Proceedings of the SAB'06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games

(http://www.mip.sdu.dk/~georgios/Proceedings)

Georgios N. Yannakakis and John Hallam (Eds.)

Rome, Italy
1 October, 2006

Workshop Committee

Georgios N. Yannakakis and John Hallam. (Eds.) (2006). *Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*: *Proceedings of the SAB'06 Workshop* (Technical Report TR-2006-2). Maersk Institute for Production Technology, University of Southern Denmark, Odense, Denmark.

Preface

These proceedings contain the papers presented at the Workshop on *Adaptive approaches for Optimizing Player Satisfaction in Computer and Physical Games* held at the Ninth international conference on the Simulation of Adaptive Behavior (SAB'06): From Animals to Animats 9 in Rome, Italy on 1 October 2006.

We were motivated by the current state-of-the-art in intelligent game design using adaptive approaches. Artificial Intelligence (AI) techniques are mainly focused on generating human-like and intelligent character behaviors. Meanwhile there is generally little further analysis of whether these behaviors contribute to the satisfaction of the player. The implicit hypothesis motivating this research is that intelligent opponent behaviors enable the player to gain more satisfaction from the game. This hypothesis may well be true; however, since no notion of entertainment or enjoyment is explicitly defined, there is therefore little evidence that a specific character behavior generates enjoyable games.

Our objective for holding this workshop was to encourage the study, development, integration, and evaluation of adaptive methodologies based on richer forms of human-machine interaction for augmenting gameplay experiences for the player. We wanted to encourage a dialogue among researchers in AI, human-computer interaction and psychology disciplines who investigate dissimilar methodologies for improving gameplay experiences. We expected that this workshop would yield an understanding of state-of-the-art approaches for capturing and augmenting player satisfaction in interactive systems such as computer games.

Our invited speaker was Hakon Steinø, Technical Producer of IO-Interactive, who discussed applied AI research at IO-Interactive, portrayed the future trends of AI in computer game industry and debated the use of academic-oriented methodologies for augmenting player satisfaction. The sessions of presentations and discussions where classified into three themes: Adaptive Learning, Examples of Adaptive Games and Player Modeling.

The Workshop Committee did a great job in providing suggestions and informative reviews for the submissions; thank you! This workshop was in part supported by the Danish National Research Council (project no: 274-05-0511). Finally, thanks to all the participants; we hope you found this to be useful!

Georgios N. Yannakakis and John Hallam
Rome, Italy
1 October, 2006

Table of Contents

Utilization of Evolutionary Algorithms to Increase Excitement of the COMMONS Game

Norio Baba[1] and Hisahi Handa[2]

[1] Department of Information Science, Osaka Educational University
Kashihara 4-698-1, Osaka Prefecture, JAPAN
baba@cc.osaka-kyoiku.ac.jp
http://www.is.osaka-kyoiku.ac.jp/lab/baba/baba-e.html
[2] Graduate School of Natural Science and Technology, Okayama University
Tsushima-Naka 3-1-1, Okayama 700-8530, JAPAN
handa@sdc.it.okayama-u.ac.jp
http://www.sdc.it.okayama-u.ac.jp/~handa

Abstract. In this paper, we suggest that Evolutionary Algorithms could be utilized in order to let the COMMONS GAME, one of the popular environmental games, become much more exciting. In order to attain this objective, we utilize Multi-Objective Evolutionary Algorithms to generate various skilled players. Further, we suggest that Evolutionary Programming could be utilized to find out an appropriate point of each card at the COMMONS GAME. Several game playings utilizing the new rule of the COMMONS GAME derived confirm the effectiveness of our approach.

1 Introduction

Gaming is regarded by many people as a new and promising tool to deal with complex problems in which human decisions have far reaching effects on others. It has been used for various purposes such as decision-making, education, training, research, entertainment, etc [1]-[7]. In recent years, various approaches concerning the applications of Evolutionary Algorithms to the field of games have been proposed[8]-[12].

In this paper, we suggest that EAs could be utilized for making the COMMONS GAME [4], one of the popular environmental games, become much more exciting. In particular, in order to attain this objective, we shall try to utilize Evolutionary Algorithms in the following steps: (1) First, we shall consider a new rule for assigning a point to each colored card in the COMMONS GAME which takes the environmental changes into account. (2) Second, we shall utilize Multi-Objective Evolutionary Algorithms (MOEA) [13][14] to generate various skilled players whose choice of each card is done in a timely fashion. (3) Further, we shall utilize Evolutionary Programming (EP) [15][16] to derive appropriate combinations of the rules (concerning the point of each card) which could be used to help players fully enjoy game playing. This paper is organized as follows. In section 2, we shall introduce the original COMMONS GAME briefly. In

section 3, we shall touch upon several problems involved in the original COMMONS GAME. We shall suggest that EAs could be utilized in order to let game playing become much more exciting. We shall also show several results of game playing (utilizing the new rule derived by MOEA & FEP) which confirm the effectiveness of our approach. This paper concludes with discussions concerning the contributions of this paper and future perspectives.

2 COMMONS GAME

The COMMONS GAME was developed by Powers *et al.* in 1980 [4]. Since we live in a world having only finite natural resources such as fishes and forests (commons), it is wise to consider their careful utilization. The COMMONS GAME may be quite helpful in stimulating discussion of this problem.

In the following, we give a brief introduction to this game. Six players are asked to sit around a table. Following a brief introduction of the game, the game director tells the players that their objective is to maximize their own gains by choosing one card among the five colored (Green, Red, Black, Orange, Yellow) cards in each round. In each round, players hide their cards behind a cardboard shield to ensure individual privacy. Each colored card has its own special meaning concerning the attitude toward the environmental protection, and has the following effect upon the total gains of each player:

Green card: A green card represents high exploitation of the commons: players who play a green card can get a maximum reward. However, they loses 20 points if one of the other players plays a black card in the same round.

Red card: A red card indicates a careful utilization of the commons: Red cards are only worth about forty percent as many points as green cards.

Black card: This card has a punishing effect on the players with green cards: Players who have played a black card have to lose 6 points divided by the number of black cards played at the same round, but are able to punish green card players by giving them -20 points.

Orange card: Orange cards give an encouraging effect to red card players: Players who have played this card have to lose 6 points divided by the number of orange cards played at the same round but are able to add 10 points to red card players.

Yellow card: A yellow card denotes a complete abstention from utilization of the commons: Players who play this card get 6 points.

Depending upon the players' strategies, the state of the commons change: If players are too cager to exploit the commons, then deterioration of the commons occurs. Players have a matrix flip chart on the board representing the payoffs for the red and green cards under different conditions of the commons. Each game ends after 50 rounds. After each 8^{th} round, players have a three-minute conference. They can discuss everything about the game and decide every possible way to play in future rounds.

Table 1. Points which can be gained by the Red and the Green card players

State: -8		State: -1		State: 0		State:1		State:8	
R	G	R	G	R	G	R	G	R	G
——	0	——	70	——	100	——	130	——	200
-10	2	25	72	40	102	55	132	90	202
-8	4	27	74	42	104	57	134	92	204
-6	6	29	76	44	106	59	136	94	206
-4	8	31	78	46	108	61	138	96	208
-2	10	33	80	48	110	63	140	98	210
0	——	35	——	50	——	65	——	100	——

Remark 2.1: In each round, each player can choose one of the 5 cards. However, COMMONS GAME is quite different from traditional "card game" in which cards are distributed randomly to the players and losing cards is frequently occurred. In this game, each player can choose any card in each round in order to represent his (or her) attitude toward the environmental protection.

Remark 2.2: Red players usually can only get about 40 % of the returns that green card players receive (assuming that no players have chosen the black card at the same round). In the original COMMONS GAME, there are 17 main environmental states $(-8, -7, \ldots, -1, 0, +1, \ldots, +8)$. (Each main environmental state (except 0 state) has 10 subordinate states (the 0 state has 21 subordinate states)). Initial state of the COMMONS GAME is 0. The state of the commons changes, depending upon the players' strategies. If players are eager to exploit the commons (many players often use Green cards), the deterioration of the commons occurs and the state of the commons changes to a minus state such as $-1, -2$, and etc. When the deterioration of the commons has occurred, then the returns that green players and red players receive decrease. Table 1 shows the returns that green players can get when no players have chosen the black card and also shows the returns that red players can receive. The second table from the left shows that green players can receive 70 point (when the state of the environment is -1 and no players have chosen the red cards) which is only the 70 % of the returns which can be obtained at the initial state 0. The points that red card players receive also decrease heavily when the environmental state becomes minus. In the -1 state, red players can get only about 70 % of the returns that they could receive in the initial state 0. (On the other hand,) If almost all of the players consider seriously about the commons and execute wise utilization of the commons, then environmental state ameliorates (state of the environmental becomes positive) and the returns that green players and red players receive increase as shown in Table 1.

Remark 2.3: Utilization of a green card also incurs degradation of the subordinate state. Although natural amelioration of the commons occurs 8 times during 50 rounds, too much exploitation of the commons (that is to say, too much use of the green card) causes serious degradation of the environmental state (One green card corresponds to one degradation of the subordinate state).

Remark 2.4: Each row in the five tables in Table 1 corresponds numbers of the red cards chosen. For an example, let us consider that case that the current main state is $+1$, and numbers of the red and green card players are 5 and 1,

respectively. Then, each red card player can get point 63 which corresponds to the point written in the 6^{th} row and the 1^{st} column of the table concerning the state +1. The green card player can get point 140.

3 Evolutionary Algorithms for Making Game Playing Much More Exciting

We have so far enjoyed a large number of playings of the original COMMONS GAME. Although those experiences have given us a valuable chance to consider seriously about the commons, we did find that some players lost interest, in the middle of the game, because the COMMONS GAME is comparatively monotonous. In order to make the game much more exciting [17]-[22], we have tried to find the reason why some players lost interest in the middle of the COMMONS GAME. We have come to the conclusion that the way that each player receives points when he chooses one of the five cards sometimes makes the game playing rather monotonous.

In particular, we have concluded that the following rule make the game playing monotonous: In the original COMMONS GAME, green card players receive a penalty, -20 points, when some player chooses a black card. On the other hand, black card players receive a point -6/(numbers of players who have chosen a black card). Orange card players receive a point -6/(numbers of players who have chosen an orange card).

We consider that some change in the points -20 and -6 mentioned above would make the COMMONS GAME much more exciting. In order to find an appropriate point for each card, we shall try to utilize EP.

In section 3.1, we suggest that Multi-Objective Evolutionary Algorithms (MOEA) [13][14] can generate various kinds of Neural Network Players with different strategies. In section 3.2, we show that Evolutionary Programming can be a useful tool for finding appropriate points of the cards in the COMMONS game.

3.1 MOEAs for generating intelligent players

Multi-Objective optimization is one of the most promising fields in Evolutionary Algorithms research. Due to the population search of EAs, Multi-Objective Evolutionary Algorithms (MOEAs) can evolve candidates of Pareto optimal solutions. Hence, in comparison with conventional EAs, MOEAs can simultaneously find out various solutions. In this paper, we employ NSGA-II, proposed by Deb *et al.*, to evolve game players with Neural Networks [13]. NSGA-II utilizing crowded tournament selection, the notion of archive, and ranking method with non-dominated sort, is one of the most famous MOEAs. Most recent studies proposing new MOEAs cite their paper[14].

Neural Network model Our objective is to simultaneously construct plenty of Neural Network players with different strategies. The neural network model

Table 2. Input variables for Neural Network players

1.	Difference between the total points of each player and the average
2.	Rank of each player
3.	States of the environment: main state & subordinate state
4.	Changes in the environment
5.	Round Number
6.	Weighted sum of each card having been chosen by all of the players
7.	Weighted sum of each card having been chosen by each player
8.	The number of each card chosen in the previous round
9.	The card chosen by each player in the previous round

adopted in this paper has 26 input units, one hidden layer with 30 units, and 5 output units.

In Table 2, input variables into this neural network model are given: In order to represent the current state of the commons, two inputs, consisting of main state and subordinate state, are prepared. For inputs 6.–9., 5 different inputs corresponding to each card are prepared.

The weights of the neural network model are evolved by MOEA. That is, the number of gene in an individual is 965 (the number of weights between input layer and hidden layer 26×30, the number of weights between hidden layer and outputlayer 30×5, and the number of thresholds 5).

Table 3. Efficiency of each card

Player's card	$E_i(C)$	Situations
R	$+1$	No black player, but some green players
	-1	Otherwise
B	$+1$	No green player
	-1	Otherwise
G	$+1$	Some black players
	-1	Otherwise

Fitness Evaluation In order to evaluate each individual, it is better to let him play with various game players as possible as we can. Fitness evaluation of individuals is carried out as follows: 1) Select 30 individuals randomly from the parent population at each generation, where the 30 selected individuals indicate opponents for 6 game runs (5 individuals are needed as opponents for a single game run). 2) The child population is generated from the parent population. 3) All of the individuals in the parent and child populations face the selected opponents. 4) As a consequence of game runs, individuals are evaluated with two objective functions[3] O_v and O_e: the variance of the total number of each card chosen in each game run, and the efficiency of played cards, respectively. The variance of the number of played card is calculated by following equation: $O_v = V_{RGB} + 20 * N_{OY}$, where V_{RGB} is the variance of the total number of Red, Green, and Black cards played in each game run, and N_{OY} is the total number

[3] According to the implementation of NSGA-II, the objective functions used here are to be minimized.

of Orange and Yellow cards chosen. On the other hand, the efficiency of the cards used is calculated by accumulating the evaluations at each play, which are summarized in Table 3: $O_e = \sum_{i=1}^{50} E_i(C)$.

