Learning Human-like Opponent Behavior for Interactive Computer Games

Christian Bauckhage, Christian Thurau, and Gerhard Sagerer

Technical Faculty, Bielefeld University, P.O. Box 100131, 33501 Bielefeld, Germany
{cbauckha,cthurau,sagerer}@techfak.uni-bielefeld.de

Abstract. Compared to their ancestors in the early 1970s, present day computer games are of incredible complexity and show magnificent graphical performance. However, in programming intelligent opponents, the game industry still applies techniques developed some 30 years ago. In this paper, we investigate whether opponent programming can be treated as a problem of behavior learning. To this end, we assume the behavior of game characters to be a function that maps the current game state onto a reaction. We will show that neural networks architectures are well suited to learn such functions and by means of a popular commercial game we demonstrate that agent behaviors can be learned from observation.

1 Context, Motivation, and Overview

Modern computer games create complex and dynamic virtual worlds which offer numerous possibilities for interaction and are displayed using incredible computer graphics. Professional game development therefore has become expensive and time consuming and involves whole teams of programmers, authors and artists [2, 8]. However, despite all progress in appearance and complexity, when it comes to implementing intelligent virtual characters the game industry largely ignores scientific advances but still reverts to techniques known for more than 30 years [2].

Up to now, the most common techniques to control virtual characters are finite state machines (of admittedly complex topology) and scripts. From a player's point of view this has two major drawbacks: (1) The actions of computer controlled characters often appear artificial since they just cycle through a fixed repertoire; this provokes repetitions and thus causes ennui and frustration. (2) Finite stated or scripted behaviors cannot generalize. Thus, if a human player acts unforeseen, i.e. interaction results in a game state not envisaged by the programmers, virtual characters tend to behave 'dumb' [2].

Therefore –and certainly because of the popularity computer games enjoy among today's students– creating intelligent opponents has attracted attention in AI research [3, 4, 11]. Especially Laird identifies a need for human-like behaving characters and heralds games as the 'killer application' of artificial intelligence [8, 9]. And indeed, all cited contributions describe ontology based inference machines or reasoning mechanisms and thus are classical AI. In the following, however, we will argue that computer games also offer interesting problems and testbeds for the pattern recognition community. While the next section explains this claim in general, section 3 treats practical

In B. Michaelis and G. Krell, editors, Pattern Recognition, Lecturenotes in Computer Science 2781, pages 148-155. Springer-Verlag, 2003.

issues concerning this idea. In section 4 we present results obtained from several experiments on learning of human-like behavior and finally an outlook concludes this contribution.

2 A Pattern Recognition Perspective on Game Characters

Modern computer games create virtual worlds of enormous sizes in which the player slips into the role of a virtual character who has to fulfill a certain task. Depending on the genre, tasks can reach from building and administer cities or civilizations over solving adventurous quests to simply staying alive on a virtual battlefield.

While it is obvious that computer games are not the real world, it is important to note that most of them must not be confused with simulations either. Like in simulations, the states of a game are characterized by a huge set of parameters which encode the current configuration of the virtual world. If this configuration is thought of as a point in a high dimensional state space, the current state of a player's virtual character corresponds to a point in an appropriate subspace and the history of states a character assumes during a game forms a path in the state space. But in contrast to simulations, interactive games merely constrain the actions of a player. Of course there are rules which govern the evolution of game states but they are not necessarily tailored to the real world (in some games, for instance, teleportation is possible). Therefore, paths in the state space neither need to be linear nor smooth. And since there are dynamic interactions between virtual world, computer controlled opponents, and player characters, the state of the player's character is influenced by the player's actions as well as by events in the virtual world. As computer controlled events in the virtual world often are triggered randomly and a players can freely chose their actions, the states of player controlled characters evolve unforeseen rather than predictable.

