Behavior Acquisition in Artificial Agents

Christian Thurau

Dipl.-Inform. Christian Thurau
AG Angewandte Informatik
Technische Fakultät
Universität Bielefeld
email: cthurau@techfak.uni-bielefeld.de

Abdruck der genehmigten Dissertation zur Erlangung
des akademischen Grades Doktor-Ingenieur (Dr.-Ing.).
Der Technischen Fakultät der Universität Bielefeld
am 25.10.2006 vorgelegt von Christian Thurau
am 13.12.2006 verteidigt und genehmigt.

Gutachter:

Dr. Christian Bauckhage, Deutsche Telekom Laboratories, Berlin
Prof. Nando de Freitas, University of British Columbia, Vancouver, Canada

Prüfungsausschuss:

Prof. Dr. Gerhard Sagerer, Universität Bielefeld
Dr. Christian Bauckhage, Deutsche Telekom Laboratories, Berlin
Prof. Nando de Freitas, University of British Columbia, Vancouver, Canada
Dr. Robert Haschke, Universität Bielefeld

Gedruckt auf alterungsbeständigem Papier nach ISO 9706

Behavior Acquisition in Artificial Agents

Christian Thurau
Applied Computer Science
Bielefeld University
P.O. Box 100131, 33501 Bielefeld, Germany
cthurau@techfak.uni-bielefeld.de

October 25th, 2006

Acknowledgements

This work would not have been possible without the help and support of many people. First of all, I would like to thank my supervisor Christian Bauckhage. I could not have imagined a better advisor for my thesis. Christian continuously encouraged and trusted me to develop my own ideas, while ensuring that I was following a reasonable path. I'm very grateful to Nando de Freitas who agreed to review this thesis and despite his tight schedule also found the time to attend my defense.

I would also like to thank the people of the Applied Computer Science Group at Bielefeld University. Especially, I would like to thank Gerhard Sagerer for supporting me, giving me the opportunity to present my work at many conferences, and for creating the infrastructure necessary for carrying out this work. Over the last year I enjoyed a close collaboration with Bernard Gorman from the Dublin City University. Not only was it a great pleasure to work with Bernard, it was also a lot of fun to meet him at conferences or during his research visit in Bielefeld. I would further like to thank Dirk Stößel, Ingo Lütkebohle, Volker Wendt, Sven Wachsmuth, Marko Tscherepanow, Katharina Rohlfing, Britta Wrede, Thomas Schack, Wulf Albers, and Jan Frederik Maas for helping me at different stages of my work. Also, many thanks to Dirk, Volker, and Ingo (again ;-), Frank Hegel, Sebastian Wrede, Oscar Carrasquero, Marc Hanheide, and Nicolas Gorges for making office such an inspiring and fun place to work at.

I deeply appreciated the interdisciplinary atmosphere at the graduate program "Strategies & Optimization of Behavior". I'm most grateful for the financial support and for the great ideas how to develop my own work by interdisciplinary research.

On the personal side, I would like to thank my family, my parents and my sisters, and Anke, for their assistance, affection, and emotional support.

Table of Contents

Chapter 1

Introduction

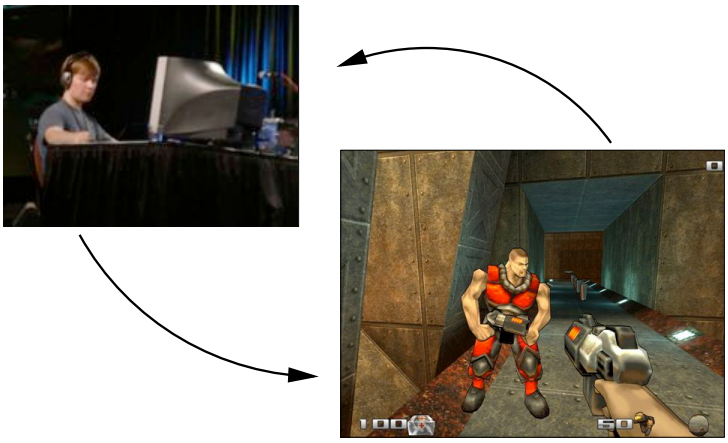


Figure 1.1: A human game player controlling a virtual avatar in a simulated environment

Looking at seemingly simple life-forms like mosquitos it astonishes the thorough observer that the behavior of biological organisms is in many respects so much superior to the performance of technical systems. This is especially striking in the case of adaptive behavior in complex and dynamically changing environments. Even a mosquito is able to fuse motor programs, goal-oriented behavior (food-acquisition) and reactive skills (evasion) in environments changing by the minute which outperform the capabilities of any existing technical system by far. Despite all insight of engineering, behavioral, neurological or biological or psychological science there is still much ground to cover in order to close the obvious gap between the natural abilities of biological organisms and man-made artifacts. Regarding this behavioral gap, robust autonomous behavior can be regarded as a key property of biological organisms that has not been satisfactorily built into a technical system, yet.

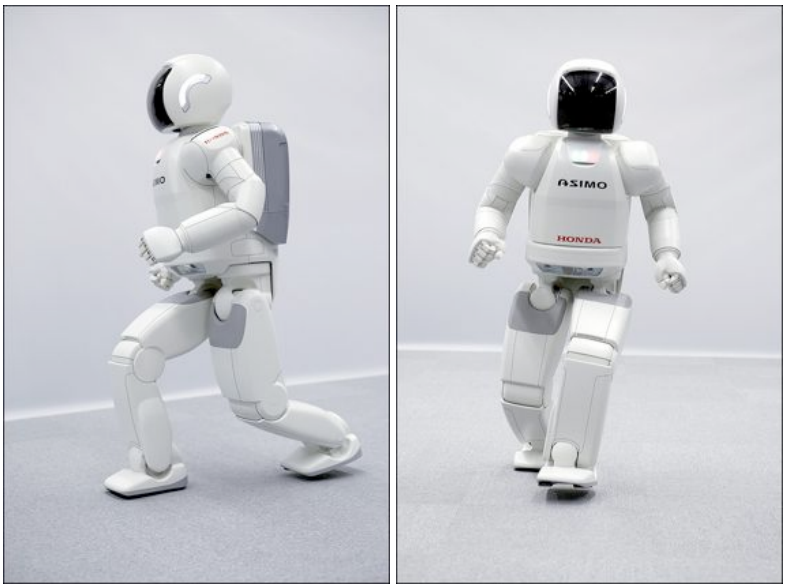


Figure 1.2: Honda humanoid robot.

Autonomous behavior in animals and humans integrates a number of different skills; most notably goal-orientation, sensor interpretation, reflexes, and transfer of planned actions into actual motor commands. Considering autonomous behavior of artificial systems in real-word scenarios, this ultimately leads to a number of disjoint problems.

First, we have to realize an appropriate sensing system allowing world perception. Although the last years have seen an impressive progress in computer vision, sensor fusion, and speech understanding, perception at a human level still seems to be out of reach. Consider for instance team-sports where the apparently simple task of tracking individual players still poses manifold challenges for computer vision systems [Okuma et al., 2004, Figueroa et al., 2004].

Second, we have to develop an actuator system for world interaction. The capabilities of robots increased massively throughout the last years. For instance, the research laboratories of the Honda corporation Japan developed an impressive robot called 'Asimo' in 2000, see also Figure 1.2. Asimo is able to walk, climb stairs, and manipulate simple objects, thus showing already very good capabilities in real world interaction. Nevertheless, what applies to the world perception of artificial systems is also fact for actuator systems: interacting with the world at a human-level still offers an incredible challenge.

Third, we have to elaborate the underlying intelligence responsible for evolving smart behavior. The Artificial Intelligence community now worked to achieve the goal of smarter machines for several decades, so far with only mediocre success

considering the high expectations back in the seventies. In fact, making machines more intelligent is often considered to be the hardest part in designing artificial systems, notably artificial intelligence at a human level is only to be found in science fiction movies.

1.1 Virtual environments

In this work we concentrate on *machine intelligence* in *simulated environments*. The most important advantage is that by using a simulation, we can concentrate on the *behavioral aspects* of intelligence and avoid the difficulties in sensor interpretation and actuator control.

Approaching behavioral intelligence in simulated environments leads to an important question: can we really compare human behavior in real environments to human behavior in virtual worlds? As noted by Wolpert et al. [2001], from a computational viewpoint, the brain is a system that converts input into outputs. Consequently, it shouldn't matter if the input comes from a virtual world and the output translates into virtual movements. The underlying basis of action and intelligence is very likely to be the same. Up to a certain level, it shouldn't matter to a human whether he is walking in a real environment or steering a character through a virtual one. However, designing a simulation with the goal of behavior modelling in mind is difficult. Self-made simulations are often critized for concentrating on only a few selected aspects while neglecting others, thereby obviously simplifying the task at hand. At worst, we can not in all kinds of simulations expect truly human behavior as intended. As soon as the level of abstraction of a simulation tends to get too high, it is more likely that it is not human behavior or cognitive abilities showing the best performance. An excellent example is the game of chess. In chess most humans are easily outclassed by machines simply because the game allows for a tree-like structuring of game states and thereby enters a domain that is extremely well suited to the fast computational processing as provided by modern computers. The true challenge in behavior modelling of artificial systems is found in the less abstract environments where the real-world can be considered the ultimate challenge.

With the task of human behavior modelling in mind, we believe that the virtual environment needs to fullfill certain requirements to actually call for human behavior: we need an environment where human actors pursue real goals in a realistic environment and have developed profound skills. These requirements are generally not met by self designed simulations. Fortunately, modern *computer games* provide an environment that comes very close to our expectations and have for some time now been considered a great tool in researching human-level intelligence [Laird and v. Lent, 2000]. As noted by Juul [2005], computer games are *half-real* - they are real in that they consist of real rules and that winning or losing a game is a real event. On the other hand, games take place in virtual worlds and as such are truly

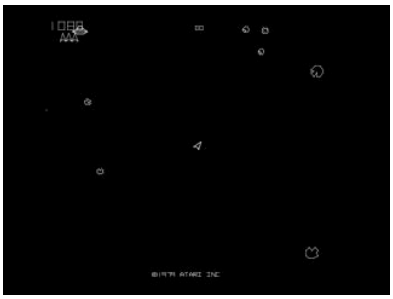
fictional. Modern games provide a level of immersion that can not be compared to classical card or board games. For instance, it can not only be difficult to get the attention of a player during an ongoing game, players are often mixing up the virtual environment with real-world actions by, for instance, trying to physically look around virtual corners. For the reader not familiar with computer games we have to point out that modern games are not to be confused with the games that appeared two decades ago as they try to create more and more realistic environments inspired by the real world (see also Figure 1.3 for a quick comparison). Thereby games provide a new and challenging environment and close the gap between completely artificial simulations on the one side and the real-world on the other side.

In this thesis, we used the popular computer game QUAKE II® (developed by ID Software) as an experimental environment, see Figure 1.1. It belongs to the game genre of First-Person-Shooter games that is well known for trying to achieve more and more realism and greater levels of immersion. The game will be explained in detail in Chapter 2.

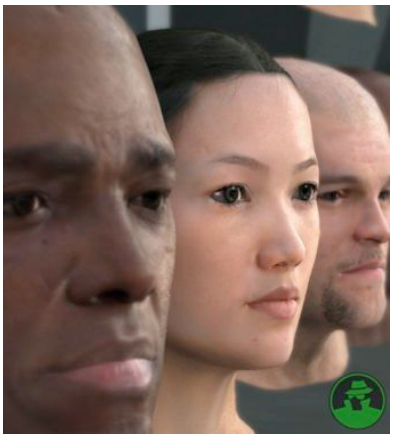
1.2 Behavior acquisition through imitation learning

Behavior defines how a human or a virtual character acts in his environment. The problem of behavior modeling in artificial systems can be approached by different means. We could specify suitable behaviors in advance (for general reading on behavior based approaches see for instance [Arkin, 1998]), or let them acquire at run-time using methods such as reinforcement learning (for a good introduction, we recommend [Sutton and Barto, 1998]). We decided on a different approach, i.e. *imitation learning*. Imitating someones else's behavior is a fundamental mean of skill learning in humans. Lately, imitation learning gained a lot of interest in the humanoid robotics community and is widely believed as one of the most promising behavior acquisition methods [Schaal, 1999]. Here, we do not follow classical robotics' approaches by directly observing a human teacher's actions. Instead, we record and store human behavior and afterwards apply methods suited for learning by example. Thereby we also skip requirements of classical imitation learning approaches, for example, the mapping of an observed action onto ones own body. The used methodologies to learn and synthesize behaviors are taken from the fields of *machine learning* and *pattern recognition*. Still, we have to decide on specific methods to apply and refine.

The past years yielded an increasing interest in psychologically and biologically motivated approaches for computational learning tasks. In many settings, the success of biological organisms over engineered machines is apparent. Therefore, concepts in humans and animals found their way into the computer sciences. For example, the concept of neurons specialized on spatial mapping, so called *place cells*, found its way into robot navigation [Arleo, 2001], or the biological structur-



(a) ASTEROIDS® (1977)



(b) CRYSIS® (2006)

(c) Crysis (2006)

Figure 1.3: Computer games evolved: starting with ASTEROIDS® , see Figure 1.3(a), nowadays games such as CRYSIS® , Figure 1.3(b) and Figure 1.3(c), do not only create realistic environments but also have more and more real-looking artificial humans inhabiting these virtual game-worlds.

ing of movements with *movement primitives* [Ghahramani, 2000] were applied for action synthesis in virtual characters [Fod et al., 2002]. Apparently, it is often by copying of biological mechanisms that good results in control problems in artificial agents can be obtained. Since this project was funded by the interdisciplinary graduate program "Strategies & Optimization of Behavior", we enjoyed an inspiring collaboration with the faculties of Economics, Psychology, Neuro-science, and Behavioral-Biology of the University of Bielefeld. This had a huge impact onto the choice of methods used within the presented work. A lot of the presented approaches are directly motivated by biological and psychological findings.

1.3 Organization of this thesis

Behavior acquisition in artificial agents is a topic of great interest. When acquiring behaviors from observations of a human teacher the underlying view on what be-

haviors are, and how the observation data implicitly encodes them is important. In Chapter 2 we introduce the principal theoretical view on *behavior acquisition*. On the one hand, behavior is interpreted as a problem of *function approximation* i.e. fitting functions according to "game situation - human action" (more general: state-reaction) observation pairs. On the other hand, we interpret behavior as a problem of *probabilistic action selection*, which often comes down to density estimation for conditional state-action probabilities. In probabilistic action selection, we rely on a set of learned or predefined prototypical actions. It was found that two distinct views on behavior acquisition are indeed necessary, since not all considered behaviors could be learned with a single approach. In fact, as the later chapters will show, certain behaviors can only be presented convincingly as a regression model while others require appropriate action selection. To give an idea how this work relates to current research, we summarize related work in the game-AI/learning-in-games domain. Finally, we introduce and explain the used experimental environment. Since the used environment has different behavioral requirements, e.g. the agent has to follow long-term goals while at the same time short-term reactive skills are equally important, we had to provide a first separation of behaviors into different layers, i.e. *reactive*, *tactical*, and *strategic behaviors*. This allows for concentrating on more primitive behaviors in each layer instead of a global but unfeasible "learn everything" approach. The subsequent chapters therefore deal with each separate layer developing, testing, and refining various approaches.

In chapter 3, we elaborate approaches for acquiring *reactive behaviors* in artificial agents. Reactive behaviors are instantaneous reactions on a given situation with a short-term effect e.g. simple movements or, in the case of QUAKE II®, aiming behaviors. We follow the underlying behavior learning idea by presenting two separate approaches. The first approach implements behavior as a state-action regression model. Different combinations of artificial neural networks serve as universal function approximators. The second approach realizes behavior as probabilistic action selection where the priors are learned from observations of humans controlling virtual avatars. Furthermore, we give a short introduction to biological place cells and movement primitives and explain the connection to the presented approaches.

In chapter 4 the focus is on *tactical behaviors*. Tactical behaviors are in our case actions dependent not only on the environment but also on other players' actions. This obviously requires a modified state representation and also more sophisticated learning approaches to deal with dynamic situations. Again, two distinct approaches are elaborated. We use a Mixture-of-Experts neural network architecture to imitate tactical situation handling where each expert represents parts of a learned regression model. The second approach utilizes extensive preprocessing of data characterizing the situation to allow for adequate action selection in dynamic situations including other avatars.

Approaches for imitating *strategic behaviors* are presented in chapter 5. Strate-

gic behaviors implement long-term goal achievement. We utilize an approach based on learned artificial potential fields to guide an agent through its world and to achieve goals. In QUAKE II® this comes down to collecting various items that are distributed throughout the game world. A player simply tries to maximize its profit by reaching and picking up the most valuable items. Since the value of items changes with varying game states, the potential field guidance has to be adequately adapted. In addition, we introduce a Bayesian approach inspired by recent psychological findings on imitation learning in infants.

A first look on a more thorough *learning architecture* is presented in chapter 6. Here approaches from chapter 3 and chapter 5 are fused and integrated. Although the focus of this thesis is on acquisition of single behaviors, we decided to include first results towards the integration of behaviors operating on different time-scales. In cooperation with Bernard Gorman from Dublin City University we furthermore present experiments on *believability testing*. We asked humans to judge from video recordings how human-like the actions of the implemented artificial agents are. This showed to be a very good indicator for the performance of behavioral models which is in general very difficult to measure. The results clearly show that human behavior can be successfully mimicked, and that completely observation based approaches are very well suited for doing so. Chapter 7 finally summarizes and concludes this thesis. Nevertheless, the task of computational imitation learning still poses various challenges. An outlook on future work, with a special emphasize on more global learning architectures will give an impression on what to expect in upcoming years.

Chapter 2

Behavior modeling & acquisition

The process of *behavior learning* in humans can take place in at least two distinct ways. On the one hand, we explore our environment and constantly learn and integrate knowledge by received feedback. For example, we learned to avoid fire or sharp edges because of negative feedback. On the other hand, humans developed *social learning*, i.e. learning from a teacher, learning by observing others, or learning by communicating. Social learning enabled us to evolve a rapid progress in cultural and technological advances, while feedback learning helps us in avoiding possible dangers in every day life.

Behavior acquisition in robots and simulated agents has been a topic of increasing popularity throughout the last years. Most methods are inspired by either feedback learning or social learning. Feedback learning methods, as for instance *reinforcement learning* [Sutton and Barto, 1998], usually suffer from a very long learning time, and thus are generally not considered an appropriate way of acquiring complex behavior. Instead, they are more useful for adapting existing skills. The process of bootstrapping behavior can be done more effectively using computational social learning methods, for example *imitation learning* [Schaal, 1999]. By imitating someone else's behavior one can avoid the expensive trial and error learning process in reinforcement learning, and thus even complex behavior can be learned in reasonable time.

In this chapter, we develop a computational approach for imitation learning or *learning from observation*, and thereby ground the approaches elaborated in the subsequent chapters. Consequently, in the first parts of this chapter, the idea of how to interpret behavior in a computational imitation learning method are explored. Although imitation learning has been known for quite some time, it was rarely used in the context of a multiplexing of goal oriented, reactive, and tactical behavior, which is essentially what we do when bootstrapping the behavior of *computer game agents* acting in complex game worlds. Therefore, we had to develop a more universal applicable foundation. However, since behavior can only be interpreted in a specific context, we can not completely neglect the connection to the used experimental

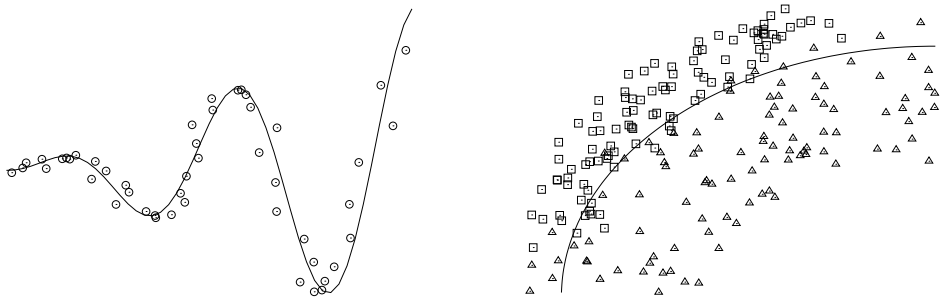
environment. Therefore, in the later sections of this chapter, we introduce the commercial computer game QUAKE II® that serves as an experimental environment in this thesis. We explain the game itself, its rules, and behavioral requirements, and we explain why we actually decided on a computer for the task of behavior learning in artificial agents. To compare our own work to the current research, we also summarize well known work from the AI, and learning in games literature.

2.1 Computational behavior acquisition

From a computational viewpoint, *behavior* can be understood as finding an appropriate *reaction* to a given world *situation*. Meaningful sequences of reactions can be interpreted as behavior. The idea of a *meaningful* behavior is usually not contained within the computational approach, instead, it is merely a human interpretation of a given action sequence. Ideally, a human observer could label a learned and generated action sequence as a specific behavior. Obviously, such an interpretation of actions is rather subjective. In this work, we concentrate on a couple of selected behaviors in the context of games. Since the behaviors are learned from the observations of virtual avatars controlled by humans, we take these behavioral recordings and interpretations as ground truth.

In most tasks that have to be carried out by artificial creatures in robotics or virtual environments, deliberative approaches were preferred so far. They are easy to develop, since the required behavior can be modelled top-down by an expert. When developing agents in a top-down approach, we could specify a set of common situations in conjunction with an optimal reaction (in the following, we often replace situation by *state vector* and reaction by *action vector*, since both can be interpreted as points in a p dimensional Euclidean state-action space). The simple top-down solution leads to an optimal behavior, at least in pre-known situations. However, this way of behavior modeling is not only time-consuming, it is also getting difficult with an increasingly complex state-action space, since it is impossible to think of all known state-action pairs in advance. To deal with new situations, we would have to extract *rules* and *patterns* from already known situation-action data. This is exactly what we want to achieve in this work. We understand behavior as implicitly defined by sets of state-action observations. Our goal now is to find an underlying mapping of state vectors on action vectors that also works for novel situations. As already noted in the introduction of this thesis, we interpret learning from observation as a *regression* problem on the one hand (action as a situation dependent function), and as a *classification* problem on the other hand (behavior as situation dependent action selection). As we will see in the later chapter, both interpretations are required.

The basic idea is simple: based on observations of a human, an artificial agent should learn how to react in a novel situation - ideally this would lead to a clone



(a) Regression (b) Classification

Figure 2.1: Two views on behavior learning: 2.1(a) in the regression task, one is interested in fitting a function into a set of data points, respectively behavior observations of a human player; 2.1(b) the classification task searches for boundaries between classes of data points, in the case of behavior learning, actions are classified based on game state observations.

of the observed human. The environment's state is represented by a state vector $\mathbf{s}_i$, the reaction by an action vector $\mathbf{a}_i$. Pairs of states and reactions $\{\mathbf{a}_i, \mathbf{s}_i\}$ denote points in a p dimensional Euclidean state-action space. The problem of reaction synthesis and behavior learning can thus be understood as a regression problem, where a reaction $\mathbf{a}_i$ can be expressed as a function of the world's state $\mathbf{s}_i$:

$$\mathbf{a}_i = f(\mathbf{s}_i) \tag{2.1}$$

Figure 2.1(a) illustrates a simple didactic regression example, based on a number of observations (the small dots) the underlying behavior function should be deduced. Since behavior follows certain temporal dependencies, we have more likely a functional dependency on the last n state vectors:

$$\mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}) \tag{2.2}$$

where $\mathbf{a}_{t+1}$ denotes the reaction at timestep $t+1$, and $\mathbf{s}_t$ denotes the world's state at timestep t respectively. Certain aspects of the game's state will remain unobservable or unpredictable, they will be further denoted as environmental influences $\mathbf{e}_t$. Thus, the complete model can be written as:

$$\mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{s}_{t-1} \ldots \mathbf{s}_{t-n}, \mathbf{e}_t, \mathbf{e}_{t-1} \ldots \mathbf{e}_{t-n}) \tag{2.3}$$

Instead of modelling behavior as a problem of function approximation, we could also interpret behavior as a problem of appropriate context dependent action selection. This means, given a set of u discrete player actions $A = \{\mathbf{a}_1, \mathbf{a}_2 \ldots, \mathbf{a}_u\}$, we want to find the action $\mathbf{a}_i$ corresponding to a state vector $\mathbf{s}_i$, see also Figure 2.1(b) for a small didactic example on classification.

Here, we decided on a probabilistic classification approaches. A *Bayesian* view

on behavior acquisition has certain advantages. Bayesian approaches have been shown to be robust for various tasks; for example, they have been applied with great success in speech analysis, computer vision, or behavioral imitation [Rao et al., 2004]. Moreover, recent results indicate a connection between human reasoning and Bayesian statistics, for instance, humans internally represent a statistical distribution of tasks and uncertainty in sensors for sensorimotor learning [Körding and Wolpert, 2004]. It was also suggested that humans integrate, update, and access knowledge in a logic similar to Bayesian statistics [Griffiths and Tenenbaum, 2006, Tenenbaum et al., 2006]. Thus, it seems promising to explore the applicability of Bayesian classifiers for imitating complex human behavior. In the Bayesian reformulation of the behavior learning problem we have to pick the action $\mathbf{a}_j$ maximizing the following probability:

$$\mathbf{a}_{t+1} = \underset{j}{\operatorname{argmax}}\, P(\mathbf{a}_j|\mathbf{s}_i) \tag{2.4}$$

where $\mathbf{a}_{t+1}$ denotes the selected action for the next timestep, $\mathbf{a}_j \in A$, and $P(\mathbf{a}_j|\mathbf{s}_i)$ denotes the probability of selecting $\mathbf{a}_j$ if a state $\mathbf{s}_i$ is observed. When again taking the last state vectors $\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}$ and environmental influences $\mathbf{e}_t, \mathbf{e}_{t-1} \ldots \mathbf{e}_{t-n}$ into account, the action at timestep $t+1$ should be picked as the action $\mathbf{a}_j$ maximizing:

$$\mathbf{a}_{t+1} = \underset{j}{\operatorname{argmax}}\, P(\mathbf{a}_j|\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}, \mathbf{e}_t, \mathbf{e}_{t-1} \ldots \mathbf{e}_{t-n}) \tag{2.5}$$

where $\mathbf{s}_t$ denotes the state at timestep t, and $\mathbf{e}_t$ denotes environmental influences at t.

For both suggested models a computational behavior imitation approach consists of three steps: first, we have to record actions $A = [\mathbf{a}_1, \ldots, \mathbf{a}_k]$ corresponding to state sequences $S = [\mathbf{s}_1 \ldots \mathbf{s}_k]$. Then, we have to extract a prediction model based on the recorded state-action vectors. Finally, reaction predictions for new state vectors can be used to synthesize behavior in test environments, and thereby evaluate the learned behavior.

The simple nature of equations 2.3 and 2.5 may imply a quick ad-hoc solution. However, we have to keep in mind that we are dealing with human behavior after all. Despite considerable efforts in robotics research that lead to automatic skill acquisition capabilities, for example, a tennis-swing [Schaal, 1999] or a button pressing [Breazeal and Scassellati, 2002], imitating human behavior in artificial systems is generally considered a difficult task. Furthermore, the more complex behaviors that are considered in this thesis provide a new challenge to computational behavior learning approaches.

Although learning from observation seems to be a common way of behavior acquisition, surprisingly little was contributed for games or similar domains. This might be because of certain principal problems of applying machine learning ap-

proaches for behavior acquisition. Although the two very general models might suggest otherwise, it is up to now completely unclear how to actually extract a suitable behavior model from human observation data. Moreover, due to the versatility of human behavior in complex environments, we are very likely to depend on huge quantities of training data to make use of classical machine learning methods.

Regarding the problem of training data availability, modern computer games provide a unique possibility. The popularity of networked games leads to millions of people playing multiplayer games over the Internet. The network traffic contains information about what the player perceives, the actual world state, and how he reacts to it. This exactly conforms to our definition of behavior: implicitly, the network traffic of Internet based games encodes human playing behavior. Given the idea of interpreting network traffic as state-action observation pairs, the problem of training data availability vanishes.

Still, it remains unclear which methods might be suited for the task of behavior learning. Consequently, in this thesis, we test and refine various behavior modelling approaches. All approaches try to find appropriate expressions for equation 2.3 and 2.5, respectively. A direct approach of finding these models showed to be unrealistic, at least at the level of complexity of the considered behaviors and environments. Since we aim at behavior modeling from scratch, we do not use predefined skills and we avoid the usage of expert knowledge as much as possible (in fact, only the process of selecting appropriate features was optimized by hand). Using predefined skills might greatly simplify the creation of complex behavior models but it also may result in unrealistic acting. The basis for our learners are the most basic observations possible; everything learned is solely data dependent.

2.2 From strategies over tactics to reactions

Behavior learning and modelling is a highly task dependent problem. In robotics, people usually tend to concentrate on only one specific behavior or a number of similar skills. In contrast, complex environments, such as encountered in modern computer games, require a broad variety of different behaviors. For instance, players might have to look for items, engage in combat, or jump over ledges. Therefore, as mentioned earlier, a single monolithic "play the game" behavior for artificial agents showed to be unrealistic (as it probably would be for robotic systems in real world scenarios). Still, some behaviors are very similar to others and can be grouped. Thus, it seems reasonable to combine and separate behaviors into behavioral categories, in order to make the problem of computational behavior learning feasible.

Based on a widely acknowledged psychological hierarchy of human behavior [Hollnagel, 1994], we decided to discriminate behaviors into three different categories. These categories are: *strategic behavior*, for achievement of longterm goals,

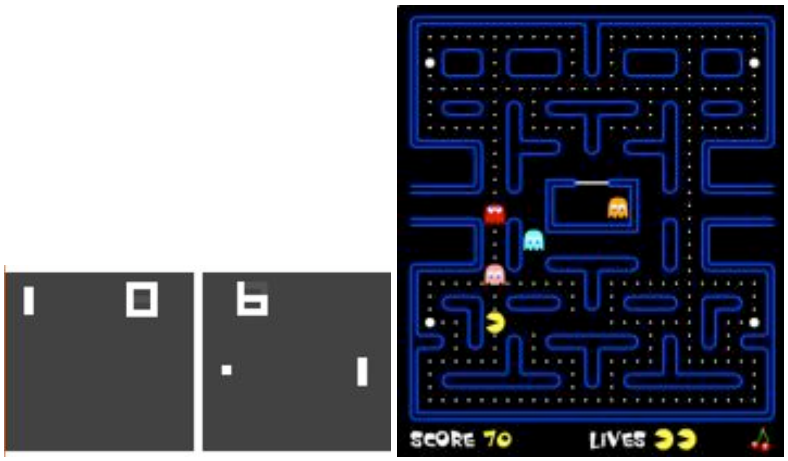
tactical behavior, for mid-term goal achievement, and *reactive behavior*, which denotes simple movements and solely sensor dependent actions. In Section 2.4, we explain in detail how these layers can be interpreted in our experimental environment.

Layered approaches for behavior execution are known for quite some time and are commonly used in behavior based robotics, see for instance [Arkin, 1998] for a brief review on traditional deliberative planners. Regarding simulated environments, Laird [2001b] introduced a behavioral structure for QUAKE II® which is similar to the introduced behavioral layers. Unlike Laird, we interpret the behavioral layers as concurrently active and not as a hierarchy. Interpreting the layers as a hierarchy implies a functional dependency of cognitive higher level layers (strategies) on cognitive lower level layers (reactive). Although we obviously also have to synthesize actions, and thereby, for example, translate strategies into actual movements, action synthesis is not to be confused with reactive behavior. We interpret certain observed behaviors as being reactive, tactical, or strategic. For instance, an aiming behavior can be seen as reactive since it mostly depends on the current position of an opponent. In contrast, picking up an important item is a strategic behavior, because it usually involved (strategic) planning. Both need to *synthesize actions* in order to execute the associated behavior. Each layer is responsible for translating desired reactions into actual in-game movements. Thereby, it is possible to learn and generate behavior in each layer independently of each other. Dividing behavior into different layers leads to the question, on how to integrate behavior from a multiplexing of reactive, tactic, and strategic behavior. We present a first approach that combines aspects of strategic behavior learning and reactive motion synthesis in Chapter 6.

2.3 Behavior learning in games

As Laird [2001b] pointed out, computer games are an excellent tool for researching human-level intelligence in artificial systems, even more so than conventional simulations, since humans usually perceive games as more real than simulations. As mentioned before, computer games are half-real, something in between reality and simulation, which obviously provides interesting aspects for elaborating behavior learning methods. We can expect humans to act in a much more natural way than what we are used to in other simulated environments. After all, an experienced human player encounters an environment he knows well, he is used to it, and has already developed profound skills. These are typical characteristics of human behavior, usually only found in behavior acquired in the real-world.

The game we use in this work is taken from the popular genre of *First-Person-Shooter* (FPS) games. FPS games try to create a realistic 3D simulation that is inspired by the real world, in which humans control virtual avatars. These games



(a) Pong (b) Pac-Man

Figure 2.2: The famous classical games Pong 2.2(a) and Pac-Man 2.2(b).

are complex and learning to master them takes its time, even in humans. We could have made the task of behavior modelling a lot easier by selecting a much more simplified game, for example, Pong or Pac-Man (see Figure 2.2). However, developing a human-like player for Pong would not only be simplistic behavior-wise, it would probably also lead to a player being worse than the actual computer player. In QUAKE II®, the opposite is true as conventional game agents in FPS games are not only less fun to play against, they are also pretty weak opponents. While their aim and reflexes are superhuman, they clearly lack more sophisticated playing skills. Consequently, developing a human-like agent for FPS games is not only a challenging research topic, it would also impact the computer game industry by creating more realistic gaming experiences.

From the behavior modelling perspective, the realistic FPS games might provide even more appealing aspects than other games or environments. Modelling a player for QUAKE II® brings us much closer to modelling of human intelligence than creating, for example, a virtual player for an abstract tennis game like Pong, or than modelling an agent for an arbitrary simulated environment. Imitating humans wandering around a 3D virtual world and performing tasks of different complexity might provide further insights on the modeling of human behavior in general, since the level of abstraction compared to the real world is a lot lower than in Chess, Tetris, or conventional simulations.

As already mentioned, the most important advantage is that by using a commercially successful game, we have access to an incredible huge database of behavioral recordings of humans. We interpret the network traffic of ongoing games as behavioral observations. In QUAKE II®, these recordings are stored in so called *demo*

Figure 2.3: Game players at a so called LAN-Party. Multiplayer network gaming is equivalent to recording training samples for behavior acquisition.

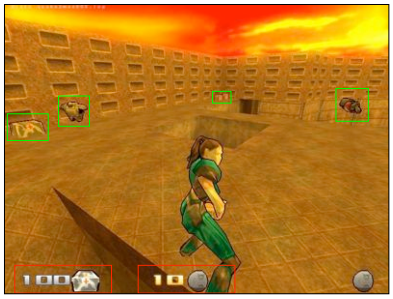
files, and the QUAKE II® engine is able to redisplay them. We can access demo files from various websites (people like other people to watch their matches) or by recording live games (Figure 2.3 shows a LAN-party, a possible location for gathering vast amounts of observation data). Having an almost unlimited amount of (noiseless) training samples, showing nothing less than humans performing complex tasks in complex environments is a unique and previously unrecognized opportunity.

The training data we are dealing with are records of the network traffic. They contain information about the exact locations (x, y, z) the player assumed, nearby items, and other players. Temporary entities like sounds and flying projectiles are included as well. There is no need for a visual analysis of game scenes, since all necessary information is already available on a higher cognitive level. This also applies to player actions which are included as velocity and position vectors. Before coming to the details of how to make use of behavioral recordings, we first want to introduce our experimental environment QUAKE II® in more detail. The reader already familiar with FPS games can safely skip the next section.