Experimental Results Fig. 1 depicts the individual distributions at the initial and final generations. In the individual distributions at the final generation, a Pareto set is constructed. Since fitness measurement in this paper is relative one, i.e., opponents are randomly chosen at every generation, some initial individuals in which the efficiency of the use of cards is equal to -48 and variance is 400, seem to indicate better performance. However, it is just caused by games with naive players. In fact, the final individuals have become much more sophisticated compared with all of the initial individuals.

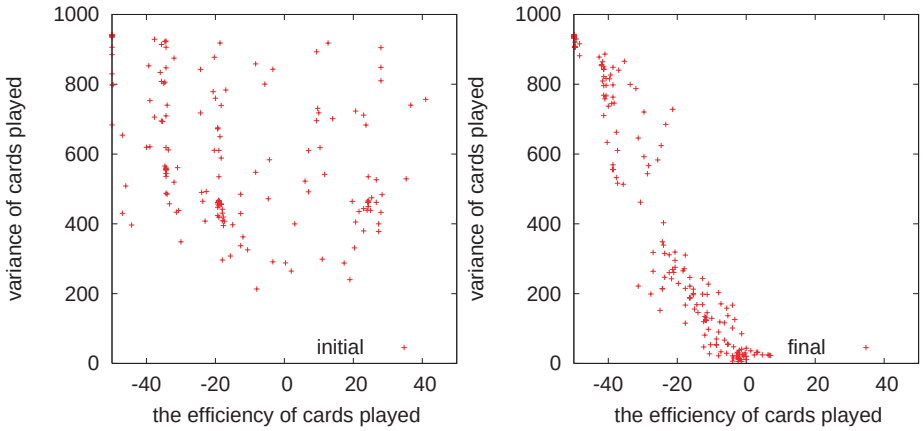


Fig. 1. Individual distributions at the initial (LEFT) and final (RIGHT) generations

3.2 Fast Evolutionary Programming to Design Appropriate Game Rules

Fast Evolutionary Programming Fast Evolutionary Programming (FEP) proposed by Xin *et al.* [16] is used in this paper because the Fast Evolutionary Programming is easy to implement and performs well due to the Cauchy distribution mutation. Individuals in Evolutionary Programming are composed of a pair of real valued vectors (X, η), where X and η indicate design variables in given problems and variance parameter used in self-adaptive mutation [15].

Utilization of FEP for Constructing a New Rule of the COMMONS GAME In this paper, we employ three variables, W_G, A, and W_o to represent new rules. The meaning of them is described as follows:

1. Penalty P_G for green players – we shall propose an appropriate way for penalizing green players which takes the environmental changes into account:

$P_G = -W_G \times$ (Gain G), where W_G means the numerical value that is determined by the Evolutionary Programming, and "Gain G" denotes the return that the green players can get if any other player does not choose "black Card."

2. Point AOB that black players loose – we shall propose an appropriate way (for asking black players pay cost in trying to penalize green players) which takes the environmental changes into account: $AOB = OB/NOB(OB = -A \times (GainR))$, where A means the numerical value that is determined by the Evolutionary Programming, and NOB means the number of players who have chosen the black cards, and "$GainR$ denotes the return that the red player can get.
3. Point OR that orange players add to the red players – we shall propose an appropriate way (for helping red players maintain the commons) which takes the environmental changes into account: $OR = W_o \times$ (Gain R), where W_o means the numerical value that is determined by the Evolutionary Programming.

Fitness Evaluation In order to evaluate the rule parameters mentioned in the previous subsection, 200 games per an individual are carried out. Before runs of Evolutionary Programming, six neural network players in the final population of MOEA are randomly chosen per game. Namely, 1200 neural network players are selected (including duplicated selection).

After each game, a rule parameter is evaluated by the following:

a) A game in which black cards & orange cards are seldom chosen (almost all of the players only choose a green card or a red card) is quite monotonous. Therefore, the more black (or orange) cards are chosen in a game, the higher value of the evaluation function should be given.

b) A game in which the ranking of each player often changes is exciting. Therefore, the more changes of the top player during a game run, the higher evaluation value should be given. A game in which there is a small difference between the total points of the top player and those of the last player is very exciting. Therefore, the small variance in the total points of each player at the end of a game, the higher the evaluation value should be given.

c) When the environmental deterioration had occurred heavily, each player can receive only a small amount of return. Under such a state of environment, the game should become monotonous. Therefore, a small evaluation value should be given if a game has brought heavy deterioration to the environment. On the other hand, a high evaluation value should be given when a game has brought a moderate final state of the environment.

By taking into account the above points, we have constructed the following evaluation function $T(x)$:

$$T(x) = f(x) + g(x) + h(x) + \alpha(x) + \beta(x),$$

where x denotes a chromosome. The function values of $f(x)$, $g(x)$, $h(x)$, $\alpha(x)$, and $\beta(x)$, correspond to the followings:

$f(x)$**:** The environmental state at the end of the game;
$g(x)$**:** The total numbers of the black card having been chosen;
$h(x)$**:** The total numbers of the orange card having been chosen;
$\alpha(x)$**:** The sum of the variance of the points having been gained by each player;
$\beta(x)$**:** The total numbers of the changes of the top player.

Due to limitation of space, we don't go into details concerning the above 5 functions. Interested readers are kindly asked to attend our presentation.

3.3 New Rules Obtained by MOEA and FEP

Evolutionary Programming with 10 population size has been carried out for 50 generations. The changes of averaged fitness value during the evolution are depicted in Fig. 2. This graph is plotted by averaging over 20 runs. The changes over the whole generations are not so much. However, the main contribution of these changes are caused by the changes of $\beta(x)$. This means that the utilization of EP has contributed a lot in changing top players. The reason why other sub-functions, such as $f(x)$, $g(x)$ and so on, did not affect the evolution process is that the neural network players are already sophisticated enough: They play various kinds of cards, including black and orange cards so that the final environmental state has not been deteriorated so seriously.

We analyzed the best individuals in 20 runs. there are two groups in the best individuals: they are around $W_G = 0.27$, $A = 0.04$, and $W_o = 0.12$, and $W_G = 0.29$, $A = 0.2$, and $W_o = 0.1$, respectively. This analysis reveals that much penalization of green players causes the frequent changes of the top players.

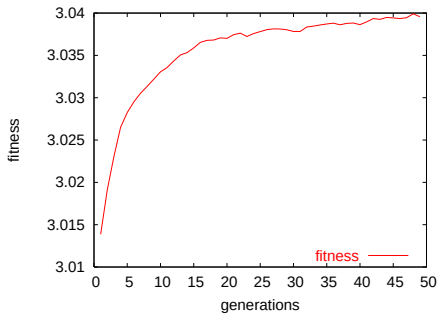


Fig. 2. Changes of averaged fitness value during evolution

3.4 Game Playings Utilizing the New Rules

In order to investigate whether the new rules having been obtained by MOEA and FEP are appropriate or not, one of the authors asked his 12 students (who have experienced several game playings of the original COMMONS GAME) to play the game according to the new rules constructed using the results by MOEA and FEP. The authors watched the attitudes of the students who participated

in the new game[4]. They felt that almost all of the students concentrated more on the game then before. After the game, they asked the students for their impressions of the new game.

Answers from the 12 students can be summarized as follows:

(1) 11 students out of 12 expressed their opinions that the new game is far more exciting than the original game.

(2) Some of the 11 students (who expressed positive impression toward the new game) explained the reason why they have felt the new game exciting: **(2-1)** In the new game, players can easily choose the Black Card. Therefore, all of the players can enjoy dynamic game playing. **(2-2)** In the original COMMONS GAME, penalty point to the green card player by a black card is always -20. However, in the new game, its point depends on the environmental state. When the environmental state is 0 or +1, damage due to the use of green card is heavy. This causes the new game rather exciting.

(3) One student (who has shown us negative impression on the new game) explained the reason why he prefers the original game to the new game: Players (enjoying the game playing of the original game) should be very careful in using Black card since its utilization causes them considerable minus point. However, a black card gives heavy minus point to the green card players wherever the state of the environment is. This makes a good thrill to him.

4 Conclusions

In this paper, we have tried to utilize two kinds of EAs, i.e., MOEA and EP for making the COMMONS GAME exciting. The MOEA has been used for generating various types of skilled players. Further, the EP has been introduced to find out appropriate combinations of the point of each card. As shown in the Fig. 2, we have succeeded in finding highly advanced rules compared with that of the original COMMONS GAME. Several game playings of the COMMONS GAME (using the new rule (derived by using MOEA and FEP)) by our students suggest the effectiveness of our approach. However, this has been suggested only by several game done by our students. The future research is needed to carry out lots of game playings by various people for the full confirmation of our approach.

Acknowledgement

The authors would like to thank the anonymous reviewers for their kind (and important) suggestions and comments. They also would like to express their thanks to the Foundation for Fusion of Science & Technology (FOST) and the Grant-In-Aid for Scientific Research (C) by Ministry of Education, Science, Sports, and Culture, Japan, who have given them partial financial support.

[4] The new game with the parameter values $W_g = 0.27$, $A = 0.04$, and $W_o = 0.12$ has been played.

References

1. Duke, R. D.: Gaming: The Future's Language, Sage Publications (1974)
2. Shubik, M.: Games for Society, Business and War, Elsevier (1975)
3. Duke, R.D. and Duke, K.: Development of the Courail Game, Operational Gaming: An International Approach. Stahl, I. Editor, IIASA, (1983) 245–252
4. Powers, R.: The Commons Game. Instruction Booklet (1980)
5. Ausubel, J. H.: The Greenhouse Effect: An Educational Board Game. Instruction Booklet, IIASA (1981)
6. Baba, N., *et al.*: A Gaming Approach to the Acid Rain Problem. Simulation & Games **15** (1984) 305–314
7. Baba, N.: The commons game by microcomputer. Simulation & Games, **15** (1984) 487–492
8. Fogel, D.B.: Evolving behaviors in the iterated prisoner's dilemma. Evolutionary Computation, **1(1)** (1993) 77–97
9. Chong, S.Y., Yao, X.: Behavioral diversity, choices and noise in the iterated prisoner's dilemma. IEEE Trans. Evolutionary Computation **9(5)** (2005) 540–551
10. Barone, L., While, L.: Adaptive learning for poker. Proceedings of the Genetic and Evolutionary Computation Conference (2000) 560–573
11. Moriarty, D. and Miikkulainen, R.: Discovering complex othello strategies through evolutionary neural networks. Connection Science **7(3)** (1995) 195– 209
12. Lucas, S.M., Kendall, G.: Evolutionary computation and games. IEEE Computational Intelligence Magazine **1(1)** (2006) 10–18
13. Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T.: A Fast and Elitist multiobjective Genetic Algorithm: NSGA-II. IEEE Trans. Evolutionary Computation **6(2)** (2002) 182–197
14. Coello Coello, C.A.: Evolutionary multi-objective optimization: a historical view of the field. IEEE Computational Intelligence Magazine **1(1)** (2006) 28–36
15. Bäck, T., and Schwefel, H.-P.: An overview of evolutionary algorithms for parameter optimization. Evolutionary Computation, **1(1)** (1993) 1–23
16. Yao, X., Liu, Y., and Lin, G.: Evolutionary programming made faster. IEEE Trans. Evolutionary Computation **3(2)** (1999) 82–102
17. Iida, H., Takeshita, N., and Yoshimura, J.: A Metric for Entertainment of Boardgames: Its Implication for Evolution of Chess Variants. Entertainment Computing: Technologies and Applications, pp. 65-72 (2002)
18. Rauterberg M.: About a framework for information and information processing of learning systems. E. Falkenberg, W. Hesse, A. Olive (eds.), Information System Concepts - Towards a consolidation of views, pp. 54-69 (1995)
19. Tesauro, G.: Temporal Difference Learning and TD-Gammon. Communications of The ACM, **38** (1995) 58p-68
20. Baba, N.: An Application of Artificial Neural Network to Gaming. Proc. SPIE Conference, Invited Paper, **(2492)** (1995) 465–476
21. Baba, N., Kita, T., and Takagawara, Y.: Computer Simulation Gaming Systems Utilizing Neural Networks & Genetic Algorithms. Proc. SPIE Conference, Invited Paper, **2760** (1996) 495–505
22. Baba, N.: Application of Neural Networks to Computer Gaming. Soft Computing in Systems and Control Technology, Tzafestas, S.G. Editor, World Scientific Publishing Company, **3** (1999) 379–396

Smarter Teammates – Applying Hidden Markov Models in Sports Games

Christian Thurau[1] and Christian Bauckhage[2]

[1] Bielefeld University, Applied Computer Science
33501 Bielefeld, Germany
http://www.techfak.uni-bielefeld.de/ags/ai/
[2] Deutsche Telekom AG, Laboratories
10587 Berlin, Germany
http://www.telekom.de/laboratories

Abstract. This paper considers statistical machine learning as an approach to realizing artificial game characters that act more human-like, especially for games where intelligent teamplay is of primary concern. Focusing on a multiplayer soccer game, we propose analyzing data contained in the network traffic of game sessions. Given these data, we apply Hidden Markov Models (HMMs) for behavior recognition in the virtual game world. First results indicate that this technique which already has numerous applications in temporal pattern recognition may also provide an avenue towards smarter artificial teammates.

1 Introduction

Compared to some 30 years ago, computer game technology and design have evolved considerably. However, although modern games dispose of highly realistic graphics, sophisticated physics simulation and absorbing story lines, convincing artificial game agents (often called *gamebots*) are still rare. As of today, behavior programming (also known as *game AI*) mainly relies on conceptually simple techniques. First and foremost, the ideas of scripted behavior or of behavior selection governed by finite state machines do prevail. This is despite the fact that it almost has become a common sense notion that they fail to create an impression of life-likeness [1, 2].