Given these observations, we see that evolving player states constitute discrete time series. And in a simple approximation, their evolution could be assumed to depend only on the last time step. I.e. if we assume the state of player p at time t to be given by a vector $\underline{\mathbf{s}}_t^p$, the player's state at the next time step $t+1$ could be modeled as

$$\underline{\mathbf{s}}_{t+1}^p = \underline{\mathbf{s}}_t^p + \underline{\mathbf{e}}_t + \underline{\mathbf{a}}_t^p(\underline{\mathbf{s}}_t^p) \tag{1}$$

where $\underline{\mathbf{e}}_t$ denotes environmental influences at time t and $\underline{\mathbf{a}}_t^p(\underline{\mathbf{s}}_t^p)$ represents the action player p accomplishes according to his current state. If we restate this equation as

$$\underline{\mathbf{a}}_t^p = f(\underline{\mathbf{s}}_{t+1}^p, \underline{\mathbf{s}}_t^p, \underline{\mathbf{e}}_t) \tag{2}$$

we see that player actions correspond to what Arkin calls *reactive behaviors* [1]. I.e. the actions of a player first of all depend on his or her state and on the current environmental influence. Furthermore, we see that, given suitable training data, prototypical actions $\underline{\mathbf{a}}_t^p$ or situated behaviors which we define to be sequences of actions $\{\underline{\mathbf{a}}_{t_i}^p, \underline{\mathbf{a}}_{t_{i+1}}^p, \ldots, \underline{\mathbf{a}}_{t_{i+n}}^p\}$ of a player might be *learnable* using techniques like statistical classifiers, HMMs, or neural networks.

Actually, Markovian approaches and conditional expectation maximization have already been successfully applied to behavior learning in human-machine interaction [5,

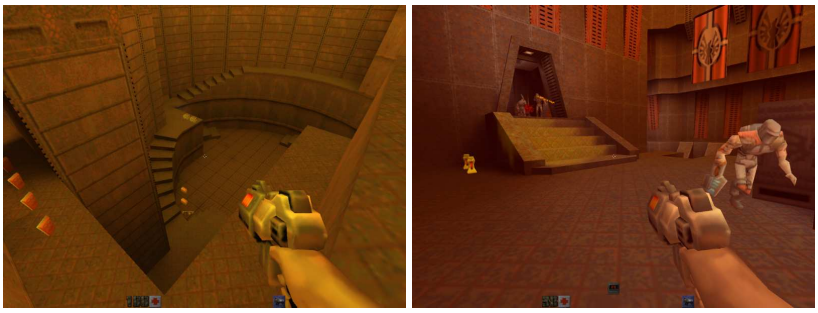


Fig. 1. Screenshots taken from ID's (in)famous FPS game Quake II.

7]. Although these methods offer interesting perspectives for computer games, too, we aimed at finding a non-probabilistic approximation of the mapping f in Eq. (2). We thus opted for neural network based approaches. But before we discuss first corresponding results in learning of opponent behavior for commercial games, the next section shall point out that appropriate training data is easily available for most present day games.

3 Practical Issues in Behavior Learning for Game Characters

For the reminder of this contribution, we will focus on the infamous but nevertheless popular genre of *first person shooter* (FPS) games[1]. In a FPS game, the player moves through a virtual world (called *map* in gamers terminology) which he perceives from the first person perspective (s. Fig.1). Though variations exist, his main task is to battle against every other character on the map. In doing so, the player will loose health, armor, and ammunition which can be compensated by collecting corresponding items distributed all over a map. The state of a FPS character therefore is almost completely determined by its current position and view on the map and its current armament and health conditions.

The ideas discussed below resulted from the fact that nowadays it is actually possible to earn a living from playing computer games. There are professionals who regularly compete at tournaments worth several 10.000 dollars of trophy money. Observing these professionals playing FPS games, one realizes that they do not reflect the situations they encounter in a game but simply react. Their reactions result from long term practice and usually only depend on the current state of their virtual character as well as on its current environmental context. Our reactive behavior model in Eq. (1) thus seems perfectly applicable to this genre. Moreover, game athletes also solve the problem of getting training data for behavior learning: most present day computer games allow to record matches. These so called *demos* can be viewed afterwards and show the game from the perspective of the player who recorded it. A demo therefore encodes the series of states $\underline{\mathbf{s}}_0^p, \underline{\mathbf{s}}_1^p, \underline{\mathbf{s}}_2^p, \ldots, \underline{\mathbf{s}}_N^p$ the recording player p underwent during a game.