2.4 Experimental environment

QUAKE II® belongs to the popular First-Person-Shooter (FPS) game genre. The

(a) Item places ...



(b) ... can be reached easily ...

(c) ... or require special maneuvers

Figure 2.4: A few typical game situations. A player, the character in the foreground, perceives the world in a first-person perspective (for clarification purpose we decided for a third person perspective for visualization). Numerous items, highlighted by green boxes, can be picked up to increase player stats, health and ammunition highlighted by red boxes. Some items can be reached easily, other require acrobatic skills, as the jump visualized by the blue arrow in Figure 2.4(c).

game concept is rather simple. Basically, in a First-Person-Shooter, player avatars' navigate a simulated 3D world with the task of surviving and shooting enemy players. Game agents can be controlled either by human players or by programs. Artificial players, game avatars controlled by the computer, are often referred to as *gamebots*.

The FPS takes place in a world that is directly inspired by the real world. However, especially in games such as QUAKE II®, game-play is favored over realism. This allows certain extreme moves, for example, very high jumps, or falling down from great heights. The game worlds themselves, in game terms *map*, appear realistic. This does not only apply to the map's architecture, but also to the map behavior, for instance, maps might have doors that can be openend, or stairs, and elevators to reach higher places. Commonly, a map consists of a number of rooms which are interconnected by hallways.

Various *items* are distributed throughout a map at fixed places and can be picked

up by players. Usually, items increase player attributes: better weapons inflict more damage, better armor allows for taking more damage, and health packages refill lost energy. Once a player's energy drops to zero, the opponent is awarded a point. The player with the most points after a fixed time period wins the game. Figure 2.4 gives a first impression on the graphical visualization of a FPS game.

At the first sight, FPS games may rely solely on reactive behaviors, namely movement and shooting. Although aiming and shooting are obviously an integral part of game-play, FPS games shouldn't be confused with the mindless shooter games (for instance ASTEROIDS®) that appeared two decades ago. In fact, FPS games demand a huge amount of strategical and tactical reasoning. These requirements arise from various aspects of game-design:

Important items are placed at fixed map positions. This implies that each player tries to claim the best items for himself while denying access to them for enemy players. This strategy of securing items is often referred to as *map control*.

All weapons behave differently. Each weapon has a specific purpose, for example, some are better at short ranges, others are better at long ranges. Appropriate weapon selection combined with adequate map control is often the key to success.

The separation of behaviors into strategic, tactical, and reactive behaviors can also be done for a FPS game. Figure 2.5 illustrates some commonly observed behaviors. In the following, we describe behaviors in each layer to give an idea of what to expect from the learning approaches elaborated later on. Explaining the considered behaviors should make clear that it is indeed complex human behavior on different cognitive levels that we are imitating.

2.4.1 Long-term strategical behaviors

"What differentiates the good players from the great are their strategies and their abilities to outsmart their opponents"
(Dennis "Thresh" Fong, author of "Thresh's Quake Bible")

On the cognitively top-most level reside strategic behaviors which are used to achieve long term goals. In our case, besides the obvious goal of winning the game, several other subgoals can be identified. Since strategies are coupled to item places, and since items always appear at fixed positions, strategies are highly map dependent.

There are a number of different items players can pick up (see Figure 2.6 for a quick introduction to items and a short description of their importance for strategies). Each item has a special purpose. In general, there are two groups of items. One group is increasing offensive attributes, for example, better weapons or ammunition, the second group is increasing defensive attributes, for instance, health packages or armor.

At the beginning of a match each player is equipped with only a weak weapon,

Figure 2.5: Using in-game observations of human players, a layered learning approach should generate life-like behaviors in artificial players.

and no armor. Consequently, most try to maximize their offensive attributes by picking up their favorite weapon, whereas cautious players try to increase their defensive attributes by picking up armor. Fixed item positions yield the possibility of waiting next to an important item. Once an item is picked up, it will disappear, and reappear after a fixed time period. By guarding important items, and shooting approaching opponents, a player can effectively lock an item so that it will be difficult for an opponent to access it.

In most cases it won't be enough to just secure one item, since there are multiple important items distributed. It is far more effective, but also more difficult, to control a whole area. For a successful strategy a player has to find the most attractive area to control. Effectively, this comes down to path and thereby item selection. Given the complex architecture of most maps this is not an easy decision in most cases, Figure 2.7 illustrates a common game situation where a player has to decide

(a) Weapons for offensive power (b) Armor for defensive attributes

Figure 2.6: Items define the most important strategical component in QUAKE II® . They either enhance a player's offensive power 2.6(a) or refill lost health and armor 2.6(b). The most important item places are essential for winning the game.

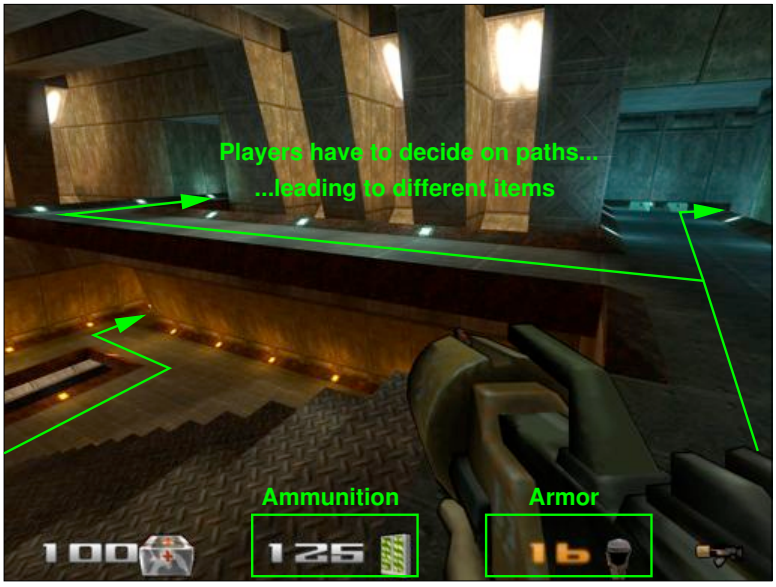


Figure 2.7: Players have to select paths, and thereby decide for items they want to pick up.

on his path at a crossing. The attractiveness of an item changes with the game state. A player has to adapt his strategy based on his own internal state. For example, if a player is low on health, it is often better to look for a health package to refill lost energy instead of guarding weapons or armor. The same is valid for the beginning of a match. A player might first head for his favorite weapon, before going to a specific area and guard it.

Learning and synthesis of strategic behavior requires two things: first, we have to identify a game state representation capable of expressing strategies. This includes a method in extracting possible goal states. Second, we have to find a way to adequately synthesize goal-reaching behavior within a live game. For the latter, a strong coupling to movement synthesis is needed. In Chapter 5, we present and evaluate computational approaches for imitating strategic behavior.

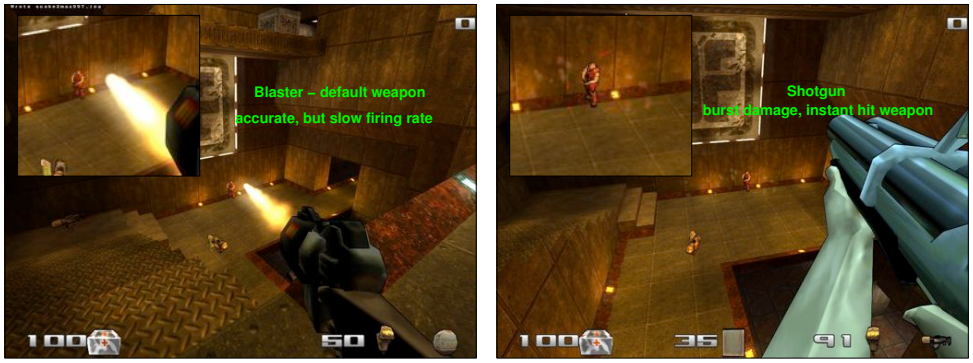
2.4.2 Mid-term tactical behaviors

Tactical behavior can be roughly characterized by smart, localized situation handling. While the strategy tells the player about the next important region on a map, the tactics are responsible for evading possible threats on the way. Tactical behaviors control important player vs. player interactions, where weapon handling, weapon selection, and in-combat movements are possibly the most interesting and gameplay relevant behaviors.

Weapon handling: QUAKE II® players can pick up various weapons where each weapon is handled differently. Weapons vary greatly in projectile flying speed and fire rate. This leads to completely different handling and thus aiming behaviors. For slow projectiles anticipation is more important than accurateness, since a future opponent's position has to be predicted correctly. Other projectiles have a huge damage radius, therefore they do not require overprecise hits. Other weapons require accurateness while anticipation is not necessary at all, for example, the notorious Rail-Gun (a visualization on some weapon's behavior can be seen in Figure 2.8).

Weapon selection: A player tactically picks his weapon based on numerous possible state variables and based on his own preferences. Often, the distance to an opponent is one of the key factors in weapon selection. Some weapons are better at close range (shotgun, rocket-launcher), while others are better at long range (railgun). Other game state variables might also be important. Since not all weapons are available all the time, a player usually has some internal weapon selection ranking, appropriate for varying game situations.

Tactical movements: When engaging in one vs. one situations players have to move tactically. Normal movements, to pick up items or just navigate the map, are very much position dependent, whereas tactical movements are centered around player and opponent positions. These close combat movements are quite important for successful gameplay. Both players try to evade and attack simultaneously. More



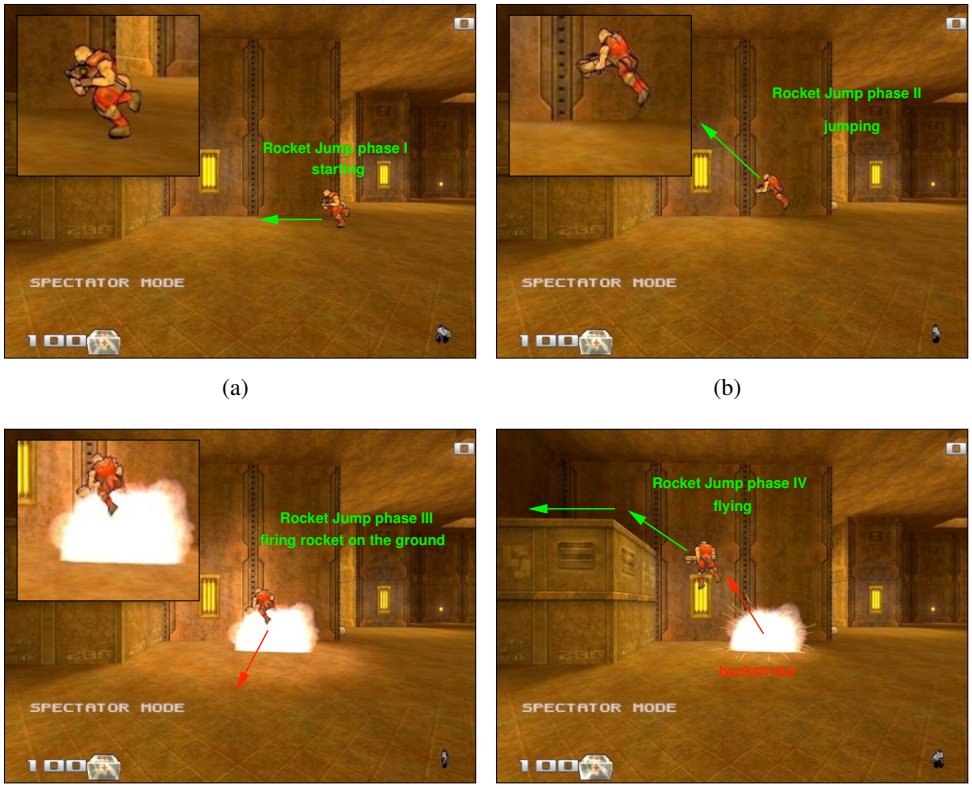
(a) Blaster (b) Shotgun

(c) Railgun

Figure 2.8: All QUAKE II® weapons behave differently. Figure 2.8(a) shows the blaster, a slow but accurate weapon. Figure 2.8(b) shows the shotgun, due to its burst-fire it is only good at close range. Figure 2.8(c) shows the rail gun, which is very accurate at all ranges but has a slow firing rate.

advanced moves include circle strafing, a player moves in circles around an opponent while holding him in his view, but also constant side movements to become a harder target.

Obviously, a lot more and other tactical behaviors exist. The sheer amount makes it impossible to include or mention them all within this work. However, some tactical behaviors that could be considered in the future are tactical use of the environment (hiding behind boxes or walls), anticipatory behavior (prediction of opponent positions), or special behaviors, for instance, an ambush. In Chapter 4 we present approaches on learning tactical behavior.



(a) (b)

(c) (d)

Figure 2.9: A rocket-jump. A player fires a projectile on the ground while simultaneously jumping, thereby leveling him up to much higher places than otherwise reachable.

2.4.3 Instant reactive behaviors

The cognitively most basic layer is reactive behavior. Here we find simple reactions to audio-visual percepts as well as sequenced moves and jumps [1].

By popular definition reactive behaviors are immediate reactions on a state vector $\mathbf{s}$ [Arkin, 1998], generally, this also holds for games. There are arguably a number of different situations requiring instant reactions of a player. Obviously, for a FPS game, a player has to aim and fire weapons, but he also has to safely navigate the game-world. Most maps contain a complex room architecture and various dangerous areas, ledges, or pitfalls. Thus, a player has to avoid pitfalls and make sure he won't run into walls.

In addition, numerous other, more complex, moves can be observed. In order to

[1] It should be noted that all moves are the result of mouse and keyboards input commands. Consequently, the modelling of human motions skills and reactive behaviors is in fact an abstract learning of human mouse movements and keyboard inputs.

reach certain places or items, players have to jump. Besides ordinary jumps, there exist more complicated and thus harder to master movement sequences. With these moves, for example the notorious rocket- (see Figure 2.9) or strafe-jump, a player outsmarts the game's physics by doing a simulatenous side-forward jump, which allows for larger jumps. Some of these specials are indeed hard to master and can only be executed by expert players.

Further, the visual in-game movement appearance of an artificial player is very important for a human-like impression. Although natural motion is a well known topic in computer graphics and robotics, the situation here is different. In computer graphics, smoothness of animations is achieved by interpolation and introduction of constraints. Here, we additionally have to model a specific human player. Therefore, we can not focus on smoothness alone but on player specific realistic motion synthesis. In Chapter 3, we report on approach for acquiring reactive behaviors.

2.4.4 Evaluation of synthesized behavior

Playing styles are different among human players. Most players have elaborated individual strategies and certain weapon and item preferences. While some almost completely rely on their aiming skills, others prefer more strategical play. Further, the way how players steer their characters can vary. Certain complex moves allow for taking shortcuts and thus enable players to reach places faster. Although the maximum forward velocity of all players is fixed, certain ways of exploiting game rules were found and are frequently used. Consequently, mimicking a human player does not only have to pay attention to a player's strategical and reactive behaviors, but also to the way movements and motions are executed. It is not sufficient to know where to go next, we also have to know how to induce the adequate motion to reach a place.

Evaluation of learned behavior is an important, yet difficult problem. Although our main goal is the synthesis of *human-like* behavior in artificial game-agents, it is not ultimately clear when we reach this goal. Obviously, the Turing test, i.e. in our case human observers decide upon the humanness of an artificial game agent, is a probable mean for evaluation. However, as long as we are dealing with individual behaviors, it is very unlikely that an artificial agent could convince a human observer. The variety in behavior might be too small to create the illusion of a human player. Consequently, we decided for different evaluation mechanisms. We evaluate each approach with respect to the intention of the generated behavior. This means that we measure the accurateness, with respect to a human teacher, of a reproduced action for reactive behaviors, or that we measure the goal success for strategic behavior. Since tactical behaviors are in between reactive and strategic behavior, we measure both the accurateness and, in the case of tactical movements, the similarity to an actual human movement. For an approach towards integrating different behavioral layers and thereby creating more versatile acting, in Chapter 6, we conduct

a believability testing, similar to the mentioned Turing test.

However, the importance of generating believable, human-like actions can not be completely neglected for the imitation of single behaviors. Therefore, we report on the visual appearance with respect to humanness of each synthesized behavior. Although this is a rather subjective way of evaluation, it should give an idea how close we can get to our ultimate goal of creating human-like behavior.

A direct comaparison of, for example, reactive and tactical learning approaches is not intended. This is also not very likely to produce meaningful results, since each approach is targetted at specific problems within its layer. The individual behaviors in each layer are usually too different to be comaparable.

2.5 Related work

For summarizing related work, we decided to explicitly concentrate on the domain of behavior modelling in games and virtual environments. Behavior modelling for artificial creatures in real world environments is up to now not comparable to behavior modelling in virtual environments. Due to noiseless sensor data and virtual actuators, far more sophisticated behavioral patterns can be realized in simulations. Obviously, this leads to approaches which are in most cases not comparable.

Games are big business [Cass, 2002]. Non-player-character (NPC) control obviously demands some sort of Artificial-Intelligence (AI). While the AI for games like Pong or Pac-man was rather straight forward and could be programmed in a few lines of code, the situation changed completely with the increasing game complexity. Nowadays, game AI is an integral part of the whole production process. Numerous book contributions deal with general and specific AI approaches in games [Funge, 2004, Champandard, 2004], most notably the "Game Programming Gems" [DeLoura, 2000, 2001, Treglia, 2002, Kirmse, 2004, Pallister, 2005, Dickheiser, 2006] and "AI Game Programming Wisdom" [Rabin, 2002, 2004, 2006] series.

The next sections gives a comprehensive overview over common game AI approaches. First, we briefly introduce more general game AI methods. Then, we concentrate on conventional First-Person-Shooter AI methods, before finally reviewing recent machine learning approaches in games, in particular FPS games. A final look on recent publications about believable, human-like game AI will close this section.

2.5.1 Game AI

Only a few years ago, classical game AI often came down to finite state machines, search-trees and the A^* algorithm. With the huge success of the gaming industry, an enormous variety of different approaches emerged during the last years. A lot of these approaches are other formulations of finite-state machines, and most of

them are targetted at specific games. The following paragraph summarizes the more general approaches.

Finite-state machines (FSMs) are often said to be the most common game AI technique [Champandard, 2004, Funge, 2004, Fu and Houlette, 2002, Cass, 2002]. Various modifications to FSMs exist, to make them more suited for specific game demands or to allow easier implementations [Tozour, 2004, Rosado, 2004]. However, it should be noted that Finite-state machines are mostly applied because of their rather simple nature. FSMs are probably the most intuitive and quickest way to approach game AI problems. Despite their simplicity, they require a time-consuming implementation, all possible situations have to be thought of in advance. This makes them very inflexible. However, a very complex finite-state machine covering a huge set of possible states and actions might be able to create convincing game AI for the casual gamer. More often finite state-machines lead to "holes" within the game AI [Spronck et al., 2002], situations where unrealistic behavior emerges from unanticipated states.

Modern games often take place in simulated 2D and 3D environments. Therefore, pathfinding is one of the major topics of commercial game AI. Pathfinding is responsible for steering characters, units, or enemies through game environments. Obviously, the calculation of collision free paths is mandatory. Besides some ad-hoc solutions to game specific path planning [Mika and Charla, 2002, Surasmith, 2002], various game AI related publications deal with the A^* algorithm [Matthews, 2002, Higgins, 2002a], its application in games [Higgins, 2002c], and improvements to decrease computational demands [Higgins, 2002b, Cain, 2002].

Also more sophisticated path planning approaches were integrated into game AI. For instance, probabilistic roadmaps are a well known mechanism for motion modelling in computer graphics and robotics [Geraerts and Overmars, 2006]. Roadmaps are a graph based approach where nodes denote entity places, and edges denote collision free paths between them. Nodes and edges can be automatically acquired in a preprocessing step. Various extensions showed the principal applicability for pathplanning in games [Nieuwenhuisen et al., 2004, Overmars, 2005].

Besides pathfinding and FSMs, scripts are another commonly found technique in commercial game AI. However, scripts are more an effective tool of implementation. Instead of hardcoding everything into the game engine, a Non-Player-Character's (NPC's) intelligence is defined in a scripting language and interpreted at runtime [Berger, 2002b,c,a, Cass, 2002]. Scripts can contain predefined movie-like action sequences, finite state machines, or other sorts of sequenced action commands [Tozour, 2002c]. Various extensions to standard scripting mechanisms were proposed throughout the last years [Barnes and Hutches, 2002, Spronck et al., 2003]. For example, Spronck et al. [2003] introduced "dynamic scripting", a method for online adaption of NPC scripts. In dynamic scripting, a set of predefined rules is weighted at runtime using reinforcement-learning. It is applied in AI

systems for role-playing games and real-time strategy games [Ponsen and Spronck, 2004]. In general, scripts are predefined (re)actions to game states. Thus, the more states and actions are anticipated in advance, the more convincing the AI will be.

2.5.2 FPS AI

Tozour [2002a] presented a generic approach for FPS AI. He describes four basic components of an FPS AI; first, the movement layer. The movement layer is responsible for determining how to move, for instance, to avoid obstacles, or follow other agents. Second, the animation layer. It is responsible for selecting character animations or playing animation sequences. Third, the combat layer. The combat layer selects and fires weapons, or decides for tactics. And fourth, the behavior layer, which is described as an overarching system, and determines goals and interacts with other systems to reach these goals. This discrimination is inspired by an already mentioned hierarchical order of behavior in robotics [Arkin, 1998], and is very similar to the structuring presented in Section 2.2. Tozour [2002a] finally gives some ideas about implementation where he suggests rather common methods for each layer. For pathfinding, the A^* algorithm is proposed. For the behavior controller, he suggests usage of a FSM where typical states could include *Patrolling, Combat, Fleeing, Searching*.

Champandard [2004] gives a very detailed overview on "AI Game Development", with a special emphasize on FPS games (see also the AI-Depot website maintained by Champandard [2006a]). In his book, the FEAR QUAKE II® AI project serves as an exemplary implementation [Champandard, 2006b, 2004]. The book describes a vast amount of AI methods, including FSMs, Fuzzy logic, but also machine learning approaches, genetic algorithms, reinforcement learning, neural networks, and Markov-decision processes. Champandard gives an overview over almost all facets of generating intelligent behavior in game agents. Unfortunately, learning from observations is only touched in a few paragraphs and was apparently not applied in games.

A very successful bot implementation for the game QUAKE III® is introduced by van Waveren [2001]. The bot is controlled by a layered architecture. The central control unit is effectively a large FSM, referred to as AI network. The approach further introduces Fuzzy logic to create less predictable opponents. Probably the most interesting novelty is the Area Awareness System. Instead of making use of classical waypoint maps, movement is based around separate 3D bounded hulls. In these defined areas, a bot can move from any point to any other point, thus movement complexity is minimal. Areas overlap, thus it can be easily measured if a movement from one area to another is possible. Summarized, the QUAKE III® Arena Bot is a good example for a well defined and strong conventional artificial opponent, although it does not create a convincing illusion of a human player. Notably, van Waveren also created the famous QUAKE II® Gladiator- and Omicron-bot [van

Waveren, 2001], in Chapter 6 the Gladiator-bot is compared against the imitation learning agent introduced in this work.

In Academia, conventional AI methods for FPS games had been thoroughly applied by Laird [2001b] using the SOAR engine. Although his approaches lead to respectable opponents for the game QUAKE II® (see also [Laird, 2001a] for an extended approach adding anticipation capabilities to an artificial player), it mostly relies on predefined behaviors which are combined by an heuristic approach using the SOAR engine [Soar, 2006]. In [Laird and Duchi, 2000] the SOAR bot is evaluated letting human players judge the humanness of the artificial agent. The assumption is that the Turing test assists in iteratively finding and adjusting behavior parameters, for example, aiming skill values, to finally create a more human-like agent. Therefore, Laird's method can be seen as the top-down approach of creating human-like game agents. In contrast, our work is approaching the same goal in a bottom-up manner by directly imitating and learning human behavior from observational data.

A BDI (Belief-Desire-Intention) agent based approach was presented by Norling and Sonenberg [2004]. BDI is considered a psychological way of agent implementation, implementing reasoning, "the way people think they think" [Norling and Sonenberg, 2004]. By definition BDI approaches combine purely reactive and deliberative techniques. Typically, a BDI system consists of an agent's beliefs about the situation, possible goals he desires, and plans on how to reach these goals. For acquiring player specific goals and plan rulesets, Norling interviewed three expert QUAKE II® players. The interview consisted of a number of questions related to situation dependent decisions ("If you'd just respawned and could hear but not see a fight nearby, what would you do?") or global opinions ("What makes a good sniping spot?"). Finally, belief conditioned rulesets serve as a basis in game agent implementations. The model itself does not include player specific modelling of atomic actions, however, global known errors in human reaction time and mouse movement errors known from HCI studies are incorporated. Norling's overall approach is indeed interesting and yielded promising experimental results. The main criticism arises from the BDI concept itself. It is questionable, if an introspective analysis of behavior can ultimately lead to realistic, human-like behavior, since it would require the interviewed expert players to be fully aware of their decision processes. It would also require the selection of questions to cover the whole range of possible situations, which is also very unlikely. Nevertheless, it is up to now one of the most successful approaches in mimicking humans in video-games.

Khoo and Zubek [2002] developed "Groo" and "tt14m (trash-talking 14-year-old-moron)", two artificial opponent projects for HALF LIFE®, another popular representative of the FPS genre. Groo could be considered a classical approach utilizing a behavior based [Arkin, 1998] architecture. The main focus was on applicability in live games and computational efficiency. However, as a chat bot, "tt14m"

is a different approach, focusing on the social aspects of multiplayer online gaming; and indeed, some humans apparently began a conversation with the agent, and believed they were talking to a human player. Although, certain stylistic problems helped players to "break through the illusion" [Khoo and Zubek, 2002], for instance, players refer to each other by abbreviations of their real nicknames, "Rev" instead of "Reverand Phunk", or "Sebb" instead of "Sebbdawred", which is obviously difficult to implement. Nevertheless, "tt14m" showed that also social aspects are important for artificial players. Regarding social aspects in games, Spyridou et al. [2004] conducted investigations on how humans use speech in FPS team-games. In fact, the requirements for speech understanding appear to be rather low, since even teams consisting of only human players use a surprisingly simple language. For example, common conversations include (example according to Spyridou et al. [2004]):
Player 1: I'm heading for the shaft
Player 2: Right. I'm going for the lift then. There is one of them coming down
Further, they could show in Wizard of Oz experiments that human players would prefer artificial team members being capable of understanding team speech. Despite some basic effort in commercial games (SOCOM® , SOCOM II® , RAINBOW SIX 3® , S.W.A.T.®), this is a so far unrecognized testbed for agent architectures in collaborative environments.

Aside from complete agent implementations, numerous improvements for specific behaviors exist. Tozour [2002b] introduces basic aspects of ranged weapon combat. As before, it is mostly a top down approach, where for instance aiming and hitting is expressed as an trigonometric function of target and player positions.

Towards strategic behaviors and internal NPC goals, Orkin [2004] developed a goal-directed action planning for the game NO ONE LIVES FOREVER 2® . Effectively, the system comes down to a FSM where the goal state is one node in the FSM. Going from a start to a goal node usually requires a fixed node sequence. For example, to push a button, the agent first has to move towards it, consequently, state transitions correspond to in-game actions. The right sequence of actions and state transitions is finally calculated at runtime and should end at the desired goal state.

A lot of games use waypoints for agent navigation, i.e. agents walk on interconnected nodes within a graph structure instead of walking directly in 3D. Paths on the waypoint-graph can be easily translated to paths in 3D and thus allows for fast and robust collision free navigation. Lidén [2002] proposes the direct integration of tactical information about the relation between waypoints into the waypoint map. For instance, for FPS games the visibility of gameworld positions is important. These additional information could be stored at each waypoint node and thereby mark positions as safe or dangerous. The approach is straight forward and can be easily applied in most FPS games (gameworld positions could also be marked as ambush locations, or cover places). In addition, tactical node information can also be used for pathfinding, for instance to penalize unsafe paths. Hancock [2002] suggests a

similar approach for navigating doors, elevators, and other obstacles. Based on a node map, specific actions are coupled to special nodes, for example, to enforce a jump or a door opening action. As noted, the approach requires "good editing tools and people who take the work seriously. If the input is sloppy, the results will be too" [Hancock, 2002]. Reed and Geisler [2004] further expanded these ideas and implemented it in the commercial game SOLDIER OF FORTUNE 2®.

2.5.3 Machine learning in games

During the past three years we were witnessing an increased academic interest in machine learning applied to the design of believable computer game characters. A boosting factor behind this growing interest was already mentioned in the introduction of this contribution and was also noted by authors such as Cass [2002] or Nareyek [2004]: as we showed in the last section, commercially available games still mostly rely on deliberative AI techniques like finite state machines or A^* searches. On the other hand, subsymbolic machine learning as a tool to produce life-like game agents has been largely neglected by the scientific community. This, however, seems to change.

For sake of completeness, we have to mention machine learning approaches applied to more conventional games, for instance, chess, othello, poker and various other board and card games. A good overview can be found in [Fürnkranz, 2001], a more recent work on reinforcement learning of strategies for the popular board game Settlers of Catan is presented in [Pfeiffer, 2004]. However, these games usually do not have much in common with modern computer games although certain requirements in strategical thinking also apply in video games.

Machine learning has not been used widely in games. Some games create an illusion of learning by giving the AI access to more predefined behaviors over time. With the exception of the lately released Forza Motorsport®, where race car driving behaviors are learned from observations [Drivatar™- Driving Avatar, 2006], up to now, machine learning does not play an important role in commercial game AI. According to [Togelius et al., 2006], the learned driving behavior in Forza Motorsport® is the outcome of stored trajectories for each player and each racing track, instead of acquiring sophisticated steering behaviors from observational data.

In video games, mostly online learning approaches had been applied so far. Recent work by Spronck et al. [2003] introduced reinforcement learning to the task of rule selection for agent behavior in a commercially available role playing game. Earlier, the same authors reported on a hybrid coupling of genetic algorithms and neural networks for offline learning in a simple strategy game [Spronck et al., 2002]. Yannakakis and Hallam [2004] introduced an evolutionary optimization approach for opponent behavior generation in Pac-Man like games. A successful application of reinforcement learning for versatile opponent modelling in a Beat-Em-Up game was presented by Graepel et al. [2004].

The idea of using human generated data to train game agents was first reported by Sklar et al. [1999] who collected the key-strokes of people playing Tron in order to train neural networks. Notably, Lipson et al. [2003] used the log-files for their game Touring to cluster game-data to generate different models of play. Using these, they were able to generate human-like playing styles in an artificial player.

Learning from observations was so far mostly done in simplified games or utilizing basic sets of predefined behaviors. The presented work is trying to close the gap by elaborating approaches for behavior acquisition in complex games by means of imitation learning. Although the focus of this thesis is on basic skill acquisition, we plan to use learned behaviors in higher level learning architectures.

2.5.4 Machine learning in FPS games

Recently, Le Hy et al. [2004] described advanced probabilistic action selection for a commercial game using Baysian networks trained by human generated input. Appropriate behaviors are selected during a live game by a human expert player, afterwards state dependent behavior selection is reproduced by an artificial player. The approach is very promising and could provide means of integrating learned basic behaviors, as proposed in this work, in a more higher level imitation learning system. The only disadvantage is the indirect control of the player avatar during the training sample recording. Since the avatar is controlled by selecting an appropriate behavior and not by directly steering it, similar criticisms as already brought against the BDI architecture by Norling and Sonenberg [2004] apply here as well: it is questionable if behavior synthesis based on a player's introspection will result in realistic acting.

Lately, Bakkes et al. [2004] presented a evolutionary team behavior learning approach. Basic behavioral patterns were supplied in advance and combined during live games, resulting in challenging and more versatile play.

In his Master thesis, Geisler [2002] let an agent learn combat behaviors of an expert player by utilizing MLPs, a naive Bayesian classifier, and ID3 decision tree learning [Quinlan, 1990]. He successfully managed to mimic speed of acceleration, direction of movement, direction of facing, and jumping. The approach is straight forward by simply selecting combat related feature vectors as an input, and possible action vectors as an output. Comparing the different approaches, apparently the MLP performed best. However, it remained unclear how the agent really performed in-game, since no direct behavioral analysis is supplied.

Zanetti and Rhalibi [2004] applied MLPs to behavior learning in QUAKE III® . They combined three MLPs, responsible for movement, aiming and shooting, and fighting. Each MLP was supplied with a set of relevant features. Training samples were recorded from observations of human players. The approach seemed to be able to capture certain behaviors, however, as noted by the authors, it is far from being a competitive agent. It also fails at capturing human-like behavior "... The objective

of human like behavior and strategies is still far away." [Zanetti and Rhalibi, 2004].

2.5.5 Believable game AI

Since no one would be interested in dumb game agents, all described game AI techniques aim at the creation of believable, realistic artificial characters. However, as noted by Livingstone [2006], there is a difference between realistic and human-like characters. A realistic game agent has to create the illusion of a specific character, for example, a medieval knight in a fantasy role-playing game. In contrast, a human-like game agent has to create the illusion of being controlled by a human, which obviously only makes sense in multiplayer games.

Laird and Duchi [2000] had human observers judge the SOAR QUAKE II® agent in videos in which the bot battled an expert human player. The judges had to rate the humanness of the observed game agents. All in all, the SOAR bot did quiet well in producing a human-like impression. Interestingly, it were the often criticized superhuman playing capabilities of artificial players that helped to unmask them, i.e. instant reaction times, or perfect aiming. It was further noted that an agent should be supplied with at least some basic tactical reasoning, purely reactive agents mostly failed in convincing the participants.

In [McGlinchey and Livingstone, 2004], believability tests were carried out for an artificial Pong player (see Figure 2.2 for Pong) which was modelled by means of a self-organizing map trained on human playing data (see McGlinchey [2003] for the original approach). The results were convincing, indicating that it is possible to fool a human player/observer. However, the human judges never saw a human or an artificial Pong player before the test took place, this obviously makes the results less reliable. The reported results only made that more evident since the judges appeared to be able to distinguish between artificial and human player, they just mislabeled them occasionally.

2.6 Summary

Behavior acquisition in artificial systems is a very important, yet underexplored area. In this chapter, we proposed two models as the theoretical basis for behavior acquisition by means of imitation learning. Both are build on the assumption that we can access a sufficient number of situation-action data pairs. These data samples implicitly describe the behaviors that we want to imitate by assigning an action to a specific situation. To goal of imitation learning now is to find an underlying mapping of situations to reactions that also accounts for novel situations. The first approach consists of a regression model, in which actions of a human player are interpreted as functional mappings of situation vectors on reaction vectors. A second approach describes behavior in a probabilistic model, in which state dependent probability distributions are learned from observational data to predict the most

likely action for a given situation.

Imitating human behavior is a demanding task. One of the major difficulties lies within the versatility and flexibility of the behavior itself. Behaviors are seamlessly integrated and combined, although they are possibly operating on completely different time-scales. For example, in soccer a player is able to think about the next best strategical pass while at the same time looking for opponents, and team members, but also controlling the ball. The considered computer game worlds might not be as complex as the real world, they still require a large repertoire of distinct behaviors. An artificial agent needs to implement a multitude of skills to develop human-like playing behavior. Modelling, or learning, a monolithic "play the game" behavior in an artificial agent is very likely to fail. In this chapter, we motivated the use of a hierarchical behavior model, based on a well known psychological hierarchy. The model separates behaviors into three categories; strategic, tactical and reactive behavior. Although this might suggest a sequential execution of behaviors where each layer controls the lower laying layer (e.g. the tactical layer selects an appropriate action), it is to be understood as a concurrent set of behaviors. The step of separating behaviors comes in naturally and greatly simplifies behavior learning in artificial systems. By developing methods targetted at specific behaviors in each category, the problem of behavior acquisition becomes feasible.