This paper describes an alternative, data-driven approach to behavior programming. Dealing with the genre of sports games, we strive to create life-like acting artificial team members which support the avatars controlled by human players. Since sophisticated teamplay depends on the current spatio-temporal context, it is mandatory for such gamebots to recognize and to react to teammate- and opponent behavior. Our idea is to apply statistical machine learning in order to *learn* these capabilities from examples. Due to the popularity of multiplayer network gaming (see Fig. 1), this is actually straightforward: game data transmitted over local or wide area networks completely describe the situations the players encounter in the game world. Since the data also also shows how the players react to these situations, recordings of game network traffic contain comprehensive and unobstructed representations of situated human behavior. On the internet, corresponding recordings abound! For pattern recognition and machine learning, this poses the challenge of acquiring, classifying and synthesizing gamebot

Fig. 1. Gamers at a LAN party.

Fig. 2. Screenshots from Tao of Soccer (left) and PRO EVOLUTION SOCCER® (right).

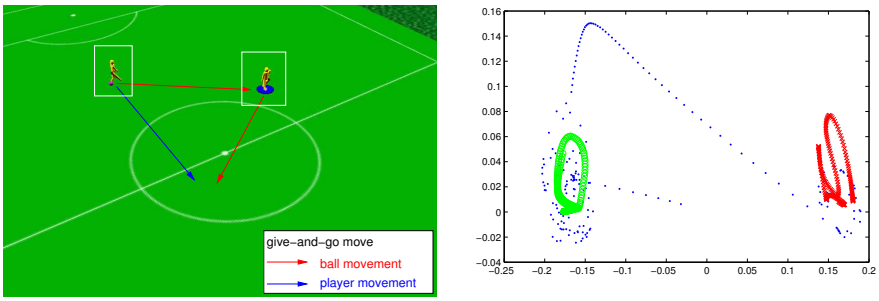
behavior from this wealth of data. The difficulty is that modern games take place in complex, highly realistic virtual environments (see Fig. 2) and require strategic and tactical thinking.

For our purpose, we consider recordings of human teamplay in the game "Tao of Soccer"[3] (see Fig. 2). Tailored to the problem of developing artificial teammates, we propose a two-step approach. In the next section, we describe a transformation of game data into a rotation and translation invariant representation. Given these canonical reference frames, we apply Hidden Markov Models (HMMs) for team behavior classification. Section 3 briefly summarizes the underlying theory and section 4 presents results we obtained from this approach. A conclusion and an outlook will end this contribution.

2 Features for Behavior Classification

For the remainder of this paper, we focus on tactical interactions between teammates in "Tao of Soccer". Player positions $\boldsymbol{p_i}$, teammate positions $\boldsymbol{t}_i$ and ball positions $\boldsymbol{b}_i$ are expressed in (x, y) coordinates extracted from network traffic. A rotation and transla-

[3] `http://soccer.sourceforge.net`



(a) (b)

Fig. 3. 3(a) Visualization of give-and-go behavior in Tao of Soccer; 3(b) corresponding bird's view of this behavior where player and ball positions are plotted in canonical coordinates.

tion invariant representation of these data is required so that general team behaviors, i.e. behaviors that are not bound to certain field-positions, can be considered.

Extending an idea originally proposed in the context of tactical combat games [3], we apply principal component analysis (PCA) to sequences of player- and of ball positions and therefore obtain a canonical reference frame. With i denoting the current *game-tic* and f indicating temporal context (i.e. the length of sequences of observations), we compute the centroid

$$\boldsymbol{\mu}_i = \frac{1}{3}\big(\boldsymbol{p}_i + \boldsymbol{t}_i + \boldsymbol{b}_i\big). \tag{1}$$

After normalization to zero mean $\tilde{\boldsymbol{p}}_j = \boldsymbol{p}_j - \boldsymbol{\mu}_j$, where $i - f \leq j \leq i$, player positions are stacked into sample matrices

$$\boldsymbol{P}_i = \big[\tilde{\boldsymbol{p}}_{i-f}, \tilde{\boldsymbol{p}}_{i-f+1}, \ldots, \tilde{\boldsymbol{p}}_i\big]^\mathsf{T}. \tag{2}$$

Together with corresponding matrices $\boldsymbol{T}_i$ and $\boldsymbol{B}_i$ for teammate and ball positions, this allows for computing covariance matrices

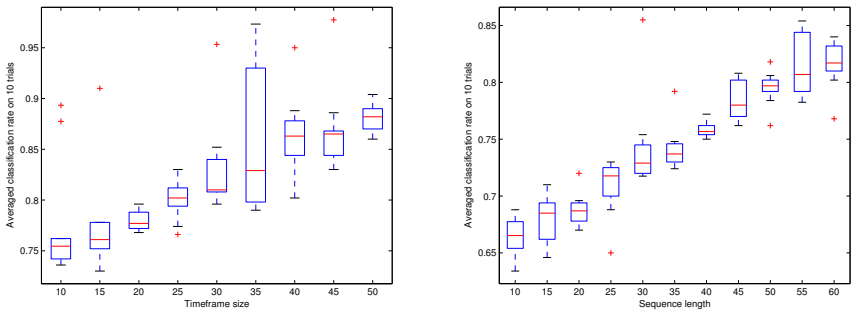
$$\boldsymbol{C}_i = \frac{1}{3f}\Big(\boldsymbol{P}_i^\mathsf{T}\boldsymbol{P}_i + \boldsymbol{T}_i^\mathsf{T}\boldsymbol{T}_i + \boldsymbol{B}_i^\mathsf{T}\boldsymbol{B}_i\Big) \tag{3}$$

Canonical player position vectors $\boldsymbol{p}_i^c$ then simply result from

$$\boldsymbol{p}_i^c = \boldsymbol{D}_i^\mathsf{T}(\boldsymbol{p}_i - \boldsymbol{\mu}_i) \tag{4}$$

where the matrix $\boldsymbol{D}_i$ consists of the eigenvectors of $\boldsymbol{C}_i$. Analogous transformations apply to the $\boldsymbol{t}_i$ and $\boldsymbol{b}_i$. Figure 3(b) plots an example of an in-game situation in canonical coordinates.

Obviously, the parameter f influences the transformation. While too high a value results in a mere rotation of the coordinates, too low a value leads to an unintended dimensionality reduction. Figure 4(a) shows how f impacts the classification of team



(a) (b)

Fig. 4. 4(a) size f of sliding window for canonical coordinate transformation vs. average classification rate over 10 test runs; 4(b) pattern sequence length h vs. average classification rate over 10 test runs.

behavior described below. It indicates that values between 20 and 25 yield good results and also allow for computations fast enough for live games (a value between 20 and 25 corresponds to 2 to 2.5 seconds of play).

3 Synopsis of Hidden Markov Models

Since they are well suited and well established for analyzing temporal patterns and because of their success in sports video annotation [4, 5], we opted to apply HMMs for our team behavior classification task.

A HMM is a statistical model used to represent Markov processes with unknown parameters. The challenge is to determine these hidden parameters from observable data. Once the parameters have been found, the model can recognize patterns similar to the ones used for training.

While for a regular Markov process over a set of states $S = \{s_1, s_2, \ldots, s_n\}$ the only parameters are the state transition probabilities

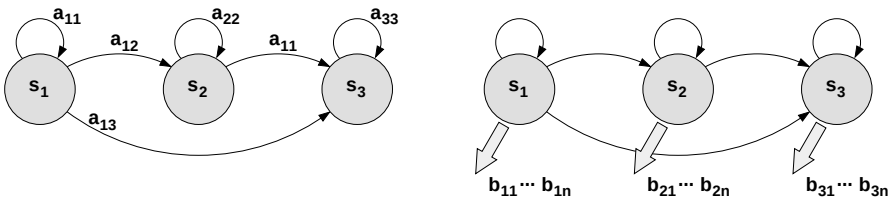
$$a_{ij} = P(s(t+1) = s_i | s(t) = s_j), \tag{5}$$

a Hidden Markov Model also generates outputs. For these, each state has a probability distribution

$$b_{ij} = P(o(t) = o_j | s(t) = s_i) \tag{6}$$

where $O = \{o_1, o_2, \ldots, o_M\}$ is a set of output tokens. Consequently, looking at a sequence of tokens generated by an HMM does not directly indicate the sequence of states.

In a supervised training phase with exemplary observation sequences, the *Baum-Welch algorithm* is applied to determine the most likely set of state transition and output



(a) transition probabilities (b) emission probabilities

Fig. 5. An example of a Hidden Markov Model (HMM) with three states s_1, s_2 and s_3. According to the state transition probabilities in 5(a), the model is of *left-to-right* topology. Therefor, it may encode temporal processes (such as human behavior) which consist of distinct phases. For instance, for a *give-and-go behavior* in soccer, state s_1 may model the phase where a player receives the ball, s_2 may represent the phase where he stops the ball, and s_3 may correspond to the phase where the ball is passed on. Since each of these phases may take some time, there are self-transitions for every state. Since sometimes the ball is not exactly stopped but immediately passed on, there is a direct transition between states s_1 and s_3. In practice, it is not necessary to encode world knowledge like this into the HMM topology. It suffices to fix the number of model states and to provide exemplary data for training. Given a random initialization of the model parameters, there are algorithms to determine suitable transition probabilities from the training data. This also holds for the emission probabilities highlighted in 5(b). The output generated by an HMM may be of symbolic or numeric nature. In our case, it is given by the canonical feature vectors described in section 2. Even though the set of possible outputs is the same for each state of the HMM, it is obvious that different phases of a process are characterized by different feature vectors. For the *give-and-go* behavior, for instance, the feature vectors which describe receiving the ball will differ from those describing the passing of the ball. Consequently, the probability b_{1i} that the HMM will emit a certain output token o_i in state s_1 will differ from the probability b_{3i} of emitting o_i in state s_3.

probabilities for the given examples. Given several trained models and a new sequence of observations, the *forward algorithm* is used to compute the different likelihoods of the sequence being produced by the different models. This obviously allows for classification: the newly observed sequence is usually classified to the model with the highest likelihood. Figures 5 shows an exemplary HMM and presents an example intended to help grasping the intuition behind this pattern recognition method. For a more detailed discussion of HMMs and the accompanying algorithms the reader may refer to Rabiner's classic text [6].

4 Results of Team Behavior Classification

In a series of initial experiments, we classified player observation data into one out of five behaviors which are typically encountered in a game of soccer:

Given-and-go: the ball is being passed back and forth between two moving players (see the example in Fig. 3(a)).

Dribbling: one player is moving with the ball (possibly along a zigzagged trajectory), another player is following.

Ball retrieval: one player runs towards the resting ball and immediately passes it to the other player.

Solo: the player with the ball does not pass the ball but constantly moves in one direction; the other player is running alongside.

Long pass: the ball is passed over a longer distance (w.r.t. the field size) but not returned to the passing player.

For each of these $k = 1, \ldots, 5$ behaviors, we trained a Hidden Markov Model

$$\lambda_k = (\boldsymbol{A}_k, \boldsymbol{B}_k, \boldsymbol{\pi}_k) \tag{7}$$

using prototypical, manually labeled game data which was generated by two human players. As discussed in the previous section, the matrices $\boldsymbol{A}_k = \{a_{ij}^k\}$ and $\boldsymbol{B}_k = \{b_{ij}^k\}$ contain the state transition- and emission probabilities, respectively; the vector $\boldsymbol{\pi}_k = [\pi_i^k]$ contains the initial state probabilities. The data used for training consists of sequences Ω_i of length h which are given by

$$\Omega_i = [o_{i-h}, o_{i-h+1}, \ldots, o_i] \tag{8}$$

where each $o_j = [\boldsymbol{p}_j^c, \boldsymbol{t}_j^c, \boldsymbol{b}_j^c] \in \mathbb{R}^6$ comprises canonical player-, teammate- and ball positions.

Since team behaviors must be classified fast enough to allow for online application, the constant h is constrained to cover a history of four to five seconds of play. Assuming a game-tic every 0.1 seconds, this corresponds to a maximum of $h = 45$. Figure 4(b) plots different sequence lengths h versus the average classification rate (the parameter f for canonical coordinate transformation was set to $f = 25$). Although longer sequences lead to better results, the empirical findings in this figure indicate that considering 45 game-tics provides a good compromise between applicability in live games and reasonably high classification rates.

Classification is done by selecting the behavior k corresponding to the HMM λ_k with the maximum log likelihood $\log P(\Omega_i|\lambda_k)$ for a sequence Ω_i. As the observations are continuous, the emission probabilities $\{b_{ij}^k\}$ are approximated by a multivariate Gaussian mixture model for each HMM state (see Fig. 6). The optimal number of HMM states and Gaussians for each behavior were determined by means of experimental cross validation. In these experiments, the log likelihood was maximized with respect to a set of test cases consisting of 40% random samples from the original training set. The best overall performance resulted from 10 states and 2 Gaussians per state.

Each HMM was trained using 400 sequences. For some behaviors, there was a slight overlap between the training sequences. Testing was done on 250–300 test sequences per behavior. They were randomly selected from a database of observations of behavioral patterns observed and recorded at various field positions.