[1] Even though ethically controversial, this genre is (pioneered by Laird) commonly to be found in literature on intelligent computer controlled opponents.

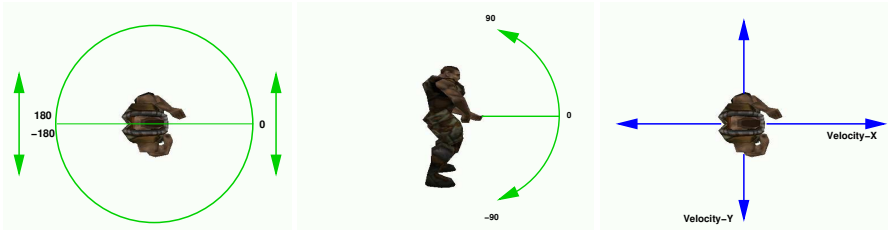


Fig. 2. A virtual character's movement depends on its field of view (fov) which is determined by the signed angles YAW and PITCH and on the character's velocity $\underline{\mathbf{v}} = (v_x, v_y)$.

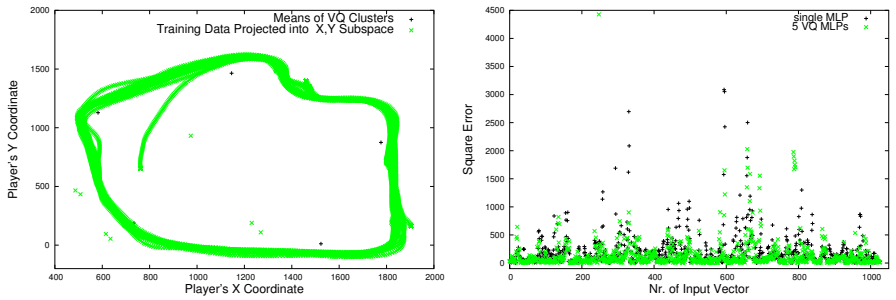
As there are millions of demos available on the Internet, the idea was to try to use this data to train neural networks which map human-made state vectors to human-made (re)action vectors. Such networks would realize artificial opponents (called *bots* in gamers terminology) which will imitate human players and thus certainly will behave human-like. In order to verify this idea, we experimented with ID's game QUAKE II, which was chosen for the following reasons:

1. Its C++ source code is freely available on the Internet [6].
2. Its network protocol is simple so that it is easy to extract game state data from recorded demos.
3. Though no longer state of the art in computer graphics, it is still popular and demo resources are nearly unlimited.

4 Experiments

In a first series of experiments, we tried to learn moving and aiming behaviors from recorded demos. In doing so, we assumed the state of a QUAKE II character to be comprehensively described by its position $\underline{\mathbf{x}} \in \mathbb{R}^3$ on a map, its distance $d \in \mathbb{R}$ to the nearest opponent, and the horizontal angle φ and the vertical angle ϑ to this opponent. As QUAKE II encodes angles such that $\varphi, \vartheta \in [0°, 180°]$ both require a signum $\sigma(\varphi), \sigma(\vartheta) \in \{-1, 1\}$ in order to cover the whole angular range. Consequently, in our experiments, a player's state was represented by an 8 dimensional vector $\underline{\mathbf{S}}$.

Since we only considered movement and aiming, we assumed a player's action to be determined by adjustments of its field of view (fov) and its velocity (s. Fig.2). In the experiments reported below, we thus realized virtual opponents by means of at least two neural networks where one was specialized in fov adjustment and the other was responsible for velocity adapting. Correspondingly, the overall reaction of our bots was composed from the output of these expert networks. As QUAKE II represents the fov by means of the angles YAW $\in [0°, 180°]$ and PITCH $\in [0°, 90°]$ with $\sigma(\text{YAW}), \sigma(\text{PITCH}) \in \{-1, 1\}$ and since we assumed the velocity to be given by a vector $\underline{\mathbf{v}}$ where $v_x \in [-400, 400]$ and $v_y \in [-200, 200]$, the networks for adjusting fov and velocity thus map 8 state parameters onto 4 or 2 action parameters, respectively.