Since behavior is usually coupled to a specific environment, we introduced our own experimental setup. We make use of the opportunities gained from modern computer games for analyzing human behavior. It is basically two important aspects that make computer games a suitable environment for the task of behavior imitation: games are often perceived as more real than conventional simulations, therefore we can expect more natural behavior which is essential for the task at hand. Furthermore, by using popular networked games, we can access vast amounts of noise free observation data, which allows the application of common machine learning and pattern recognition approaches. Regarding the experimental environment, we decided for the game QUAKE II®, which belongs to the genre of First-Person-Shooter (FPS) games. This game was used by various researchers throughout the last years and was named an excellent tool in researching human-level AI [Laird, 2002]. Modern FPS games create realistic virtual worlds, in which humans control virtual avatars and play against computer controlled avatars. It is important to notice that QUAKE II® is not a simple game and indeed requires strategical and tactical acting for successful gameplay. To further point out that the problem of creating a realistic, human-like agent for games is by no means solved, we reported on the most successful approaches carried out in the games industry and academia. Up to now, no one could show that an artificial agent for computer games can come up to human-level intelligence or playing behavior. The presented techniques were mostly inspired by classical AI research. Recently, the idea of agent behavior modelling by means of machine learning received increased attention, so far with only

mediocre success for FPS games.

The task we want to achieve with this work is to use the network traffic recordings of human players to (a) explore the data using different learning approaches, and (b) synthesize behavior in computer controlled artificial players. The goal obviously is to mimic the playing behavior of a human as accurate as possible. The following chapters introduce various approaches, targetted at different behaviors, on achieving this goal. We will first concentrate on specific behaviors within the reactive, tactic, and strategical layers.

Chapter 3

Learning reactive behaviors

Reactive behaviors include, among others, directly sensor stimuli based movements and motions. In this chapter, we outline methods in imitating reactive behavior. The behaviors we consider are common reactive behaviors found in the game QUAKE II® , our experimental environment. In particular, we elaborate approaches for computational imitation of movement, aiming, and complex motions. Besides, we introduce basic pattern recognition methods that are also important for the later elaborated tactical and strategical behavior learning.

Following ideas discussed in Section 2.1, we consider two distinct approaches for learning reactive behaviors. The first approach interprets behavior as functional mapping of sensor inputs on reactions in order to extract underlying behavioral functions. The second approach focuses on state dependent action selection with the goal of human-like motion synthesis. This chapter contains work published in [Bauckhage et al., 2003, Thurau et al., 2003, 2005a,b].

3.1 Reactive behaviors as functional mappings onto actions

In Chapter 2, we introduced the idea of interpreting behavior as a functional mapping of a game state $\mathbf{s}_t$ at time step t on an action vector $\mathbf{a}_t$:

$$\mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{s}_{t-1} \dots \mathbf{s}_{t-n}, \mathbf{e}_t, \mathbf{e}_{t-1} \dots \mathbf{e}_{t-n}\mathbf{e}_t) \tag{3.1}$$

where $\mathbf{e}_t$ denotes environmental influences at t and $\mathbf{a}_{t+1}$ represents the action a player accomplishes according to the last n states. Although the state history, the states $\mathbf{s}_{t-1} \dots \mathbf{s}_{t-n}$, could be important, we for now only have a look at a few past states or only direct sensor based reactions i.e. the state at timestep t. This is because of the obvious computational constraints in real-time decision making. Moreover, reactive behaviors, as they were introduced in robotics, are meant to be dependent solely on the current state and the current environmetal influences

[Arkin, 1998]. By focusing on the current state at timestep t, and by neglecting environmental influences $\mathbf{e}_t$ which we cannot observe anyway, we can restate equation 3.1 as follows:

$$\mathbf{a}_{t+1} = f(\mathbf{s}_t) \tag{3.2}$$

Given a set of human observation data of a reactive behavior, we now approximate the underlying behavioral function f using the observational data as training samples. For this task, we will first consider *Multi-layer perceptrons* (MLPs) which are basically a universal function approximator and thus should be suited for the task at hand. The next Section briefly introduces MLPs. In addition, we also introduce *Time-delay neural networks* (TDNNs) which are a type of MLPs suited for learning temporal dependencies. In the following, we mostly concentrate on standard MLPs. Nevertheless, we found that including a limited state history is beneficial for certain behaviors. In these particular behaviors, Time-delay neural networks were showing an improved performance compared to standard MLPs. However, due to the already mentioned computational constraints, we are usually only including a very short state history, corresponding to a maximum value of $n = 4$ for Equation 3.1.

Here, one could argue that there are more advanced, and thus more capable learners, for example, support vector Regression. However, Neural Networks are inspired by concepts in neurons in biological organisms, this simple fact makes them an interesting candidate for imitating human behavior. There is no clear evidence that biologically inspired approaches outperform others in imitation learning, still, they also showed to be effective in numerous uncommon (from a machine learning perspective) applications as for instance chess playing [Thrun, 1995]. Moreover, given the basic idea of this thesis and the collaborations that supported this work, whenever possible we decide for biologically or psychologically motivated approaches. Furthermore, given the rather shallow preliminary work in the field of behavior acquisition in artificial agents, we decided to first have a look at simple, and well established approaches.

3.1.1 Multi-layer Perceptrons

A *Multi-layer Perceptron* is the most common type of *artificial neural networks* (ANN). It consists of several connected layers of *Perceptrons*. The basic computational unit in a MLP is named neuron, since its basic functionality is inspired by concepts in biological Neurons. Each neuron is a computational unit generating a numerical output by weighting and summing up several numerical inputs, see also Figure 3.1(b). The idea of combining Perceptrons in a larger network is motivated by the limitations of single Perceptrons with respect to the type of functions they are able to model. Theoretically, networks containing two layers of weights are able to approximate any continuous function. Since MLPs are a common method

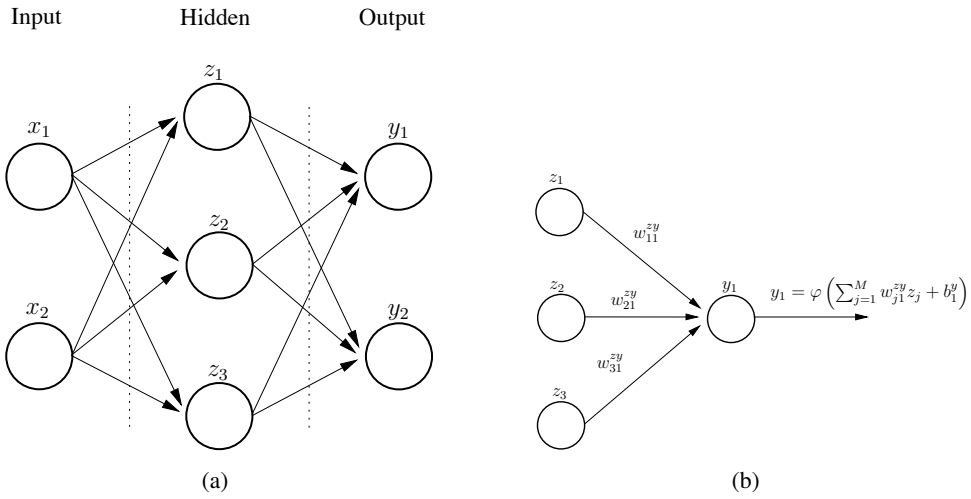


Figure 3.1: Figure 3.1(a) shows a didactic example for a Multi-layer Perceptron. Figure 3.1(b) shows how a single neuron within a feedforward network calculates its output.

in pattern classification and function approximation tasks, we will only summarize them briefly, for further reading we recommend [Bishop, 1996].

In the most common form a MLP consists of 3 layers of neurons, an input layer receiving input signals $\mathbf{x} \in \mathbb{R}^n$, a hidden layer, and an output layer generating an output $\mathbf{y} \in \mathbb{R}^m$. Given a sufficient number of hidden neurons, a MLP is able to approximate any continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$.

Figure 3.1 shows a simple MLP, and a single neuron and how it calculates its output. The output of a single neuron is calculated by a weighted linear sum of inputs which is put through a (usually) non-linear activation function φ. In Figure 3.1(b), z_j denotes inputs from the predecessing layer, w_{ij}^{zy} denote the input weights, and b_1^y denotes a bias term for the neuron y_1. As an activation function φ, commonly the logistic sigmoid function $y(x) = 1/(1 + \exp(-x))$, or tangens hyperbolicus $y(x) = \tanh(x)$ are used.

For regression problems, the *mean-squared error* (MSE) E with respect to a collection of training samples is optimized:

$$E = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - g(\mathbf{x}_i))^2 \tag{3.3}$$

where N denotes the number of training samples, $\mathbf{y}_i$ denotes the known target values for the network output (or the unknown function f), and $g(\mathbf{x}_i)$ denotes the output generated by the MLP for an input vector $\mathbf{x}_i$. Implicitly, the function f is already defined via the training samples. Learning in a MLP is equal to approximating the unknown function f by adjusting MLP weights w_{ij}. Various methods

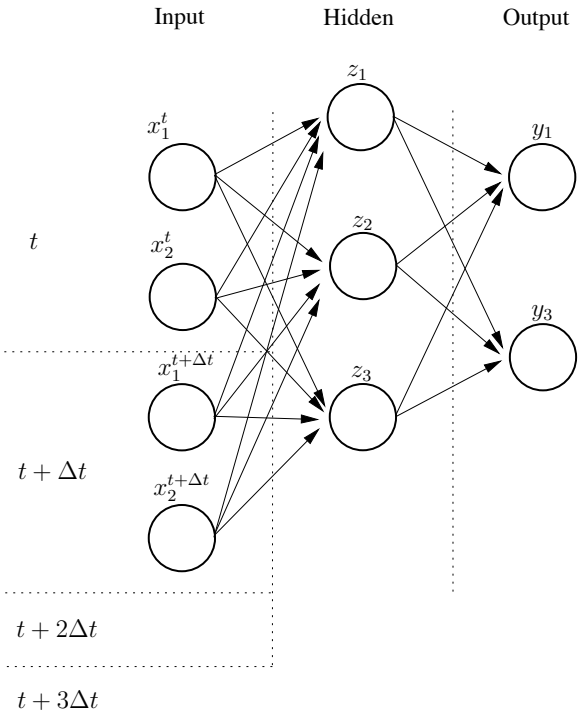


Figure 3.2: A simple Time-delayed neural network. Instead of using only one input signal a TDNN uses sequential input data, to learn temporal relationships.

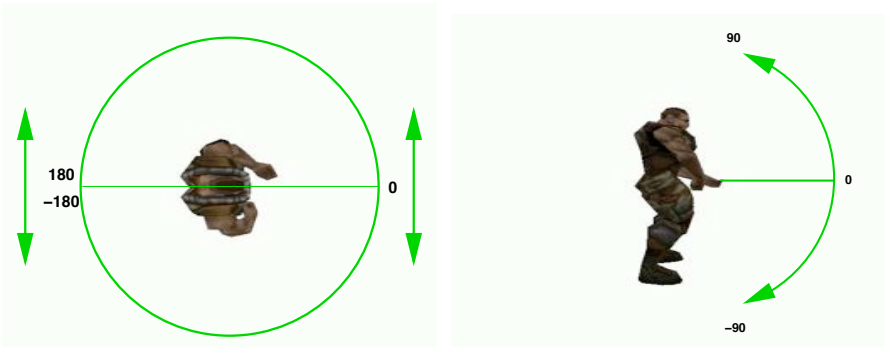
for finding appropriate weights w_{ij} are known. The most common is the backpropagation algorithm. However, in this work we make use of the (also very common) Levenberg-Marquardt algorithm, both are explained in detail in [Bishop, 1996].

3.1.2 Time-delay neural networks

A *Time-Delay Neural Network* (TDNN) is a special type of Multi-layer Perceptron used for learning temporal relationships. Similar to a MLP, it consists of a layer of inputs, a hidden layer, and an output layer. The difference to MLPs lies within the input layer. In TDNNs, an input signal $\mathbf{x}$ is sampled in a series of discrete intervals Δt. This leads to a temporal sequence of input signal $\mathbf{x}^t, \mathbf{x}^{t+\Delta t}, \ldots, \mathbf{x}^{t+n\Delta t}$ that can be used as sequential input data for a TDNN, as can be seen in Figure 3.2. The networks internal processes of output generation are the same as in standard MLPs. For weight adaption, the backpropagation algorithm can be used.

3.1.3 Features, behaviors and evaluation

Using behavioral data to train a MLP approximates the underlying behavioral function f given by $\mathbf{a}_t = f(\mathbf{s}_t)$. Thereby, we reproduce a human player's reaction $\mathbf{a}_i$



(a) Yaw angle (b) Pitch angle

Figure 3.3: Yaw and Pitch game agent viewangles.

for a new state vector $\mathbf{s}_i$, and thus create similar behavior for novel game situations. Obviously, the selection of features for $\mathbf{a}_t$ and $\mathbf{s}_t$ is important and closely coupled to the actual behavior we intend to learn. Essentially, the selection of features defines what action to consider for which aspect of the game-state. Consequently, including action variables for controlling the agent's velocity would result in movement behavior, whereas including a fire action would result in learning aspects of a player's combat behavior. However, the selection of relevant state space features for $\mathbf{s}_i$ is not as intuitive.

A human player perceives the game-world through 3D visualization of game-data, and reacts by moving the mouse or using keyboard inputs. A computer vision analysis of the game's visualization could be used to extract information about objects or other players. However, here we rely on a different method by directly accessing game-state information. As already mentioned, the network traffic of QUAKE II®, which we interpret as observation data, already contains a higher level state description. It includes, for example, entity positions (objects, players in x, y, z), character velocities (in $\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta z}{\Delta t}$), and internal state descriptors such as player health or armor values. The player actions are included as well, they contain the player viewangles (yaw and pitch), velocities (in $\frac{\Delta x}{\Delta t}, \frac{\Delta y}{\Delta t}, \frac{\Delta z}{\Delta t}$), and special actions as for instance fire or jump [1]. Since the focus of this thesis is on behavior acquisition, we decided to mostly hand-pick features we considered essential for the considered behaviors (in the case of situative reactive behaviors we use a simple Principal Component Analysis based feature extraction which will be explained in Section 3.1.7). This turned out to be sufficient for the behaviors considered in this work. However, future work might consider automatic approaches for feature selection, for instance boosting methods (see e.g. [Redpath and Lebart, 2005]).

[1] All in all, our underlying API, in MATLAB®, to interact with QUAKE II® can provide a total of about 260 different features.

(a) Perception (b) Reaction

Figure 3.4: A game-bots perception and reaction spectrum.

For the later, we carried out experiments where movement and aiming behaviors are to be learned. We selected for the game-state $\mathbf{s}_t$ the player's origin $\mathbf{o}_t = [x_t, y_t, z_t]$, an opposing player's distance $\in [0, 2000]$, target yaw $\in [0°, 180°]$ and pitch $\in [-90°, 90°]$ viewangles, yaw being assigned a signum $\sigma(\text{YAW}) \in \{-1, 1\}$, in order to cover the full range as Figure 3.3 illustrates.

$$\mathbf{s} = \begin{pmatrix} position\ X \\ position\ Y \\ position\ Z \\ opponent\ distance \\ opponent\ target\ yaw\ angle \\ opponent\ target\ yaw\ sign \\ opponent\ target\ pitch \end{pmatrix}, \mathbf{a} = \begin{pmatrix} yaw\ angle \\ yaw\ anglesign \\ pitch\ angle \\ velocity\ forward \\ velocity\ side \\ velocity\ up \end{pmatrix}$$

Figure 3.5: The agent's state $\mathbf{s}$ and reaction $\mathbf{a}$ vector variables used in the experiments.

Regarding relevant features for the player's action vectors $\mathbf{a}_t$, we include all movement relevant actions i.e. forward-, side-, and up- velocity, and yaw-, and pitch-viewangles. Viewangle adjustments are coupled in a 3 dimensional vector, containing the player's yaw $\in [0°, 180°]$ and pitch $\in [-90°, 90°]$ angles, again including a signum $\sigma(\text{YAW}) \in \{-1, 1\}$. The player's velocity is represented by a 2 dimensional vector, containing $v_{forward} \in [-400, 400]$, $v_{side} \in [-400, 400]$, and $v_{up} \in [-400, 400]$. Figure 3.4 illustrates the bot's perception and reaction spectrum, and Figure 3.5 contains a detailed listing of the features and actions used in the later described experiments.

The behaviors we intend to learn ranged from simple movements, over more versatile and complex moves, to an aiming behavior. All training data sets are

recordings of a human performing the described action. Thus, successfully learning and mimicking these behaviors is equal to imitating human behavior. We used the following behavioral observation data:

Experiment 1 - Simple movement The first experiment is about reproducing a simple movement behavior. A human player was recorded while running around a map on a fixed route.

Experiment 2 - Simple aiming The second experiment introduces an aiming behavior. A human player was recorded while aiming at an opponent.

Experiment 3 - Combined aiming and movement A combination of experiments 1 and 2, thus, two distinct behaviors should be learned within one architecture. Thus,

As already mentioned, evaluation is an important aspect of behavior synthesis. Evaluation for reactive behaviors is done by computing the *mean squared error* (MSE) for a set of novel state/action data pairs. Given that MLPs are optimized towards the MSE, and that reactive behaviors in this context only have a short-term relevance, this seems appropriate. As opposed to strategies, where we find a long-term relevance of actions, it is useless to measure goal success, since we do not have any specific goal coupled to a reactive behavior. The performance of the learner is measured by computation of the mean squared error E_d of given data set d, containing n state $\mathbf{s}$ and reaction $\mathbf{a}^{desired}$ vector pairs.

$$E_d = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}_i^{desired} - \mathbf{a}_i(\mathbf{s}_i))^2 \tag{3.4}$$

Moreover, in addition to the MSE computation, we also decided to report on the general appearance of the generated behavior. In the case of reactive behavior it is important that the behaviors are learned and synthesized successfully, and, that their in-game execution creates the illusion of a human player.

3.1.4 Simple reactive behaviors

In a first approach, we realized our agent by means of 1 MLP, responsible for viewangle adjustment and velocity adjustment with a 6 dimensional output vector, and a 7 dimensional input vector (see again Figure 3.5 for the features). Although we treated viewangle and velocity adjustment as equally important player reactions, in-game evaluation showed that viewangle adjustment has a greater influence on overall bot performance; it is also harder to learn. MLPs were trained using the Levenberg-Marquardt algorithm. We tested on various parameter settings, varying the number of hidden layer neurons as well as varying training and test behavior datasets.

It should be noted that although most behaviors were successfully imitated, the approach also showed shortcomings. It appeared that a single MLP is not able to successfully imitate a larger set of behavior. The outcome of experiment 3, where two separate behaviors are included in the training data, shows this very well. The problem appears to reside in the versatility of observed behavior. Effectively, multiple functions have to be learned, with at times compromising data pairs. There seems to be interference in function representation. Before we analyze the experimental results in detail in Section 3.1.8, we first introduce an extension to the MLP learning in the next Section.

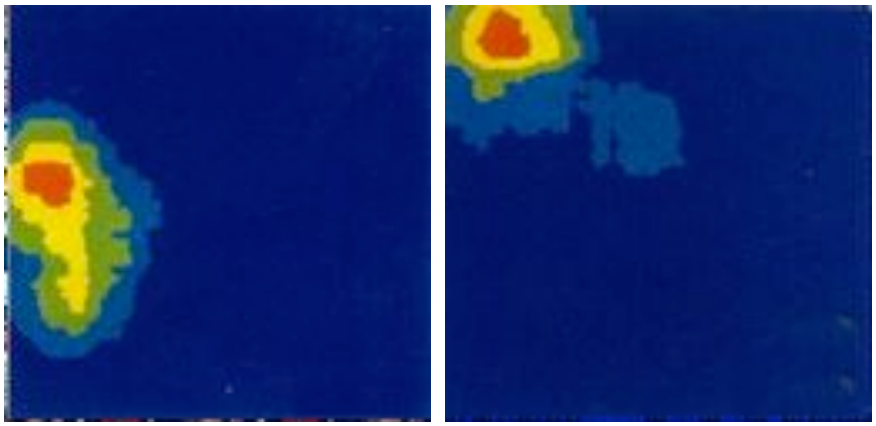
We noticed that by taking into account multiple MLPs and establishing a more concrete situational dependency, the described problems can be avoided. Effectively, we integrate the well established concept of biological *place cells* to allow for specialized, situation aware learners.

3.1.5 Situation aware reactive behaviors

Animals build up environment representations using so called *place cells* Arleo [2001]. Given visual cues, place specific cells respond, and thereby tell the animal its location and orientation. In Figure 3.6 we present additional background on this interesting concept. Here, we wanted to make use of a similar method: instead of establishing direct functional dependencies, we decided to first roughly separate states by making behavior learning dependent on activation of place cell like structures. This means that we learn different behavioral functions, dependent on the highest activation of one specific situational or place cell.

Place cell activity in humans and animals is build up on visual cues and integration over the past movements. Since we do not have to rely on visual input for game-agents, we can directly access game agent position vectors. Therefore, place cell activity can be estimated straight forward by comparing state vectors to place cell prototypes. The principal concept of place cells is in fact very similar to topological representations of environments, also often used in conventional Game AI where it is called *waypoint-map*. However, unlike usual methods in waypoint-map acquisition, we build up and learn representations from data which is not restricted to position vectors. In fact, the introduced place cells are more likely *situational cells*, since they correspond to not only spatial information representation, but can also contain other information, for instance, positions of an opponent. To avoid further confusion, we stick to the established name of place cell.

In order to make use of the concepts of place cells, we have to derive a place cell like structure from the game-data. For that matter, we apply the *Neural Gas* vector quantization algorithm to game observation data which we will first introduce, before reporting on the details of the behavior acquisition approach.



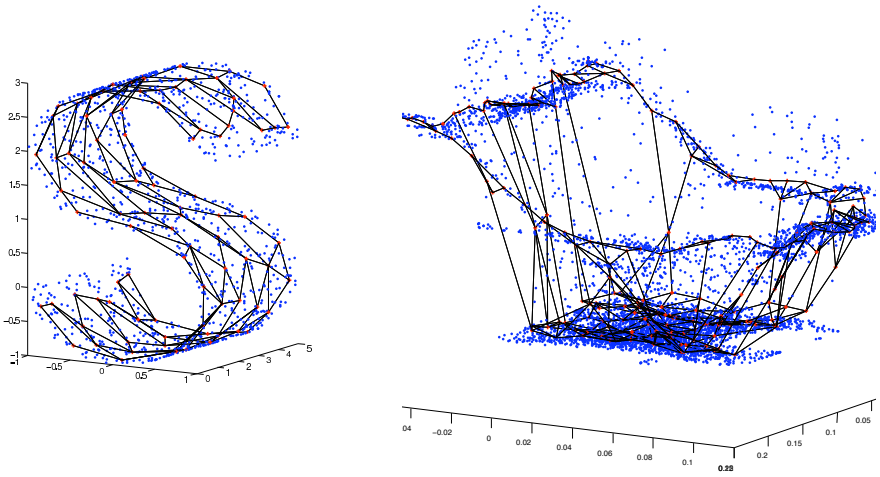
(a) (b)

Spatial memory is of great importance for animals and humans. Recent findings indicate that specialized neurons, so called *place cells*, are responsible for recognizing spatial position. Coupled to each place cell is a *place field*, an area of response. This place field is defined by its target, and results in a higher activation once a location is reached where locations are generally defined by fixed spatial cues (i.e. objects). Place cell responses give us precise information about our position in an environment. The pictures show place fields of two place cells in rats in a square room. The red regions mark the best responding area for a cell. In blue regions, the cells are not responding (both pictures were taken from [Arleo, 2001]).

A few years ago, Maguire et al. [1998] investigated brain activity in humans during navigation in complex environments. Findings indicate, that the right hippocampus is associated with knowing exactly where places are located. They also found further activation in other brain areas, probably involved in non-spatial aspects of navigation. Interestingly, the experimental environment they used was the famous First-Person-Shooter game DUKE NUKEM® , which is very similar to QUAKE II® . The proven existence of place cells, even during the playing of a FPS game, further supports our own ideas of integrating *artificial place cells* into imitation learning approaches.

Furthermore, Ekstrom et al. [2003] showed evidence for a neural code of cells located in the hippocampus region that respond at specific spatial locations. They also showed that other cells located in the parahippocampal region responded to views of landmarks. Further, cells responded to navigational goals and to conjunctions of place, goal, and view. Although this is still different to our proposed *situational cells*, it is clearly showing biological means towards sensor integration, and to a more unifying representation of spatial and other information.

Figure 3.6: Place Cells in humans



(a) (b)

Figure 3.7: A Neural Gas clustering of a 3.7(a) swiss-roll toy example, and of 3.7(b) player positions extracted from the network code of QUAKE II® .

3.1.6 Neural Gas

The Neural Gas algorithm [Martinetz and Schulten, 1991] is a soft vector quantization algorithm. Similar to other clustering algorithms, the Neural Gas algorithm fits a number of k prototype vectors $W = [\mathbf{w}_1, \ldots, \mathbf{w}_k], \mathbf{w}_i \in \mathbb{R}^n$ to represent a number of input signals $V = [\mathbf{v}_1, \ldots, \mathbf{v}_m], \mathbf{v}_i \in \mathbb{R}^n$ (for the Neural Gas algorithm the prototypes are often named neurons to underline the biological background). In contrast to winner takes all clustering algorithms, as for instance k-means, the Neural Gas algorithm adapts not only a winner prototype according to an input signal, but also more or even all prototypes. One characteristic of the algorithm is that at the beginning generally more prototypes beside the winner are adapted, at the end it is usually only the winner prototype which is modified. Moreover, the Neural Gas algorithm introduces a Hebbian learning of a topological structure which interconnects prototypes $\mathbf{w}_i$, thereby reflecting the underlying structure of input signals $\mathbf{v}_i$. Figure 3.7 gives two examples of a resulting clustering and learned topology. In the following we explain the clustering as well as the Hebbian learning of the topological structure:

For a new input signal $\mathbf{v}_i$, the distance $D_{\mathbf{v}_i} = \| \mathbf{v}_i - \mathbf{w}_u \|, u = \{1, \ldots, k\}$ to each prototype vector $\mathbf{w}_u$ is computed, and ordered as follows

1. Assign initial values to the prototypes $W = [\mathbf{w}_1, \ldots, \mathbf{w}_k], \mathbf{w}_i \in \mathbb{R}^n$
2. Select an input signal $\mathbf{v}_i$ from $V = [\mathbf{v}_1, \ldots, \mathbf{v}_m], \mathbf{v}_i \in \mathbb{R}^n$
3. Determine the distances of each prototype vector to $\mathbf{v}_i$, and rank the prototypes in W:

   $\{\mathbf{w}_m^1, \mathbf{w}_m^2, \ldots, \mathbf{w}_m^k\}$, where $\| \mathbf{v}_i - \mathbf{w}_m^1 \| \leq \| \mathbf{v}_i - \mathbf{w}_m^2 \| \ldots \| \mathbf{v}_i - \mathbf{w}_m^k \|$

   According to the ordering assign each prototype $\mathbf{w}_i$ a value d_i that reflects its ranking ($d_m = 1$ for the winner, $d_m = 2$ for the second ...).
4. Adapt each prototype according to its ranking d_u:

   $$\mathbf{w}_u = \mathbf{w}_u + \epsilon(t) \exp\left(\frac{1 - d_u}{\lambda(t)}\right)(\mathbf{v}_i - \mathbf{w}_u) \tag{3.5}$$

   where

   $$\lambda(t) = \lambda_s(\lambda_e/\lambda_s)^{(t/t_{max})} \qquad \text{and} \qquad \epsilon(t) = \epsilon_s(\epsilon_e/\epsilon_s)^{(t/t_{max})} \tag{3.6}$$
5. Increment the time parameter t: $t = t + 1$
6. If $t \leq t_{max}$, continue with step 2

Figure 3.8: Neural Gas vector quantization

$$\{\mathbf{w}_m^1, \mathbf{w}_m^2, \ldots, \mathbf{w}_m^k\}, \text{where } \| \mathbf{v}_i - \mathbf{w}_m^1 \| \leq \| \mathbf{v}_i - \mathbf{w}_m^2 \| \ldots \| \mathbf{v}_i - \mathbf{w}_m^k \| \tag{3.7}$$

According to the ordering each $\mathbf{w}_i$ is assigned a value d_i, reflecting its ranking. The prototype $\mathbf{w}_m^1$ which is closest to $\mathbf{v}_i$ is assigned a value of $d_m = 1$, the second closest $\mathbf{w}_m^2$ a value of $d_m = 2$, till the last prototype $\mathbf{w}_m^k$ which is assigned a value of $d_m = k$. Each prototype is now adapted according to

$$\mathbf{w}_u = \mathbf{w}_u + \epsilon(t) \exp\left(\frac{1 - d_u}{\lambda(t)}\right)(\mathbf{v}_i - \mathbf{w}_u) \tag{3.8}$$

where d_u denotes the ranking value, $\epsilon(t) \in [0, 1]$ denotes a learning rate which is decreased over time, and $\lambda(t)$ influences how many ranked prototypes are adapted with each step. According to [Martinetz and Schulten, 1991], $\lambda(t)$ and $\epsilon(t)$ can be defined as follows:

$$\lambda(t) = \lambda_s(\lambda_e/\lambda_s)^{(t/t_{max})} \qquad \text{and} \qquad \epsilon(t) = \epsilon_s(\epsilon_e/\epsilon_s)^{(t/t_{max})} \tag{3.9}$$

1. Initialize a connectivity Matrix $C, C_{ij} \in \{0,1\}$ with $i,j = 1 \dots M$. Set all C_{ij} to zero
2. Select an input signal $\mathbf{v}_i$ from $V = [\mathbf{v}_1, \dots, \mathbf{v}_m], \mathbf{v}_i \in \mathbb{R}^n$
3. Determine the distance of each prototype vector to $\mathbf{v}_i$, and rank the prototypes in W:

   $\{\mathbf{w}_m^1, \mathbf{w}_m^2, \dots, \mathbf{w}_m^k\}$, where $\| \mathbf{v}_i - \mathbf{w}_m^1 \| \leq \| \mathbf{v}_i - \mathbf{w}_m^2 \| \dots \| \mathbf{v}_i - \mathbf{w}_m^k \|$

   According to the ordering assign each prototype $\mathbf{w}_i$ a value d_i that reflects its ranking ($d_i = 1$ for the winner, $d_j = 2$ for the second ...).
4. Determine the two best matching prototypes $\mathbf{w}_i, \mathbf{w}_j$ with $d_i = 1, d_j = 2$
5. If there is no existing connection between the two best matching prototypes $\mathbf{w}_i, \mathbf{w}_i$, set $C_{ij} = 1$
6. Increment the time parameter t: $t = t + 1$
7. If $t \leq t_{max}$, continue with step 2

Figure 3.9: Neural Gas Hebbian topology learning

where λ_s, and ϵ_s denote initial values, and λ_e, and ϵ_e denote end values, and t_{max} denotes the maximum number of iterations. For their examples, Martinetz et al. [1993] used the following parametrization : $\lambda_s = 10, \lambda_e = 0.01, \epsilon_i = 0.5, \epsilon_f = 0.005, t_{max} = 40000$. Generally, λ_s should be given a higher value than λ_e. The complete algorithm is summarized in Figure 3.8.

The Hebbian learning of a topological structure is usually done after the adaption of prototype vectors. It could also be done in parallel to the adaption process, therefore an additional decay parameter to remove edges needs to be included. For a detailed explanation of the latter we recommend [Martinetz and Schulten, 1991].The principal idea of the topology learning is to establish an edge between two prototypes $\mathbf{w}_i, \mathbf{w}_j$ that have a ranking of $d_e = 1$ and $d_k = 2$ for a new input signal $\mathbf{v}_k$.

First, initialize a connectivity Matrix $C, C_{ij} \in \{0,1\}$ with $i,j = 1 \dots k$ to describe the neighborhood relationship of all prototypes. Since an index $C_{ij} = 1$ means that prototype $\mathbf{w}_i$ is connected to prototype $\mathbf{w}_j$, initialize all C_{ij} to zero. After the initialization process, the learning of the topological structure starts by setting the starting time parameter $t = 1$. For a new input signal $\mathbf{v}_k$, the winner and the second ranked prototype $d_i = 1, d_j = 2$ is determined according to Equation 3.7. If there is not already an existing connection between the two winner prototypes $\mathbf{w}_i, \mathbf{w}_j$, set $C_{ij} = 1$. Figure 3.9 summarizes the approach for the

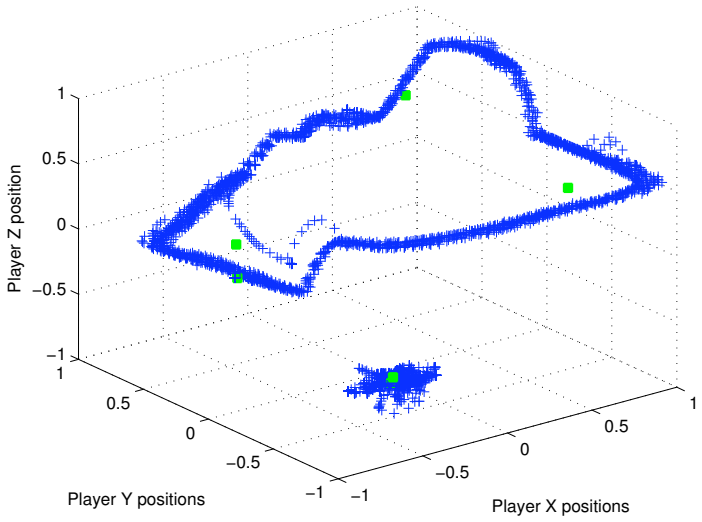


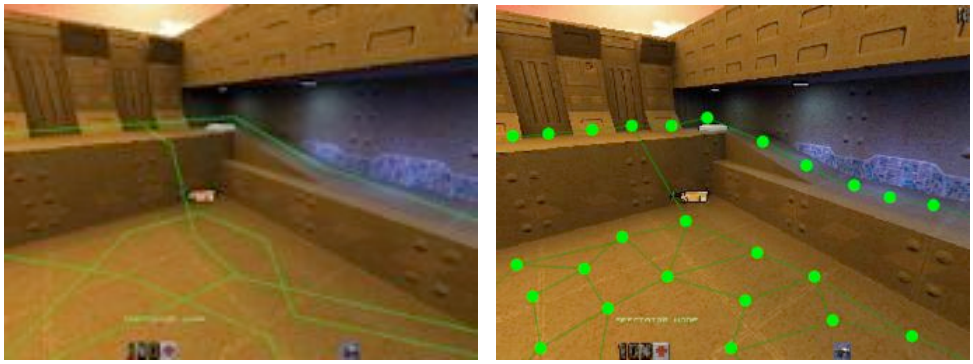
Figure 3.10: Blue crosses mark observed player positions $\mathbf{p} = [p_x, p_y, p_z]$, larger green dots mark resulting place cell or waypoint map cluster centres.

Hebbian topology learning.

3.1.7 Combining simple behaviors using place cells

In the experiments, where we used a single MLP, we observed problems in behavior synthesis as soon as the training data for learning consisted of more than one behavior, or more complex behavior. We now address this problem by introducing situation aware learners i.e. we first determine a rough situation classification using a place cell inspired approach, and then select a specialized MLP for generating reactions. For this approach, we first have to determine how to distinguish game-situations. In addition, we have to distribute the training data among different learners, to imitate specialized behavior.