Overall, we achieved convincing results with this approach and real-time behavior classification based on canonical coordinate sequences appears possible. Table 1 displays the confusion matrix for the experiments where the sequence length was set to

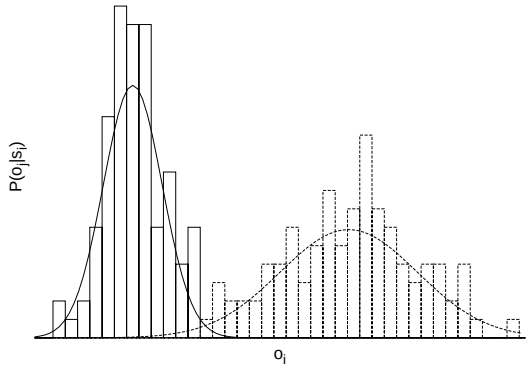


Fig. 6. In a Gaussian mixture model, an arbitrary probability distribution $P(x)$ is approximated as a linear combination of (high-dimensional) Gaussian distributions: $P(x) = \sum_{k=1}^{l} a_k \mathcal{N}(\mu_k, \sigma_k)$ where μ_k and σ_k denote mean and standard deviation (or –in the high-dimensional case– the covariance matrix) of distribution k . In the example shown in this figure, the probability $b_{ij} = P(o_j|s_i)$ for emitting token o_j in state s_i is modeled by a mixture model of $l = 2$ Gaussian modes. A typical approach to determining suitable parameters μ_k and σ_k from given data is the *expectation maximization* (EM) algorithm [7]. It consists in iterating two steps: first, an expectation (E) of the likelihood of the current (initially random) parametrization fitting the data is computed. Second, a maximization (M) step computes the maximum likelihood estimates of the parameters by maximizing the expected likelihood found in the first step. The parameters resulting from this M step are then fed into another E step and the process is repeated until convergence.

$h = 45$ frames and the sliding window for computing canonical coordinates covered $f = 25$ frames.

Three out of the five higher-level behaviors we considered were classified with high success rates. The behaviors long-pass and solo-play are an exception in that they tend to get confused with each other. A closer in-game inspection of these two activities revealed considerable similarities among some of the recorded sequences. For instance, before the ball is passed in a long pass situation, one player usually possesses the ball and the other player is moving away. However, a very similar characteristic is also found in subsequences of solo play. There, one player possesses the ball while the other one is moving alongside. In essence, the first part of a long pass sequence consists of vectors very close to solo play behavior.

The per-frame classification rates in Tab. 1 can be improved by exploiting the fact that behavior is a temporal phenomenon. Using temporal context on a higher level of abstraction, we apply a majority voting scheme to the results of the frame-wise classification. If one or several frames of a longer sequence are misclassified, such temporal filtering will increase the confidence level of the HMM-based classification. The figures in Tab. 2 confirm this intuition for they show a considerable increase in classification accuracy.

Table 1. Confusion matrix resulting from testing the five behavior classifiers.

behavior	dribbling	give-and-go	ball retrieval	long pass	solo play
dribbling	**199**	16	0	25	12
give-and-go	0	**145**	12	0	27
ball retrieval	0	9	**113**	0	17
long pass	0	0	0	161	**186**
solo play	0	1	15	124	**205**

Table 2. Confusion matrix after temporal smoothing.

behavior	dribbling	give-and-go	ball retrieval	long pass	solo play
dribbling	**218**	9	0	25	0
give-and-go	0	**151**	9	0	20
ball retrieval	0	12	**112**	0	15
long pass	0	0	0	163	**184**
solo play	0	0	13	119	**213**

In addition to confusion matrices, classifier performance is often characterized by means of two more abstract measures called *precision* and *recall* [8]. They are based on four categories of classification results: in our case, for a given HMM λ_k, the true positives (TP) are test sequences correctly assigned to the behavior represented by λ_k. The false positives (FP) are behaviors that should not have been classified to λ_k; true negatives (TN) are behaviors correctly assigned to another HMM, and false negatives (FN) refer to behaviors that should have been assigned to λ_k but were not. Given these notions, recall and precision are defined as

$$\text{recall}(\lambda_k) = \frac{TP}{TP + FN} \qquad \text{and} \qquad \text{precision}(\lambda_k) = \frac{TP}{TP + FP}. \tag{9}$$

Recall thus measures the fraction of behaviors of type k that were correctly labeled while precision denotes the fraction of behaviors that were labeled k and indeed are of type k. Table 3 displays the corresponding, more abstract performance indicators, we obtained from our experiments with the majority voting scheme.

5 Conclusions and Outlook

Although modern computer games are usually of high technical quality with regard to graphics and physics simulation, the majority still lacks convincingly acting artificial game agents. Especially for genres where the player has to control a whole team of agents, team members that act inconsistently when taken over by the game AI may ruin tactics or strategies devised and intended by the human player and thus will spoil the

Table 3. Precision and recall values corresponding to the classifier responses in Tab. 2.

	dribbling	give-and-go	ball retrieval	long pass	solo play
precision	86.5	84.2	80.6	47.0	61.7
recall	100.0	88.1	83.6	53.1	49.3

fun. A first step to remedy such unsatisfying behavior will be to provide gamebots with *a sense of what is going on* or with *a sense of their mates' activities*. In this paper, we therefore presented an approach to classifying human team behavior in sports computer games. Exploiting opportunities provided by modern multiplayer game technology, we use recordings of network traffic generated by human players to train behavior classifiers for a soccer game. In order to be able of recognizing behaviors independent of the current field location of the involved players, position vectors of players and ball are transformed into canonical coordinates. Sets of canonical player-, team-mate- and ball trajectories are used to train different HMMs which model different behaviors. Since, in the game domain, we have to deal with the real time constraints, the time-frame for classification is limited to less than 50 game tics. Subsequent temporal filtering lifts the recognition rates to satisfactory levels. Several series of initial experiments reveal that this framework indeed is computationally inexpensive and therefore may be applied in commercial products. However, although the overall performance appears to be robust, for some behaviors there still is room for improvements.

Consequently, we are currently working on extending our approach towards *anticipation*. A reliable classification of certain behaviors requires information on the future development of an observed movement sequence. Since augmenting the representations introduced here with expected future canonical positions may overcome this problem, we are testing the use of Kalman filters and Markov Chain Monte Carlo methods which generate predictions based on recently observed game data. Regarding the classification of whole episodes, future work will also explore the use of hierarchical HMMs.

Acknowledgements

This work was supported by the German Research Foundation (DFG) within the graduate program "Strategies & Optimization of Behavior". We are grateful to Thomas Hettenhausen for his endeavors in providing us with test and training data. Also, we want to thank the developers of "Tao of Soccer" for making it available for public use.

References

1. Cass, S.: Mind games. IEEE Spectrum (2002) 40–44
2. Nareyek, A.: Artificial Intelligence in Computer Games – State of the Art and Future Directions. ACM Queue **1**(10) (2004) 58–65
3. Sukthankar, G., Sycara, K.: Automatic recognition of human team behaviors. In: Modeling Others from Observations, Workshop at IJCAI. (2005)

4. Assfalg, J., Bertine, M., Colombo, C., del Bimbo, A., Nunziati, W.: Semantic annotation of soccer videos: automatic highlights identification. Comp. Vis. Image Underst. **92**(2–3) (2003) 285–305
5. Jaser, E., Kittler, J., Christmas, W.: Hierarchical decision making scheme for sports video categorisation with temporal post-processing. In: Proc. CVPR. Volume II. (2004) 908–913
6. Rabiner, L.: A tutorial on hidden malov models and selected applications in speech recognition. Proc. of the IEEE **77**(2) (1989) 257–286
7. Dempster, A., Laird, N., Rubin, D.: Maximum likelihood from incomplete data via the EM algorithm. J. of the Royal Stat. Soc., Series B **39**(1) (1977) 1–38
8. Davis, J., Goadrich, M.: The relationship between Precision-Recall and ROC curves. In: Proc. Int. Conf. on Machine Learning, ACM (2006) 233–240

Using Hierarchical Machine Learning to Improve Player Satisfaction in a Soccer Videogame

Brian Collins and Michael Rovatsos

School of Informatics
The University of Edinburgh
Edinburgh EH8 9LE
United Kingdom
BrianC@BrianLCollins.com
mrovatso@inf.ed.ac.uk

Abstract. This paper describes an approach to using a hierarchical machine learning model in a two player 3D physics-based soccer video game to improve human player satisfaction. Learning is accomplished at two layers to form a complete game-playing agent such that higher-level *strategy* learning is dependent on lower-level learning of basic *behaviors*. Supervised learning is used to train neural networks on human data to model the basic behaviors. The reinforcement learning algorithms Sarsa(λ) and Q(λ) are used to learn overall strategies mapping game situations to these basic behaviors. We compare learning and non-learning agents and provide game results. Performance in self-play is analyzed to obtain a deeper understanding of the agent's learning performance. Seventy people participated in a survey in which the learning agent led to a more dynamic and entertaining experience, while the non-learning agent was a slightly more difficult opponent.

1 Introduction

A problem often experienced in video games is that the results obtained by directly specifying artificial player ("agent") behavior may be very predictable or just not very good. Machine learning (ML) [1] offers many techniques that can be applied to games to create more dynamic and realistic AI agents that can adapt to new situations. In an attempt to increase entertainment in games, we apply a layered ML architecture to a 3D physics-based soccer video game. This involves learning lower level behaviors to facilitate the learning of successively higher level behaviors.

An additional benefit of learning is that it makes the design of artificial game players methodologically easier. Without ML, a programmer needs to manually "search" for and compare strategies used by agents. Considering the current availability of low-cost computational power, machines are much more suited to this task and can find better strategies in less time. Unfortunately, learning a model that directly maps inputs to outputs is too complicated for most games. Even if it were possible, it would probably not lead to a good general playing

strategy, since it would be very difficult to manually adjust such a model and correct poor behavior. Layered ML alleviates these problems and allows for more control over the learning process. A custom hierarchy of behaviors to be learned can be designed specifically for a game, allowing the benefits of problem domain knowledge to be fully realized. Behaviors at any level of the hierarchy can either be hand-designed or learned. Using a layered learning model is flexible and allows for the learning of complex behaviors.

The main goal of the research presented here is to implement learning agents with acceptable in-game performance against people and other agents. Furthermore, agent behavior should be "human-like", i.e. appear natural and dynamic. Ultimately, we wanted to provide an entertaining game experience. To do this, we used neural networks to model low level behaviors based on human data. We used reinforcement learning (RL) to learn high level strategies based on these behaviors.

Stone applied a similar layered learning architecture to simulated robotic soccer [2], but the behaviors to be learned were different; entertainment and "human-like" behaviors were not required. An alternative method to learn based on human data in games is presented in [3], where rules are learned rather than neural networks. Another way to learn and use rules in games is presented in [4].

This project is a case study of considerable size examining the potential for using layered learning in video games to increase their overall entertainment value. We identified human behaviors in simple training games that were easier to model than others. We compared the performance of different RL algorithms [5] in a fairly complex environment, which exceeds the complexity of toy environments often used in the literature. Given enough time for learning, agents using learning consistently won games against agents using the best non-learning agents. Survey participants found the best non-learning agent to be more difficult, while the best learning agent was more dynamic. Overall, people found playing against the learning agent to be a more entertaining experience.

2 The Soccer Game

The problem domain is a 3D real-time one-on-one soccer game (screenshot in Fig. 1). The physics engine supports spheres with linear and angular momentum, as well as stationary planes, cylinders and boxes. Relevant physical equations are solved numerically in real-time. Players are represented as spheres and the soccer ball is a smaller sphere. Each player scores a point for a goal and spheres can collide with the walls. To control a player, a human or an automated agent provides a two-dimensional vector representing acceleration. They have the option to kick the ball at any time, releasing it in a straight line. Physical objects in motion are gradually slowed by friction and drag. Players acquire the ball by colliding with it; then the ball is attached to the player and is automatically repositioned in front of the player as they move. The ball can be stolen from an opponent by touching the ball then quickly getting it away from them.

The game is complex enough that it is highly improbable that an agent exists which performs optimally in every situation. Hand-designed agents are not very

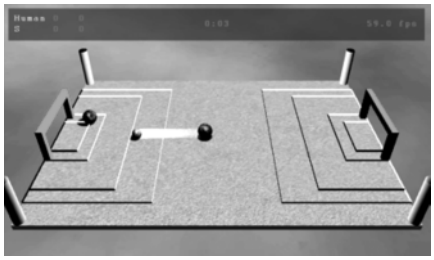


Fig. 1. The Soccer Game - A player is about to score a goal. The soccer field is flat and the white lines are zone markers used in representing strategies.

likely to be optimal. Optimal agents would need to be able to adapt their strategy to a wide range of situations. Low level behaviors, such as moving across the field can be difficult to specify or learn. The physics engine works with fairly complex differential equations (for friction, drag, angular momentum, ball handling, etc). Objects with angular momentum do not move in straight lines and non-zero velocities never remain constant. As a consequence, obtaining optimal low-level behaviors is a difficult process. For example, using acceleration to simultaneously control velocity and position is difficult within the game.

3 Layered Learning Architecture

We developed a two-layered learning architecture for this game. The lower layer allows for learning "basic behaviors" that are designed to accomplish simple tasks. The behaviors were chosen to be simple and robust, yet complex enough to be non-trivial. Each behavior needs to represent an opportunity for ML. An individual player's strategy can be formed by selecting a single basic behavior to use at a given time. Given the state of the world and a set of available behaviors, it is not trivial to select an appropriate one for a situation. Therefore, the higher "strategy learning" layer uses RL to learn which basic behavior to apply in each state from experience using information about rewards obtained previously. In other words, the higher layer learns a meta-strategy over the basic behaviors learned by the lower layer.