(a) (b)

Fig. 3. (a) Training data (projected into the (x, y) plane) and means of VQ cluster. **(b)** Differences between expected and calculated fov adaptions obtained on a test set of 1027 vectors. The plot compares the performance of a single MLP with that of a combination of 5 MLPs.

Given several training series $\underline{\mathbf{s}}_t^p, t \in \{1, 2, \ldots, N\}$ of recorded state vectors, the state parameters $\underline{\mathbf{S}}$ described above can immediately be extracted while the parameters $\underline{\mathbf{A}}^f$ and $\underline{\mathbf{A}}^v$ for corresponding fov and velocity adjustments result from the difference $\underline{\mathbf{s}}_{t+1}^p - \underline{\mathbf{s}}_t^p$. The goal is thus to minimize the square errors

$$E_x = \frac{1}{2} \sum_{t=1}^{N-1} \left\| \hat{\underline{\mathbf{A}}}_t^x(\underline{\mathbf{S}}_t) - \underline{\mathbf{A}}_t^x(\underline{\mathbf{S}}_t) \right\|^2, \quad x \in \{f, v\} \tag{3}$$

between automatically generated adjustments $\hat{\underline{\mathbf{A}}}_t^x$ and known ones. As the next paragraphs indicate, for basic behavior learning tasks this can successfully be done by different combinations of simple multilayer perceptrons (MLPs) based on Levenberg-Marquardt training[2].

Learning Efficient Paths: Experienced FPS players do not move arbitrarily through a game environment. They have learned to cycle a map such that the effort for collecting items is minimal. In a first experiment, we investigated if such paths can be learned from training data. To this end we recorded a demo containing a total of 2064 state vectors which show five efficient runs around a map. Figure 3(a) shows a projection of the corresponding 8 dimensional data into the subspace of the player's (x, y) coordinates.

Integrating MLPs for fov and velocity adjustments into Quake II, i.e. implementing a virtual character, allowed to literally observe their performance. This revealed that a bot based on monolithic MLPs tended to become stuck at certain parts of the map. A solution was provided by clustering the training data using vector quantization. Figure 3(a) also shows the means of clusters that resulted after the number of partitions was chosen to be 5. After MLPs with 12 hidden neurons were trained for each of the resulting clusters, we could observe the bot smoothly pursuing the path encoded in the

[2] Details on the choice of network topologies discussed in the following can be found in [10].

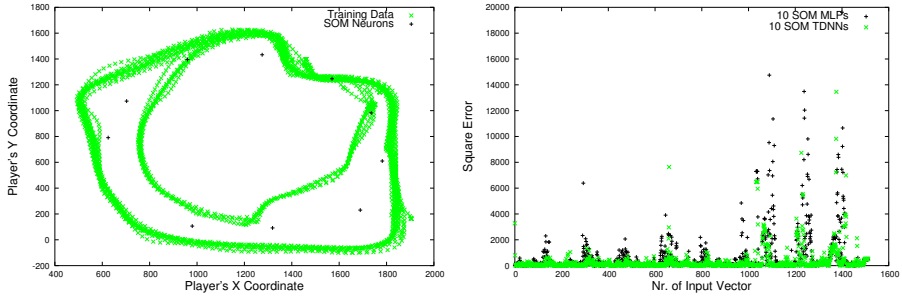
Experiment	Classifier		Training			Test	
	Task	Network Topology	Epochs	N_{Train}	E_{Train}	N_{Test}	E_{Test}
single path run	adapting fov	single 8-24-4 MLP	150	2064	35.29	1027	190.76
	adapting velocity	single 8-24-2 MLP	150	2064	2455.95	1027	2957.5
	adapting fov	5 VQ 8-12-4 MLPs	150	2064	15.1	1027	140.87
	adapting velocity	5 VQ 8-12-2 MLPs	150	2064	2453.82	1027	2788.75
crossing paths run	adapting fov	10 SOM 8-24-4 MLPs	150	2840	119.22	1513	644.1
	adapting velocity	10 SOM 8-24-2 MLPs	150	2840	3214.18	1513	7214.18
	adapting fov	10 SOM 24-12-4 TDNNs	150	2840	30.56	1513	313.17
	adapting velocity	10 SOM 24-12-2 TDNNs	150	2840	2411.85	1513	6032.1
integrated aim and movement behavior	adapting fov	6 SOM 8-12-4 MLPs	150	2361	18.42	1598	4585.5
	adapting velocity	6 SOM 8-12-2 MLPs	150	2361	1599.94	1598	2834.38
	adapting fov	6 SOM 5-12-4 MLPs	150	2361	15.5	1598	119.06
	adapting velocity	6 SOM 8-12-2 MLPs	150	2361	1599.94	1598	2834.38