For separating situations we make use of the concept of place cells by assigning game-states to specific place cells. Thus, in a first step, place cells are derived by clustering training data samples $D = [(\mathbf{s}_1, \mathbf{a}_1), \ldots, (\mathbf{s}_n, \mathbf{a}_n)], \mathbf{s}_i \in \mathbb{R}^k, \mathbf{a}_i \in \mathbb{R}^l$ in state space $\in \mathbb{R}^k$ using a Neural Gas clustering. This leads to m prototypes of $P = [\mathbf{s}^p_1, \ldots, \mathbf{s}^p_m]$. By assigning each training sample pair $(\mathbf{s}_i, \mathbf{a}_i)$ to the prototype $\mathbf{s}^p_o$ with the minimum Euclidean distance $E_o = \parallel \mathbf{s}_i - \mathbf{s}^p_o \parallel$, the training data is distributed among prototypes. Each prototype (or place cell) $\mathbf{s}^p_i$ is now associated with a number of training samples $D_i = [(\mathbf{s}^i_1, \mathbf{a}^i_1), \ldots, (\mathbf{s}^i_u, \mathbf{a}^i_u)]$. Implicitly, the Voronoi partition in the surrounding of each training sample cluster yields a situative behavior description and each Voronoi region corresponds to a *place field*. Individual data



(a) (b)

Figure 3.11: If applied to data representing the paths a player did run during a match (3.11(a)), Neural Gas clustering will result in a structured set of waypoints which captures the topology of the corresponding map(3.11(b)).

sets D_i can now be separated and distributed among different learners. A exemplary distribution of Neural Gas neurons in input state space is displayed in Figure 3.10, a didactic example of clustering the player positions is shown in Figure 3.11. Since the correct number of Neural Gas neurons for place cell computation is impossible to guess, we estimate an optimal number through experimental validation. A detailed analysis follows in the next Section.

In a second step, MLPs are assigned to each subset of training data D_i, to learn the associated behavior for each prototypical situation. The MLPs learn how to map state vectors $\mathbf{s}_t$ to an appropriate player reaction $\mathbf{a}_t(\mathbf{s}_t)$, this is the same approach as outlined in Section 3.1.4. Whenever a bot's reaction to a given situation $\mathbf{s}_t$ has to be generated, the MLP associated with the best matching Neural Gas cluster will be selected for behavior generation.

Since we are now dealing with disjoint behaviors that are learned individually, we should again have a look at feature selection. It is very unlikely for a multitude of different behaviors to depend on the same state space features to establish a state-action association. For example, for an aiming behavior, it is obvious that the action is dependent on the position of an opponent, whereas a movement behavior is more likely to depend on the map position. For an automatic behavior learning approach, it would be impossible to select relevant features in advance, since we do not know anymore which behaviors are learned by which MLP. Therefore, we decided for a simple variance based approach for feature selection i.e. *Karhunen-Loewe transformation*, see for instance [Hastie et al., 2001]. In Karhunen-Loewe transformation, a number of relevant features is selected based on the variance of data samples. First, *Principal-Component-Analysis* (PCA) (see also [Hastie et al., 2001]) is applied to a data set. Resulting from PCA, the eigenvectors of the covariance matrix of a data set can be used to project the data to an optimum representation. By using only the

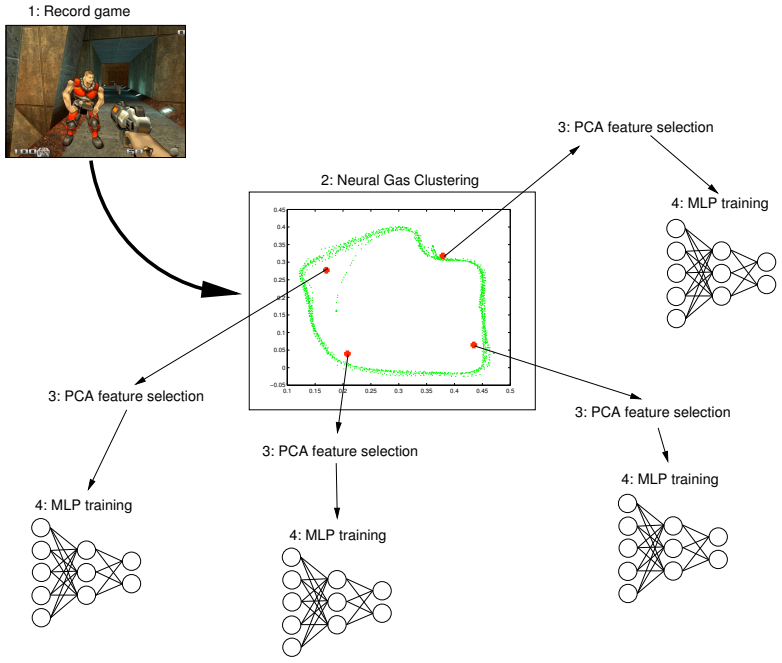


Figure 3.12: A simple concept for reactive behavior learning: training samples are first assigned to prototypical state cluster centres. Then, relevant features are selected using Karhunen-Loewe transformation, finally coupled MLPs are trained and later used for in-game synthesis of behavior.

eigenvector with the corresponding largest eigenvalues, the dimensionality of a data set can be reduced. Selecting relevant features for each learner using Karhunen-Loewe transformation could indeed improve the overall performance. Effectively, selecting about $2-4$ features did improve our overall results, where $2-4$ features usually were able to reconstruct 98% of the complete state vector (the actual number varies among behaviors). Figure 3.12 visualizes the whole learning approach.

3.1.8 Results

In the following we report on the results of imitation learning using single MLPs, and the Neural Gas and MLP combination. In three experimental setups we applied the outlined methods to an imitation learning task in QUAKE II®. Most importantly, we were interested in the principal usability of the approach for behavior acquisition. In addition, we estimated the correct parametrization for (a) Neural Gas clustering (number of prototypes), (b) MLPs/TDNNs (number of hidden Neurons, number of TDNN timesteps), and (c) number of relevant features for different behaviors. The number of TDNN timesteps n for a state vector $\mathbf{s}_t$ at t corresponds to a sequence consisting of the n last state vectors i.e. $\mathbf{S} = \{\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}\}$. The

# NG cluster	# MLP Neurons	# time steps	E_{test}
1	4	1	**0.388**
3	12	1	0.205
9	12	1	**0.175**
9	16	1	0.202

# NG cluster	# MLP Neurons	# time steps	E_{test}
1	4	4	**0.372**
1	16	4	0.263
5	8	4	0.208
9	12	4	**0.188**

Table 3.1: Experiment 1: MSE on test samples for simple movement behavior. The bold results indicate the best and worst results from different configurations.

number of relevant features refers to the number of selected features for Karhunen-Loewe transformation after applying PCA.

Experiment 1: Our first set of training samples showed a player navigating various routes on the map "The Edge - q2dm1", and thus implicitly encoded a running behavior. The demos we used for training contained a total of 2275 state/reaction vector pairs, our evaluation set contained 975 test samples. As the relevant feature vectors in $\mathbf{s}_t$, we decided for agent positions $\mathbf{p} = [p_x, p_y, p_z]$.

Although the Neural Gas is switching between different MLPs, it is not really switching different behaviors. Every MLP represents the same behavior only that it is specialized on a certain part of the state space. Increasing the number of Neural Gas clusters did slightly lower the overall test mean squared error. A single MLP still showed good performance and was capable of learning the behavior encoded in the training set. Increasing the number of learners to more than 2 did not lower our overall results considerably. In addition, the usage of TDNNs could also not improve the results. Details on the results are presented in Table 3.1 and Figure 3.13.

A visual inspection of the gamebots in-game behavior showed a very good imitation of a movement behavior. We did notice a difference when varying the number of coupled learners, whereas weak performers did end up more often in front of walls or fell down ledges. We also noticed, despite an appropriate reproduction of behavior, motion appeared overly smooth. This seems to be a result of the very good interpolation abilities of MLPs. Instead of performing quick turns at corners as they were observed in the training data, the agent started a turning maneuver earlier, and it took him longer to finish it. While the smoothness of motion might sound like a benefit, it did sometimes lead to a rather artificial appearance, which is something we intentionally wanted to avoid.

Experiment 2: The second set of training samples showed an aiming behavior. An overall number of 2059 training samples had to be learned. For evaluation we used a set of 883 test samples. In contrast to the first experiment, we used

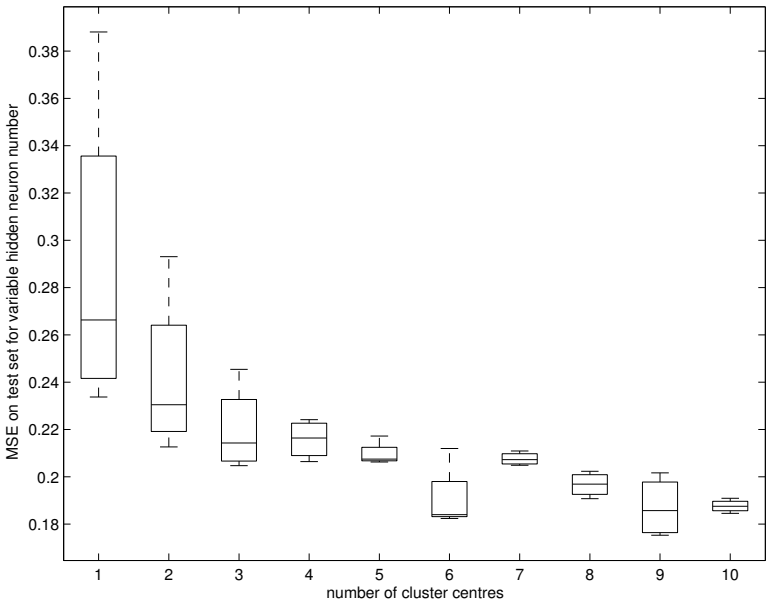


Figure 3.13: Selected MSEs for the simple-run experiment using different numbers of state space cluster. Although an increase in cluster numbers (X-Axis) did lead to a lower MSE, even a single MLP was able to learn the "simple run" behavior.

the opponent position vectors as state descriptors. Detailed results can be seen in Table 3.2 and Figure 3.14

Increasing the number of NG cluster centres did improve the test MSE. Again, choosing more than 2-3 cluster did not have significant impact. Interestingly, the corresponding in-game evaluation showed good aiming behavior for all learners. Even a single MLP managed to learn the connection between opponent positions and player viewangles, thus allowing for successfull imitation of an aiming behavior.

As we already mentionend, the temporal context might also be important for reactive behaviors. We used Time-delay neural networks with a varying number of past states to consider. For the aiming behavior, we could indeed improve the performance, as Table 3.2 shows. Including a rather short state history (we considered the last 4 states) resulted in more accurate behavior reproduction for the test data.

Experiment 3: The third experimental set was a combination of samples of experiment 1 and 2. A total of 4334 training samples and 1858 test samples were used. Relevant features were selected for every learner using Karhunen-Loewe transformation as described in the last section.

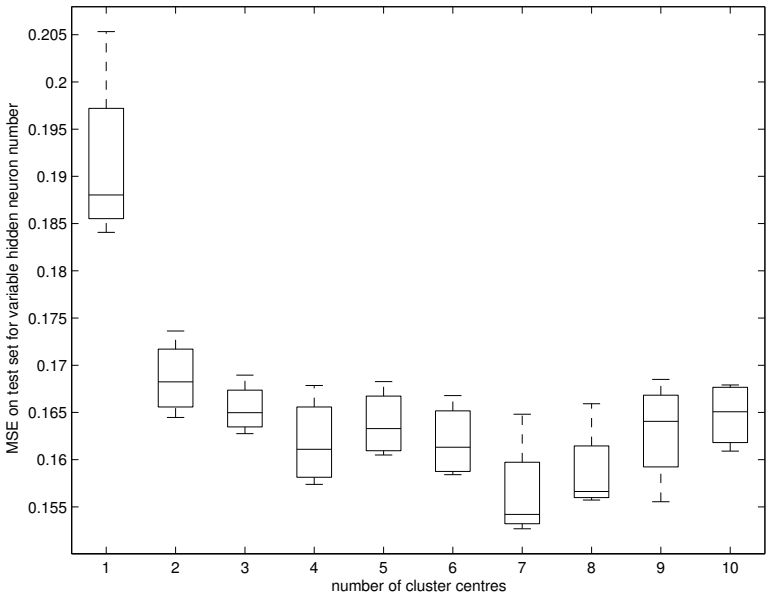


Figure 3.14: MSE for the "aiming" experiment using different numbers of state space cluster.

# NG cluster	# MLP Neurons	# time steps	E_{test}
1	4	1	0.205
1	16	1	0.184
2	12	1	0.167
7	16	1	**0.152**

# NG cluster	# MLP Neurons	# time steps	E_{test}
1	4	4	**0.151**
4	8	4	0.109
4	12	4	0.097
4	16	4	**0.091**

Table 3.2: Experiment 2: MSE on test samples for aiming behavior. The bold results indicate the best and worst results from different configurations.

The best results could been achieved for more than 7 Neural Gas clusters and coupled MLPs, indicating that behavior learning for multiple behaviors can indeed be improved using the proposed distributed approach. Online evaluation showed again some interesting behavior. Due to state space separation and feature selection the artificial player decides on one active behavior at a time. This means, the agent is either aiming or moving. Although there are some in-between behaviors they are only activated in special situations, for instance, the agent starts aiming only when he is close to a place where the aiming training data was collected, or when an opponent was very close to him.

Selecting large numbers of Neural Gas cluster centres did not necessarily result in an increased performance, as can be seen in Figure 3.15 and Table 3.3. The usage of too many learners suffers from overfitting and a possibly insufficient number of

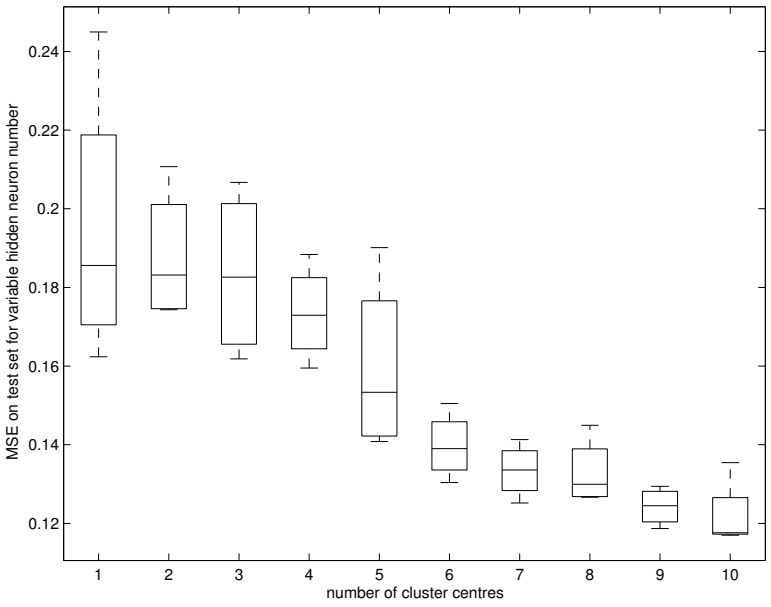


Figure 3.15: MSE for the combination of movement and aiming behavior using different numbers of state space cluster and 4 time steps.

# NG cluster	# MLP Neurons	# time steps	E_{test}
1	4	1	**0.231**
2	16	1	0.188
7	**4**	1	0.138
10	16	1	**0.122**

# NG cluster	# MLP Neurons	# time steps	E_{test}
2	4	4	**0.211**
2	16	4	0.1743
5	**8**	4	0.141
10	8	4	**0.1176**

Table 3.3: Experiment 3: MSE on test samples for combined aiming and movement behavior. The bold results indicate the best and worst results from different configurations.

training samples.

3.1.9 Intermediate conclusion

Although a single MLP is generally able to learn the mentioned behaviors, it showed problems if there are different behaviors included in the training data. In case of multiple implicitly encoded behaviors, the divided approach yielded considerably better performance than the single learner. Moreover, the feature selection using Karhunen-Loewe transformation showed to be quite usefull. For instance, in the third experiment an aiming and movement behavior were included in the training set. The state space separation of the observation data leads to at least two clusters, one cluster representing the situation when an opponent is in view, the other clus-

ter corresponding to situations when no enemy player is visible. The data variance analysis using Karhunen-Loewe transformation showed that for aiming the variables describing the opponent were selected, whereas for simple movements, it was mostly the player positions. Thus feature selection did indeed lead to an improved performance for the coupled learners (here, we are not comparing the MSEs that are obviously not comparable as soon as the number of considered features is reduced).

An interpretation of a single learner as a meaningfull behavior, for example a specialized aiming or jumping MLP, is not possible in all cases. For example, it is possible that multiple MLPs are responsible for movement behavior and only one for aiming. Also, there is no direct connection between the number of learners, implicitly encoded behaviors, and error rates for learning behavior functions. As a rule of thumb we can say that the more behaviors are implicitly encoded within the observational data, the more learners are needed to sufficiently represent and learn the underlying behavioral functions. Also it is interesting to note that an increase of hidden layer neurons in MLPs couldn't compensate for too few learners. Consequently, a modularized architecture seems to be favorable over a monolithic system.

Following these first results, it is obvious that motion synthesis has to be expanded, as this was one of the major problems in the MLP learning approach. While the approach leads to a sufficient program level imitation of behavior i.e. the actions were reproduced successfully, it also leads to a rather artificial action synthesis. Moreover, more complex actions require rather awkward input of mouse and keyboard commands. As the MLP generally led to overly smooth motion, these complex motions showed to be very hard to learn. Consequently, for motion synthesis, we switched to another approach which is directly inspired by movement generation in biological organisms.

3.2 Bayesian learning of reactive movement behaviors

The question of storage and synthesis of movement commands is a popular topic in biology and robotics. Recent results indicate that also movements in humans are very likely not stored as functional mappings, but rather as *movement primitives* which provide a less demanding way of movement encoding [Ghahramani, 2000]. A movement primitive is a unit located in a sensorimotormap that projects limb-trajectories onto motor commands. In the following, we will further outline the concept of movement primitives, and how they can be used for behavior learning in virtual agents. First, however, we have to further develop the underlying learning approach for behavior acquisition.

Here, we make use of a Bayesian framework for behavior learning. Movement synthesis is dependent on appropriate action selection for a given state vector. As already suggested in Chapter 2, if we assume the state of an agent at time t given

by $\mathbf{s}_t$, and the agent's reaction corresponding to timestep t given by $\mathbf{a}_t$, then a naive reactive Bayesian model for estimating the next action is given by:

$$\mathbf{a}_{t+1} = P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}, \mathbf{e}_t, \mathbf{e}_{t-1} \ldots \mathbf{e}_{t-n}) \tag{3.10}$$

Again, we can focus onto the latest observed states when interpreting the model as truly reactive behavior:

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{e}_t) \tag{3.11}$$

where $\mathbf{s}_t$ denotes the game's state at time-step t, and $\mathbf{a}_k$ denotes a specific action. The next action $\mathbf{a}_{t+1}$ is selected by picking the action k maximizing $P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{e}_t)$. If we neglect environmental influences $\mathbf{e}_t$, the probability of executing a specific action $\mathbf{a}_i$ can be denoted as

$$P_{\mathbf{a}_i} = P(\mathbf{a}_i|\mathbf{s}_k) \tag{3.12}$$

Although this model considerably simplifies equation 3.10, it should be accurate for short-term relevant actions as intended by reactive behavior learning. However, in a sequence of actions, not every action can be executed as a successor of another one. Also, on the one hand humans tend to move in a smooth way, at least compared to what would be possible for an artificial player. On the other hand, humans are bound to physical limitations of their hand motion, besides, some players might have certain *habits*, they tend to jump for no reason or make other kinds of useless, yet very human movements (as Chapter .6 will show, these are indeed very important to create a human-like agent impression). To reflect those aspects, the probability of executing an action $\mathbf{a}_i$ as a successor of a $\mathbf{a}_t$ should be incorporated. It can be denoted as

$$P_{\mathbf{a}_i} = P(\mathbf{a}_i|\mathbf{a}_t) \tag{3.13}$$

Assuming the conditional probabilities $\mathbf{a}_t$ and $\mathbf{s}_k$ are independent (the current state should only influence the next action), the next action can be selected according to:

$$\mathbf{a}_{t+1} = \underset{i}{\operatorname{argmax}}\, P(\mathbf{a}_i|\mathbf{s}_t, \mathbf{a}_t) = \underset{i}{\operatorname{argmax}}\, \frac{P(\mathbf{a}_i|\mathbf{a}_t)P(\mathbf{a}_i|\mathbf{s}_t)}{P(\mathbf{a}_i)} \tag{3.14}$$

In order to extract the required priors from human observation data, we first need an appropriate representation for states and action data $\{\mathbf{s}_i, \mathbf{a}_i\}$. The representants should be suited for generating the desired motion sequences. Regarding state descriptors, we further extend the model of situation aware learners proposed in Section 3.1.5 by first deriving a topological state representation. For action representants, we make use of the already mentioned movement primitive concept.

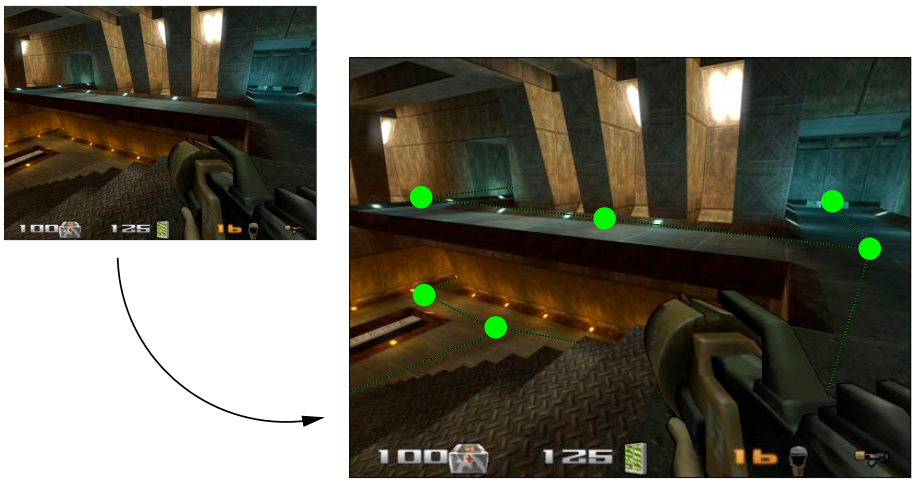


Figure 3.16: A simple map and its possible corresponding topological representation. The dashed green lines are only have didactic purpose, nodes are not interconnected.

3.2.1 Learning a topological representation

Again, the chosen state representation is inspired by biological place cells. For a representation of the virtual world we are learning a topology using a Neural Gas algorithm which we already explained in Section 3.1.6. Here, the training data used for Neural Gas learning consists of all locations $\mathbf{p} = \{x, y, z\}$ a human player visited during various plan executions. Application of a Neural Gas algorithm to the player's positions results in a number of prototypical positions. Since we focus, in this Section, on action synthesis of complex moves, it is important to mention that the prototypical positions are implicitly interconnected by the player's actions. However, for this approach, we do not need the toplogical structure of the prototypes given by the Neural Gas edge learning. Figure 3.16 shows a small map and its corresponding topological representation, edges were only drawn for clarification reasons. In contrast to place cell structures learned in the Section 3.1.5, we found that for a probabilistic approach a much finer graining is needed. This effectively results in more accurate representations of player positions, and thereby also of the map's structure.

By assigning training samples of recorded player actions to cluster centres in the topological map, similar to the approach in section 3.1.5, small sets of localized training samples are generated. Each separated training set defines the legal actions for a specific region, not only in the topological map, but also in the simulated 3D world.

The storage and organization of movement in biological organisms is an extremely complex task. When viewing it as a control problem, the motor system receives thousands of sensory inputs to effectively control hundreds of motor units [Wolpert et al., 2001]. The sheer number of muscles within an organisms body leads to an unmanageable number of degrees of freedom. An example by Wolpert et al. [2001] illustrates the problem as follows: ".. consider the 600 or so muscles in the human body as being, for extreme simplicity either contracted or relaxed. This leads to 2^{600} possible motor activation patterns, more than the number of atoms in the known universe".
Obviously, animals and humans are still able to internally deal with the challenging task of motor representation. Instead of directly controlling motor activation commands, recent results indicate that complex movements are the outcome of a combination of *movement primitives* [Ghahramani, 2000]. A movement primitive is in this context a computational element in a sensorimotormap that transforms desired limb-trajectories into actual motor commands [Thoroughman and Shadmehr, 2000]. For instance, [Fod et al., 2002] interpreted this concept in a computational approach, by linear superposition of primitive motor commands. As they noted, movement primitives can be interpreted as a basis set of motor commands. Superposition of these can finally lead to complex movements.
[Fod et al., 2002] further introduced *action primitives* which instantiate certain motor commands, and thus denote a specific superposition of movement primitives. Interestingly, recent findings in modern sport-psychology suggested *basic action concepts* [Schack, 2004, Schack and Mechsner, 2006] as a way for internally structuring complex movements. Basic action concepts denote a characteristical motor unit within a sequence of complex movement patterns. Although basic action concepts are introduced in a different context, they come very close to the concept of action primitives.
The images show an interpretation of basic action concepts (or action primitives) in the context of QUAKE II® by visualizing prototypical action units. Red arrows denote the view direction, green arrows the movement direction.

Figure 3.17: Movement primitives in humans

3.2.2 Extracting movement primitives from observation data

Evidence form neuroscience indicates that complex movements in humans and animals are built up by combinations of simpler motor or movement primitives [Ghahramani, 2000]. Thus, for a more life-like appearance of a computer game character's motion, a biological approach utilizing movement primitives seems promising. In Figure 3.17 we summarize a little background information on how movement is encoded in biological organisms, and possibly artificial agents might work.

In our experimental environment, an action of a player usually comes down to a 7 dimensional vector. In contrast to the actions in the MLP learning, we decided to also include a fire action. This was necessary in order to learn complex moves such as the rocket jump:

$$\mathbf{a} = \begin{pmatrix} yaw\ angle \\ yaw\ anglesign \\ pitch\ angle \\ velocity\ forward \\ velocity\ side \\ velocity\ up \\ player\ fire \end{pmatrix}$$

A motion vector completely defines an action the player executed[2]. To identify the underlying set of basic movement-primitives, we apply a variance based approach, *Principal-Component-Analysis* (PCA) (see for instance Hastie et al. [2001]). PCA is applied to the training samples, the resulting eigenvectors $\mathbf{e}$ provide the elementary movements of which a motion vector can be constructed. No dimension reduction is applied at this stage, PCA is simply used for computing an optimal representation. Thereby a projection of the observed motion vectors onto the eigenvectors can be viewed as a reconstruction of player movements by their movement primitives. Although dimensionality reduction might be possible, it is not needed due to the low dimensionality of motion vectors. Formally, PCA does not change the data and is merely a rotation. Thus, it is also not required at this stage. We simply included it to explain the connection to the biological movement primitives.

Frequently executed player movements can be grouped to gain *action primitives*. To acquire a set of *action primitives*, the projected motion vectors are clustered using a *k-means* vector quantization algorithm (for a good introduction to

[2] A motion vector corresponds to a human player's mouse and keyboard inputs. While we effectively only see the outcome in the virtual world, we have to keep this in mind. We are, after all, modelling human motion - though more likely hand motion. These motion vectors can be directly send to a server, no further mapping on agent motor commands is needed.

k-means we recommend Hastie et al. [2001] [3], thereby staying close to the action primitive derivation suggested in [Fod et al., 2002]. This results in a set of cluster centers, each of which represents a single action unit or action primitive.

The number of action primitives does have an influence on the overall smoothness of the learned motion sequences. The right number of cluster centers depends on the number of training samples and on the variety of observed motions. However, the representation of a large number of training samples could not be further improved by choosing a higher number of cluster centres. Instead, we found substantial overfitting when increasing the number of primitives over a certain value. This indicates that there might be a fixed number of action primitives which guarantees a smooth execution of motion sequences. A detailed analysis follows in the next Section.

3.2.3 Results

To test the presented approach, we carried out a set of eight smaller experiments. Each experiment dealt with a separate training sample set, in which (at times) complicated movement sequences were executed several times by a human player. The observable motions varied from simple ground movements to more complex jumping or shooting maneuvers (or combinations of both). In addition, a larger training set, this time a real match between two human players, was used. Here, we decided for a different set of behaviors as in the MLP learning approach. Using the probabilistic approach, numerous different behaviors can be learned, especially more complex movements and motions.

The acquisition of the priors (see again Equation 3.14) needed for action synthesis is relatively simple. Each movement vector can be assigned to a node in the topological representation and it can be assigned to an action primitive. Counting the evidences of action primitives in all nodes results in a $m \times n$ matrix, where m denotes the number of nodes in the topological map and n denotes the number of action primitives. A matrix entry at position k, i denotes the probability $P(\mathbf{a}_i|\mathbf{s}_k)$ of executing an action primitive $\mathbf{a}_i$ for node number k. The probabilities can be extracted from the training samples by inspecting the observed action primitive sequence, resulting in a $n \times n$ transition matrix, where n denotes the number of action primitives. The probabilities for $P(\mathbf{a}_u|\mathbf{a}_l)$ are calculated and stored in a similar fashion.

When placed in the game world, the next action for the artificial game character is chosen randomly using a roulette wheel selection according to the probabilities for each $\mathbf{a}_i$ in Equation 3.14, or by selection the action $\mathbf{a}_i$ corresponding to

[3] We decided to skip a more detailed explanation of the k-means algorithm because of the similarities to the already introduced Neural Gas algorithm. By restricting the attraction of neighboring units in the Neural Gas algorithm, we could effectively extract a similar clustering as by using k-means, since the algorithms are then almost identical.

# NG cluster	# action primitives	E_{test}
5	10	0.6096
30	30	**0.2709**
30	70	0.3618
5	80	**0.7314**

# NG cluster	# action primitives	E_{test}
5	10	0.1109
30	30	0.0639
30	70	**0.0610**
5	80	**0.1590**

Table 3.4: MSE results on rocket (left) and a longer run with a long jump. MSE results are not really comparable, due to the different complexity in observed behaviors in both cases. The right table shows results of a longer movement sequence, where walking in one direction was dominant, thus, leading to a rather low MSE.

Figure 3.18: The artificial game character, observed while executing one of the experiment movements - a jump to an otherwise unreachable item

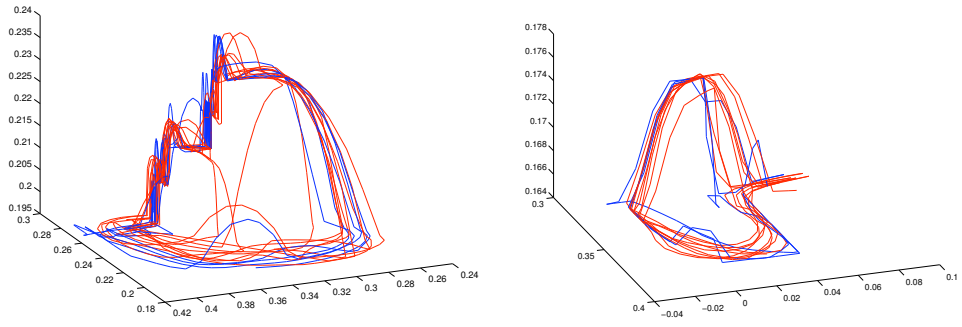
$$\mathbf{a}_{t+1} = \underset{i}{\operatorname{argmax}} \frac{P(\mathbf{a}_i|\mathbf{a}_t)P(\mathbf{a}_i|\mathbf{s}_t)}{P(\mathbf{a}_i)} \tag{3.15}$$

For evaluation, since we are still dealing with reactive behavior, we calculate the mean squared errors on test sets. In all tests, we excluded 40% of training data, and used it for evaluation. MSEs are calculated as follows:

$$E_d = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{a}_i^{desired} - \mathbf{a}_{t+1})^2 \tag{3.16}$$

where $\mathbf{a}_{t+1}$ is the outcome of Equation 3.15, or selected via a roulette wheel selection. For in-game movement synthesis, we usually preferred the roulette wheel selection. For MSE calculation we usually used the argmax_i for the probabilities for each $\mathbf{a}_i$. Table 3.4 summarizes MSE results for learning the rocket jump, and for a longer movement sequence. Generally, all movement sequences were learned sufficiently, when considering the optimum parametrization for each individual experiment. Since the MSE tables, and graphs, are rather similar, we decided to only include the more interesting behaviors.

Additionally, we decided to evaluate based on generated movement trajectories, and based on visual inspection of synthesized behaviors. In the experiments, we



(a) (b)

Figure 3.19: Comparison of the artificial player's (blue) movement trajectories in 3D and the trajectories of a human player (red). Figure 3.19(a) shows a combination of different jumps to reach an item located on a platform. Figure 3.19(b) shows a repeated jump through a window.

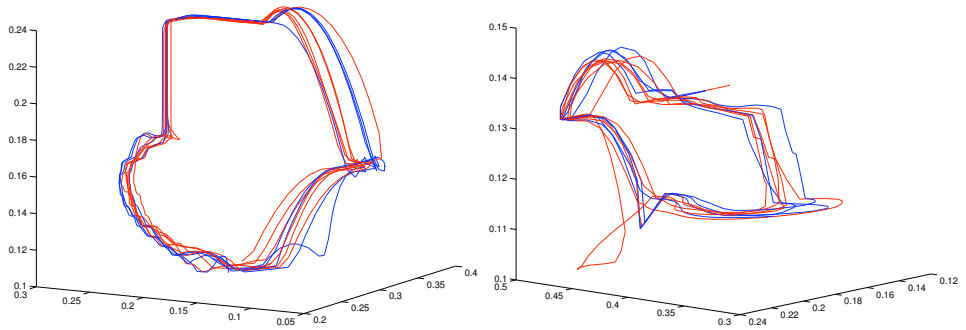
were interested in finding sufficient numbers of action primitives for each test set, and sufficient number of situation prototypes.

Generally, the approach managed to learn and imitate the observed behaviors. Even complicated actions, for instance, the already mentioned rocket jump, could be learned and synthesized correctly. Occasionally, in other movement behaviors, the agent ran into a wall and stopped moving. However, this appeared to be the exception. The agent does not have a direct sensing of obstacles which is clearly a disadvantage when dealing with narrow hallways or obstacles.

Although staying very close to the training set movements, an exact reproduction almost never occurred because of the randomness in sequence generation. While this might seem like a disadvantage, it definitely adds to a more human-like impression. Despite the random action selection mechanism, the synthesized behaviors can be easily identified as the observed behavioral patterns, indicating that the agent indeed imitates human behavior. Figure 3.19 and Figure 3.20 show the trajectories of the artificial player compared to a human test player in 3D. However, not staying as close as possible to the human observation data can have some disadvantages, as already mentioned.

Regarding the number of place cells, we found that this is indeed very much dependent on the actual map size, and on the complexity of a certain movement. To sufficiently cover an average sized map, we found that a number of $600 - 800$ prototypes is needed. However, since this also very much depends on the map's architecture, usually, a sufficient number of prototypes has to be selected manually.