3.1 Supervised Learning of Low-Level Basic Behaviors

Seven basic behaviors were chosen for the game and we created a corresponding "training game" for each one. These training games are used to obtain data from human players and to objectively measure learning performance, the focus being on achieving "human-like" behavior at this level, rather than optimal performance in terms of the game in question. Supervised learning is used, since desired output vectors are known for the input vectors. During training data collection, a person plays the game and the game state is recorded at a fixed

rate. The games are designed to present the player with random situations from the set of all game situations where the behavior could potentially be used. The aim is for a generalized representation of a basic behavior to be distilled from data obtained from the training games. Although this process cannot be guaranteed to work, very good results were obtained.

The basic behaviors chosen for the game are as follows: intercepting the ball (*Intercept*), retreating to the goal (*Retreat*), attempting to get close to the potential path of a kicked ball (*Defend*), aiming the ball toward the opponent's goal (*LineUp*), advancing toward the opponents goal (*Advance*), stealing the ball (*Steal*), and preventing an opponent from stealing (*KeepBallSafe*). Each of these are available to the higher learning layer as a primitive action with the exception of *LineUp*, for which three variations are provided to aim the ball toward different sections of the goal, thus allowing for a greater variety of strategies. Finally, there is a *Kick* behavior that does not require learning in this game.

The design of the training games has a fundamental effect on the behaviors that are learned. For example, it is possible that high training game performance may not be related to high performance against humans at the same task. For some behaviors, artificial opponents are needed in the training games to provide a consistent measurement of performance. In most training games the opponents select randomly from a set of simple hand-designed behaviors.

At the "basic behavior" layer, feedforward fully connected Multi-Layer Perceptrons (MLPs) were used to model and learn the behaviors. We used MLPs with a single sigmoid (tanh) hidden layer and linear output units, which can represent any bounded continuous function to an arbitrarily small error [6] and learn such functions from training data. The training algorithm used was stochastic gradient descent backpropagation with momentum [1] and training was off-line, so that the MLPs are fixed during actual games. A mean squared error (MSE) function was used to measure how well a given MLP models a training dataset. The neural networks for each basic behavior have two real number outputs (for 2D acceleration) and between six and ten real-valued inputs that are derived from game state information. For optimal performance, network inputs are first scaled to have a mean and variance of approximately 0 and 1, respectively. Each MLP had ten hidden neurons and approximately 100 real-valued weights to be learned. This appeared to be a good MLP size for the amount of available training data. The networks were trained for between 100 and 20,000 epochs. In some cases, MLPs with the best performance were found quickly and further training did not yield better results. In other cases, in-game performance continued to improve after thousands of epochs.

Twenty percent of the collected training data was used as a validation dataset. We also applied early stopping, where training is stopped in the event that the validation error has not improved for a certain number of epochs, i.e. allowing training to continue was not beneficial. The most consistent results were obtained when the validation data was taken from several disjoint segments of the collected data.

3.2 Reinforcement Learning for Higher-Level Strategies

A policy for playing the game can be defined to be a mapping from game states to actions (basic behavior implementations). The class of RL algorithms applied here is that of temporal difference (TD) methods [5], which learn from experience by continually improving an estimate of the optimal policy, i.e. the policy that leads to maximum long term reward. They are model free in the sense that they do not make use of explicit models of the environment (e.g., transition probabilities between states). Bootstrapping allows faster learning and updates estimates of reward based on other estimates. In terms of learning strategy we used on-line learning, where an agent learns while they play as this has the potential to lead to a dynamic experience for a person playing against the agent. However, this adaptiveness comes at the price of not being able to guarantee convergence of learning since a person's changing strategy leads to a dynamic environment.

However, for a fixed-strategy opponent, a policy learned using RL will, under certain conditions, converge to the optimal policy [5]. The conditions necessary for convergence were not created in the system, since it may not be desirable within a videogame; policies cease to be adaptive when convergence occurs. Despite this, it is important to note that minor changes to the RL implementation could allow for guaranteed convergence (such as decreasing the exploration rate over time such that it becomes zero in the limit).

RL heavily relies on an appropriate definition of state and action spaces, whereby states can be complex structures built up over time that implicitly contain information about past occurrences and the requirement is that all information needed to make a decision is implicitly in the knowledge of the current state. To learn a strategy in a reasonable amount of time, the most important information must be encoded within a small set of states. Additionally, in the case of on-line learning in a video game, learning has to happen within a reasonable amount of time as game play only extends over a limited period and human players need to be able to experience their artificial opponent's adaptiveness to make the game appear interesting.

The state space defined for our domain was based on quantized values of the most important information, such as the player's proximity to the soccer ball and some historical information about recent ball possession. Without any historical information, an agent may observe that they do not have the ball immediately before they score a goal and may learn that not having the ball is good. Rewards are given for goals and negative rewards are given when the opponent scores. Rewards are also given in some other situations to increase the speed of learning, for example when the player gains possession of the ball or loses it.

The RL algorithm used was the standard version of Sarsa(λ) [5]. This algorithm uses so-called eligibility traces to determine how each observation can be used to update predictions made in the past. The result is a table of estimated Q-values, which estimate the expected long term reward for taking an action in a state. A variant of Q(λ), another algorithm which estimates Q-values, referred to as "naive Q(λ)" in [5] was also implemented, but the time needed for

learning was much longer. The one-step Sarsa method (equivalent to Sarsa(0)) performed significantly better than one-step Q-learning. The likely cause is that Q-learning needs more time to implicitly model state transitions. Sarsa takes actual state-action sequences into account and learning was faster.

A Q-table alone does not provide enough information for selecting an action to apply in a given state. There is always a trade-off between exploration and exploitation of current knowledge. Exploitation may lead to higher rewards, while exploration may lead to increased knowledge and ultimately higher rewards. Epsilon-greedy (ε-greedy) exploration and Softmax exploration [5] were both implemented and Softmax provided better performance. It includes an exploration temperature τ, which allows for a blend of exploration and exploitation. For high temperatures, actions are essentially selected at random. For low temperatures, actions with low Q-values will rarely be selected, while there is a high probability of selecting an action with a relatively high Q-value. This focuses exploration on actions that are estimated to be nearly optimal.

4 Experiments

4.1 Basic Behaviors

In all training games besides *Advance*, we were able to find a neural network which consistently performed better than a human expert. Note that performance is indicated by training game score. For some behaviors, the ideal outcome occurred and the networks with the lowest validation dataset error had the highest training game performance. An example of this is the *Intercept* behavior in Fig. 2. For other behaviors such as *Retreat* (Fig. 2), the highest performing networks were found earlier in the training process while validation set error was still dropping. Figure 2 shows how validation set error was slightly higher than training set error throughout the training process. The correlation of training game performance to data set error varied between behaviors, but was strong for the simplest behaviors.

In two cases, obtaining high performance networks through training was particularly difficult. It is possible that not enough high quality training data was obtained. Low dataset error cannot guarantee high in-game performance, but the process worked well for most of the behaviors. Using highly processed information from the game state as input to the networks was not effective, since relationships between inputs and outputs became more complex and more difficult to learn.

Hand designed basic behavior implementations were created with varying amounts of effort and manual tuning. In two cases, the MLP implementations performed better than the hand designed alternatives. For these two networks, the correlation of training game performance to data set error was strong and networks with lower dataset error performed better in the game. In other cases, MLP performance was comparable to hand-designed behavior performance. In all instances the MLP based behaviors appeared to be more "human-like" and less rigid than their hand-designed counterparts. Noise could be added to the

outputs of the hand-designed behaviors, but results are not likely to be as good. Human actions in the game can be imprecise and the MLPs implicitly model the types of deviations a player might make from their desired paths.

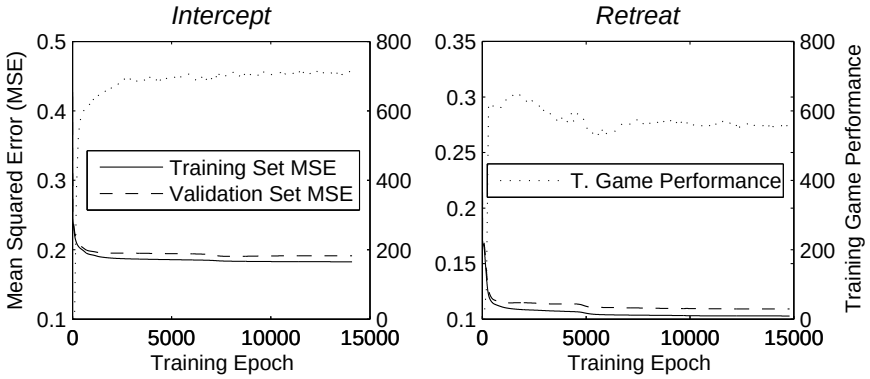


Fig. 2. The neural network training process for two basic behaviors. Actual training game performance is not necessarily related to mean squared error on the validation dataset.

4.2 Higher-Level Strategies

Experimentation was performed to find good values for the many parameters of the Sarsa(λ) algorithm. After reasonable initial values were selected, many games were run against an agent with a hand-designed deterministic strategy to optimize the parameters. A state space with 108 states and 432 state-action pairs was used, which led to high performance policies and required less learning time than larger state spaces. When the number of states used was 500, up to ten times more learning time was required to obtain the same results. After extended periods of learning, performance never exceeded the performance of the small state space. Increasing the number of states may be necessary when playing against opponents that have highly complex strategies.

For each experiment, two parameters were varied simultaneously in pursuit of optimal values. The λ parameter determines how much observed information is used to update past predictions and the best value was 0.7. Softmax exploration led to the best overall performance. Higher exploration temperatures, of around 2.0, led to increased performance at the conclusions of the sessions but relatively low performance throughout the sessions. Balanced exploration (temperatures ranging from 0.5 to 2.0) led to better overall results and decent performance at the conclusions of the sessions. The best value for the learning rate α was 0.03. The best values of the reward discount factor γ were between 0.925 and 0.98. The $\gamma\lambda$ product is closely related to the eligibility trace and intermediate values of this product, between 0.55 and 0.75, led to the best results.

4.3 Combining the Learning Layers

The best implementations of each type (ML/non-ML) for each learning layer were combined. Strategy implementations were: RL (Sarsa(λ) with Softmax ex-

ploration), a hand-designed stochastic policy, and a hand-designed deterministic policy (these policies are essentially based on finite-state machines). MLP based behaviors and hand-designed (HD) behaviors were tested in combination with each (high-level) strategy. Stochastic fixed policy agents won almost all of their games against deterministic ones.

Overall, the players using MLP based behaviors outperformed the counterparts that used HD behaviors. On average, agents using HD behaviors won 42% of their games while agents using MLP behaviors won 57% of them. This does not coincide with the training game performance of the behaviors. Some behavior implementations display weaknesses during game-play which cannot be seen in the training games. Perhaps the MLP behaviors are more robust despite slightly inferior (on average) training game scores. For games lasting 25 game-hours (simulated in around 3 minutes), both RL players won the majority of their games. They clearly outperformed all fixed policy agents. When the Sarsa/MLP player competed directly with the Sarsa/HD player, the Sarsa/MLP player won most games. It seems to be easier to learn a strategy based on the more complex MLP behaviors. Since the MLP behaviors are more robust, a higher proportion of strategies based on them (as opposed to HD behaviors) lead to acceptable performance. This increases the chances of finding an MLP based strategy through learning that performs consistently well. When self-play was simulated, two learning agents with initially no knowledge competed directly. Actions in such games were initially random. After some time, simple strategies such as heavily offensive strategies emerged. After enough time, complex and well-balanced strategies could be observed. Provided the exploration temperature τ was significantly high, around 0.5 to 2.0 (depending on the lower level behavior implementation), this process consistently generated useful and balanced strategies.

Learning times were reduced to one game-hour and Q-tables learned in various 25 hour games were used as initial Q-tables. Figure 3 illustrates some of these results. The players benefited from any initial knowledge, particularly when the exploration temperature τ in previous game was high, 2.0 in this figure. A lower τ in the next game of 0.5 allowed the players to immediately exploit their knowledge and simultaneously adapt it to the new situation. These are the τ values used for Fig. 3. Both RL players benefited the most from their experience against the stochastic policy agent. There was a 50% increase in the number of one hour games won against various (learning and non-learning) automated opponents. Experience from self play was beneficial to both RL players, leading to a 35% increase in games won. Each RL player also benefited from any experience obtained by the other, which lead to a 28% increase in games won. This indicates that the process can lead to generally good strategies. Self play is an interesting case, because it leads to a dynamic non-Markovian environment. RL for non-Markovian processes is still an active area of research. Results cannot be guaranteed [5], but good results were obtained leading to increased confidence in the underlying methods. The amount of time needed for learning is important for a real-time game. Good strategies were learned from no initial knowledge in

less than an hour of game time, which is a reasonable amount of time for a person to play such a game. Simulated games can be run much faster than real-time to obtain prior knowledge, which allowed for significantly better results in the early stages of learning.

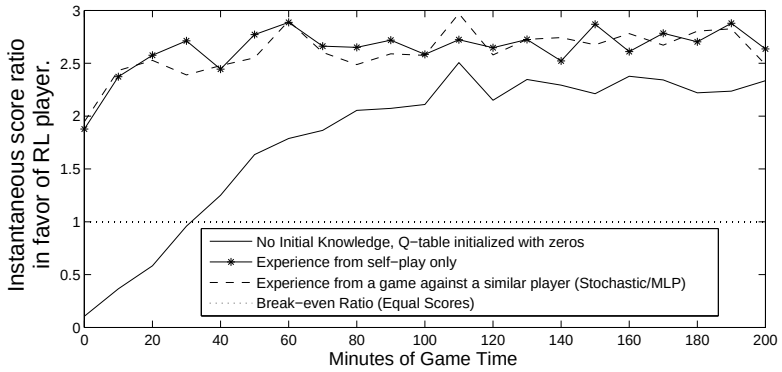


Fig. 3. Average performance of an RL agent as time increases. The agent had varying initial knowledge and used MLP based behaviors. The opponent was the stochastic/HD agent. The score ratio in favor of the RL agent is presented. This is an instantaneous ratio consisting of points scored within 10 game minutes. An agent with no initial knowledge began outperforming their opponent, who uses a fairly complex strategy, in around 30 game minutes on average. Knowledge from self-play is seen to be very useful.