Table 1. Summary of offline evaluation results.

training data. This is documented in Fig.3(b) and the upper part of Tab.1 which summarize results from an offline evaluation of this experiment. Both compare the performance of the different solutions on an independent test demo of 1027 state vectors.

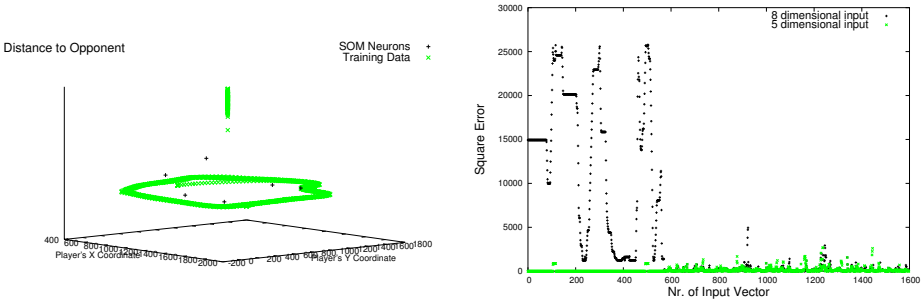
Learning to Run Crossed Paths: Already in realizing simple running behaviors, the assumption we made in Eq. (1), i.e. the idea that a reaction just depends on the current state, proves to be insufficient. Rather, knowledge of temporal context, e.g. knowing one's last position, can be crucial for intelligent movement behavior. This becomes apparent if we consider Fig.4(a) which superimposes data from two demos. Both demos contain cyclic runs around a map which overlap to a certain extend but happened to be directed contrary. Thus, without knowing where it was last a bot that has to move in the overlapping section of the paths would be lost for it simply would randomly choose between velocity adjustments learned from the one or the other demo. This assertion is backed up empirically in Fig.4(b) and the middle rows of Tab.1. Both compare results obtained with combinations of regular MLPs with results from combined time delay neural networks (TDNNs). In these experiments, the training data was separated into 10 clusters using self organizing maps (SOMs). On the one hand, usual MLPs with 24 hidden neurons were trained to learn movement behaviors. On the other hand, we trained TDNNs which not only regarded the current state but also the last two states to derive fov and velocity adjustments. Consequently, the input layer of these networks is of dimension 24. But even if their hidden layers contain only 12 neurons, this architecture lowers the errors E_f and E_v considerably.