Regarding the number of action primitives, the results are more interesting. Figure 3.21 the influence of number of action primitives on the MSE. What we usually



(a) (b)

Figure 3.20: Comparison of the artificial player's (blue) movement trajectories in 3D and the trajectories of a human player (red). The experiment underlying Figure 3.20(a) contained an elevator usage and a long jump to a platform. While Figure 3.20(b) displays the trajectories extracted from the experimental results illustrated in Figure 3.18

find is a minimum number of action primitives for optimal results, for each sample set. In the case of only a rocket jump, $30 - 50$ action primitives appear sufficient. Although the amount of training data and observed motion complexity have a huge influence, even larger training sample sets, or more complex moves, showed a similar characteristic. This can be seen in Figure 3.2.3, where a more complex training set was used (2 players fight each other). In that example we used 11768 training samples and 5044 test samples (QUAKE II® records the training data at a rate of 10Hz), and still could find that a rather low number of action primitives was sufficient (about $80 - 120$). Moreover, increasing the number of primitives above approx. 200 did lead to worse results, and did also not improve the overall smoothness of motion synthesis (at least not to a degree that is noticeable for a human observer). This indicates that a there might be a fixed number of action primitives that guarantees appropriate movement generation.

Besides the basic reconstruction of certain movements, the artificial player's motions themselves, his way of turning, jumping and running, were naturally looking, thus creating the illusion of a human player (this applies to motion, not the tactical/strategic decisions).

The more realistic training set, two human players competing on a smaller map, finally resulted in a very good imitation of a broader repertoire of a human player's motion, indicating, that our approach is suitable for larger scale problems. The strong coupling between the game characters position in the topological representation and the selection of motion primitives made the character appear smart, by acting in an appropriate way to the architecture of the 3D game world - jumping

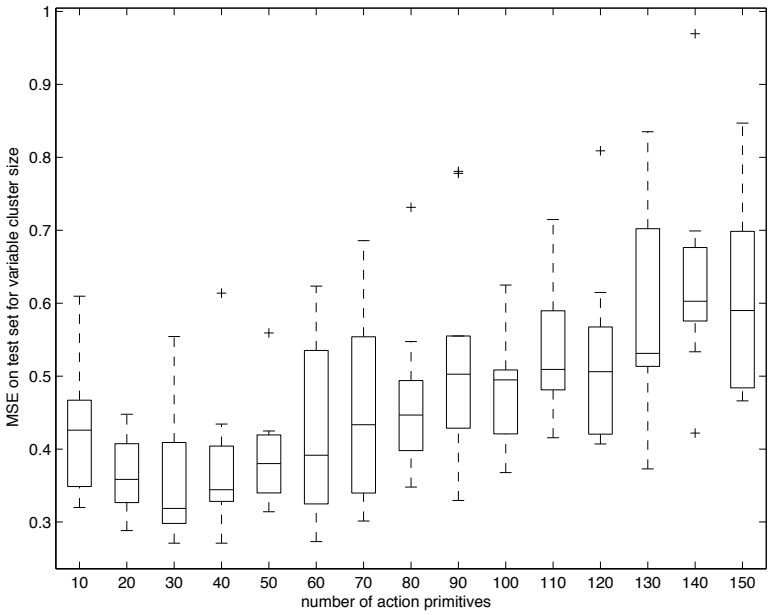


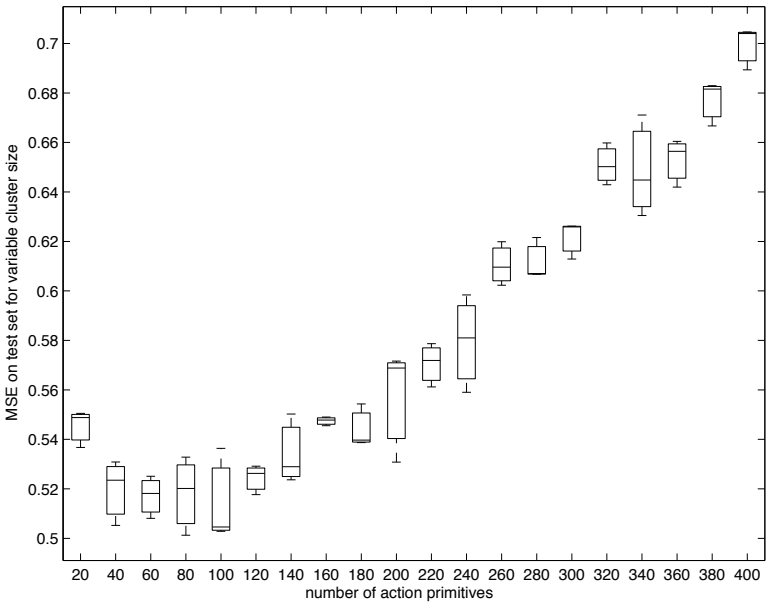
Figure 3.21: MSE for the synthesis of a rocket-jump under varying number of movement primitives and varying number of state cluster. The horizontal axis denotes number of primitives.

over cliffs or standing still when using an elevator. In addition, the approach preserved certain player habits by (in one case) executing senseless jumps from time to time. Since no information about the enemy player was introduced during live play, some kind of shadow fighting could be observed, as if an enemy would be present (approaches in the next Chapter will take the opponent position into account).

3.3 Summary

Reactive behaviors represent the most basic behaviors, usually immediate reactions on sensory input data. In our computer game test environment, this comes down to basic movements, but also aiming behavior. In this chapter we outlined approaches in mimicking reactive behaviors in artificial agents. Given the basic model discussed in the first chapters of this thesis, we explored a functional and a probabilistic approach on imitating these behaviors.

For the function approximation, a *Multi-layer Perceptron* (MLP) serves as a simple universal function approximator. One MLP was trained using the observational data of a human player performing specific behaviors. To account for multiple behavior included in the observational data, we extended the approach to multiple MLPs where the training samples are first partitioned using a *Neural Gas* vector



[Rocketjump results on varying cluster numbers] MSE for varying action primitive cluster numbers on the map q2dm1.

quantization algorithm, afterwards a single MLP is trained on each individual set. The rough categorization of training data is similar to concepts in spatial orientation in biological organisms, so called place cells. Thereby, multiple specialized MLPs are responsible for creating the overall agent's behavior. The divided approach finally managed to learn multiple behaviors at once, and could sufficiently synthesize actions for novel situations. We also used *Time-delay neural networks* (TDNN) to account for temporal dependencies. These managed to gain better results for certain behaviors.

In the probabilistic approach, we introduced the concept of *movement primitives* within the learning in games domain. In detail, we first derived action primitives (action units) by k-means clustering over all observed player actions. Here, we constructed action units by a superposition of movement primitives, in our case movement primitives are the result of an application of PCA to motion observations. The movement and action primitives are inspired by the recently discovered concepts in animals and humans. Then we clustered player positions to extract a topological world representation, similar to the place cell inspired approach in the MLP learning. Based on movement observations, we derived action primitive probabilities dependent on nodes within the topological map, thereby realizing a situation aware selection of action units. In addition, we introduced a dependency on the last executed action primitive to reflect physical limitations in a human player's input (usually mouse and keyboard) devices. This approach led to a sufficient reproduc-

tion of more complex moves. Moreover, the appearance of the game bot was very close to a human player.

In general, both approaches were able to learn and generate behavior. Although the MLP based approach worked well for movement and aiming behaviors, it clearly lacked a life-like appearance in movement generation. In contrast, the probabilistic approach showed a very realistic, life-like generation of movement or motion behavior, where even complex moves consisting of sequential execution of action units could be learned. Evaluation is an important topic for mimicking of behavior. The question when a behavior is learned correctly is not easy to answer. In our experiments, we could see that it is not always a low mean squared error with respect to test data samples that guarantees a successful and realistic behavior synthesis. Although the MLP based learning lead to a sufficient approximation of training data, the action synthesis tended to look artificial. It can be far more important to directly inspect the behavior success rates, and the overall appearance of the generated behavior for novel situations.

Chapter 4

Learning tactical behaviors

While strategic or reactive behaviors can be expressed in a rather straight forward manner, for instance, strategic behaviors implement a goal-attainment behavior, this is not as intuitive for *tactical behaviors*.

According to our definition in Section 2.4.2, tactics are behaviors in between reactive and strategic behavior. The main problem in tactical behaviors is due to the shire number of possible behaviors to consider; a huge amount of different behaviors is imaginable. Due to the versatility in behavior, a single approach learning all of these behaviors is unrealistic. In FPS games, tactical behaviors can be often found in direct player vs. player interactions. Consequently, we decided to approach imitation learning of tactics for a few exemplary behaviors i.e. behaviors needed for intelligent player-opponent interactions. Given the rather specialized tasks, the underlying approaches are summarized more briefly, in contrast to the methods for learning reactive or strategic behavior.

Parts of the work in this chapter were published in [Bauckhage and Thurau, 2004] and [Thurau and Bauckhage, 2005].

4.1 Learning tactical situation handling

For the first approach we consider tactical weapon handling. This is, in the case of QUAKE II® , situation dependent weapon selection and weapon specific aiming behavior. In the last Chapter, a *Multi-layer Perceptron* (MLP) based approach sufficiently learned an aiming behavior from human observation data. Notably, the learned aiming behavior did not consider the used weapon. As already mentioned in Section 2.4.2, all available weapons behave differently. Therefore, a single aiming behavior can not produce sufficient results for all supplied weapons. Besides successfull aiming, the problem of appropriate weapon selection arises. Weapons do not only behave differently, they also have different purposes. Therefore, we can not only focus on weapon handling alone, but also have to consider situation

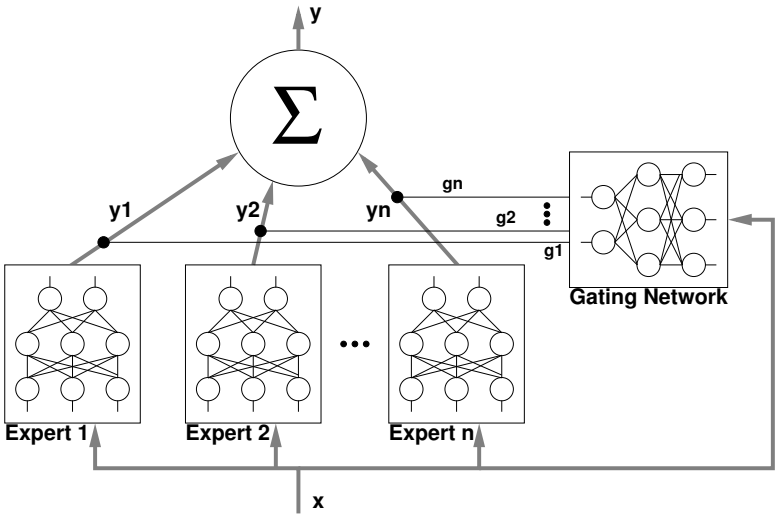


Figure 4.1: A mixture of experts architecture consists of a set of expert networks and a gating network. Given an input $\mathbf{x}$, the output $\mathbf{y}$ of the system is computed as the sum $\mathbf{y} = \sum_j g_j(\mathbf{x}) y_j(\mathbf{x})$.

dependent weapon selection.

Following the positive results for learning reactive behaviors using MLPs, in the following, we elaborate a learning approach consisting of multiple MLPs. For learning an aiming behavior, these are combined in a *Mixture of Experts* (MOE) architecture [Jacobs et al., 1991]. This approach is similar to the already introduced combination of MLPs using a topological structure. Both approaches try to make a complex learning problem feasible by partitioning training data in the corresponding state space. In contrast to the place cell approach, the Mixture of Experts provide a soft partitioning of the training data. This means that the samples are not divided based on a clustering in state space, instead, they are divided based on the actual approximation quality of each individual expert. The output is, instead of the winner takes all selection of the place cell approach, a weighted sum of outputs of each expert. In addition, for learning the situation dependent weapon switching, we are applying a simple *Time-delay neural network*(TDNN). Before coming to the details of our implementation, we first briefly introduce the MOE architecture in the next Section.

4.1.1 Mixture of Experts

As introduced in Section 3.1.1, given a training set of pairs of vectors $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ where $\mathbf{y}_i = f(\mathbf{x}_i)$, a MLP can solve the regression problem by learning a function g that approximates f through minimizing the sum of errors

$$E = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - g(\mathbf{x}_i))^2 \tag{4.1}$$

However, as we could see in the last Chapter, a single MLP is not in all cases able to sufficiently approximate the underlying function. Especially, for different tasks to be learned at the same time we can expect interferences. The MOE approach as introduced by Jacobs et al. [1991] tackles this problem by means of a divide and conquer strategy. Instead of fitting a single function into the training data, it uses a mixture of local experts which are moderated by a gating network. As shown in Figure 4.1, the basic idea of the MOE technique is to compute the vector $\mathbf{y}$ as a weighted sum of outputs produced by n expert networks. Given a vector $\mathbf{x}$, the corresponding $\mathbf{y}$ thus results from

$$\mathbf{y} = F(\mathbf{x}) = \sum_{j=1}^{n} g_j(\mathbf{x}) f_j(\mathbf{x}) \tag{4.2}$$

where $g_j(\mathbf{x})$ denotes the weighting for each expert i, dependent on the input signal $\mathbf{x}$. Learning in a MOE network is equal to finding appropriate parameters for the experts and the gating network. Assuming MLPs for the local experts and the gating network, the corresponding weights W_i for each learning structure need to be adapted. It is common to require $\sum_j g_j = 1$ which can be realized by assuming the g_j to be soft-max functions of the output layer values s_j of the gating network:

$$g_j(\mathbf{x}) = \frac{e^{s_j(\mathbf{x})}}{\sum_k e^{s_k(\mathbf{x})}} \tag{4.3}$$

Apparently, this suggests a probability interpretation of the weights g_j. The naive approach for estimating the parameters would be, to minimize the error function E

$$E = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{y}_i - F(\mathbf{x_i}))^2 \tag{4.4}$$

by, for instance, a gradient descend. However, more sophisticated approaches for estimating the parameters include usage of the *expectation maximization* (EM) method (see for instance [Dempster et al., 1977], or [Rao et al., 1997]). Note that a MOE model simultaneously learns a segmentation of the input space and the mappings from the segments to the output space. Unlike clustering techniques like the Neural Gas algorithm, the MOE approach yields a soft partition of the input space. Finally, extensions to hierarchical MOE models are possible where each expert itself consists of a mixture of experts.

4.1.2 Context aware weapon handling

(a) Targeting its expected position . . .

(b) . . . an sending the missile there . . .

(c) . . . will most likely hit the opponent.

Figure 4.2: Example of experienced handling of the QUAKE II® rocket launcher. In contrast to the 'direct hit' weapons in the arsenal, rockets are relatively slow. This requires to anticipate the opponent's movement and to target its expected rather than its current position. A gamebot will have to reproduce this behavior in order to appear life-like.

In a series of experiments, we examined whether a Mixture of Experts architecture can learn human-like handling of different weapons in QUAKE II® . Thereby, we focused on a subset of three weapons and considered the *blaster*, the *rocket launcher* and the *railgun*. As already mentioned in Section 2.4.2, these weapons do have different characteristics. The railgun is very accurate at long-distances, but comes with a considerable recharge delay. The rocket launcher is more intended for mid-distance combat. Human players avoid its use in close combat because the splash of rockets can cause harm to oneself. The blaster is a handgun with a high bullet frequency suited for close combat. According to their characteristics, these weapons are typically used in different (spatial) contexts. However, their use does not only differ with respect to the distance to an enemy player; their ballistics vary as well. This requires to add anticipation to the handling of the rocket launcher, as is exemplified in Figure 4.2, instead of directly targeting their opponents, ex-

training method	pre-trained experts	# hidden neurons	E_{TRAIN}	E_{TEST}
$\nabla_\theta L$	no	4	0.01318	0.01280
EM	no	4	0.01332	0.01293
EM	no	7	**0.01125**	**0.01133**
EM	yes	7	0.01153	0.01167

Table 4.1: Results in learning human-like aiming behavior using a Mixture of Experts architecture.

perienced players fire rockets towards a location where their adversaries will most likely appear in a few moments.

4.1.3 Results

The training data consisted of different sequences where each weapon specific aiming behavior could been observed for a few minutes. For the blaster, 3651 frames were recorded, for the rocket launcher we obtained 5641 frames and for the railgun 8378 frames. The recorded demos perfectly reflected the recharging times of the considered weapons. While for the blaster the ratio of frames where the player fired a shot to the total number of frames was approximately $3/4$, for the railgun it was about a mere $2/7$ on average.

The data that was extracted from this demos consisted of pairs $\{\mathbf{a}_i, \mathbf{s}_i\}$ where the state vectors $\mathbf{s}_i \in \mathbb{R}^6$ encoded the spatial angle $\{\text{opponent yaw}, \text{opponent pitch}\} \in [-180, 180]$, and the distance between the player and its opponent $\{\text{opponent distance}\} \in [0, 2000]$ as well as the current activated weapon indicated by a flag $\{\text{rail gun,blaster,rocket launcher}\} \in [0, 1]$. For the experiments, we used a short state history i.e. the last 5 state vectors were considered, in the case of QUAKE II®, this corresponds to a period of 0.4 seconds. Effectively, resulting in a 25 dimensional state vector, consisting of $\mathbf{s}^\beta = [\mathbf{s}^\alpha_t, \mathbf{s}^\alpha_{t-\Delta t}, \ldots, \mathbf{s}^\alpha_{t-4\Delta t}]$.

The action vectors $\mathbf{a} \in \mathbb{R}^7$ consist of a flag indicating whether to shoot or not $\in [0, 1]$, and a corresponding target viewangle $\{\text{yaw angle}, \text{pitch angle}\} \in [-180, 180]$.

Weapon handling consists of weapon selection and aiming. Since weapon selection and aiming are in fact completely different behaviors, we decoupled the, and learned them individually. Thus, the task of imitating weapon handling becomes easier. It showed that weapon selection can be implemented in a straight forward manner by using TDNNs. Effectively, weapon selection seemed to be mostly based on the distance of an opponent. Since we used TDNNs in the already introduced reactive behavior learning in Section 3.1, we decided to not include further details

about this approach. However, the output of the TDNN, a binary output vector $\mathbf{a}_w \in \mathbb{R}^3, \mathbf{a}_w = [\text{rail gun}, \text{blaster}, \text{rocket launcher}] \in [0, 1]$ consisted of flags indicating the type of weapon to be used, and is used as an input vector for the MOEs, responsible for an aiming behavior. Learning appropriate aiming seems to be the harder problem though.

For learning weapon handling, we experimented with a MOE architecture of three experts and a gating network. The gate was chosen to have 10 hidden neurons, the experts were tested with 4 and 7 hidden neurons, respectively. Training was done in 5000 epochs using gradient descend or the EM algorithm. For the sake of comparison, we also tested an architecture where the three expert networks were pre-trained, one for each weapon, and the gate was subsequently trained to select the most appropriate expert.

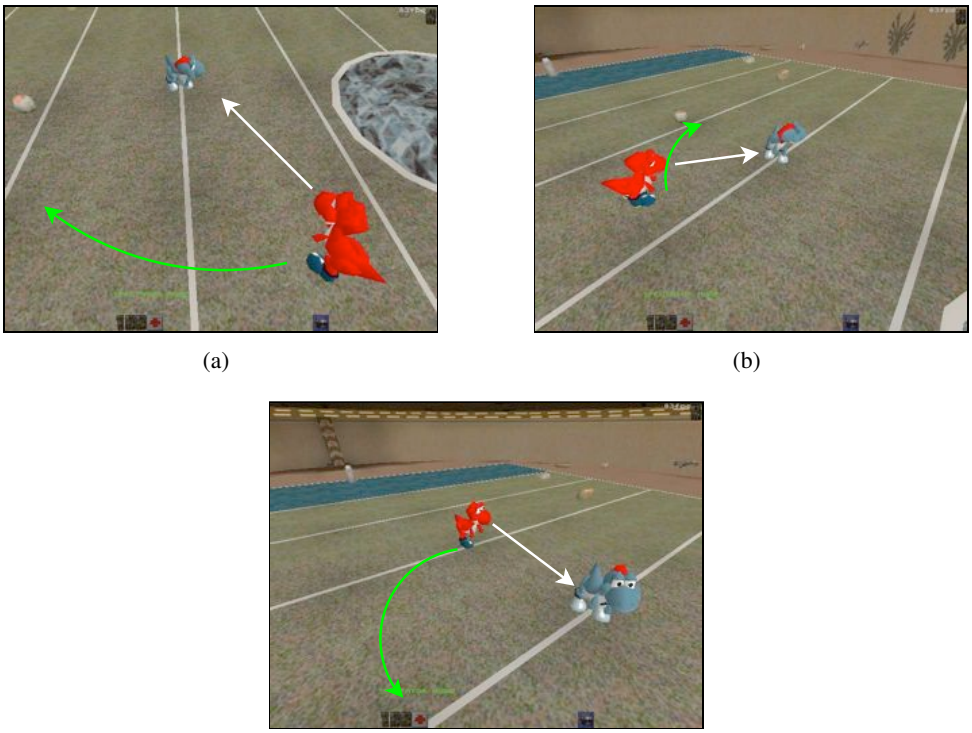
An in-game examination of the resulting network architectures showed gamebots that were able to produce the desired behavior. Using the TDNN for weapon selection, if the opponent was close, the bots selected the blaster, in mid-distance combat they referred to the rocket launcher and for long-distance fights they used the railgun. Relying on the MOE architecture, the bots reproduced the aiming behavior contained in the demos. When using the blaster or the railgun they directly targeted their enemies; when using the rocket launcher they 'predicted' their adversaries trajectory and fired accordingly. Finally, the shooting frequencies characteristic for the different weapons were learned as well.

Table 4.1 displays some of the results we obtained from an offline evaluation. Obviously, the architectures with expert networks with 7 hidden neurons perform better even though, compared to the ones with 4 hidden neurons, the gain is small. What is noticeable, is that the difference between training error and test error is small in every case; this backs the above observation of a good reproduction of the behavior encoded in the training material. However, for the evaluation, the in-game synthesis of behavior seemed to be more important.

4.2 Bayesian learning of tactical movement behavior

As already mentionend, tactical movements are usually found in 1 vs. 1 interactions of players. For example, players try to evade the direct focus of an opponent, or they try to gain an advantage by moving around an opponent and keeping him in their field of view. The latter, a so called *circle-strafe* behavior, is illustrated in Figure 4.3.

In Chapter 3, we introduced the concepts of biologically motivated movement synthesis using movement and action primitives. These very promising results led to the assumption that these concepts are also applicable in the context of tactical movement imitation. Notably, the situation for synthesis of tactical moves is considerably different than for environment dependent reactive behaviors. As we will see, the representation of the spatial relations of the player and opponent are crucial.



(a) (b)

(c)

Figure 4.3: The circle strafe maneuver: the player circles around the opponent while keeping him in focus.

Following Chapter 3, a probabilistic expression for choosing an action $\mathbf{a}_i$ at timestep $t+1$ can be stated as follows:

$$\mathbf{a}_{t+1} = \underset{j}{\operatorname{argmax}}\, P(\mathbf{a}_j|\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}, \mathbf{e}_t, \mathbf{e}_{t-1} \ldots \mathbf{e}_{t-n}) \tag{4.5}$$

where $\mathbf{s}_t$ denotes the state at timestep t, and $\mathbf{e}_t$ denotes environmental influences at t. For reactive behaviors, we restated Equation 4.5 to only consider states at the latest observed state $\mathbf{s}_t$. Although we are dealing with tactics, we decided to also restrict the number of states to take into account. Here, the meaning of tactical behavior is not grounded in the assumption of depending on a longer state-history than reactive behaviors. It is only dependent on a *meaningful* interpretation as a tactic behavior. While this is, as already noted, rather subjective, the movement behaviors we intend to learn in this Chapter are still considerably different than the reactive movements introduced in Chapter 3. However, implicitly a temporal dependency is included in the representation of states which will be introduced in the next Section.

By assuming the behaviors to be dependent solely on the states at timestep t, and by neglecting environmental influences, we can restate Equation 4.5 as follows:

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t) \tag{4.6}$$

Probably the most important features for describing tactical movement behavior in 1 vs. 1 situations are the positions of the two involved players. Based on the relative positions to each other, the players will decide about their next action. For instance, if they come too close, they will most probably back off, whereas if they are in a good shooting distance (which then usually applies to both players), the players are very likely to perform circle strafes or other dodging behaviors. Thus, tactical movements are mostly dependent on the current state of the player $\mathbf{s}_t^p$, and the state of the opponent $\mathbf{s}_t^o$. Besides, since we again aim at life-like imitation of motion, we have a dependency on the latest executed action $\mathbf{a}_t$, as already introduced in Section 3.2. Therefore, we will substitute the state vector $\mathbf{s}_t$ by a triple of player and opponent state vectors, and past action vectors $(\mathbf{s}_t^p, \mathbf{s}_t^o, \mathbf{a}_t)$:

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t^p, \mathbf{s}_t^o, \mathbf{a}_t) \tag{4.7}$$

by assuming independence of past actions $\mathbf{a}_t$ and states $\mathbf{s}_t^p, \mathbf{s}_t^o$, the action selection probabilities can be restated as follows:

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, \frac{P(\mathbf{a}_k|\mathbf{s}_t^p, \mathbf{s}_t^o)P(\mathbf{a}_k|\mathbf{a}_t)}{P(\mathbf{a}_k)} \tag{4.8}$$

where both $P(\mathbf{a}_k|\mathbf{s}_t^p, \mathbf{s}_t^e)$ and $P(\mathbf{a}_t|\mathbf{a}_k)$ can be estimated from observations of

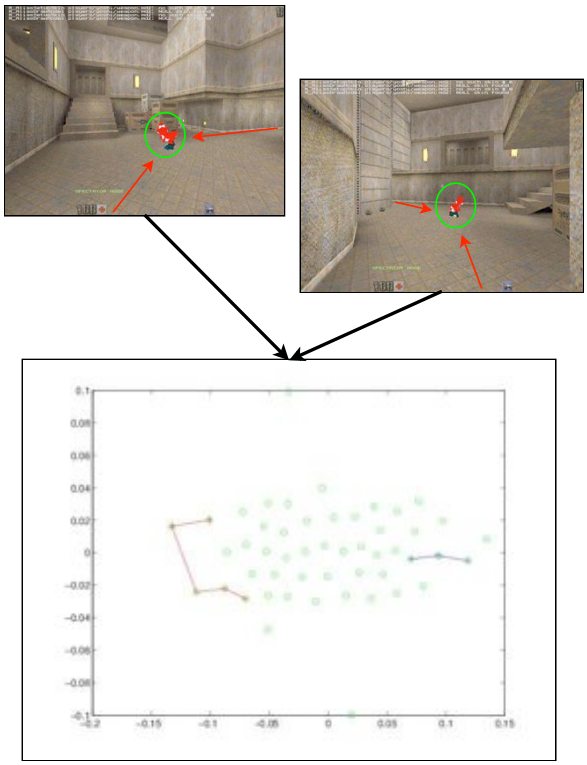


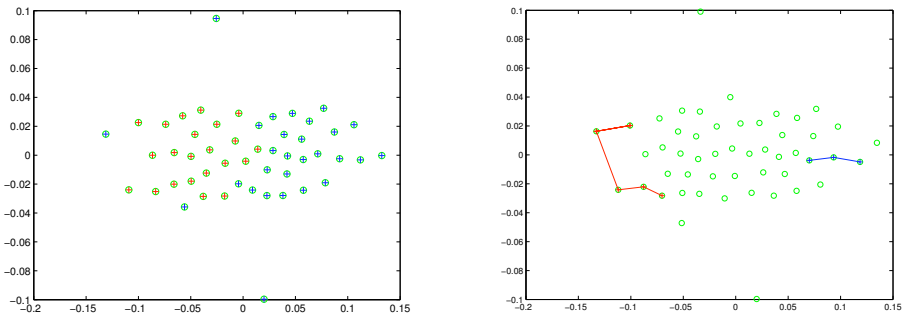
Figure 4.4: For a usable representation the player positions are transformed into the same canonical coordinate system, which captures the essentials of both pictures: despite the shift in viewangles both situations show an opponent standing in front of the player.

human players. The next Sections will provide details concerning the states and actions used in our implementation.

4.2.1 Canonical state representations

Using 3D player coordinates as a feature for behavior learning does not allow for recognizing behaviors that are independent of their environment. This is simply because the same underlying behavior might generate different trajectories in the 3D space. Therefore, for behaviors that are not dependent on the environment, an appropriate, invariant representation is needed.

In [Sukthankar and Sycara, 2005], an approach for classification of behaviors in the game UNREAL TOURNAMENT® was presented. In order to achieve a rotationally and translationally invariant representation of player positions, the player positions were centered around their overall mean and rotated according to the principal axis of the positions within a certain time window. In our work, we make use of this idea and apply *Principal Component Analysis* (PCA) to sequences of player and opponent positions, similar to an approach we recently applied to classifying



(a) (b)

Figure 4.5: Tactical waypoint maps: (4.5(a)) the red marked nodes are positions usually occupied by the observed player, while the blue marked nodes are usually occupied by an enemy player. (4.5(b)) The red path shows a path of the observed player, the blue the path of an opponent.

team behavior in sports computer games [Thurau et al., 2006].

Given a set of observed player positions $P = [\mathbf{p}_1^p, \mathbf{p}_2^p, \ldots, \mathbf{p}_n^p]$ and opponent positions $P_o = [\mathbf{p}_1^o, \mathbf{p}_2^o, \ldots, \mathbf{p}_n^o], \mathbf{p}_i^o$, where $\mathbf{p}_i^o = [p_x, p_y, p_z], \in \mathbb{R}^3$ and $\mathbf{p}_i^o = [o_x, o_y, o_z], \in \mathbb{R}^3$, we can derive a canonical representation by rotating the vectors according to the resulting eigenvectors of the matrices A_i with

$$A_i = \frac{1}{2fs} \sum_{k=i}^{k=i+fs} (\mathbf{p}_k^p - \mu_i)(\mathbf{p}_k^p - \mu_i)^T + (\mathbf{p}_k^o - \mu_i)(\mathbf{p}_k^o - \mu_i)^T \tag{4.9}$$

where fs denotes the framesize, and μ_i denotes the centroid of the positions of a frame i. The eigenvectors of the covariances A_i can now be used to represent the positions in a canonical coordinate system. This can be done by a rotation, according to

$$\mathbf{p}_i^c = B_i^T(\mathbf{p}_i - \mu_i) \tag{4.10}$$

where B_i are the eigenvectors of A_i, and $\mathbf{p}_i^c$ denotes the resulting canonical representation of the original state vector $\mathbf{p}_i$ at timestep i. Transforming all training vectors results in a map independent representation of player and opponent state vectors, with the principal axis usually aligned along the displacement between the player and opponent positions. However, we should make sure that either player or opponent positions are usually projected on the same "side" of the resulting representation (see also Figure 4.5), which can be achieved by an inverting the eigenvectors B_i, in case the positions are projected on the wrong side.

The framesize fs considerably influences the appearance of the canonical repre-

sentation: too high a value might lead to a mere rotation of the original coordinate system, too low a value might result in (unintended) dimensionality reduction by aligning position vectors along their displacement axis. We found that values of 10-15 for fs provide sufficient results, and still allow for fast computations of the projection needed in live games. This holds for a variety of game genres, see for instance Thurau et al. [2006]. Interestingly, the rotation in the canonical reference frame manages to capture the dynamics in the player-opponent interactions. For example, if the two players are not moving at all for fs gametics, they will be aligned along their displacement axis.

Representing the player positions in angular positions would have been an alternative to the canonical representation. However, considering the canonical representation allows us to again apply the concept of biological place cells. Therefore, we extract prototypical state vectors from the training samples. As usual, this can be achieved by applying a Neural Gas algorithm to the rotated representations. The resulting prototypical player and enemy positions $S = [\mathbf{s}_1, \ldots, \mathbf{s}_m]$ can be interpreted as *tactical place cells* or a *tactical waypoint map*. Understanding the resulting position prototypes as a tactical waypoint map implies that we can project player and enemy positions to specific waypoint nodes using a nearest neighbor criterion, resulting in sequences of prototypical positions $S_p = [\mathbf{s}_1^p, \ldots, \mathbf{s}_n^p]$, and $S_o = [\mathbf{s}_1^o, \ldots, \mathbf{s}_n^o]$ respectively.

4.2.2 Tactical movement primitives

As already introduced in Section 3.2.2, we characterized player actions using action primitives. Again, following [Fod et al., 2002], these are extracted from the observation data using k-means clustering. For the experiments described later on, we decided to use a simplified selection of actions and describe an action $\mathbf{a}_i$ as:

$$\mathbf{a} = \begin{bmatrix} \text{yaw angle} \\ \text{forward velocity} \\ \text{sideward velocity} \end{bmatrix}$$

where the player's viewangle is represented by yaw angle $\in [-180°, 180°]$, the player's velocity is represented by $v_{\text{forward}} = \Delta s_{\text{forward}}/\Delta t, \in [-400, 400]$, and $v_{\text{sideward}} = \Delta s_{\text{side}}/\Delta t, \in [-400, 400]$. Clustering of actions results in a set of action primitives, similar to the approach outlined in Section 3.2. For the considered behavior, we experienced sufficient results, and smooth artificial player movements, when using approximately $100 - 140$ action primitives. To interpret the movement primitives in conjunction with the tactical waypoint map, we have to rotate them according to the rotation already applied to the state vectors. Thereby, actions, too are invariant to different orientations of the recorded players.

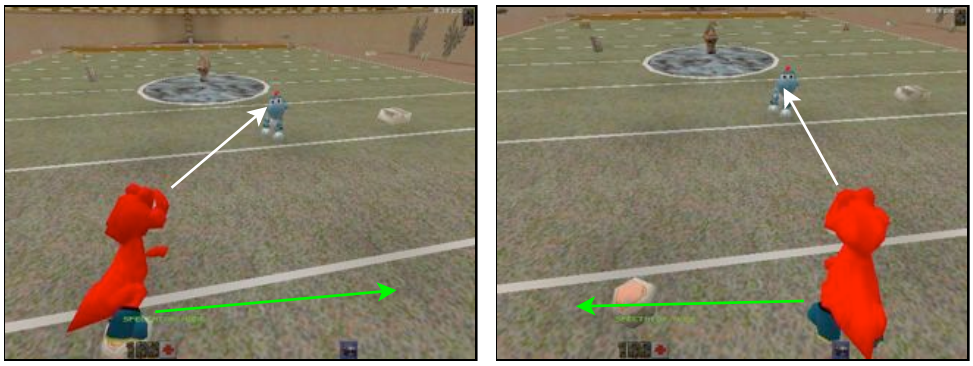


Figure 4.6: While aiming the artificial player moves from left to right.