4.4 Survey

In order to evaluate the more "human-centric" goals of the project, an online survey was conducted in which 70 people participated. Participants played five minute games against two different opponents in a random order. The ML player used Sarsa(λ) for the higher layer and MLPs for low-level behaviors. The non-ML player used a hand designed stochastic strategy and hand designed behaviors. People usually found the non-ML player to be slightly more difficult. In order to give the ML player a chance of providing a good challenge, it had been trained against the non-ML player and could defeat it consistently. People were not told which opponent used ML. An average of 41% of people won their first game, while 50% of people won their second game. This did not seem to depend on which opponent was encountered first. Overall, people found the non-ML player to have a better strategy and be more difficult. However, it was significantly easier for players to predict the actions of the non-ML player. In general, people felt that the ML player moved around more fluidly and behaved in a much more dynamic 'human-like' way. Some people commented that player behaved like a human who was not an expert at the game and still made some mistakes. Despite occasionally making mistakes, the ML player proved to be an entertaining and challenging opponent. People may have felt the non-ML player had a better strategy because its strategy is more consistent. The ML player sometimes appears to change its playing style several times during a game, which may have had a confusing effect

on players not anticipating it. Overall, people were more satisfied with the ML player. Some noted that it adapted based on their own strategy. A representative comment about the ML player was that it "felt less 'cookie cutter AI' ... he sort of moved around more fluidly and didn't resort to the same patterns." Many people appreciated the adaptive nature of the ML player. Some others preferred the more traditional challenge offered by the non-ML player and the pleasure of finding a strategy that consistently outperformed this non-adaptive opponent.

5 Conclusion

In this paper, we described the implementation of a layered learning architecture in a soccer videogame. It was based (a) on learning basic behaviors based on data obtained from humans performing simple tasks and (b) on learning a game strategy that uses these basic behaviors as primitive actions by means of reinforcement learning.

At the lower level, our method was very successful for learning human-like, natural-looking behaviors, provided that the behaviors to be learned were simple enough. Attempts to learn more intricate behaviors in this way led to disappointing results, as training data was noisy and complex. The design of the training games used for data collection is very important at this stage. At the higher level, the RL implementation led to very good results with reasonable amounts of learning time. The ML agents performed well against other agents and were well received by human players, who found them dynamic and entertaining.

The hierarchical learning approach is very flexible and led to efficient learning. Much control over the learning process was retained and undesirable actions could be corrected with relative ease. The hierarchical learning model used could easily be extended to more interesting situations with multiple agents on each team. Various policies could be learned through RL for common roles in a soccer team, using different reward functions. In the future, we plan to add a third "social" learning layer which would enable artificial players to learn optimal teamwork coordination strategies.

References

1. Mitchell, T.M.: Machine Learning. McGraw Hill (1997)
2. Stone, P.: Learning in Multi-Agent Systems. PhD thesis, Computer Science Department, Carnegie Mellon University (1998)
3. van Lent, M., Laird, J.: Learning hierarchical performance knowledge by observation. In: Proc. 16th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA (1999) 229–238
4. Laird, J.E.: It knows what you're going to do: adding anticipation to a quakebot. In Müller, J.P., Andre, E., Sen, S., Frasson, C., eds.: Proceedings of the Fifth International Conference on Autonomous Agents, Montreal, Canada, ACM Press (2001) 385–392
5. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press (1998)
6. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals, and Systems **2** (1989) 303–314

Adaptive Generation of Dilemma-based Interactive Narratives

Heather Barber and Daniel Kudenko

University of York, Heslington, York, YO10 5DD, England

Abstract. In this paper we present a system which automatically generates interactive stories that are focused around dilemmas to create dramatic tension. A story designer provides the background of the story world, such as information on characters and their relations, objects, and actions. In addition, our system is provided with knowledge of generic story actions and dilemmas which are based on those clichés encountered in many of today's soap operas. These dilemmas and story actions are instantiated for the given story world and a story planner creates sequences of actions that lead to dilemmas. The player interacts with the story by making decisions on these dilemmas. Using this input, the system adapts future story lines according to the player's preferences.

1 Introduction

In recent years computer games from most genres have included a progressive story line to increase the immersive experience of the player and their enjoyment of the game. However, stories are often linear, and in almost all cases pre-defined, which reduces the replay value of these games. Research into interactive narrative generation (or interactive drama) tries to overcome these weaknesses. Most interactive drama systems (prominent examples include [2, 7, 9, 1, 3, 8, 11, 4, 10]) are focused on generating short story lines and do not adapt to the player (see Section 8 for exceptions).

In this paper, we propose a system that generates interactive stories which are long (potentially infinitely so), and that adapt to the player's behaviour. To add dramatic tension, the story incorporates dilemmas as decision points for the player. These dilemmas are based on the clichés found in many contemporary soap operas, such as the between personal gain and loyalty to a friend. Overarcing stories connect these dilemmas as points of interaction within a coherent plotline that is dynamically created, based on the player's response.

Our goal is to keep the story designer's input to a minimum. In the proposed system, the story designer provides the story background in the form of character information and other knowledge that relates to the world in which the story is to be created (e.g., a pirate ship). The system then instantiates all generic knowledge on story actions and dilemmas accordingly and thus creates the narrative.

Our paper is structured as follows. We first give a general overview of our system, followed by a discussion on how the story background is specified. We

then proceed with a description of dilemmas, the story generator, and the player modelling component. We finish the paper with a brief overview of related work and conclusions.

2 System Overview

The interactive drama system consists of: a knowledge base (which contains information regarding the characters); story actions; and dilemmas which can occur in the storyworld. These components are drawn upon in the generation of a narrative through planning. The user is able to interact with the narrative generator, and their actions effect the story experienced. A user model is employed to ensure that the story's dramatic interest is maximised. The interactions between the system components are shown in figure 1.

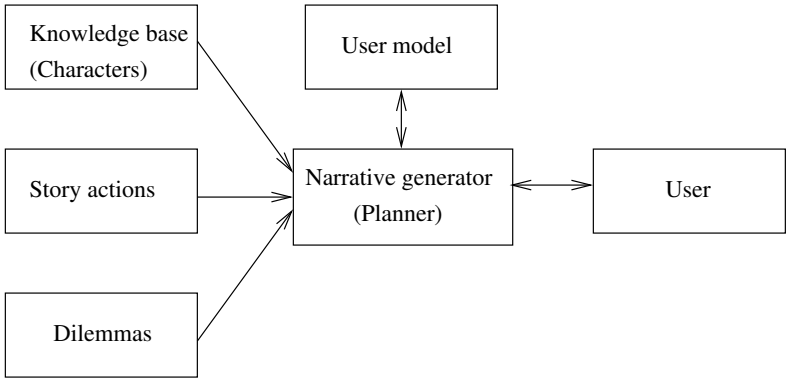


Fig. 1. This figure shows the components of the system and how they interact.

3 The Storyworld

The storyworld consists of characters and actions in which the characters can participate. These characters have associated characteristics, personalities and relationships with one another.

In the current storyworld, each character has associated domain independent characteristics, including: attractiveness, generosity, morality, selfishness, gender, sexuality and age group. A range of values is associated with each attribute. It is also possible to specify particular personalities, such as bad_boy and busybody. These genre specific character descriptions are those which are not fully deducible from other aspects of a character's personality and will relate to specific storylines within the current domain. A character's nature affects

which actions they can participate in and also, ideally, the user's opinion of that character. These personality traits should be apparent to the user from the way the characters act within the storyworld.

Characters also have storyworld relationships with one another, including friendship and love. They are able to disapprove of one another's partnerships. This can be for any one of a variety of reasons, including an age difference or snobbery.

The characters hold storyworld principles, such as monogamy. Under specified pressures and circumstances, principles can be broken (or their associated strength of belief reduced). Characters also have aspirations, for example wanting a baby. These affect which actions a character will participate in and the dilemmas in which they will become involved.

A series of genre-specific locations are required by the storyworld. At any given time in the story, each character will be at one of these locations. Direct interactions between characters can only take place if they are at the same location.

Those actions which can take place within the storyworld must be specified for each domain. Every possible action should be included. Actions have associated preconditions (for execution) and effects (required to update the storyworld in accordance with their successful completion).

4 Dilemmas

Field [6] states that "drama is conflict", that the dramatic interest in a story centralises on its conflicts. In genres which make use of clichéd storylines, these are usually found to be essentially conflicts (or dilemmas). Writers utilise these dilemmas in the creation of stories. A general form of each such clichéd dilemma can be determined, and a computerised storywriter can create an interactive drama around these.

Since the focal point of an interactive drama is the user, each dilemma should represent a conflict to that user. Within the course of the experience, they will be required to make fundamentally difficult decisions which will have negative outcomes whatever choice they make.

Our experience showed that when more than two characters were involved in a dilemma, it was either expandable to multiple two character dilemmas, or the characters receiving payoffs naturally divided into two groups with the same resultant utility. Therefore a user decision on a dilemma will involve only two recipients of utility payoffs. Five such dilemma categories were identified. These do not consist of all payoff matrices for two players. Many such matrices would not involve a dilemma for the character making the decision. The relevant categories are: Betrayal (dilemma 1), Sacrifice (dilemma 2), Greater_Good (dilemma 3), Take_Down (dilemma 4) and Favour (dilemma 5). In order to involve a dilemma for the user, these may require characters to be friends or enemies. Where relevant, this is stated with the dilemma utility matrices in dilemmas 1 to 5.

In these dilemmas: A_X represents the decision of character X being to take action A; u_C^i represents the utility of character C for the respective action; and i denotes the relative value of the utility, i.e., u_C^1 is greater than u_C^2.

$$\frac{A_X \mid (u_X^1, u_Y^2)}{\neg A_X \mid (u_X^2, u_Y^1)} \wedge friends(X,Y) \; - Betrayal \tag{1}$$

$$\frac{A_X \mid (u_X^2, u_Y^1)}{\neg A_X \mid (u_X^1, u_Y^2)} \wedge friends(X,Y) \; - Sacrifice \tag{2}$$

$$\frac{A_X \mid (u_X^1, u_Y^1)}{\neg A_X \mid (u_X^2, u_Y^2)} \wedge enemies(X,Y) \; - Greater\_Good \tag{3}$$

$$\frac{A_X \mid (u_X^2, u_Y^2)}{\neg A_X \mid (u_X^1, u_Y^1)} \wedge enemies(X,Y) \; - Take\_Down \tag{4}$$

$$\frac{A_X \mid (u_Y^1, u_Z^2)}{\neg A_X \mid (u_Y^2, u_Z^1)} \; - Favour \tag{5}$$

For example, the first dilemma represents a conflict where a player's action A_X will lead to a higher utility for himself than not performing A_X. For the player's friend Y, however, this action will lead to a lower utility.

As can be seen, dilemmas 1 and 2 are the inverse of one another, as are dilemmas 3 and 4. This means that any dilemma which falls into one of these categories can be inverted to become a dilemma of the other category. All five categories are kept to increase ease of dilemma identification within specific genres. From these categories (given in equations 1 to 5) dilemma instances can be found and generalised within each domain. From the generalised form of the dilemma the system will be able to create new dilemmas. In the presentation of these to the user wholly original stories are created.

It will not be possible to create great literature in this way – the use of clichéd storylines prevents this. However, such stories are enjoyed by many people and this method is common in such genres as James Bond films, soap operas (soaps) and "chick flicks". The story is built around the cliché, and it is the cliché as well as the story which the audience appreciate, the very repetitiveness and familiarity of the dilemmas adding to the dramatic interest. It can only be imagined how much more enjoyment could arise from the user becoming a character in such domains, and experiencing the dilemmas first hand.

5 The Narrative Generator

Prior to a dilemma being presented to the user certain conditions must be met within the storyworld. These are the preconditions of the dilemma. It is the task of the storywriting system to achieve these preconditions. This constitutes the build-up – the essence of the story itself. Given actions within the storyworld, the system can use planning to satisfy a dilemma's preconditions. In this way, a plan to achieve a dilemma becomes a storyline. The interactive drama is made

up of a series of such substories, dynamically selected according to dramatic interest.

The storyworld actions are domain specific. They include characters falling in love, becoming pregnant and being involved in crimes – such as drugging or murder.

The potential consequences of each decision must be clear to the user before they make their choice. Once they have chosen, these repercussions on the storyworld will be implemented.

On being passed a dilemma, a STRIPS planner finds all plans to achieve this dilemma given the current storyworld state and background knowledge. Dilemmas are not running concurrently, therefore the build-up cannot be too extensive as the user could become disinterested in the experience. As a result, the system is designed to prefer shorter plans. Stories of the same length will involve more drama if plotlines are shorter. From these plans, that which is most dramatically interesting can be selected. Once the user has made their choice, the system updates the storyworld state in accordance with that choice. The system can then plan from the new state in order to be able to present another dilemma to the user – thus continuing the interactive drama. This sequence of events is demonstrated in fig. 2. From this figure it can be seen that the planner finds all plans in the story dependent on the current state and a given dilemma. If no plan can be found for this dilemma, another will be selected. Once all plans have been found, the most dramatically interesting can be presented to the user, resulting in a new state from which the story will continue.