Learning to Switch between Movement and Aiming Behaviors: Up to now, we only described neural network architectures that imitate a single human-demonstrated behavior. Figure5(a) depicts a projection of a demo where another character came across the recording player who correspondingly slowed down and aimed at that opponent. What we can see are the (x, y, d) coordinates contained in the demo, i.e. the 8 dimensional state vectors $\underline{\mathbf{S}}$ are projected into the subspace describing the player's spatial



(a) (b)

Fig. 4. (a) Superimposed projections of crossed, contrary directed paths. **(b)** Corresponding performance in fov adjustment by hybridly coupled regular MLPs and hybridly combined TDNNs



(a) (b)

Fig. 5. (a) (x, y, d) space projection of a demo where another player came across. **(b)** Errors in fov adjustment obtained from a combination of SOM clustered MLPs for 8 dimensional input and from the same architecture applied to PCA reduced input.

coordinates as well as his distance to an opponent. An ingame evaluation yielded that a combination of only two MLPs for fov and velocity adjustment cannot really reproduce the switching between behaviors that was implicitly encoded in the training data. SOMs, in contrast, automatically unfold evenly into the space of training data and, in fact, Fig.5(a) shows that one out of 6 SOM neurons emerged as a representative of the situation where the opponent came close. Consequently, the MLPs attached to this neuron automatically assume the role of experts for aiming and the capability to switch between behaviors automatically emerges from input space clustering.

However, as we see in Fig.5(b), our accustomed technique of assigning MLPs which process 8 dimensional input to SOM neurons does not perform well either. The figure displays the error E_f in fov adjustment. While the first 600 state vectors in the analyzed test data correspond to a situation where an opponent was near, the remaining 998

vectors represent usual player movements. Thus, as for the first phase of this demo the observed error oscillates considerably, the aiming behavior was not learned well. A solution resulted from reducing the input data dimension using PCA. Unsurprisingly, this yielded that mainly field of view parameters are responsible for aiming. And in fact, as the second plot in Fig.5(b) as well as the lower most rows of Tab.1 underline, reducing the input to the fov parameters only improved the quality of aiming.

5 Conclusion and Future Work

Even though computer games have become an enormous business that exhausts considerable intellectual efforts, current commercial realizations of game characters (also called *bots*) are far behind the scientific state of the art. In this contribution, we promoted the idea of understanding bot programming as a learning task. And indeed, first shot experiments with several neural network architectures indicate that it is possible to realize bots which behave human-like simply because they learned from human-generated training data.

Currently, we extend our approaches to more complex behavior and to other games. In doing so, we also investigate online learning for game characters as well as more sophisticated neural network architectures like mixtures of experts. Furthermore, we plan to examine data mining techniques in order to determine which information is relevant to generate and switch between appropriate behaviors. Finally, besides its long-term commercial impact, we believe that the game domain also provides interesting impulses for behavior learning in general. A mid-term vision, for instance, is a robocup-like league where bots from different research groups compete in order to foster the development of intelligently behaving game characters.

References

1. R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.
2. S. Cass. Mind games. *IEEE Spectrum*, pages 40–44, December 2002.
3. K. R. Dixon, R. J. Malak, and P. K. Khosla. Incorporating Prior Knowledge and Previously Learned Information into Reinforcement Learning Agents. Technical report, CMU, 2000.
4. C. Fairclough, M. Fagan, B. MacNamee, and P. Cunningham. Research Directions for AI in Computer Games. Technical report, Trinity College Dublin, 2001.
5. A. Galata, N. Johnson, and D. Hogg. Learning Variable-Length Markov Models of Behaviour. *Computer Visiosn and Image Understanding*, 81(3):398–413, 2001.
6. http://www.idsoftware.com/business/home/techdownloads/.
7. T. Jebara and A. Pentland. Action reaction learning: Automatic visual analysis and synthesis of interactive behaviour. In *Proc. 1st Int. Conf. on Computer Vision Systems*, volume 1542 of *Lecture Notes in Computer Science*, pages 273–292, 1999.
8. J. E. Laird. Using a Computer Game to develop advanced AI. *IEEE Computer*, pages 70–75, July 2001.
9. J. E. Laird and M. v. Lent. Interactice Computer Games: Human-Level AI's Killer Application. In *Proc. AAAI*, pages 1171–1178, 2000.
10. C. Thurau. Untersuchung über Lernverfahren für künstliche Agenten in virtuellen 3D-Umgebungen. Master's thesis, Bielefeld Universtiy, March 2003.
11. J.M.P. van Vaveren. The Quake III Arena Bot. Master's thesis, TU Delft, June 2001.