4.2.3 Results

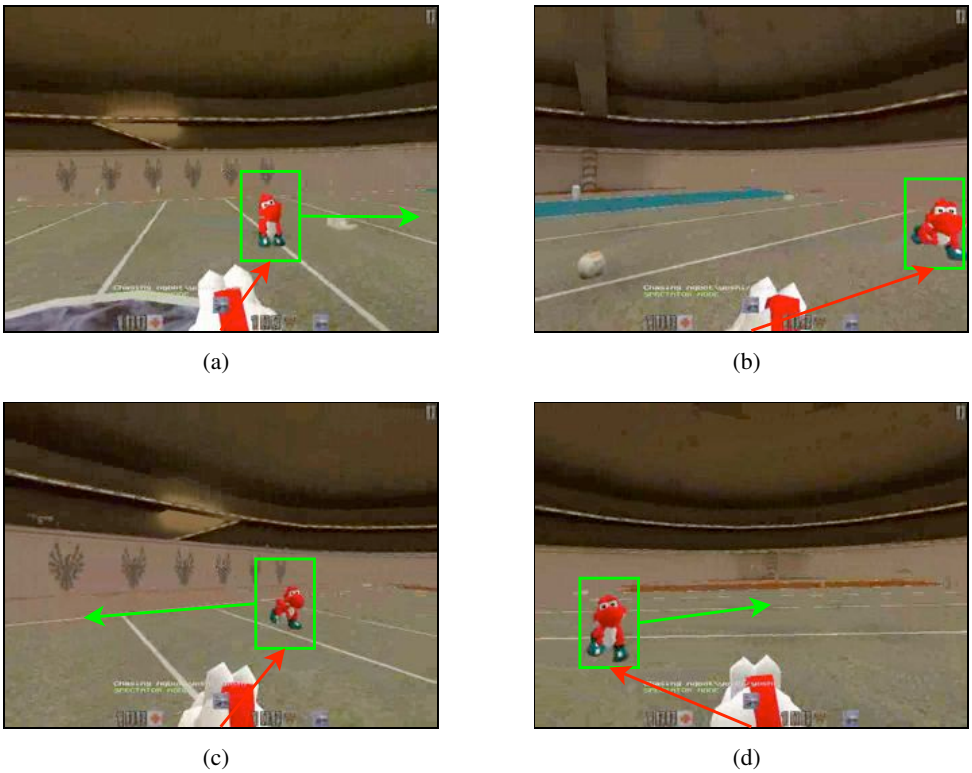
Given these preconditions, the final probabilities for action $\mathbf{a}_i$ in $P(\mathbf{a}_i|\mathbf{s}_t^o, \mathbf{s}_t^o, \mathbf{a}_t)$ (see also equation 4.8) can be estimated in a straight forward manner by analyzing recorded demo files of human players. In a live game, at every time step t, based on the player and opponent positions mapped on the place cell structure $\mathbf{s}_t^p$,$\mathbf{s}_t^o$, and the last selected action $\mathbf{a}_{t+1}$, we have estimates for selecting the next action a_{t+1} (again, we could also select the next action based on a roulette wheel selection):

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}} \frac{P(\mathbf{a}_k|\mathbf{s}_t^p, \mathbf{s}_t^o)P(\mathbf{a}_k|\mathbf{a}_t)}{P(\mathbf{a}_k)} \tag{4.11}$$

For verification of the given approach, we conducted one larger experiment. For this, we used training samples from a real 1 vs. 1 match of two human players. The map consisted of only one room where the players could pick up one weapon. Obviously, the recorded behaviors are more focused on tactical movements than on, for example, strategic item pickups. Having only one available item does not allow for sophisticated strategies.

From observing the players' behavior, we could spot a few dominant movements. Mostly, both players were strafing around each other, strafing from side to side (see also Figure 4.6), while still trying to focus their opponent as accurately as possible. Both tried to keep the opponent at a good shooting distance. This included evasive movements once the opponent decided to come closer. We expected similar behavior from the game agent. For the experiments we used a total of 200 action primitives, and 50 state prototypes, and approx. 5000 training sample vectors. Due to the simple, flat nature of the chosen map, we decided to reduce 3D state position vectors to 2D ($\mathbf{s} = [x, y]$)).

As already mentioned, we want to evaluate behaviors with respect to their intention. It is therefore important to directly inspect the resulting behaviors, with respect to the *goal* of the player, i.e. do the synthesized behaviors result in appro-



(a) (b) (c) (d)

Figure 4.7: The artificial player (the red character in the background) successfully manages to keep the opponent (the white character in the foreground) at a safe distance. When the opponent is chasing him, he manages to evade the attack and at the same time moves around him, back to a safe distance (pictures were taken from a recorded test run).

priate evasion, or circle-strafe behaviors. This is, in the case of tactical movements, only possible by inspecting the agent's behavior in 1 vs 1 situations beside a human player. As already mentionend, the life-like synthesis of motion is another important aspect of movement imitation. Consequently, we relied on visual inspection of the learned behaviors.

The artificial player, driven by the Bayesian model, managed to reproduce most of the observed behaviors. We tested the artificial agent on a different map, which provided more open spaces, thus we could avoid wall collisions. Generally, the agent showed the same strafing and circle-strafing behaviors as observed in the training samples. When a human test player was approaching the agent, it moved to the side and still managed to keep the opponent in his view. The agent tracked the opponent throughout the whole game but it always stayed at a distance which appeared to be similar to the one the human teacher preferred. Besides, synthesis of movements using a weighted roulette wheel selection over the movement primitives

made the agent's motion look very similar to the movements of the original player. Figure 4.7 shows one exemplary generated behaviors. Besides reproducing the tactical behaviors correctly, the agent's motion created a life-like agent impression.

4.3 Summary

Tactics are found to reside in between reactive and strategic behavior. In QUAKE II®, players have to decide how to use the different weapon types appropriately, and when to use each weapon. Although weapon handling could be considered a reactive behavior, it has certain tactical components, since weapons behave differently and as such require sophisticated skills. Further, bringing oneself into a good position towards an enemy player, or special maneuvers like circle strafing are considered tactical behaviors.

First, we outlined a Mixture of Experts based approach on weapon handling. Here, a Mixture of Experts architecture was used to learn and synthesize weapon specific aiming. A simple hierarchical combination of MLPs was able to handle differently behaving weapons in one architecture. Synthesizing and testing of this approach showed good aiming skills in the artificial player, also when using different weapon types. The weapon selection was realized using a simple Time-delay neural network, the output of these were used as additional input variables for the Mixture of Experts.

Besides weapon handling, appropriate movement in player-opponent situations is very important. In humans, a number of behaviors can be observed. For instance, circle strafes i.e. a player circles around an opponent while aiming. In a Bayesian action selection approach, we learned tactical movement behaviors in artificial agents. For this, a rotationally and translationally invariant data representation was required. Based on this, finally, actions could been sequenced in live games, to produce tactical moves; circle strafes, and dodging behaviors.

In the context of tactics, a lot of different behaviors can be observed in humans. As this Chapter showed, it is very unlikely that a single approach could be able to capture all of these behaviors. Moreover, we could think of a couple of tactical behaviors that would possibly require even more specialized approaches. Consider for instance tactical movements taking advantage of the environment, for example, hiding behind a wall or obstacles. These behaviors would first of all demand a representation capable of grasping possible hiding places, in addition, appropriate acting towards a possible hiding place is also important (where exactly to hide?). The gaming industry did so far approach these problems by top-down approaches where the agent is told what to do for a specific situation, see for example [Paull and Darken, 2004].

Chapter 5

Learning strategic behaviors

So far, we concentrated on behavior that only had a short-term relevance. For reactive behaviors, the outcome of an action is immediately visible. In contrast, *strategic behaviors* are targeted at long-term goal achievement. Therefore, when learning strategic behavior, we have to rely on approaches especially suited for recognizing action patterns that lead to *goal-states*. In addition, the extraction of possible goal-states is very important, since they are usually not known in advance.

Based on a certain *state* a human decides about the next goal-state to attain. Since we intend to synthesize actions within each behavioral layer independently of each other we also need appropriate movement imitation. Consequently, for learning strategies from human observation data, we need to (a) derive an appropriate representation for states and goal-states, (b) establish a mapping between states and goal-states (which goal to attain for a given situation?), and (c) synthesize adequate motion to generate a goal reaching behavior.

This Chapter is based on work published in [Thurau et al., 2004, 2005a,b].

5.1 Potential fields for modeling strategic behaviors

Fortunately, regarding the extraction of goal-states, we can rely on a few known facts for our game scenario: obviously, the main goal would be to win the match. Since we can safely assume that a player pursues this goal, we can concentrate on various subgoals. As already mentioned in Chapter 2, although subgoals differ among players, most are centered around important items or important areas which a player tries to reach. These decisions usually depend on the current *game-state* as well as on the individual player. Each player usually has his own opinion about the value of items or areas. Following the strategies considered in Section 2.4.1, learning a strategy is equivalent to learning goal-directed movement in 3D dependent on the current internal player statistics. In the following, we denote the *game-state* or *internal state* of a player at timestep i by a vector $\mathbf{s}_i \in \mathbb{R}^n$, whereas the position of

the player is denoted by a vector $\mathbf{p}_i \in \mathbb{R}^3$. For this first part on learning strategies, it is important to note that the position $\mathbf{p}_i$ of the player is not a component of the game-state vector $\mathbf{s}_j$. However, we will see that the actions of a player are also determined by his position.

Artificial potential fields [Arkin, 1998] provide an computationally inexpensive and well examined approach for navigational, goal-oriented movement control, and allow for an overlay of concurrent goals. Therefore, we consider them for learning goal-oriented behavior in game agents[1]. An artificial potential field can be described by a function $E : \mathbb{R}^m \rightarrow \mathbb{R}$, assigning every point in a m-dimensional space a scalar value f. These values will be referred to as *potential*. A potential field is characterized by an originating vector μ, and its corresponding field strength $\mathcal{F}$. A potential field's strength decreases with the distance to the field's origin. To calculate the potential $f_{\mathbf{x}}$ at an arbitrary point in space $\mathbf{x}$, we can use a simple inverse squared distance metric, thus:

$$f_{\mathbf{x}} = \frac{\mathcal{F}}{(\mu - \mathbf{x})^2} \tag{5.1}$$

When used for movement control, an artificial potential field can either attract or repulse an agent. The potential field's force $\mathcal{F}$, and the distance $(\mu - \mathbf{x})^2$ determine the degree of attraction or repulsion. For an agent, guided movement by potential fields is equivalent to following the steepest gradient of potentials.

According to Section 2.1, the underlying model for interpreting behavior as a function approximation task assumes a strong coupling of a player's decision $\mathbf{a}_t$ at timestep t and the world state $\mathbf{s}_t$ (again, we neglect environmental influences and concentrate on the latest state vector):

$$\mathbf{a}_{t+1} = g(\mathbf{s}_t) \tag{5.2}$$

where g denotes the behavioral function. Considering potential fields for agent navigation, the action $\mathbf{a}_t$ is the result of following the strongest local potential field force. Consequently, we want to learn the mapping $\mathbf{s}_t \rightarrow \mathcal{F}_i$ between world states and corresponding potential field forces, effectively resulting in an idealized function

$$\mathbf{a}_{t+1} = g(\mathcal{F}_i(\mathbf{s}_t)) \tag{5.3}$$

However, for proper guidance in complex worlds, usually more than one active potential field is required, thus

[1] Although we prefer biologically and psychologically motivated approaches in this thesis, we decided on artificial potential fields which are not, to our knowledge, inspired by biological or psychological findings. Still, the idea of interpreting certain areas or items as attractive, and thereby associating them with an attraction force, seems to be an intuitive approach.

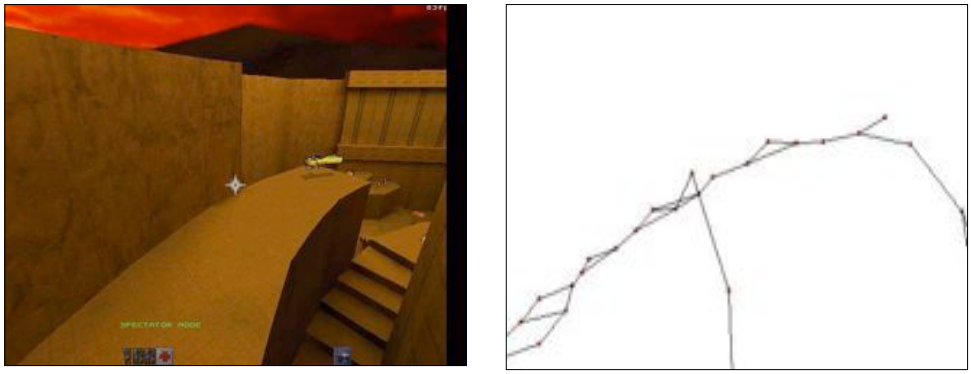


Figure 5.1: A screenshot of a small part of the map and the corresponding waypoint map

$$\mathbf{a}_{t+1} = f(\mathcal{F}_1(\mathbf{s}_t), \ldots, \mathcal{F}_n(\mathbf{s}_t)) \tag{5.4}$$

Thereby, field forces are changed depending on the current game's state, and thus guide an agent. These mapping should be derived from the observation data of a human player.

However, first of all the potential fields require a *medium* through which they are transmitted. Although we could have relied on the 3D game-world as a medium, a place cell inspired representation showed numerous advantages.

5.1.1 Topological gameworld representations

Following ideas presented in Section 3.2.1, for a representation of the virtual world we are learning a place cell like structure using a Neural Gas algorithm. The training data used for Neural Gas learning consists of all positions $O = [\mathbf{o}_1, \ldots, \mathbf{o}_n], \mathbf{o}_i \in \mathbb{R}^3, \mathbf{o}_i = \{x, y, z\}$ a human player held during various plan executions. Applying a Neural Gas algorithm to the player positions results in a set of position prototypes $P = [\mathbf{p}_1, \ldots, \mathbf{p}_m]$, i.e. places a player visited frequently.

In contrast to our regular application of the Neural Gas algorithm, this case also requires to interconnect the position prototypes. The reason for this is simple: we plan to use the topological structure as a medium for transmitting potential field forces. This requires edges over which the forces will be transmitted.

Compared to directly moving in the 3D game-world, moving on a topological representation has certain advantages, especially in conjunction with artificial potential fields. Consider a potential field placed at an arbitrary 3D position in the world of QUAKE II®. Estimating the potential for nearby positions could only be done sufficiently, if we obey the map's structure i.e. we do not let forces propagate through walls or similar obstacles. Using a learned graph representation for navigation should allow for collision free paths, and it should provide a suitable medium

1. Initialize a connectivity Matrix $C_{ij} \in \{0,1\}$ with $i,j = 1 \dots M$. Set all C_{ij} to zero. Initialize the counter t to 1.
2. Select an input signal $\mathbf{v}_t$ from $V = [\mathbf{v}_1, \dots, \mathbf{v}_m], \mathbf{v}_i \in \mathbb{R}^n$
3. Determine and store the best matching prototype $\mathbf{w}_i^t$ corresponding to $\mathbf{v}_t$:
$$\mathbf{w}_i^t = \underset{i}{\operatorname{argmin}} \parallel \mathbf{v}_t - \mathbf{w}_i \parallel$$
4. if $t > 1$, and $C_{\mathbf{w}_i^{t-1},\mathbf{w}_i^t} \neq 1$, set $C_{\mathbf{w}_i^{t-1},\mathbf{w}_i^t} = 1$
5. Increment t: $t = t + 1$
6. If $t \leq m$, continue with step 2

Figure 5.2: Neural Gas sequence based edge learning

for guidance by potential fields. However, the edges connecting the nodes are not computed using the classical Hebbian topology learning, which is originally used in the Neural Gas algorithm.

Drawing edges based on concurrent activity, as introduced in the Hebbian learning, would not result in an accurate representation of possible movement paths. It could easily lead to connections between nodes that are, for example, separated by a wall. Therefore, we only connect two nodes if there was an observable player movement from one node to the other. The following approach is briefly summarized in Figure 5.2. For the Figure, we decided to keep to the already used variables for the Neural Gas learning in Section 3.1.6, since it is effectively an alternative approach to Neural Gas edge learning, suited for sequences:

Given a number of position prototypes $P = [\mathbf{p}_1, \dots, \mathbf{p}_m]$, and given a sequence of player positions $O = [\mathbf{o}_1, \dots, \mathbf{o}_n], \mathbf{o}_i \in \mathbb{R}^3$, we can assign each player position to the prototype $\mathbf{p}_o$ with the minimum Euclidean distance $E_o = \parallel \mathbf{o}_i - \mathbf{p}_{o_i} \parallel$. Thus, we can express player movements by a sequence of position prototypes $P_O = [\mathbf{p}_{\mathbf{o}_1}, \dots, \mathbf{p}_{\mathbf{o}_n}]$.

Considering the topological structure, the edges connecting the nodes are stored in a $m \times m$ matrix $C_{i,j} \in \mathbb{N}^+$ where m denotes the number of position prototypes. If $C_{i,j} = 0$ unit i is not connected to unit j, if $C_{i,j} = 1$ unit i is connected to unit j. Thus, for connecting nodes based on subsequent movements, we set $C_{i,j} = 1$ if there was an observed movement between the two prototypes $\mathbf{p}_i$ and $\mathbf{p}_j$. . Figure 5.1 shows a resulting waypoint map.

The node distances, measured via the topological structure, can be computed in advance and are stored in a $m \times m$ matrix D, where $D_{i,j} \in \mathbb{R}^+$ denotes the distance from node i to node j.

Interconnecting the prototypes based on the observed movements of a human

player results in an accurate representation of the game-world's topology and of the possible movement paths. For example, prototypes that are next to each other but are separated by a wall are not interconnected, whereas prototypes that are not next to each other in the 3D space, but are connected via a teleporter (a quite common entity in QUAKE II®), would also be connected in the graph structure, because of the observed subsequent player positions.

For an artificial agent, a movement from one node to another is only possible if a connection between the two exists, and can be translated to movements in 3D. Obviously, directly moving from node to node bears certain problems that will be mentioned later on.

5.1.2 Estimating potential field parameters

As already mentioned, for strategic behavior, the world state is best described by the internal state of a player. In our experiments, we used the player's inventory[2] as well as his current armor and health value as state vector variables. The internal state is assumed to contain the most relevant information for strategic decisions. For a recording of strategic behavior, the sequence of internal player states is given by $Z = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$ and $\mathbf{z}_i \in \mathbb{R}^{10}, \mathbf{z} = \{\text{health}, \text{armor}, \text{weapon1}, \ldots, \text{weapon8}\}$.

Situations that are similar to others will most likely lead to similar strategic decisions. Therefore, we are grouping similar observed player states using the k-means clustering algorithm. This results in a number of state vector prototypes $S = [\mathbf{s}_1, \ldots, \mathbf{s}_m]$, representing more common game situations. In a second step, each observation of a world state can be assigned to a state prototype using a nearest neighbor criterion, resulting in a sequence of state prototypes $S_Z = [\mathbf{s}_{\mathbf{z}_1}, \ldots, \mathbf{s}_{\mathbf{z}_n}]$. Corresponding to these, we already derived a sequence of player position prototypes $P_O = [\mathbf{p}_{\mathbf{o}_1}, \ldots, \mathbf{p}_{\mathbf{o}_n}]$.

We assume that a player changes his strategic behavior and his desired goal-state, as soon as his internal state corresponds to another state prototype $\mathbf{s}_i$. Consequently, each movement behavior begins at a specific time-step t if the sequence S_Z is entering a new state space cluster ($\mathbf{s}_{\mathbf{z}_t} \neq \mathbf{s}_{\mathbf{z}_{t-1}}$), and ends at a time-step e if a state change occurs ($\mathbf{s}_{\mathbf{z}_{e+1}} \neq \mathbf{s}_{\mathbf{z}_e}$). Accordingly, the movement pattern is described by the position prototype sequence, going from time-step t to e: $[\mathbf{p}_{\mathbf{o}_t}, \ldots, \mathbf{p}_{\mathbf{o}_e}]$. Each movement pattern can be interpreted as a state dependent, goal-oriented movement behavior.

Most state changes occur when items are picked up. Therefore, almost all movement patterns end with an item pickup, implicitly defining item positions as the *goal states* to attain (or defining the internal state of a player where he already possesses the desired item, as a goal-state). A single movement pattern D can be denoted as a sequence of nodes in the topological representation:

[2] The set of items a player picked up in the course of a game

$$D = [\mathbf{p}_{\mathbf{o}_i}, \ldots, \mathbf{p}_{\mathbf{o}_j}] \tag{5.5}$$

where $\mathbf{p}_i$ denote a single node, and $j - n$ denotes the length of the movement pattern D. Based on these assigned movement patterns, we have to shape potential field forces, in order to mimic the observed movement behaviors.

Potential fields originating in a node in the waypoint map produce a certain potential on all nodes, which decreases with the distance to the potential field source. Since any node is a possible potential field source, the potential f_j at node j is denoted as:

$$f_j = \sum_{i=1}^{n} \frac{\mathcal{F}_i}{d_{i,j}^2} \tag{5.6}$$

where $d_{i,j}$ denotes the distance from node i to node j and $\mathcal{F}_i$ denotes the potential field strength at node i.

For every given state space cluster $\mathbf{s}_i$, we have to find a potential field force configuration $\mathcal{F}_{\mathbf{s}_i} = [\mathcal{F}_1, \ldots, \mathcal{F}_m]$, where m denotes the number of place cells. The configurations should contribute to the observed movement patterns.

Since the movement in a potential field does always follow the steepest gradient, there has to be an increase in the potentials f_i found on every node along a specific movement pattern. Consequently, by assigning ascending discrete values to each node in the sequence $D_{\mathbf{s}_i}$ corresponding to a state prototype $\mathbf{s}_i$, we construct a set of possible potentials for every node along a certain movement pattern, thus:

$$[f_{\mathbf{p}_{D_1}} = 1, \ldots, f_{\mathbf{p}_{D_n}} = n] \tag{5.7}$$

where $f_{\mathbf{p}_{D_i}}$ denotes the assigned potential for the i-th waypoint map node taken from the Sequence D. The final node, and possible goal-state, $\mathbf{p}_{D_n}$ is always assigned the highest value. This is done with all observed movement paths belonging to a certain state space cluster $\mathbf{s}_i$, giving more weight to more common plans by summing up all observed potential distributions. Thus, for each node $\mathbf{p}_u$ associated with a state prototype $\mathbf{s}_i$, we can estimate a potential $f_{u,\text{desired}}$. Arbitrary movements which do not approach the final goal node $\mathbf{p}_{k,n}$ are always assigned lower potentials than their successor nodes. Thereby, random (non goal-oriented) moves are filtered out to a certain extend, as can be seen in Figure 5.4.

Given the potentials $f_{i,\text{desired}}$, we can interpret the behavior learning problem as a regression problem. By knowledge of a few desired potentials given a certain situation, we can estimate the potentials for all other nodes in the topological representation. This will be done by appropriate selection of potential field forces $\mathcal{F}_{j,\mathbf{s}_k}$ corresponding to state prototypes $\mathbf{s}_k$.

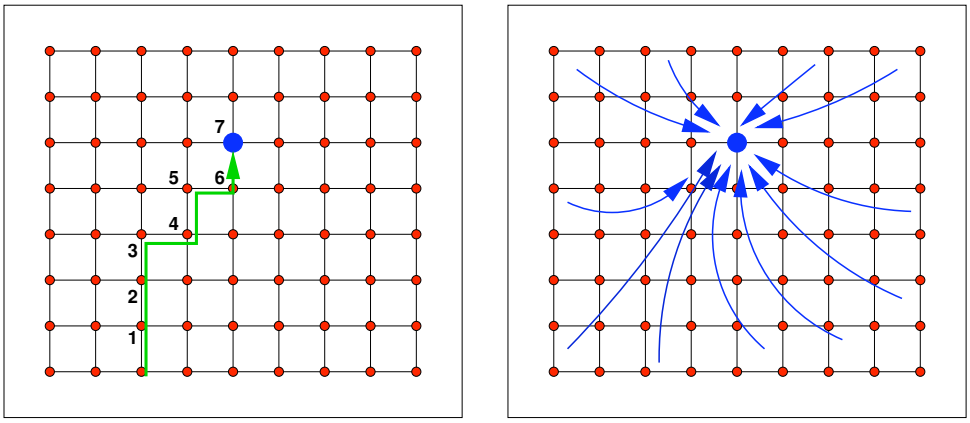


Figure 5.3: The left figure shows the observed movement path and the assigned discrete potentials, the right the learned potential field forces

The computation of the field strengths $\mathcal{F}_j$ can be interpreted as an optimization problem, where the following error function has to be minimized:

$$E = \frac{1}{2} \sum_{i=1}^{n} (f_{i,\text{desired}} - f_i)^2 \tag{5.8}$$

where $f_{i,desired}$ denotes the sum over all assigned potentials of a node belonging to a state prototype $\mathbf{s}_k$. A gradient descent leads to the update rule for the potential field force $\mathcal{F}_i$ of a node i:

$$\mathcal{F}_i = \mathcal{F}_i - \alpha \sum_{k=1}^{n} \left(f_{i,desired} - \sum_{l=1}^{n} \frac{\mathcal{F}_l}{{d_{k,l}}^2} \right) \frac{1}{{d_{k,i}}^2} \tag{5.9}$$

where α denotes a learning rate which is decreased during the training process. In each step, the forces of one nodes are adapted, then, the next node $\mathcal{F}_{i+1}$ is modified, until the learning rate $\alpha \leq 0$. It showed to be usefull to only allow positive potential forces $\mathcal{F}_i$, and to restrict the number of originating potential fields to the actual number of observed $f_{i,desired}$.

Figure 5.3 illustrates the relation between the observed movement path and resulting potential field forces, given a simple goal oriented movement pattern.

On the first sight, this methods seems a bit ineffective, since a shortest path algorithm, as for instance elaborated by Dijkstra [1959], might provide the same results. However, the problem we are facing is different from finding the shortest path to a specific map location. First of all, the goal states are not known in advance. Since we aim at behavior learning from observation data, and since goal-states tend to be different among players, we can not define them by expert knowledge. Moreover, learning a strategic movement behavior does not imply that we are searching

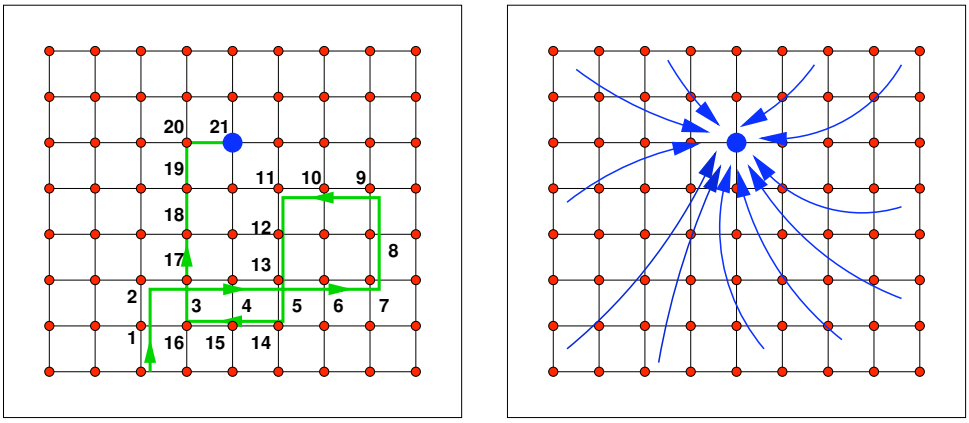


Figure 5.4: An observed movement pattern with some arbitrary, non goal oriented movement, can still produce goal oriented potential fields.

for the shortest path. Instead, we try to learn the strategically most appealing path. This could also mean to guide the agent along a longer path, if this is an observed behavior (for instance, a player might prefer a path in darker areas), Figure 5.5 illustrates how the approach can learn a more player specific path. Moreover, by using potential fields, we can also provide guidance for seldom visited nodes.

However, the usage of potential fields comes along with some problems. On the one hand, we might encounter local maxima, on the other hand, we have a relatively weak guidance when the distance to originating potential field nodes increases. The next section deals with both problems.

5.1.3 Avoiding the past potential fields

Avoiding the past potential fields [Balch and Arkin, 1993] serve as a local spatial memory, to avoid already visited nodes, in order to not get stuck at local maxima. At fixed time intervals a repulsing potential force is deposited at the current node. To reduce interference with goal based potential fields, the avoid-past potential field forces are reduced over time, thus, effectively resulting in a pheromone trail known from ant-algorithms [Bonabeau et al., 1999]. In addition, the area of effect is reduced compared to the goal based potential field forces.

These pheromone trails have two purposes. First, local maxima can be avoided by pushing the agent to nearby nodes, hopefully back to the field of influence of potential fields offering a better guidance. Second, areas which provide poor guidance because of the distance to goal oriented potential fields can be escaped by reinforcing a chosen direction.

The overall potential distribution on waypoint map node m_j at timestep t is now computed as follows:

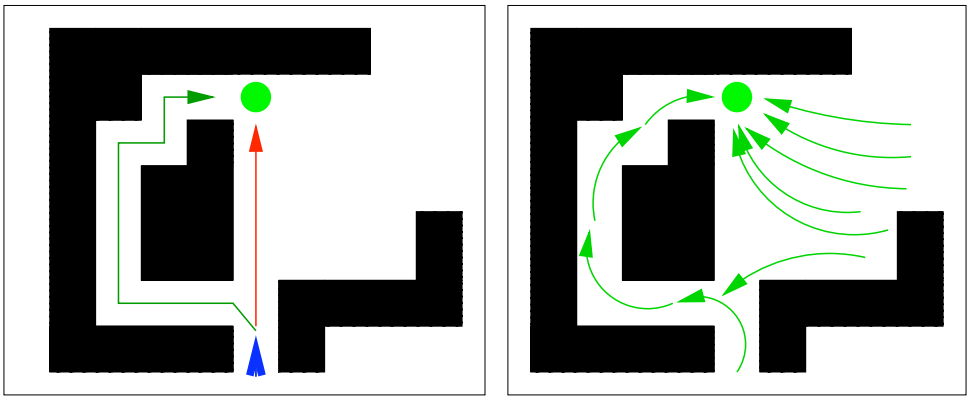


Figure 5.5: The left figure shows an observed movement path, where the green arrows denotes the actual observation, the red one the shortest path to the goal. The right figure shows the learned potential field forces, which would guide the agent along the longer path, when approaching the goal from the initially observed direction.

$$f_j(t) = \sum_{i=1}^{n} \frac{\mathcal{F}_i}{{d_{i,j}}^2} - \sum_{i=1}^{n} \frac{P_i(t)}{{d_{i,j}}^3} \tag{5.10}$$

where $P_i(t)$ denotes the Pheromone concentration at node i, at timestep t, and with $P_i(t+1) = P_i(t) - \varrho P_{evap}$, where ϱ is a diffusion parameter and P_{evap} is the fixed evaporation parameter for an occupied node.

Both parameters P_{evap} and ϱ have to be chosen manually to provide the expected effects and to not interfere too much with the goal oriented potential field forces. Generally, we decided on rather low values to leave most of the guidance with the potential fields.

5.1.4 Results

The agent control mechanism is straight forward. With knowledge about the potential field forces, the potentials for all nodes, connected to the currently occupied node, can be computed. From the list of possible successor nodes, the one with the highest potential is selected, thereby, the agent follows the steepest gradient of potentials. Environmental influences or item pickups change the agents state vector $\mathbf{s}_i$ and result in a change to the overall potential field forces, thereby leading the way to other nodes.

The training data which we used consisted of the observation of 20 different tasks on one map, mainly item pickup sequences. In every demo, we recorded a human player steering a game agent. The human was told to pickup certain items in a fixed sequence. Thereby, various combinations of item pickup-sequences were tested. In the most simple setup, the human picked up only one item, whereas

Task	human player time in sec.	starting p. 1 time in sec.	starting p. 2 time in sec.	starting p. 3 time in sec.	starting p. 4 time in sec.
Set 1	20.2	13.89	17.60	16.51	8.31
Set 2	38.4	33.30	25.71	24.50	31.42
Set 3	20.6	8.42	9.00	11.43	3.02
Set 4	17.8	22.49	19.30	23.62	14.89
Set 5	14.2	11.64	9.21	13.61	14.01
Set 6	20.0	16.82	16.52	22.70	16.42
Set 7	28.7	33.30	35.03	36.40	31.54
Set 8	19.2	28.01	–	–	–
Set 9	44.6	41.50	41.24	41.80	46.33
Set 10	24.9	20.02	14.71	24.82	15.71
Set 15	28.0	39.19	46.48	41.20	36.21
Set 20	32.8	40.2	55.10	32.41	50.91

Table 5.1: The bot had to complete every learned task when starting from four different locations. To give an indicator about the agent's performance, we measured the time in sec. it took him to reach all goals in the predefined order, thereby reproducing the human movement behavior.

the more complex demos showed 4-5 subsequent items to attain. For each task a separate bot was trained.

To estimate the game-bot performance, we evaluated four different starting locations from which the game-agent had to find its way according to the human behavior in the training samples. With respect to the intention of strategic behavior which obviously is a goal-reaching behavior, we evaluated the success rate in goal attainment. Letting the game-agent start from different locations tested the program-level imitation, i.e. how well strategic goals of the behavior are imitated for novel situations. Here it is first of all important to reach the desired items from each starting location. Notably, the human observation data usually only contained one starting location. Thus, the agent did produce behavior, with respect to the movement path, that was not directly observed in the training data (notably, the waypoint-map was learned by taking into account all player recordings). Performance can be measured over the number of goals the bot reached, and the time it took him to do so. Table 5.1 shows some of the results. The game-bot's movement speed is comparable to a human player, since he usually took the shortest path to the goals, and thereby reproduced the observed behavior.

Generally, our approach showed very good program-level imitation, failing to reach the goals in the predefined order only once. Regarding task number 8, the bot got stuck in three out of four different runs. The main problem in that experiment was that the first goal was located very close to the latest goal, thus, the agent

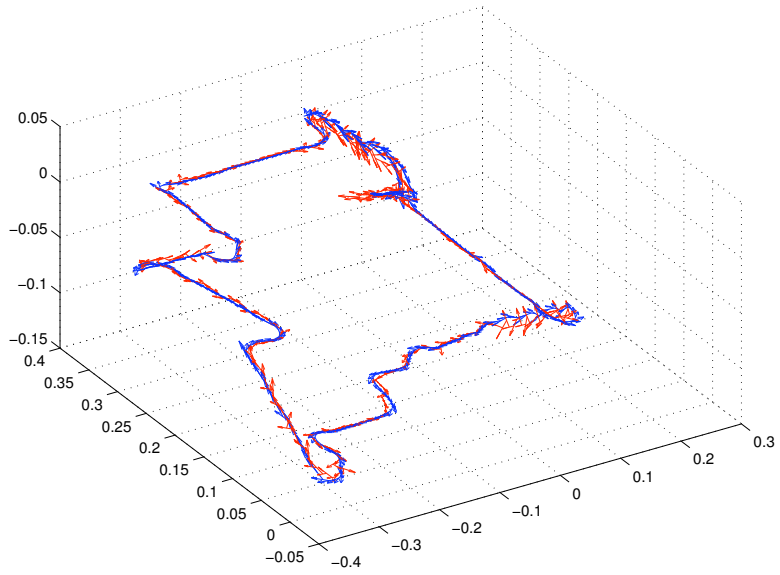


Figure 5.6: Visualization of a human and a game agent's movement path in x, y, z. The blue arrows indicate the game bot's current positions and its velocity, the red arrows show the observed human player's position and velocity, which was used for learning the potential field forces. The bot started from the same positions as the human player and managed to fully imitate the human movement behavior.

accidentally picked up the last item at the beginning of its run which lead to an internal state which was associated with only very weak potential field forces, and therefore resulted in too weak guidance.

Visual comparison of the human and artificial player's movement paths gives further insights how the potential fields are able to learn human movement patterns. Figure 5.6 shows the observed game agent's movement path in comparison to the human's movement path. Figure 5.7 illustrates a complete goal directed movement sequence.

Since strategic movements do not depend on the execution of a single, isolated movement pattern but on a combination of adequate movement paths, we decided to test the described approach with more complex behaviors. Here, we instructed a human player to play the game Quake II as if an enemy was present (again we simplified the situation in order to focus on the strategical aspects). This resulted in a set of training samples where the player collected various items in different orders, dependent on the initial starting position, and after that securing important areas by cycling different types of armor (a quite common behavior shown by many players).