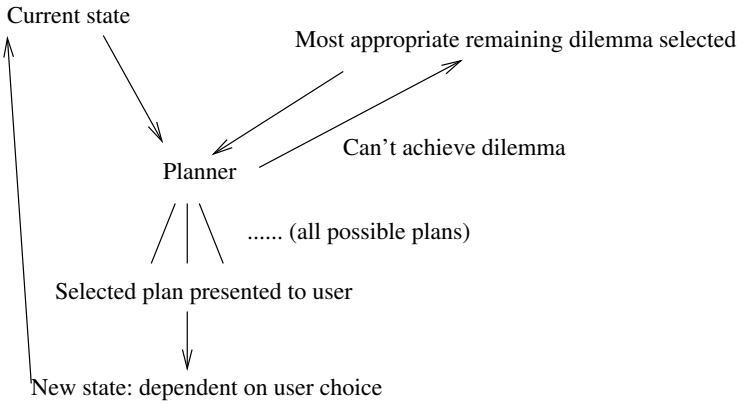


Fig. 2. This figure gives an overview of the system moving between states dependent on plans, dilemmas and user decisions.

The sequence in which the dilemmas are selected for planning is dependent on dramatic interest and the user model.

6 The User Model

The user of an interactive drama system should be modelled rather than controlled. The story should adapt to the user's interactions rather than forcing the user to follow a particular storyline.

The user model will ideally be used to identify which dilemmas are going to be most conflicting and dramatically interesting for the current user. There will be an "interestingness" value associated with each dilemma. This value is initally fixed but will ideally adapt to suit the user and their modelled personality. The system will search for the most interesting story path to a pre-defined fixed depth (dependent on the size of the search space and the speed of the search and planning algorithms).

Each dilemma has associated assumptions as to how the modelled values will change dependent on the user decision. Once they have made their choice, the user model will be updated accordingly. A selection probability will be associated with each criterion, so that the credibility given to the user model will depend on how many times it has been updated. It may additionally depend on how recently the criterion being utilised was updated – since the user and their opinions may well change through the course of the interactive drama. This user model will then be employed to approximate the probability of a user making a particular choice within a dilemma. It then calculates the expected total "interestingness" of that path. The system will present that dilemma which has the highest chance of leading to the most dramatically interesting experience for the user. A section of this search is shown graphically in fig. 3.

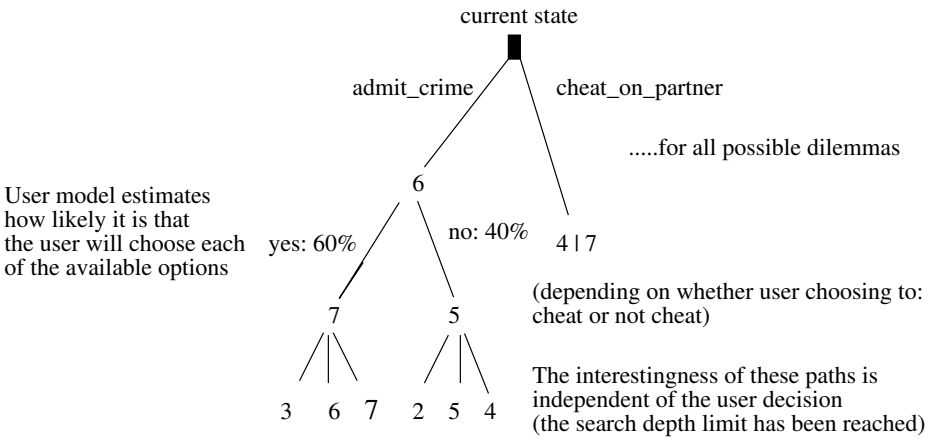


Fig. 3. This figure shows a section of a potential user model. The expected interestingness of each dilemma is as given. The highest prospected score of the admit_crime dilemma is 12.2. A similar calculation can be carried out for each path, and the most interesting subsequently selected.

In this story creation method, care must be taken to ensure that a single dilemma (or group of dilemmas) is not overused. In order to do so, the frequency of occurence for each dilemma (within the specified domain) must be considered.

Those aspects of the user which are modelled must be general enough to apply to a number of dilemmas, but also specific enough to accurately predict which choices the user is likely to make. Personality traits included in this general model include: honesty, responsibility_for_actions, faithfulness, strength_of_principle, selfishness, preference_for_relationship_or_friendship, strength_of_character and morality. There may also be domain specific factors. In addition, the user's opinions of other characters must be modelled, for example: value_for_relationship, value_of_friendship, value_of_love. Each of these criteria will have an associated value, which for certain criteria will take variables, for instance: strength_of_principle will take a separate value for each principle; value_of_friendship and value_of_love will take a separate value for each character within the storyworld.

7 Example

The techniques discussed here are applicable in any genre which places a particular emphasis on stereotypes and clichés. It was decided to intially focus on the creation of an interactive soap. This domain does not require an overall story arc but rather involves an infinite series of 'mini-stories'.

The domain of soap operas is commonly understood to revolve around stereotypical storylines. In many cases, these will involve a character being presented with a decision likely to result in negative outcomes either way. A range of such dilemmas from Neighbours, Home and Away, Coronation Street, Eastenders and Hollyoaks have been identified and generalised. These soaps were selected for their accessibility, familiarity and popularity with the general public.

For each soap, a series of dilemmas which characters had faced in recent years were identified[1]. It was found that these dilemmas fell into only three of the five possible categories, namely Betrayal (1), Sacrifice (2) and Favour (5). Figure 4 shows these dilemmas, which are generalised in fig. 5.

All domain specific background knowledge was added to the system, including STRIPS-style actions (such as why two characters may fall in love) and locations (for example club and house) which appear in the considered soaps. In fig. 6 a Prolog action from the system is shown with its pre- and postconditions.

Once a plan for a dilemma has been found, the system will execute this dilemma. In order to do so, the current state must be updated in accordance with each action in the plan. These actions are shown to the user, so that they understand the dilemma and will consequently be more interested in it. Figure 7 shows the build-up to and presentation of a sample dilemma to the user.

[1] Thanks to George Barber for his knowledge of soaps and ability to identify such dilemmas.

Hollyoaks: Becca has the opportunity to cheat on her husband Jake with Justin, a schoolboy in her class.
Eastenders: Jane has to decide whether or not to cheat on her husband Ian with the local bad boy Grant.
Coronation Street: Danny has the opportunity to cheat on his wife with Leanne, his son's girlfriend.
Home and Away: Kim has to decide whether or not to cheat on his girlfriend with his best friend Hayley.
Neighbours: Stu has the opportunity to cheat on his institutionalised wife Cindy with a local pretty girl – who previously went out with his brother.

Fig. 4. As can be seen from this small sample of similar dilemmas, the plotline of a character being presented with a dilemma involving cheating on their partner has been used in all of the examined soaps. This demonstrates the frequent use of clichéd storylines in soaps.

```
A_X: cheat_on_partner(character(X))
preconditions: partners(X,Y) ∧ loves(X,Z) ∧ loves(Z,X)
dilemma: ‘‘Would you like to cheat on your partner character(Y) with
character(Z) who loves you?’’
if user chooses to cheat:
add to state: cheating(X,Y,Z)
update user model:
honesty - lowered, faithfulness - lowered, value_for_relationship with Y -
lowered
if user chooses not to cheat:
delete from state: loves(X,Z)
update user model:
honesty - raised, faithfulness - raised, value_for_relationship with Y -
raised
```

Fig. 5. A dilemma of type Betrayal which is frequently used in soaps (see fig. 1), and can be presented to the user of this interactive drama system.

```
can(fall_in_love(Character,Other),[loves(Other,Character)],__):-
attractive(Character,A), A \== 1, attractive(Other,1),
character(Character), character(Other), Other \== Character.
adds(fall_in_love(Character,Other), [loves(Character,Other)]).
deletes(fall_in_love(__,__),[ ]).
```

Fig. 6. An action in which any characters in the system can participate. Here, an attractive person is in love with someone less attractive. In a soap world (where looks are very important) the less attractive character will fall in reciprocal love with the more attractive.

```
(a)
Background knowledge (includes): character(adam), attractive(adam,0),
character(joe)
Current state (includes): partners(joe,eve)
User model (includes): character(eve), attractive(eve,1), loves(eve,adam)
(b)
Action: fall_in_love(adam,eve)
You are going out with joe. But you are also in love with adam who loves
you. Will you cheat on your partner joe with adam?
```

Fig. 7. The necessary background knowledge and state elements are given here (a), prior to the build-up to and presentation of a user dilemma (b).

8 Related Work

Other interactive drama systems in existence use planning techniques. Mimesis [1] uses planning to achieve the story goals. This is much longer-term planning and is less flexible around the user's interactions - which will either be accommodated in re-planning or intervened with. In the I-Storytelling [3] system, hierarchal task network (HTN) planning is used. Each character is equipped with an HTN to follow in the story, which is defined before the story begins. There is very little allowance for user interactions in this system. In neither system is there any allowance for the story to be dynamically created, but only for it to be dynamically adjusted.

Fairclough's [5] utilises planning techniques to dynamically create an interactive story in the fairy tale genre. There are a finite number of subplots and the user's actions determine which is experienced. A plan is then created for the subplot, which consists of a "sequence of character actions" given to the NPCs as goals. The user has a high level of freedom but they are not entirely flexible as they must adhere to a limited number of subplots. In contrast, the system proposed here will allow the user complete freedom. The user is also modelled so that the experience is more enjoyable for them personally. The dilemmas posed to the user in our system will increase the dramatic interest of the stories.

Other systems utilise a user model. In IDA [8] this is used only to direct the user within the story's pre-defined overall plot structure. IDtension [11] uses the user model to determine the user's nature and present dilemmas accordingly. In this system, the user takes turns with the system to choose actions for the story as a whole. If they are modelled to consistently choose actions which avoid violence, the system can present them with a dilemma in which they must choose a violent action in order to achieve the pre-defined goals of the story. The dilemmas here are for the user as an external observer of the system, rather than as a character.

9 Conclusions

In this paper, we presented an interactive narrative generator that is able to create long, and potentially infinite, story lines that incorporate dilemmas to add

dramatic tension. The stories are dynamically created based on player decisions, and adapt to the player's tendencies.

In future work we plan to increase the interactivity of the system, giving the player the opportunity to influence the story between dilemmas with their actions. Also, dilemmas could be selected according to their frequency of use in the domain currently studied. There is also the potential for the creation of soap-specific dramas, with characters as in real soaps, for example an interactive *Eastenders* soap. The user model will be extended, to assess interestingness value of dilemmas to the individual user.

It is ultimately intended that these interactive drama worlds will be graphically simulated. In this way the user will see the storyworld as in conventional media but will be a character, and will be able to act as such. In the short term basic text descriptions will be used, which may be translated into their pictoral representation.

References

1. Liquid narrative. http://liquidnarrative.csc.ncsu.edu/people.php. The mimesis project was part of this group's work, with a wide range of collaborators.
2. Oz project. http://www.cs.cmu.edu/afs/cs/project/oz/web/oz.html. The Oz Project invovled a large number of group members at CMU.
3. Marc Cavazza and Fred Charles. Interactive storytelling. http://www-scm.tees.ac.uk/users/f.charles.
4. Chris Crawford. Erasmatron. http://www.erasmatazz.com/.
5. Chris Fairclough. *Story Games and the OPIATE System*. PhD thesis, University of Dublin - Trinity College, 2004.
6. Syd Field. *The Screen-writer's Workbook*. Dell Publishing, New York, 1984.
7. Barbara Hayes-Roth. Interactive theater. http://www.ksl.stanford.edu/projects/cait/. Barbara Hayes-Roth led this research group.
8. Brian Magerko. Interactive drama architecture. http://www.eecs.umich.edu/ magerko/research.
9. Michael Mateas and Andrew Stern. Façade. http://interactivestory.net.
10. Nikitas Sgouros. The defacto project. http://www.cslab.ece.ntua.gr/ defacto/default.htm. There were many other collaborators on this project.
11. Nicolas Szilas. Idtension project. http://www.idtension.com.

Using Decision Theory for Player Analysis in Pacman

Ben Cowley, Darryl Charles, Michaela Black, Ray Hickey

University of Ulster, Cromore Rd, Coleraine, Northern Ireland
{cowley-b, dk.charles, mm.black, rj.hickey}@ulster.ac.uk

Abstract. Computer and videogames have been described using several formal systems – in this paper we consider them as Information Systems. In particular, we use a Decision Theoretic approach to model players dynamically in real-time Pacman (Namco 1980). The method described provides low-level in-game data capture which can provide a simple metric of challenge and player skill, which are key components in measuring the optimality of player experience based on Flow theory. Our approach is based on the calculation of optimal choices available to a player based on key utilities for a given game state. Our hypothesis is that observing a player's deviation from an expected path can reveal their play preferences and skill, and help enhance our player models. Improved models will then enable in-game adaptation, to better suit individual players. In this paper we outline the basic principle of this approach and discuss the results of our first experiment.

Keywords: Decision theory, player modelling, formal methods, Pacman, Flow.

1 Introduction

There are several definitions of play in the literature from a variety of sources such as ludologists [1], game designers [2] and game studies researchers [3]. However, the academic study of commercial computer and video games (hereafter called games) is still a very new area and while we are coming closer to an agreement on the nature of play there is still much work to do. In particular, more study is required into the formalisations of the mechanics of interaction between a player and game. In this paper we focus on formal game and player analysis, with the aim of enabling more effective in-game adaptation of agents, entities and interactions.