This experimental setup, too, yielded good results, showing that the approach is also working reasonably well on a larger scale. The artificial agent convincingly

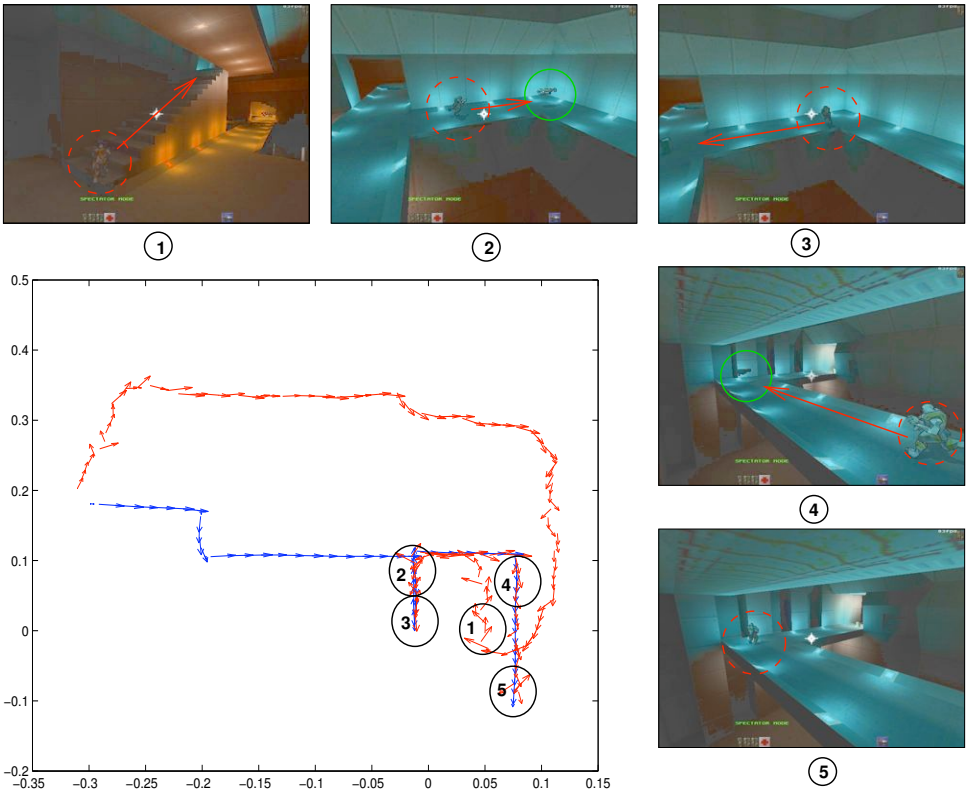


Figure 5.7: The bot started from a map location, other than the one used during the learning process. While the red arrows show all bot positions and its velocity, the blue arrows show the positions and velocity of the observed human player's movement path. Picture 1 shows, where the bot moves up a staircase and comes close to the observed movement pattern. In Picture 2, he is targeting the first item and picks it up at 3, reproducing the observed human behavior. A state change occurs, switching the potential field forces. The bot moves on to the next goal in 4 and finally reaches it at 5.

moved through the 3D game-world and managed to reproduce the observed behavior. He picked up several different types of items, before he mostly stayed at one area, usually located around an important armor item, and only moved away when driven by its pheromone trail or when there were sudden changes in its state vector (for example, a decrease in its armor value due to an attack by another player). Usually, the agent reached a state where he owned all weapons and filled up his armor value to the maximum. In comparison to the first experimental setup, no fixed movement behavior sequence was reproduced. Instead, more sophisticated behavior emerged by mixing various observed movement patterns.

5.2 Goal directed Bayesian imitation learning

So far, we introduced several computational approaches motivated by biological or psychological findings. All of these considered functional aspects that could be used to improve mechanisms of computational imitation learning. Notably, we did not yet motivate the general process of imitation learning by psychological findings.

In a recent contribution, Rao et al. [2004] introduced a *Bayesian model of imitation learning* for application in robotics. Based on experiments in developmental psychology, their probabilistic framework models four stages of imitative abilities that were observed in infant behavior. Thus, they realized human imitation learning capabilities by a Bayesian model. In the following, we apply this model for learning strategic behavior from human observation data. Here, we do not only aim at a sufficient program-level imitation, but also at realistic motion synthesis. Since a basic understanding of these stages helps grasping the approach discussed below, we summarize them briefly in Figure 5.8.

As already suggested in Section 2.1, in a probabilistic behavior learning approach, we can express a human's reaction $\mathbf{a}_{t+1}$ within a conditional probability, dependent on a number of state vectors $\mathbf{s}_i$, and environmental influences $\mathbf{e}_i$:

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_{t-1}, \ldots, \mathbf{s}_{t-n}, \mathbf{e}_t, \mathbf{e}_{t-1} \ldots \mathbf{e}_{t-n}) \tag{5.11}$$

Although we are considering strategic behavior, we for now focus on the latest observed states:

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{e}_t) \tag{5.12}$$

As already mentionend, for strategic behaviors, we also need to consider goal-states or subgoal-states. Interstingly, the Bayesian model of imitation learning proposed by Rao et al. [2004] also accounts for goals and subgoals. Within the Bayesian framework, the probability for the execution of an action $\mathbf{a}_i$ at time step t depends not only on the current state $\mathbf{s}_t$, but also on the next desired state (subgoal)

The concept and biological basis of imitation learning is known for quite some time. Interestingly, the ability for learning from observing others is already developed in infants. The development can be described in three subsequent steps.
By means of postnatal *body babbling* infants acquire a model of their body. They learn which muscular actions lead to what kind of limb configurations and thus acquire a vocabulary of useful *motor primitives*. From a computational point of view, this step can be interpreted as acquiring a forward model, connecting actions to states, and establishing an idea about consequences of actions.
This enables the *imitation of body movements* where infants map observed actions onto their own body. At the age of several weeks, for instance, they can mimic facial expressions they have never seen before, an example for this can be seen in the pictures (the pictures are taken from [Rao et al., 2004]).
In a third stage, infants start *imitating actions on physical objects* such as toys which are external to their body. By the time they are 1.5 years old, infants are experienced in interacting with other humans. Consequently, they can acquire models of agents with intentions. Forward models allow them to *infer the goals of an agent* even if they only observe unsuccessful demonstrations; inverse models are used to select motor commands that will achieve undemonstrated but inferred goals.

Figure 5.8: Stages of imitation learning in humans

$\mathbf{s}_{t+1}$, and the overall goal state $\mathbf{s}_g$. Thus, we can restate Equation 5.12 as follows (again, we neglect environmental influences $\mathbf{e}_t$):

$$\mathbf{a}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_{t+1}, \mathbf{s}_g) \tag{5.13}$$

According to [Rao et al., 2004], we assume that a forward model $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_i)$ that describes the consequences of actions in state space, is already acquired by means of *body babbling*. It is important to note that this model is also assumed to be independent of the goal state, since it is determined by the environment, thus $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_i, \mathbf{s}_g) = P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_i)$. Further, we assume a set of priors $P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_g)$ that are also already learned from, for instance, observations of a teacher. Using Bayes' theorem, Equation 5.12 can be restated:

$$\begin{aligned} \mathbf{a}_{t+1} &= \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_{t+1}, \mathbf{s}_g) \\ &= \underset{k}{\operatorname{argmax}}\, \frac{1}{C}\, P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_k)\; P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_g) \end{aligned} \tag{5.14}$$

where the normalization constant C results from marginalizing over all possible actions:

$$\begin{aligned} C &= P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{s}_g) \\ &= \sum_m P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_m) P(\mathbf{a}_m|\mathbf{s}_t, \mathbf{s}_g). \end{aligned} \tag{5.15}$$

In order to apply this model for selecting subsequent actions $\mathbf{a}_{t+1}$ for learning strategic behavior, we need to know the subgoal-state $\mathbf{s}_{t+1}$, and the current goal-state $\mathbf{s}_g$. If both goal states are known, the best matching action $\mathbf{a}_{t+1}$ can be selected.

For the task of learning a strategy for QUAKE II®, winning the game is the only true global goal. Again, we can assume a player to follow this goal anyway. Therefore, in the following we concentrate on elaborating a model for subsequent subgoal states $[\mathbf{s}_{t+1}, \ldots, \mathbf{s}_{t+n}]$. Thus, a suitable representation for states is needed that also fullfills the requirements for the forward model $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_i)$.

5.2.1 Game state representations

A state vector $\mathbf{s}_i$ should contain a problem specific description of a player's internal state, and the surrounding gameworld. With the goal of strategic behavior imitation in mind, we decided on the ever important player's position, and his inventory. In the experiments, the considered number of state variables varied, therefore, for

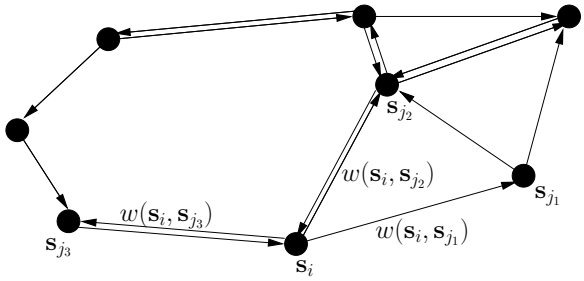


Figure 5.9: Didactic example of a state transition graph.

now a state vector is given by $\mathbf{s}_i \in \mathbb{R}^m$. As usual, in order to derive a discrete approximation of the state space, we cluster the state vectors recorded from the network traffic using a Neural Gas clustering.

To select the next subgoal $\mathbf{s}_{t+1}$, as the successor of subgoal $\mathbf{s}_t$, we also d on a probabilistic model. Thus, we select $\mathbf{s}_{t+1}$ according to:

$$\mathbf{s}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{s}_k|\mathbf{s}_t) \tag{5.16}$$

First, however, we need to provide the resulting discrete state space with a useful structure. This can be achieved by a state transition graph. In this graph, a directed edge between two state prototypes indicates a possible state transition. Moreover, edges are labeled with transition counts (see Figure 5.9). As the idea for the transition graph was inspired by the theory of *edge reinforced random walks* known from statistics [Diaconis, 1988], we use the term weights when referring to state transition counts. Formally, the transition graph is thus given as a triple $G = (V, E, w)$ where $V = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_M\}$ is a set of vertices, $E \subseteq V \times V$ is a set of directed edges and $w : E \rightarrow \mathbb{R}^+$ is a function assigning weights to the edges. Edges are drawn based on state transitions observed in the training data; i.e. we have

$$(\mathbf{s}_i, \mathbf{s}_j) \in E \iff \exists\, \mathbf{s}_t, \mathbf{s}_{t+1} : \mathbf{s}_t = \mathbf{s}_i \wedge \mathbf{s}_{t+1} = \mathbf{s}_j \tag{5.17}$$

Transition counts are recovered from traversal frequencies in the training data; the weight w of an edge $(\mathbf{s}_i, \mathbf{s}_j)$ corresponds to the number of times a human player was observed to move from state space prototype $\mathbf{s}_i$ to prototype $\mathbf{s}_j$.

Given this weighting scheme, suitable upcoming subgoals for Bayesian behavior synthesis can be determined either from a roulette wheel or a maximum a-posteriori selection over the state transition probabilities

$$\mathbf{s}_{t+1} = \underset{k}{\operatorname{argmax}}\, P(\mathbf{s}_k|\mathbf{s}_t) = \underset{k}{\operatorname{argmax}}\, \frac{w(\mathbf{s}_t, \mathbf{s}_k)}{\sum_i w(\mathbf{s}_t, \mathbf{s}_i)} \tag{5.18}$$

Again, imposing a discrete state space structure like this provides certain advantages over an unstructured collection of prototypes. Besides the already mentioned

advantages in Section 5.1.1, limiting the choice for a successor to those states that are connected to the current state considerably lowers the computation time but still allows for a reconstruction of all observed state sequences. Moreover, we stay very close to the observed human behavior by only considering recorded movement patterns, thus, certain state transitions become impossible.

5.2.2 Traversing state graphs

The state graph introduced in the last Section can only be traversed by executing actions $\mathbf{a}_t$. For instance, if the next desired state would correspond to a position in front of the player, a forward movement would result in the desired state transition. This implicit knowledge of action consequences should be available within the forward model $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_i)$.

Again, the set of available actions is given by *action primitives*, prototypical actions, which are extracted from recorded matches. For the experiments presented below, an action primitive is a 5 dimensional vector

$$\mathbf{a} = \begin{bmatrix} \text{yaw angle} \\ \text{pitch angle} \\ \text{forward velocity} \\ \text{sideward velocity} \\ \text{upward velocity} \end{bmatrix}$$

where the player's viewangle is represented by a yaw angle $\in [-180°, 180°]$, a pitch angle $\in [-90°, 90°]$, the player's velocity is represented by $v_{\text{forward}} = \Delta s_{\text{forward}}/\Delta t, \in [v_{\text{min}}, v_{\text{max}}]$, $v_{\text{side}} = \Delta s_{\text{side}}/\Delta t, \in [v_{\text{min}}, v_{\text{max}}]$, and $v_{\text{upward}} = \Delta s_{\text{upward}}/\Delta t \in [v_{\text{min}}, v_{\text{max}}]$, with $v_{\text{min}} = -400$ and $v_{\text{max}} = +400$. Prototypical action primitives result from applying k-means clustering to the given movements of human players. As already mentioned, a number of 100 to 120 action primitives is usually sufficient for synthesizing smooth looking motions.

Following the proposal of Rao et al. [2004], we apply the mechanism of body-babbling to estimate a forward model $P(\mathbf{s}_{t+1}|\mathbf{a}_t, \mathbf{s}_t)$. In the case of QUAKE II®, we do not have to rely on self-exploration of the agent, instead, we simply use the recordings of a human player's actions on states $\mathbf{s}_t$.

Following ideas discussed in Section 3.2.2, we introduced a conditional probability $P(\mathbf{a}_{t+1}|\mathbf{a}_t)$ for generating more life-like agent motions. Again, these can be learned from observing human players. Since the action vector $\mathbf{a}_t$ can be seen as part of the world state vector $\mathbf{s}_t$, we extend Equation 5.2 by integrating a dependency between actions $P(\mathbf{a}_{t+1}|\mathbf{a}_t)$. Assuming independence of $\mathbf{a}_t$, $\mathbf{s}_t$ and $\mathbf{s}_g$ (after all, physical limitations should not affect environmental conditions), this results in the following model for selecting an action $\mathbf{a}_{t+1}$ (again, we could also select an action based on a roulette wheel selection instead of maximum a-posteriori):

$$\begin{aligned}\mathbf{a}_{t+1} &= \underset{k}{\operatorname{argmax}}\, P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_{t+1}, \mathbf{s}_g) \\ &= \underset{k}{\operatorname{argmax}}\, \frac{1}{C}\, P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_k)\, \frac{P(\mathbf{a}_k|\mathbf{s}_t, \mathbf{s}_g) P(\mathbf{a}_k|\mathbf{a}_t)}{P(\mathbf{a}_k)}\end{aligned} \tag{5.19}$$

where the normalization constant C is now given by:

$$\begin{aligned}C &= P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{s}_g) \\ &= \sum_m P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_m) P(\mathbf{a}_m|\mathbf{s}_t, \mathbf{s}_g)\end{aligned} \tag{5.20}$$

5.2.3 Results

In order to test the Bayesian framework for imitating strategic behavior and motion behavior, we carried out a series of experiments. The basis of all experiments was of course formed by recordings of data generated by human players. The experimental goal was to reproduce all observed movement and strategic behaviors.

The first experiment basically tested the functionality of the approach. Several examples of a goal directed movement sequence were recorded; in each sequence, the player's motion ended at the same map position. Due to the simplicity of the task, the state vectors $\mathbf{s}_i$ that we considered only contained the observed player positions $[x, y, z] \in \mathbb{R}^3$. Since the agent traverses a discrete lattice of prototypical positions, the number of clustered state prototypes plays a vital role. Our experiments revealed that smaller maps require 50 to 100 prototypes to allow for collision free navigation. All in all, the movements were learned and reproduced successfully.

Our second experiment considered an extended maze problem; Figure 5.10 sketches its setting. The training data we considered here displayed several instances of a human player first picking up goal-item 1 (an armor) and then continuing to goal-item 2 (a better weapon). The state space dimensionality was extended to account for the player's inventory. The additional dimensions encode information about the internal armor value and the weapon currently hold.

Given state graphs computed under these conditions, we expect to observe state sequences which reflect the order of the item pickups. In order to reach a state with higher armor values, the agent has to obtain the armor item. Since in the demo data the armor pickup precedes the weapon pickup, states of higher armor value must lie on the way through state space that leads to the overall goal state. On the map, the agent therefore has to visit the location of the armor prior to continuing to the

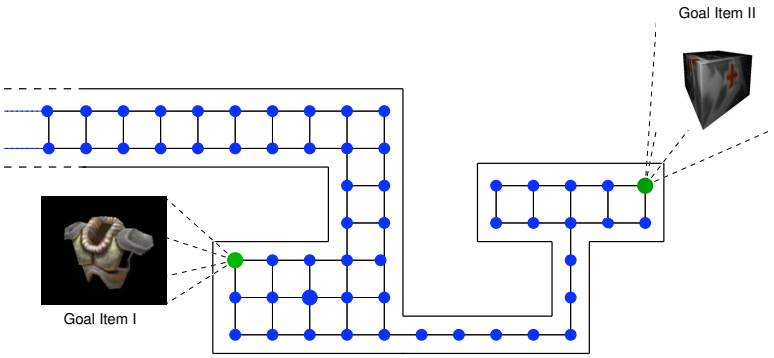


Figure 5.10: Schematic representation of an extended simple maze task. Two points in the waypoint map correspond to the locations of desirable items on the map. In the experimental setting, a subgoal of the game agent is to increase its amour value. It therefore has to devise a path trough the abstract higher dimensional state space that accords with a path through the 3D waypoint map leading to the armor item.

final destination (note that state space paths must not be confused with paths in the 3D gameworld). Figure 5.11 shows an example of a trained game-agent trying to accomplish this task and a three dimensional projection of the corresponding state graph transitions.

The number of state prototypes is crucial. Using a 5 dimensional state space and a medium-sized game-map, we found a number of $150 - 200$ prototypes sufficient for our purpose. Using this setup the observed behaviors were imitated convincingly as is illustrated by state space trajectories in Figure 5.12. The agent managed to reach both goals in the predefined order. Additionally, the movements appeared to be smooth and showed characteristics mostly seen in human players. For instance, the agent "slides" around corners thus preserving observed player habits.

However, we also noticed that the approach tended to show a less accurate program-level imitation than the potential field learning approach. For the Bayesian approach, the agent can usually only imitate state-action pairs that were observed in the training data. While this could be considered accurate imitation, it leads to problems for novel situations.

5.3 Summary

Strategic behaviors represent complex behaviors that aim at long-term goal-achievement. In computer game worlds, as considered in this thesis, strategic behaviors are often centered around specific, important positions in the game-world. Usually, a player tries to reach these places to pickup an item, or to gain a strategic advantage by staying in an important area. In this chapter, we elaborated two approaches for acquiring strategic behaviors from human observation data.

The first approach used artificial potential fields to guide an agent through the

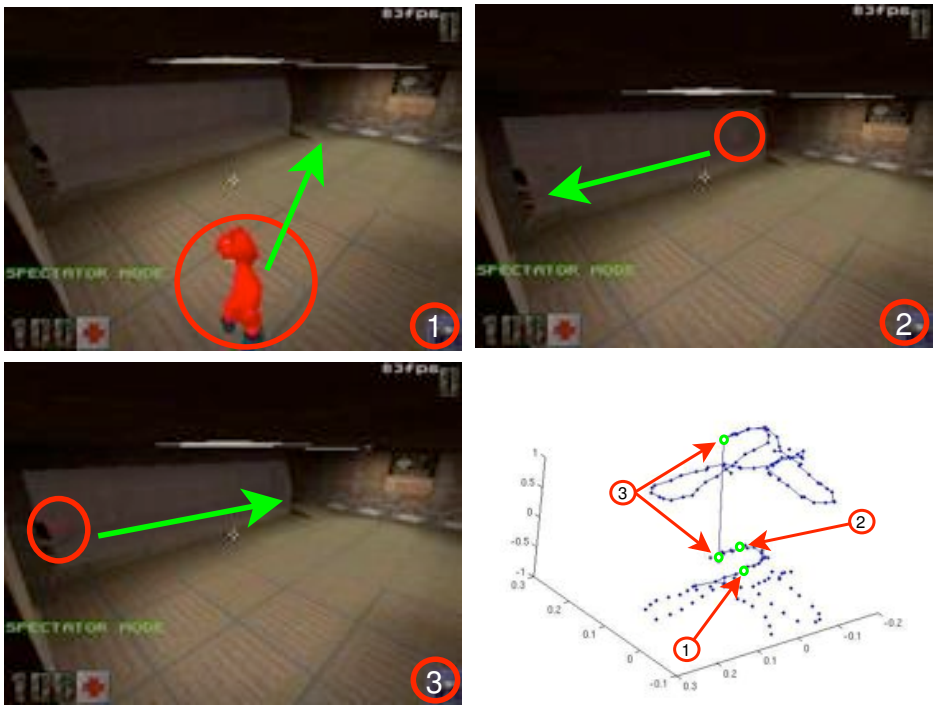
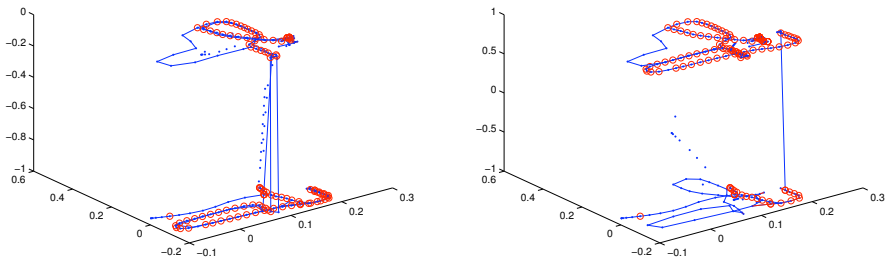


Figure 5.11: Screenshots showing an artificial player solving an item pickup task and visualization of the corresponding transitions in state space. The screenshots display the agent's actions in the game world; the plot in the lower right corner shows what is going on in a subspace of the state space considered in this experiment: the X and Y axes denote the gamebot's (x, y) positions, while the Z axis represents inventory item information. Movements in the 3D gameworld and resulting changes of the agent's internal state correspond to movements between nodes of the state graph. First, in (1), the agent is seen moving along the graph closer to a node whose (x, y) coordinates coincide with a goal item. In (2), the agent is strafing around the corner of a transparent wall. Finally, as seen in screenshot (3), it reaches the item and continues its way to the next item. The item pickup in (3) considerably increases the inventory value for this item and thus results in a 'jump' along the Z axis of the subspace shown here. Although, in visualizations like this, such state space discontinuities appear random to the human eye, they are not. In fact, the screenshots in this figure show but a part of a longer sequence of actions. At the beginning of this sequence, the agent determined its next suitable subgoal represented by a state s_{t+1} and the item pickup in this figure is actually a planned action to reach this graph node s_{t+1}.



(a) (b)

Figure 5.12: Examples of synthesized movements (plotted in state space coordinates). Again, the X and Y axes correspond to x and y positions of the agent whereas the Z axis denotes inventory armor values (5.12(a)) and inventory rail gun values (5.12(b)). The blue dots correspond to sequences of state vectors generated by human player while 'solving' the task of item cycling considered in our second experiment. Red circles indicate state prototypes reached by the artificial player.

game-world. The potential field forces were learned, based on the movements observed in a human player. Since strategies are usually dependent on the internal state of a player, we introduced state dependent variations in potential field strength distributions. In experiments, we could show that the learned potential fields were successfully able to steer an artificial player through the 3D world. However, the rigid way of moving between waypoint nodes led to at times jerky movements, which is something we intentionally tried to avoid. Nevertheless, the approach showed a very good program-level imitation by successfully imitating various observed strategic behaviors of different complexity.

The second approach is based on recent work by Rao et al. [2004]. In their "Bayesian model of imitation in infants and robots", a probabilistic behavior acquisition based on psychological findings is introduced. We applied this model to the task of acquiring strategic behavior in artificial game agents. At the same time, we used the approach to learn human-like synthesis of motion. In detail, we learned state dependent probabilistic action selection to reach implicitly defined goal states. Successfull experimental results underlined the applicability for acquisition of complex goal directed behavior.

To summarize the outcome of the experiments: the Bayesian learning showed to be more suited for generating life-like motions. While we could observe sufficient program-level imitation, generally, the artificial potential field approach was more successfull with respect to goal-reaching behaviors. Moreover, the approach utilizing potential fields also generated adequate goal following behavior for completely novel state observations and for seldom visited nodes. However, due to the artificial

movement between discrete points in 3D, it so far clearly lacked sufficient motion synthesis. Notably, the potential field learning is similar to learning a value function known from dynamic programming or reinforcemeant learning. In the next Chapter, we therefore present a first approach towards integrating decent program-level imitation, using a reinforcement learning approach, and suitbale motion modelling, using the presented Bayesian model of imitation learning. Furthermore, a more detailed evaluation by means of believability testing will provide a more objective rating about the humanlikeness of the artificial player.

Chapter 6

Towards integrating behavioral layers

So far, we could show that behavior acquisition by means of imitation learning can lead to life-like agent acting, the learned and synthesized strategic, tactic, and reactive behaviors appeared human-like. Unfortunately, the overall playing behavior of a human player is a multiplexing of these behavioral layers. Consequently, in order to generate more versatile agent behavior, we should consider integration and combination of individual behaviors.

In this Chapter, we introduce an approach towards integrating strategic program-level imitation into a Bayesian action selection framework. Besides the approach itself, its evaluation will be an important topic of this Chapter. Due to the integration of behaviors considered in this chapter, we finally are able to generate more versatile agent behavior. This, however, allows for a more objective way of evaluation. When synthesizing individual behaviors, the artificial agent usually could be identified as artificial rather easily, because of the restricted repertoire of behaviors. Therefore, an independent evaluation by human subjects did not make sense. Here, for the first time, we conducted a believability test, i.e. human subjects had to decide about the humanness of the generated agent's behavior.

This Chapter is based on a close collaboration with Bernard Gorman, it was published in [Gorman, Thurau, Bauckhage, and Humphrys, 2006a,b], we would like to thank him for the excellent cooperation during his research visit at Bielefeld University in 2005.

6.1 A framework for integration

Most of the time, in QUAKE II® , the players are either engaging in combat or they are navigating the game-world and try to pickup items. Here, we concentrate on the latter. In order to imitate the human behavior observed in item-pickups sequences,

we require *program-level* imitation (where to go next?), and a life-like *motion synthesis* (how to go there?). Notably, we already tried to achieve this by the Bayesian imitation learning introduced in the last Chapter. However, that approach did not result in the desired program-level imitation, at least not as sufficient as it is required here.

For the program-level imitation it is important that the decisions appear smart and are reasonable with respect to the situation. In the last Chapter this was achieved by imitating the strategic behavior of a human by means of artificial potential fields. For the motion synthesis, it is required that different movements can be learned and generated. Further, they need to create a life-like impression. For example, if an agent can only move forward and back, it is probably not creating a human-like impression. In contrast, an agent that is able to execute even complex jumps and difficult maneuvers is more likely to create the impression of being controlled by a human.

In the following, we integrate an approach for learning strategies and an approach for synthesizing movements. The strategy learning is based on a reinforcement learning approach introduced in [Gorman and Humphrys, 2005]. We might have also used the potential field based approach introduced in the last Chapter, since both approaches are similar. However, as we will see, the reinforcement learning approach introduces two extensions (to deal with in-game items) that make it more suited for application in FPS games. The movement synthesis is based on the Bayesian imitation learning that we presented in the last Chapter.

6.1.1 Advanced strategy learning

As usual, a topological graph structure of the game-world is derived by clustering a sequence of player positions $[\mathbf{o}_1, \ldots, \mathbf{o}_n], \mathbf{o} \in \mathbb{R}^3, \mathbf{o} = [x, y, z]$, using a k-means algorithm, resulting in a set of prototypical player positions $[\mathbf{p}_1, \ldots, \mathbf{p}_m]$. Similar to the Neural Gas edge learning in Section 5.1.1, based on sequential traversal of nodes, the nodes are interconnected and a connectivity matrix C is constructed. In a second step, similar to the approach in Section 5.1.2, observed movement paths are clustered according to the player's inventory, in order to assign movement patterns to prototypical situations.

In order to learn strategic decisions, we build a value function by assigning an increasing reward to consecutive nodes in every path taken under each state prototype $\mathbf{s}_i$. Thereby, each node $\mathbf{p}_i$ in the topological map is assigned a value $V_{\mathbf{s}_j}(\mathbf{p}_i)$, for each state prototype $\mathbf{s}_j$. For observed strategic behavior, the value $V_{\mathbf{s}_j}(\mathbf{p}_i)$ usually increases for nodes along the movement path leading to certain items. Thus, following the maximum value of nearby nodes will lead the agent to possible goal-states. Furthermore, the value iteration algorithm is applied to assign utility values to each node under any given inventory prototype. This should lead to appropriate guidance on all nodes in the topological map, for every state prototype.

Gorman and Humphrys [2005] further improved this idea by applying *fuzzy clustering* for weighting multiple state prototypes, i.e. effectively weighting multiple goals. In addition, they introduced an *item activation* variable. In QUAKE II®, once an item is picked up it won't be available for a fixed time period. The item activation variable is responsible for keeping track of this. Both, the fuzzy clustering of state prototypes as well as the item activation variable can be modelled within a membership function $m_{\mathbf{s}_k}$:

$$m_{\mathbf{s}_k}(\mathbf{z}_i) = \frac{a(o_p)|\mathbf{z}_i - \mathbf{s}_k|^{-1}}{\sum_{j=1}^{S} a(\mathbf{s}_j)|\mathbf{z}_i - \mathbf{s}_j|^{-1}} \tag{6.1}$$

where $\mathbf{z}_i$ denotes the current inventory state, $\mathbf{s}_k$ denotes a prototype inventory state, S is the number of prototypes, a is 1 if the object o at the terminal node of the path associated with prototype $\mathbf{s}_k$ is present and 0 otherwise.

In addition, an online discount factor γ is added to decrease the utility value of recently visited nodes. Thereby, the agent is driven to nearby nodes and won't reside in one area. Thus, the utility for a node $\mathbf{p}_i$ can be calculated as follows:

$$U(\mathbf{p}_i) = \gamma^{e(\mathbf{p}_i)} \sum V_{\mathbf{s}_j}(\mathbf{p}_i) m_{\mathbf{s}_j}(\mathbf{z}_i) \tag{6.2}$$

where $U(\mathbf{p}_i)$ is the final utility of node $\mathbf{p}_i$, γ denotes the online discount, $e(\mathbf{p}_i)$ denotes the number of times the player has entered cluster $\mathbf{p}_i$, $V_{\mathbf{s}_j}(\mathbf{p}_i)$ is the original value of node $\mathbf{p}_i$ in state prototype $\mathbf{s}_j$.

Thus, the next node to attain can be selected by maximizing the utility $U(\mathbf{p}_k)$

$$\mathbf{p}_{t+1} = \underset{k}{\operatorname{argmax}}\, U(\mathbf{p}_k), \quad k \in \{x | C_{\mathbf{p}_t, \mathbf{p}_x} = 1\} \tag{6.3}$$

The learned strategic behavior leads to decisions about where to go next, i.e. the next node $\mathbf{p}_{t+1}$ in the topological representation. Thereby, it supplies the intended program-level imitation.

Moving in a straight line from node to node creates a very unrealistic movement behavior, as we already saw in the learning of strategies in the last Chapter. Even worse, moving from node to node might not be possible in all cases, consider for instance a ledge that would require the agent to jump in order to reach the next node. Since we try to make behavior learning an automatic process we obviously can not associate every single node traversal with an action in advance. Consequently, the actions associated with each node have to be learned from observations as well.

6.1.2 Bayesian action selection

In earlier chapters, we successfully acquired motion and movement behaviors using probabilistic models. The very general model elaborated in Chapter 3 introduced a dependency on the latest executed action $\mathbf{a}_t$, and the latest position prototype $\mathbf{p}_t$:

$$\mathbf{a}_{t+1} = \underset{i}{\operatorname{argmax}}\, P(\mathbf{a}_i|\mathbf{a}_t, \mathbf{p}_t) = \frac{P(\mathbf{a}_i|\mathbf{a}_t)P(\mathbf{a}_i|\mathbf{p}_t)}{P(\mathbf{a}_i)} \tag{6.4}$$

where $\mathbf{a}_{t+1}$ denotes the next selected action, $\mathbf{s}_t$ denotes the state at t, and $\mathbf{a}_i$ denotes a single action. This simple model led to life-like motion synthesis of even complex moves.

In Chapter 5, we enhanced this basic model by accounting for *goal-states* and *subgoals*, thus

$$\begin{aligned}\mathbf{a}_{t+1} &= \underset{i}{\operatorname{argmax}}\, P(\mathbf{a}_i|\mathbf{p}_t, \mathbf{p}_{t+1}, \mathbf{p}_g) \\ &= \underset{i}{\operatorname{argmax}}\, \frac{P(\mathbf{p}_{t+1}|\mathbf{a}_i, \mathbf{p}_t, \mathbf{p}_g)P(\mathbf{a}_i|\mathbf{p}_t, \mathbf{p}_g)}{\sum_m P(\mathbf{p}_{t+1}|\mathbf{p}_t, \mathbf{a}_m)P(\mathbf{a}_m|\mathbf{p}_t, \mathbf{p}_g)}\end{aligned} \tag{6.5}$$

where $\mathbf{p}_{t+1}$ denotes the next subgoal, and $\mathbf{p}_g$ denotes a goal-state. Given the states $\mathbf{p}_t$, and the membership function $m_{\mathbf{s}_k}$ from the strategy learning, we can directly apply the action selection from Equation 6.5 for motion synthesis of goal-directed moves. In addition, a weighting of multiple goals needs to be included to account for the goal weighting in the strategy learning. Finally, the probabilities for each action $\mathbf{a}_i$ are computed as follows:

$$\sum_g m_g P(\mathbf{a}_i|\mathbf{p}_t, \mathbf{p}_{t+1}, \mathbf{p}_g) \frac{P(\mathbf{a}_i|\mathbf{a}_t)}{\sum_u P(\mathbf{a}_u|\mathbf{a}_t)} \tag{6.6}$$

An action $\mathbf{a}_{t+1}$ can thus be selected according to:

$$\mathbf{a}_{t+1} = \underset{i}{\operatorname{argmax}} \sum_g m_g P(\mathbf{a}_i|\mathbf{p}_t, \mathbf{p}_{t+1}, \mathbf{p}_g) \frac{P(\mathbf{a}_i|\mathbf{a}_t)}{\sum_u P(\mathbf{a}_u|\mathbf{a}_t)} \tag{6.7}$$

where $\mathbf{p}_{t+1}$, and the goal weighting m_g are supplied by the strategy learning. Thereby, the program-level behavior imitation is achieved using reinforcement learning, whereas motion modelling is implemented using the Bayesian imitation learning. Interestingly, both models depend on each other. The motion synthesis obviously demands subgoal and goal states, while the program-level behaviors need appropriate movement imitation to successfully traverse the topological graph structure.

6.2 Results

In order to test the presented approach we used various gameplay sequences. As usual, the training data showed a human player performing different movements. The gameplay sequences included a variety of item pickups and complex movement

sequences, we will give more details about the selection of moves later on. Again, action primitives $\mathbf{a}_n$ are derived by clustering player movement observations:

$$\mathbf{a} = \begin{pmatrix} \text{yaw angle} \\ \text{pitch angle} \\ \text{forward velocity} \\ \text{side velocity} \\ \text{upward velocity} \\ \text{firing} \end{pmatrix}$$

where the player's viewangle is represented by a yaw angle $\in [-180°, 180°]$, a pitch angle $\in [-90°, 90°]$, the player's velocity is represented by $v_{\text{forward}} = \Delta s_{\text{forward}}/\Delta t, \in [v_{\text{min}}, v_{\text{max}}]$, $v_{\text{side}} = \Delta s_{\text{side}}/\Delta t, \in [v_{\text{min}}, v_{\text{max}}]$, and $v_{\text{upward}} = \Delta s_{\text{upward}}/\Delta t \in [v_{\text{min}}, v_{\text{max}}]$, firing $\in [0, 1]$, with $v_{\text{min}} = -400$ and $v_{\text{max}} = +400$

State prototypes are clustered from the player's inventory $Z = [\mathbf{z}_1, \ldots, \mathbf{z}_n]$ and $\mathbf{z}_i \in \mathbb{R}^{10}, \mathbf{z} = \{\text{health}, \text{armor}, \text{weapon1}, \ldots, \text{weapon8}\}$, where health $\in [0, 200]$, armor $\in [0, 200]$, and weapon1, ..., weapon8 $\in [0, 1]$.

Given these features, the strategic behaviors included in the gameplay sequences could been successfully learned. The in-game action synthesis again yielded a life-like impression, the agent also sufficiently managed to reach the implicitly defined goal-states. Thus, the approach combines a sufficient program-level imitation with learned realistic movement synthesis. As already mentionend, in order to evaluate the generated behaviors, we let human subjects decide about the humanness of the agent. This should give an indicator for the quality of the approach with respect to our ultimate goal of mimicking human behavior.

Based on video-clips, a number of human participants had to decide about the humanness of different agent implementations. The first obvious pitfall lies in the selection of video clips. The experiment consisted of 15 groups, with 3 clips in each; these clips were, on average, approximately 20 seconds in length. The selector may deliberately choose certain clips in an effort to influence the respondents. To guard against this, we first ensure that the number of samples is sufficient to embody a wide variety of behaviors, and second, we cede control of the selection of the specific behaviors to an unbiased arbiter. In our case, we wished to compare the believability of our imitation agents against both human players and traditional rule-based bots[1]; thus, we first ran numerous simulations with the traditional agent – over whose behavior we had no control – to generate a corpus of gameplay samples, and then proceeded to use human clips embodying similar behavior both in the believability test and to train our imitation agents.

The test is presented as a website, as shown in Figure 6.1[2]. First, each rater had

[1] We used the famous Gladiator bot programmed by van Waveren

[2] The website can be accessed at http://reynard.computing.dcu.ie/sab_tests/

Figure 6.1: Believability test for game agents. A human player rates the believability of a video showing a game agent from a third person perspective.

to estimate his experience in playing QUAKE II® by assigning it a rating between 1-5 (1 corresponding to "Never played, rarely or never seen", and 5 corresponding to "Played frequently (daily)"). This helped us to interpret the test results. Each subject was presented a series of 15 pages, showing 3 clips. The human subjects had to classify each video into one out of five categories: human, probably human, don't know, probably artificial, and artificial. In order to make the results reliable, the selection of clips, and the way they are presented are crucial. For example, the clips' order and selection were randomized, and the file names were hidden. Further, we made sure that the clips do not contain any additional 'clues' that might reveal the nature of the agent, for instance nicknames, or special avatar models for visualization. More details on the construction of the test can be found in [Gorman et al., 2006b].

After a one week test period, and after discarding incomplete responses, we had a total number of 20 completed forms and 900 rated clips.

From the evaluation we could conclude that the agent managed to convince human subjects that it is a human who is controlling. In about 69% of all clips showing our agent, it was thought of as human. Interestingly, a human was also only believed to be human in 69% of all human player clips. Whereas the conventional artificial agent was usually classified as being artificial. The corresponding details can be seen in Table 6.1 and Table 6.2. It is of course interesting to see why the imitation learning agent managed to convince the human subjects:

The test allowed to also submit a comment to each clip. These comments gave a very good idea about what people did expect from a human player, and what from an artificial one, Table 6.3 contains some comments. It showed that the player habits often helped to create a human-like impression. The imitation learning ap-

Clip Type	Imitation	Human	Artificial
Human	225	145	102
Artificial	102	68	176
Neutral	32	28	22

Clip Type	Imitation	Human	Artificial
Human	15	12	1
Artificial	1	0	13
Neutral	2	0	1

Table 6.1: The tables show how all subjects classified the clips (upper table), and one exemplary very successful (for our agent) individual user rating (lower table). In a total 900 clips were rated (359 for the imitation learning agent, 241 for the human player, and 300 for the conventional artificial player). In 225 rated clips the imitation agent was being thought of as a human, in contrast, in only 102 clips it was thought of as being artificial. The ratings for the clips showing an agent controlled by a human are similar. In 145 it was rated as human, and in 68 clips it was rated as artificial. The classification threshold is taken to be a rating of 'probably artificial/human', thus interpreting a rating of 'probably human' as rated as 'human'.

Clip Type	Recall	Precision (%)
Human	68.08	78.39
Imitation	68.81	
Artificial	36.69	50.87

Table 6.2: The table shows recall and precision values based on the ratings from the web survey. The values consider classification as 'human' to be the desired results. Again, the very similar ratings for the imitation agent, and the human support the overall human-like agent impression.

Experience	Comment
5	Bunny hop for no reason, also seems to be scanning for enemies
5	Fires gun for no reason , so must be human
5	Unnecessary jumping
5	Stand and wait. AI wouldn't do this (?)
5	Human as they knew how to Rocket jump
5	The rocket jump and the short sequence of backward running at the end suggest this was human

Table 6.3: Sample comments from imitation agent clips misidentified as human. Interestingly, it is often a player's habits (unnecessary jumps, unnecessary firing of gun) that led to a human-like impression. Moreover, complex moves also supported the human-like agent impression (rocket-jumps).

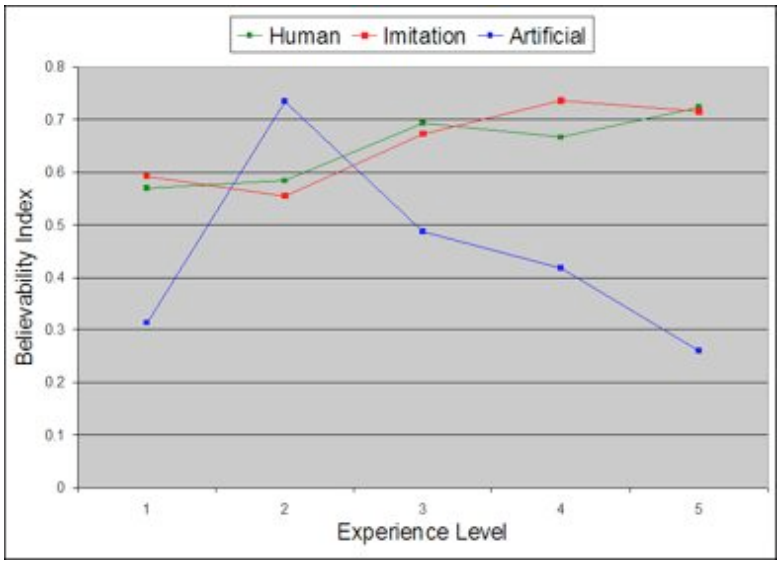


Figure 6.2: Average variation of believability with experience level. The believability index is a representation of the degree to which a given type of clip was regarded as human, in the range $(0, 1)$. What is interesting about these results is that the experienced players mostly believed that the imitation agent is a human player, moreover, they usually correctly identified the rule-based agent as being artificial. The more inexperienced players occasionally mislabeled the different agent types, but generally rated the imitation and human agent similar. The average experience level of respondents was 3.2 in the range $(1, 5)$.

proach successfully learned and generated these habits, for example, useless jumps or firings of weapon. Moreover, the movements themselves also showed to be very important. More complex moves are generally believed to be too difficult to be executed by an artificial player. Therefore, for the experienced QUAKE II® player, seeing an artificial player doing a rocket-jump was probably a surprise - and consequently it had to be human. In general, the experienced players did not have better rating results than the inexperienced, as can be seen in Figure 6.2. Most classified the imitation learning agent as a human player. However, the experienced players also usually recognized the human player as being human. Although the imitation agent also showed a decent program-level imitation by picking up items, this generally did not led to a more human-like impression (this interpretation is based on the user comments that usually did not mention item-pickups).

6.3 Summary

In this chapter, we presented an approach for integrating the imitation of strategic behavior and Bayesian motion modelling. The combination of these led to more versatile, and even more realistic agent behavior compared to what we could achieve with each single approach. The synthesis of more versatile behavior made it possible to conduct a believability test, i.e. human subjects decide about the humanness of an agent.

For the combined approach, we first presented a reinforcement learning based approach for imitating program-level behaviors. In this, a value function is build up considering different movement paths of a human player. The movement paths are expressed as node-sequences in a topological game-world representation. According to a player's internal state, various value functions are build up. Given a specific game situation, the agent can now decide on the next node in the topological map to attain by selecting the node with the corresponding maximum utility value. The iterative selection of nodes finally guides the agent along movement paths leading to important items or places. For a life-like agent's appearance and for steering the agent between nodes in the topological representation, realistic movement imitation is required. For this, we elaborated a Bayesian imitation learning approach, similar to the already introduced methods. The two approaches were integrated in a straight forward manner. The strategy imitation is responsible for selecting the next node to attain, while the motion imitation is responsible for moving between the nodes. The combined approaches finally led to an agent capable of walking along learned paths to goal-states, and at the same time generating life-like movements for doing so.

For evaluation, we conducted a believability test. Based on video clips of different agent implementations, human subjects had to decide on how human-like our agent did behave. Votings on humanness were compared to ratings of video record-

ings of a human player and a conventional game agent. The believability test was carefully designed in order to be as objective as possible. And indeed, the imitation learning bot successfully managed to convince human subjects. In most cases, it was perceived as being a human rather than artificial. In contrast, the conventional game-agent was usually correctly identified as being artificial. And also the human player was sometimes rated as artificial instead of being human. This is, except for some early work by Laird and Duchi [2000], not only the first time that a test like this was conducted for an artificial agent in FPS games, it is also the first time that an artificial player learned all its behavior from the observations of a human player and thereby managed to create the impression of being controlled by a human.

Chapter 7

Conclusion

In humans, behavior acquisition by means of learning from observations is necessary for a rapid development of behaviors. Lately, this topic gained popularity in robotics and artificial agent control, too. The idea is that by imitating someone else an artificial agent can boostrap behavior in a fast and reliable manner. In this thesis we applied this concept for behavior acquisition in artificial game-agents, in order to model human-like behavior.

Behavior acquisition can only be addressed in association with an environment. For instance, modern computer games also require sophisticated, complex behaviors in order to successfully play the game. Within such games, not only can we observe a number of complex behaviors of humans, we can also access vast amounts of observation data extracted from the network traffic of Internet games. Moreover, we do not have to deal with problems in sensor-noise or actuator control. This allows us to fully concentrate on the behavioral aspects. In this work, we considered the popular genre of First-Person-Shooter games. In these, simulated avatars are steered by human players through a simulated 3D world that is in many aspects inspired by the real-world. Therefore, FPS games require less abstraction than most other genres, and are thus a well suited application for researching human-level intelligence.

In games, we can easily access the game-state informations a human player receives as well as his reactions. In our case, the behavior of a player defines how these sensory inputs are transformed to reactions. The task now is to learn a similar mapping for usage in an artificial agent. For this, we applied methodologies from pattern recognition and machine learning. In particular, we preferred methods inspired by biological and psychological findings. Even the idea of acquiring behavior from observing others is clearly inspired by psychological findings, since it is a basic human ability already developed in infants. Besides this, we also utilize the concepts of biological "place cells" for providing a spatial memory, or "basic action units" for providing an optimized storage of motor commands, and also different types of artificial neural networks. Thus, the contribution of this thesis is

not only the idea of transferring the imitation learning task to games, it is also the unique combination of approaches grounded in biological findings.

To make the idea of behavior learning feasible, we used a well known psychologically motivated separation of behavior into three categories, i.e. (a) reactive, (b) tactic, and (c) strategic behavior. The overall behavior of a human can be understood as a multiplexing of these layers. Regarding imitation learning, we could now concentrate on each layer individually. However, we did not understand the structuring of behavior as truly hierarchical. In a truly hierarchical model the cognitive higher level layers would control the cognitive lower layers. Instead, each layer is responsible on its own for synthesizing the required in-game actions. Moreover, we do not iteratively combine atomic behaviors, as for instance obstacle avoidance, to generate complex behavior. Therefore, our view on behavior differs from the ideas proposed in the behavior-based robotics community that are usually based on combinations of atomic behaviors.

(a) Reactive behaviors denote player actions that only have a short-term relevance, for example, simple movements or aiming. For learning reactive behaviors, we developed two main approaches: the first understood the reactive learning as a regression problem, where we used neural networks; Multi-layer Perceptrons were applied to learn a suitable mapping from sensor inputs to reactions. We further enhanced this approach by applying the concept of biological place cells to roughly discriminate prototypical situations, and thus divide the learning problem among multiple MLPs. It has been shown that this approach was very well suited for learning distinct behaviors that are simultaneously included in the observation data. Behavior-wise, it led to a good reproduction of movement and aiming behaviors. The second approach utilized the already mentioned movement and action primitives, where the idea of basic action units is also supported by psychological evidence. Here, we learned a probabilistic mapping of prototypical situations onto a single action unit, i.e. basically a prototypical player action. The prototypical situations were derived from a topological game-world representation, which again is inspired by place cells. The suggested probabilistic sequencing of basic action units led to natural motion and movement imitation. Even complex moves performed by a human player were successfully imitated in an artificial agent. The idea of appropriate action sequencing is one of the main contributions and is also applied for the learning of tactical and strategic behaviors.

(b) Tactical behaviors were introduced as behaviors in-between reactive and strategic ones. We concentrated on the important, for a FPS game, behaviors in player and opponent interactions. Again, we made use of the biological concepts of place cells and action primitives. However, for a first approach on learning weapon handling we were using a Mixture of Experts architecture. In this, we aggregated multiple MLPs responsible for learning the handling of different weapon-types from the observations of a human player. The resulting aiming behavior adapted to the

different characteristics of a selected weapon. Besides the weapon handling, appropriate tactical movements for 1 vs. 1 situations were important. For example, human players might try to evade, or approach an opponent. Again, the successfull probabilistic selection of basic action units was applied here. However, this time we developed a different state representation especially suited for the dynamics of player and opponent interactions. The states were again based on deriving state prototypes. The prototypes, however, reflected the special spatial orientations in 1 vs. 1 game-states. Finally, the developed approach resulted in life-like, opponent dependent movement synthesis.

(c) For the cognitively highest layer, i.e. strategic behaviors, we considered behaviors targetted at long-term goal achievement. These behaviors provide another challenge since they require the extraction of goal- and subgoal-states. Instead of directly trying to extract the goal-states, we contributed an approach for implicitly learning them. In this, state dependent movement patterns were remodelled using artificial potential fields. Thus, the artificial agent was guided along a the steepest gradient of the learned potential fields, thereby, the agent was guided to places or items that could be interpreted as goal-states. Changes to the state vector, by for instance picking up an item, led to changed potential field forces, and thus provide further guidance. This developed approach provided successful goal-reaching behavior, i.e. program-level imitation. However, since no motion modelling was included, the movements appeared rather machine-like. To tackle the problem of motion synthesis we elaborated a second approach. Recently, based on recent results in developmental psychology, Rao et al. proposed a Bayesian model of imitation learning. We applied this model to the task of learning strategies in games. Again, behavior was the outcome of appropriate action primitive sequencing. The sequencing depends on memorized sequences of subgoal and goal-states, effectively denoting nodes in a topological represention of the game-world and internal player states. Although the program-level imitation did not reach the performance of the potential field based approach, the motion modelling using the action primitives appeared realistic and life-like.

Since the overall behavior of a human can be understood as a multiplexing of reactive, tactic, and strategic behavior, we finally developed a first approach towards integrating strategic behavior and reactive motion modelling. In this, we combined an approach for learning a strategy based on reinforcement learning, and the already mentioned Bayesian motion imitation. For imitating a player's strategical behavior, a utility value was assigned to each node in a topological game-world representation. They are assigned so that maximizing the reward for the agent would result in imitating the observed strategic movement patterns. To adapt the strategy, i.e. react to varying internal states of the human player, multiple state dependent utility distributions are build up. Finally, novel movement patterns are generated by following the node with the maximum reward. Here, the Bayesian imitation learning

of movements comes into play. Instead of walking directly from node to node, the motions between the nodes are learned within a probabilistic model. Thus, based on the current and the next desired node action primitives are selected. This unique approach allowed for a sufficient program-level imitation while at the same time generating complex motion sequences.

In order to evaluate the approach we conducted a believability test. In the test, human subjects had to make judgments on the humanness of different agent implementations based on a series of video clips. By constraining the showed behaviors, and randomizing, and anonymizing the clips we tried to make sure that the human subjects indeed only decide on the observed behaviors, and that their decisions are not influenced by hidden 'clues' that might reveal the nature of the agent. In the test, we compared our own agent, using the combined approach, against a human player and a conventional artificial agent. The results indicated that the agent is mostly perceived as being controlled by a human rather than being artificial. Thereby, we not only developed a Turing like test for believability in games, we also could confirm that the developed approaches successfully learn and generate human-like behavior.

Future work

While we managed to successfully imitate a vast amount of important behaviors in artificial agents, the agent did not yet reach a status at which it would be able to play against a human player. We obviously tested the various behaviors in live games, but a fully working agent indeed demands behavior learning and integration capabilities that are to date beyond realization. Therefore, we can not yet answer the question how good the agent actually is at playing the game. We are confident that future work will lead to the imitation of more complex human behaviors, and finally to a level at which a competition between a human and an imitation learning agent can take place.

This thesis clearly showed that single behaviors can be learned by observing others. For future research and according to our experience we would suggest a more global learning architecture. A crucial question will be, how to integrate behavior. Therefore, it is not only important to combine already learned behaviors, but also to integrate novel behavioral observations into existing behaviors. The analysis of observation data is probably another crucial topic for ongoing research. We are confident that it is possible to extract important information directly from the data samples, for example about goal-states, segments of behavior, or relations between behaviors. Moreover, the refinement and adaptation of existing behaviors will also be of great importance. Often, these abilities are considered characteristic of human-level intelligence.

Bibliography

R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

A. Arleo. *Spatial Learning and Navigation in Neuro-Mimetic Systems, Modeling the Rat Hippocampus*. PhD thesis, Swiss Federal Institute of Technology of Lausanne (EPFL), 2001.

S. Bakkes, P. Spronck, and E. Postma. TEAM: The Team-oriented Evolutionary Adaptability Mechanism. In *In Proc. Int. Conf. on Entertainment Computing - ICEC*, number 3166 in Lecture Notes in Computer Science, pages 273–282. Springer-Verlag, 2004.

T. Balch and R. Arkin. Avoiding the past: A simple but effective strategy for reactive navigat. In *Proc. IEEE Int. Conf. on Robotics and Automation*, 1993.

J. Barnes and J. Hutches. *AI Game Programming Wisdom*, chapter Scripting for Undefined Circumstances, pages 530–540. Charles River Media, 2002.

C. Bauckhage and C. Thurau. Exploiting the Fascination: Video Games in Machine Learning Research and Education. In *Proc. 2nd Int. Workshop in Computer Game Design and Technology (GDTW'04)*, pages 61–70. ACM, 2004.

C. Bauckhage, C. Thurau, and G. Sagerer. Learning Human-like Opponent Behavior for Interactive Computer Games. In *Pattern Recognition*, volume 2781 of *LNCS*, pages 148–155. Springer-Verlag, 2003.

L. Berger. *AI Game Programming Wisdom*, chapter Scripting: Overview and Code Generation, pages 505–510. Charles River Media, 2002a.

L. Berger. *AI Game Programming Wisdom*, chapter Scripting: System Integration, pages 516–519. Charles River Media, 2002b.

L. Berger. *AI Game Programming Wisdom*, chapter Scripting: The Interpreter Engine, pages 511–515. Charles River Media, 2002c.

C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, Oxford, UK, 1996.

E. Bonabeau, M. Dorigo, and G. Theraulaz. *From Natural to Artificial Swarm Intelligence*. Oxford University Press, 1999.

C. Breazeal and B. Scassellati. Robots that imitate humans. *Trends in Cognitive Science*, 6(11):481–487, 2002.

T. Cain. *AI Game Programming Wisdom*, chapter Practical Optimizations for A Path Generation, pages 146–152. Charles River Media, 2002.

S. Cass. Mind Games. *IEEE Spectrum*, pages 40–44, December 2002.

A. J. Champandard. *AI Game Development*. New Riders, 2004.

A. J. Champandard. AI Depot. URL:http://ai-depot.com, 2006a.

A. J. Champandard. FEAR Project. URL:http://fear.sourceforge.net, 2006b.

M. DeLoura, editor. *Game Programming Gems 1*. Charles River Media, 2000.

M. DeLoura, editor. *Game Programming Gems 2*. Charles River Media, 2001.

A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1): 1–38, 1977.

P. Diaconis. Recent Progress on de Finetti's Notions of Exchangeability. In J.M. Bernardo, M.H. DeGroot, D.V. Lindley, and A.F.M. Smith, editors, *Bayesian Statistics*, volume 3, pages 111–125. Oxford Univ. Press, 1988.

M. Dickheiser, editor. *Game Programming Gems 6*. Charles River Media, 2006.

E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

Drivatar^{TM}- Driving Avatar. URL: http://research.microsoft.com/mlp/forza/, 2006.

A. D. Ekstrom, M. J. Kahana, J. B. Caplan, T. A. Fields, E. A. Isham, E. L. Newman, and I. Fried. Cellular networks underlying human spatial navigation. *Nature*, 425: 184–187, 2003.

P. Figueroa, N. Leite, R. M. L. Barros, Isaac Cohen, and Gerard Medioni. Tracking soccer players using the graph representation. In *ICPR*, 2004.

A. Fod, M.J. Matarić, and O.C. Jenkins. Automated Derivation of Primitives for Movement Classification. *Autonomous Robots*, 12(1):39–54, 2002.

D. Fu and R. Houlette. *AI Game Programming Wisdom 2*, chapter The Ultimate Guide to FSMs in Games, pages 283–302. Charles River Media, 2002.

J. Funge. *Artificial Intelligence for Computer Games*. A K Peters, Wellesley, MA, 2004.

J. Fürnkranz. Machine Learning in Games: A Survey. In J. Fürnkranz and M. Kubat, editors, *Machines that Learn to Play Games*, chapter 2, pages 11–59. Nova Science Publishers, Huntington, NY, 2001.

B. Geisler. An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a 'First Person Shooter' Video Game. Master's thesis, Department of Computer Sciences, University of Wisconsin-Madison, 2002.

R. Geraerts and M. H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, 54:165–173, 2006.

Z. Ghahramani. Building blocks of movement. *Nature*, 407:682–683, October 2000.

B. Gorman and M. Humphrys. Towards Integration of Strategic Planning and Motion Modelling in Interacctive Cojmputer Games. In *Proc. Int. Conf. Computer Game Design & Technology*, pages 92–99, 2005.

B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys. Bayesian Imitation of Human Behavior in Interactive Computer Games. In *Proc. Int. Conf. on Pattern Recognition*. IEEE, 2006a.

B. Gorman, C. Thurau, C. Bauckhage, and M. Humphrys. Believability Testing and Bayesian Imitation in Interactive Computer Games. In *Proc. 9th Int. Conf. on the Simulation of Adaptive Behavior (SAB'06)*, volume LNAI. Springer, 2006b.

T. Graepel, R. Herbrich, and J. Gold. Learning to Fight. In *Proc. Int. Conf. on Computer Games, Artificial Intelligence,Design and Education*, 2004.

T. L. Griffiths and J. B. Tenenbaum. Statistics and the bayesian mind. *Significance*, 3(4):130–133, 2006.

J. Hancock. *AI Game Programming Wisdom*, chapter Navigating Doors, Elevators, Ledges, and other Obstacles, pages 193–201. Charles River Media, 2002.

T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

D. Higgins. *AI Game Programming Wisdom*, chapter Generic A Pathfinding, pages 114–121. Charles River Media, 2002a.

D. Higgins. *AI Game Programming Wisdom*, chapter Hot wo Achieve Lightning-Fast A, pages 133–145. Charles River Media, 2002b.

D. Higgins. *AI Game Programming Wisdom*, chapter Pathfinding Design Architecture, pages 122–132. Charles River Media, 2002c.

E. Hollnagel. *Human Reliability Analysis: Context & Control*. Academic Press, 1994.

R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive Mixture of Local Experts. *Neural Computation*, 3(1):79–87, 1991.

J. Juul. *Half-Real: Video Games between Real Rules and Fictional Worlds*. MIT Press, 2005.

A. Khoo and R. Zubek. Applying Inexpensive AI Techniques to Computer Games. *IEEE Intelligent System Special Issue on Interactive Entertainment*, 17(4):48–53, 2002.

A. Kirmse, editor. *Game Programming Gems 4*. Charles River Media, 2004.

K. P. Körding and D. M. Wolpert. Bayesian integration in sensorimotor learning. *Nature*, 427:244–247, 2004.

J. E. Laird. It knows what you're going to do: adding anticipation to a Quakebot. In Jörg P. Müller, Elisabeth Andre, Sandip Sen, and Claude Frasson, editors, *Proceedings of the Fifth International Conference on Autonomous Agents*, pages 385–392, Montreal, Canada, 2001a. ACM Press.

J. E. Laird. Research in Human-Level AI using Computer Games. *Communications of the ACM*, 45:32–35, 2002.

J. E. Laird and J. C. Duchi. Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot. In *AAAI 2000 Fall Symposium Series: Simulating Human Agents*, 2000.

J. E. Laird and M. v. Lent. Interactice Computer Games: Human-Level AI's Killer Application. In *Proc. AAAI*, pages 1171–1178, 2000.

J.E. Laird. Using a Computer Game to develop advanced AI. *IEEE Computer*, pages 70–75, July 2001b.

R. Le Hy, A. Arrigioni, P. Bessière, and O. Lebeltel. Teaching Bayesian behaviours to video game characters. *Robotics and Autonomous Systems*, 47(2–3):177–185, 2004.

L. Lidén. *AI Game Programming Wisdom*, chapter Strategic and Tactical Reasoning with Waypoints, pages 211–220. Charles River Media, 2002.

A. Lipson, H. Kueck, E. Brochu, and N. de Freitas. The "Touring" Test: Human-Like Play in Computer Games. In *First International Digital Games Research Conference (DiGRA)*, 2003. (Invited poster presentation).

D. Livingstone. Turing's Test and Believable AI in Games. *ACM Computers in entertainment*, 4(1), 2006.

E. Maguire, N. Burgess, J. Donnett, R. Frackowiak, C. Frith, and J. O'keefe. Knowing Where and Getting There: A Human Navigation Network. *Science*, 280: 921–924, 1998.

T. Martinetz and K. Schulten. A "Neural Gas" Network Learns Topologies. In *Artificial Neural Networks*. Elsevier Science Publisher B.V., 1991.

T.M. Martinetz, S.G. Berkovich, and K.J. Schulten. Neural Gas Network for Vector Quantization and its Application to Time-Series Prediction. *IEEE Trans. on Neural Networks*, 4(4):558–569, 1993.

J. Matthews. *AI Game Programming Wisdom*, chapter Basic A pathfinding made simple, pages 105–113. Charles River Media, 2002.

S. McGlinchey. Learning of AI players from game observation data. In *Proc. 4th International Conference on Intelligent Games and Simulation (Game-On'03)*, 2003.

S. McGlinchey and D. Livingstone. What believability testing can tell us. In *Proc. Int. Conf. on Computer Games, Artificial Intelligence,Design and Education*, 2004.

M. Mika and C. Charla. *AI Game Programming Wisdom*, chapter Simple, Cheap Pathfinding, pages 155–160. Charles River Media, 2002.

A. Nareyek. Artificial Intelligence in Computer Games – State of the Art and Future Directions. *ACM Queue*, 1(10):58–65, 2004.

D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind, and M.H. Overmars. Automatich Construction of Roadmaps for Path Planning in Games. In *In Proc. International Conference on Computer Games:Artificial Intelligence, Design and Education*, 2004.

E. Norling and L. Sonenberg. Creating Interactive Characters with BDI Agents. In *In Australian Workshop on Interactive Entertainment*, 2004.

K. Okuma, A. Taleghani, N. de Freitas, J. Little, and D. Lowe. A Boosted Particle Filter: Multitarget Detection and Tracking. In *ECCV*, 2004.

J. Orkin. *AI Game Programming Wisdom 2*, chapter Applying Goal-Oriented Action Planning to Games, pages 217–227. Charles River Media, 2004.

M. H. Overmars. Path Planning for Games. In *In Proc. International Game Design and Technology Workshop and Conference*. ACM, 2005.

K. Pallister, editor. *Game Programming Gems 5*. Charles River Media, 2005.

G. H. Paull and C. J. Darken. Integrated On- and Off-line Cover Finding and Exploitation. In *Proc. GAME-ON*, pages 20–24, 2004.

M. Pfeiffer. Reinforcement learning of strategies for Settlers of Catan. In *Proc. of Int. Conf. on Computer Games, Artificial Intelligence, Design and Education*, 2004.

M. Ponsen and P. Spronck. Improving Adaptive Game AI with Evolutionary Learning. In *Proc. Int. Conf. on Computer Games, Artificial Intelligence,Design and Education*, pages 402–408, 2004.

J. R. Quinlan. Induction of Decision Trees. In J. W. Shavlik and T. G. Dietterich, editors, *Readings in Machine Learning*. Morgan Kaufmann, 1990. Originally published in *Machine Learning* 1:81–106, 1986.

S. Rabin, editor. *AI Game Programming Wisdom*. Charles River Media, 2002.

S. Rabin, editor. *AI Game Programming Wisdom 2*. Charles River Media, 2004.

S. Rabin, editor. *AI Game Programming Wisdom 3*. Charles River Media, 2006.

A.V. Rao, D. Miller, K. Rose, and A. Gersho. Mixture of Experts Regression Modeling by Deterministic Annealing. *IEEE Trans. on Signal Processing*, 45(11): 2811–2820, 1997.

R.P.N. Rao, A.P. Shon, and A.N. Meltzoff. A Bayesian Model of Imitation in Infants and Robots. In K. Dautenhahn and C. Nehaniv, editors, *Imitation and Social Learning in Robots, Humans, and Animals: Behavioural, Social and Communicative Dimensions*. Cambridge University Press,, 2004. to appear.

D. Redpath and K. Lebart. Bossting feature selection. In *3rd International Conference on Advances in Pattern Recognition (ICAPR'05)*, 2005.

C. Reed and B. Geisler. *AI Game Programming Wisdom 2*, chapter Jumping. Climbing. and Tactical Reasoning: How to Get More Out of a Navigation System, pages 141–150. Charles River Media, 2004.

G. Rosado. *AI Game Programming Wisdom 2*, chapter Implementing a Data-Driven Finite-State Machine, pages 307–318. Charles River Media, 2004.

S. Schaal. Is Imitation Learning the Route to Humanoid Robots? *Trends in Cognitive Sciences*, 3(6):233–242, 1999.

T. Schack. The cognitive architecture of complex movement. *International Journal of Sport and Exercise Psychology*, Special Issue Part II: The construction of action - new Perspectives in Movement Science,2 (4):403–438, 2004.

T. Schack and F. Mechsner. Representation of motor skills in human long-term memory. *Neuroscience Letters*, 391:77–81, 2006.

E. Sklar, A.D. Blair, P. Funes, and J. Pollack. Training Intelligent Agents Using Human Internet Data. In *Proc. 1st Asia-Pacific Conference on Intelligent Agent Technology (IAT-99)*, 1999.

Soar. URL:http://sitemaker.umich.edu/soar, 2006.

P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Improving Opponent Intelligence through Machine Learning. In *Proc. 3rd Int. Conf. on Intelligent Games and Simulation (GAME-ON'02)*, 2002.

P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma. Online Adaptation of Game Opponent AI in Simulation and in Practice. In *Proc. 4th Int. Conf. on Intelligent Games and Simulation (GAME-ON'03)*, 2003.

E. Spyridou, I. Palmer, and E. Williams. Investigating Team Speech Communication in FPS Video Games. In *Proc. of Int. Conf. on Computer Games, Artificial Intelligence, Design and Education*, 2004.

G. Sukthankar and K. Sycara. Automatic Recognition of Human Team Behaviors. In *Modeling Others from Observations, Workshop at IJCAI*, 2005.

S. Surasmith. *AI Game Programming Wisdom*, chapter Preprocessed Solution for Open Terrain Navigation, pages 161–170. Charles River Media, 2002.

R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, 1998.

J.B. Tenenbaum, T. L. Griffiths, and C. Kemp. Theory-based bayesian models of inductive learning and reasoning. *Trends in Cognitive Sciences*, 10(7):309–318, 2006.

K.A. Thoroughman and R. Shadmehr. Learning of action through adaptive combination of motor primitives. *Nature*, 407:742–747, October 2000.

S. Thrun. Learning to Play the Game of Chess. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems (NIPS) 7*, Cambridge, MA, 1995. MIT Press.

C. Thurau and C. Bauckhage. Tactical Waypoint Maps: Towards Imitating Tactics in FPS Games. In M. Merabti, N. Lee, and M.H. Overmars, editors, *Proc. 3rd International Game Design and Technology Workshop and Conference (GDTW'05)*, pages 140–144. ACM, 2005.

C. Thurau, C. Bauckhage, and G. Sagerer. Combining Self Organizing Maps and Multilayer Perceptrons to Learn Bot-Behavior for a Commercial Computer Game. In *Proc. GAME-ON*, pages 119–123, 2003.

C. Thurau, C. Bauckhage, and G. Sagerer. Learning Human-Like Movement Behavior for Computer Games. In *Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB'04)*, pages 315–323, 2004.

C. Thurau, T. Paczian, and C. Bauckhage. Bayesian Imitation Learning in Game Characters. In *International Workshop on Automatic Learning and Real-Time (ALaRT'05)*, pages 143–151, 2005a.

C. Thurau, T. Paczian, and C. Bauckhage. Is Bayesian Imitation Learning the Route to Believable Gamebots? In *Proc. GAME-ON North America*, pages 3–9, 2005b.

C. Thurau, T. Hettenhausen, and C. Bauckhage. Classification of Team Behaviors in Sports Video Games. In *Proc. Int. Conf. on Pattern Recognition*. IEEE, 2006.

J. Togelius, R. De Nardi, and S. M. Lucas. Making racing fun through player modeling and track evolution. In *SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, 2006.

P. Tozour. *AI Game Programming Wisdom*, chapter First-Person Shooter AI Architecture, pages 387–396. Charles River Media, 2002a.

P. Tozour. *AI Game Programming Wisdom*, chapter The Basics of Ranged Weapon Combat, pages 411–418. Charles River Media, 2002b.

P. Tozour. *AI Game Programming Wisdom*, chapter The Perils of AI Scripting, pages 541–547. Charles River Media, 2002c.

P. Tozour. *AI Game Programming Wisdom 2*, chapter Stack-Based Finite State Machines, pages 303–306. Charles River Media, 2004.

D. Treglia, editor. *Game Programming Gems 3*. Charles River Media, 2002.

J. P. van Waveren. The Quake III Arena Bot. Master's thesis, Delft University of Technology, 2001.

D. M. Wolpert, Z. Ghahramani, and J. R. Flanagan. Perspectives and problems in motor learning. *TRENDS in Cognitive Sciences*, 5(11):487–494, November 2001.

G. N. Yannakakis and J. Hallam. Evolving Opponents for Interesting Interactive Computer Games. In *Proc. 8th Int. Conf. on the Simulation of Adaptive Behavior (SAB'04)*, pages 499–508, 2004.

S. Zanetti and A. El Rhalibi. Machine learning techniques for FPS in Q3. In *ACE '04: Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 239–244. ACM Press, 2004.