We investigate modelling and predicting player behaviour from in-game data, starting at a low level of analysis and building inferences using formal methods. Our approach is inspired by considering games as Information Theory systems [3], and especially the principle that we can use Decision Theory [4] to model the choices that players make on the basis of available information. The decision theoretic approach that we take is *descriptive* rather than *prescriptive* because although we can quite accurately calculate near-optimal choices that a player should take given the current game state and the most obvious and observable utilities, players generally do not play in an optimal way. There are two main reasons why players will play in a non-optimal way. Firstly because they may not be skilled or experienced enough to work out the optimal choice for a given game state in real-time – in fact the dynamic

characteristics of a game can pressurise even a skilled player into making non-optimal decisions. Secondly, players will play with their own characteristic styles for aesthetic reasons, because game strategies tend to be worked out individually, and due to personality traits of different players. Hence, an analysis of the occasions and reasons why a player does not follow an expected path as predicted on the basis of the key utilities is perhaps as interesting as the decision theoretic approach in itself.

Although it is not unusual to investigate games in terms of Information Theory, its formulae or derivations (Decision and Game Theory) are not often applied in games in real-time. In fact there is limited existing research on real-time game analysis at the level of individual game elements and their interactions, based on formal methods. So this paper addresses two difficult issues; the development of real-time adaptive mechanisms that respond on the basis of dynamic game data, and the construction of real-time player models that are reflective of different player types. We outline the first stage of our research by implementing our player modelling methodology in a testbed game – the Pacman arcade platform game – it is anticipated that the same testbed will be used in future work. In this first attempt we begin to examine the role information plays in games, and how it relates to play performance. The basic principle of our approach involves predicting player actions and comparing the results with actual performance. These predictions form a model, and over many instances of gameplay this model would help enable adaptation – Fig 1.

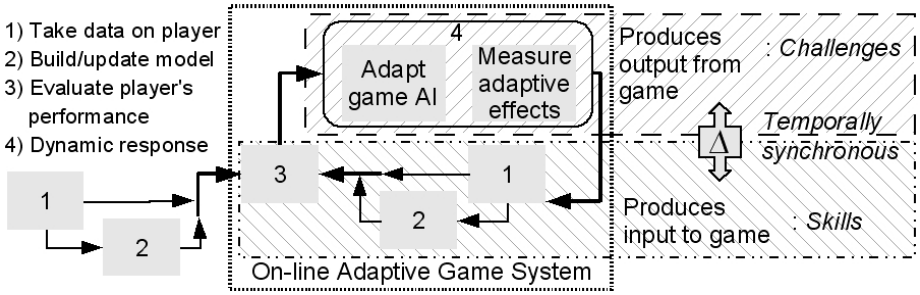


Fig. 1. A schema diagram of the game engine for adaptivity and automated Flow evaluation.

We proceed in this paper, reviewing existing player analysis research in Section 2, and in Section 3 describe the Pacman game and how it breaks down to fit a Decision Theoretic formula. Section 4 details the results of experiments on test subjects.

2 Modern Games and Player Modelling

Most modern commercial video games distinguish between players in a shallow manner. For example, it is common practice to offer players a choice of distinct difficulty levels, through which advancement is pre-set and linear. This practice assumes a player model based on "an ideal user and so some players may feel some what discontented when they advance in a manner that is counter to the ideal" [5].

To account for the variations between players during gameplay requires both dynamic player modelling, and adaptation based thereon. We propose an approach to

online data capture of gameplay that incorporates formal analysis [6] and heuristics of human cognitive processes [7]. The methodology implemented in this paper is the first, lowest-level stage of this approach, and so relies more on the former. We believe it is important to have a data-capture structure based – at the very lowest level – on proven formal techniques. From this low-level information, a low-complexity player model can be constructed. In further work, we envision utilising techniques from cognitive science in a more accurate, predictive player model.

Variation between players can be seen in Electronic Arts and iHobo audience models of gamer 'clusters' with differing skill levels and lifestyle priorities for games [8]. These levels of game playing ability, combined with personality types, have been refined into player types in models such as Demographic Game Design (DGD) [8] (based on Temperament Theory and the Myers-Briggs Type Indicator). Such models show us that gamers have differing attitudes and personality qualities such as attention span, temper, reasoning faculties, and spatial skills. There are also differing game types, such as the four types of Caillois – agon (competitive play), alea (games of chance), mimicry (impression of altered identity) and illinx (sensation of vertigo) [9] – for each of which gamers will have differing preferences.

This variation seems to be of low priority in the industry and developers often appear unconcerned with broad accessibility for their games. However, accessibility should be designed in from the beginning, as many of the games which are afforded vast budgets have no potential to tap the higher sales figures – because they only feature one mode of play. For instance, the majority of games revolve around agon – the opportunity to compete against an opponent (real or virtual) on a level playing field. However, most of the all-time top-selling games contain multiple modes of play (e.g. Halo) – and this was achieved with massive budgets and world-class game design. We hold that multiple modes of play could be easier implemented with adaptive player modelling.

We recently proposed [10] an approach to adaptive player modelling, based on the Experience Fluctuation Model (EFM – Fig. 2) [11] taken from Flow Theory [11], to describe and measure players' (optimal) experience. Flow theory proposes that the key attribute of optimal experience for an individual is a balance between challenges

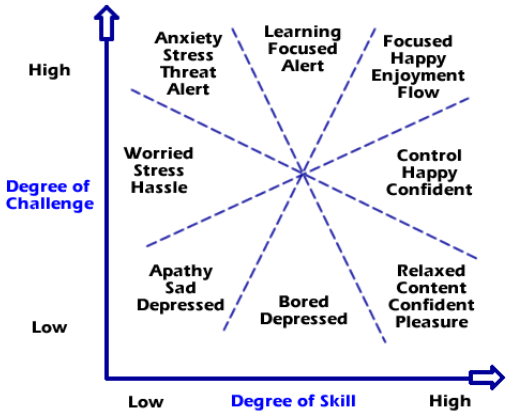


Fig 2. Experience Fluctuation Model diagram [11].

perceived in the experience, and the skills deployed respectively. The EFM relates to the current abilities of the subject, which is why it is possible to experience apathy. Flow views optimal experience as autotelic and task/goal-oriented, similar to gaming.

Our current aim is to provide a low-level building block to be used in modelling players so that a game's play structure may automatically adapt to the preferences and skills of each individual player. The motivation is to improve the experience of mainstream game playing, and to broaden the appeal of games generally since the mainstream gaming audience has a rather narrow demographic.

2.1 Current Data Capture Techniques for Game Players

Much modern work on player-centeredness for games owes something to the pioneering work of Malone [23]. Narrowing the focus to data capture for the purpose of player modelling, two broad categories cover most of the work done:

- 'In-game' – i.e. data generated and processed within the game engine.
- 'Observational' – i.e. physical measurement and sensing of the player.

In the first category, the methods mostly look at numerical data describing the player's avatar, generated by the game engine and reflecting player performance. So in a First Person Shooter (FPS), the method may consider shot accuracy, kill count, death count, and use of tactical techniques such as circle strafing, jumping and special weapons (such as smoke grenades).

Examples of this kind of work include Ryan Houlette in AI Game Programming Wisdom II [12], Scott Miller's methodology for Auto Dynamic Difficulty from Max Payne [13], and Robin Hunicke's Hamlet system for Dynamic Difficulty Adjustment in FPS games [14].

The first two methods are entirely game specific and are aimed at game designers. They require subjective definitions of the player model attributes, and thus such models wouldn't translate across games. Hunicke's application analyses specific gameplay elements related to player performance in an FPS setting, such as player health, ammo level, and the recharge 'pickups' available around the game world. Hamlet provides a quite generic approach, but to a single game genre only.

In the second category, physical sensing of the player during play is obtained from one or more specialist devices – this type of work is usually labelled 'affective'. Sensors measure player attributes including Galvanic Skin Response, facial reactions (after the work of Paul Ekman [15]), and gaze direction tracking.

Affective computing is studied in several high-profile labs worldwide, including the eMotion lab at Glasgow Caledonian University [16], the Affective Computing Group at MIT [17] and the M.I.N.D. Lab Finland at Helsinki School of Economics (HSE) [18].

The methodology detailed below falls into the first category, as it uses purely in-game data to generate a predictive model of player actions at a low level. Work which relates to both categories includes Yannakakis et al [19], where the gaming platform was used to both measure players' affect and build a player model based on the actions of the player avatar (although 'avatar' is only loosely applicable here).

3 Decision Theory Applied to In-Game Player Actions

Pacman is a fairly linear, simple game from the perspective of modern computer games, however there are still subtleties. Below we describe how the game was broken down into its compositional elements and fitted into the Decision Theoretic utility calculation formula. This enabled our implementation of the formula within the Pacman testbed code. In the following, 'the player' and 'Pacman' are interchangeable.

In all versions of Pacman the goal is to move around the game level and obtain all the collectables, thus progressing to the next level. Points are awarded for collectables and eating Ghosts. Our version of Pacman's gameplay may be broken down as follows (first caps are used to describe game entities and actions):

- The game world is a 2D graph, where the nodes of the graph can be Fruit, Pill, Dot or Empty. This constitutes a level.
- Pacman and the Ghosts Move between nodes, along two axes (↔↕).
- Ghosts move randomly, starting in a central box. They are controlled by pseudo-random numbers generated using the rand() function seeded by the system clock
- Pacman's Move becomes Move and Eat when there is an entity in his path.
- Pacman only Eats Ghosts when he has Eaten a Pill within the last t cycles, otherwise Ghosts Eat Pacman and he loses a Life.
- Whomever is Eaten respawns at their original start node, unless Pacman has run out of lives, then the game is over.
- Pacman must Eat all Fruit, Pills and Dots to finish, whereupon, the level ends whether or not that collectable was a Pill (i.e. the t cycles do not apply).
- Fruit gives an extra Life – there is 1 piece per level, it remains in one spot until eaten & doesn't return after it's eaten, differing from Namco's Pacman game.
- Ghosts can eat the Fruit before Pacman, so it is important Pacman eats it quickly
- Ghosts are permeable and do not interact with Dots/Pills, or obstruct each other.

This is a description of Constituative rules [3] – the formal rules system that underlies the game mechanics [20]. Our approach uses descriptive Decision Theory to model the Constituative rules, and by this method approximately model the Operative description – i.e. how the player sees and interacts with the game.

To see why we use Decision Theory, consider the situation where Pacman is unconstrained in a level – for example if there were no Ghosts. Time is not a factor in Pacman, so the path chosen by the player is unimportant, as any explorative path will result in the same utility – collecting all the dots. There are no variables. For the sake of argument, let us say that the player calculates and travels the shortest path, which in a level where each dot was adjacent to exactly two others would constitute a Eulerian tour of the graph of all edible items on the screen. If Pacman could follow this tour, then there are no decisions to be made (since the level ends when the tour does). Divergence is forced by higher utility *choices* – save lives by evading, or increase points by hunting Ghosts. Information Theory, and by extension Decision Theory, can be viewed as a formulation of the Uncertainty of outcomes due to non-trivial choices. The choices of where to diverge from the Eulerian tour are made by the player in real time. The current Information, from the game state and the

heuristics on player utility, is used in our implementation to make predictions of those choices. So, within the game engine, we express the choice of which nodes it is possible to move to, weighted by the utility of moving to each one, using the formal concept of Uncertainty. So we argue that Uncertainty corresponds to the difficulty of decision-making for the player on a move-by-move basis, and it provides a direct (if simplistic) index of challenge. We can fit the choices, and their utility weights, to a formula using Decision Theory. Thus we can quite accurately predict the behaviour of a rational, utility-maximising agent. The list of these predictions will give an example of an expected path. If a skilled player's actual performance is close to the prediction this would help validate our utility calculation algorithm.

A chain of the player's choices defines a plan. Decision Theory considers sets of choices, which equates to a plan. Corresponding planning by the prediction function should therefore be bounded by what is a reasonable number of moves ahead that the player could consider. We assume that the player 'chunks' game state information – i.e. they will not consider the 5 moves ahead, but all the moves up to an objective, plus the possible moves of the Ghosts, and the consequences of both. Then they can take meaningful actions, which are actions involving risk versus reward decisions. A player doing this kind of reasoning would be more likely to consider Ghost proximity than absolute location, so pathfinding (A*) has been used. In the Pacman testbed, the Information measured can be directly used to judge a simplified version of the challenge and skill involved, giving a rudimentary measure of Flow.

4 Results and Discussion

In this implementation of the Pacman game, as with most games, the mechanics of play are concerned with choosing the action which maximises a utility function, from a set of actions situated in an evolving state-space.

Our method follows [4] in formulating predictions in terms of a set *A* of possible plans $\{a_1, a_2, \ldots\}$, which comprise one or more actions. A plan considers a probability distribution **P**(*S*) over a state space *S*. In Pacman, the current state is known and the probability represents how this state may change. Our Utility function (*U*) maps the states to numerical output, $U \times S \rightarrow R$. Thus we can predict that a player should perform a plan a^* that maximises its utility by calculating:

$$a^* = ArgMax_{a_i \in A} \sum_{s \in S} p_i^j U(s^j) \quad \textbf{(1)}$$

Where p_i^j is the probability assigned by $P_i(S)$ to the state s^j. This formula gives us the capability for dynamic predictions of optimal utility movements in Pacman. However, implementing *U*, the utility function, will always be game-specific. In this game, eating the Ghosts affords far more points than eating Dots, x10 for the Pill and x20 for each Ghost. With 180 Dots, and 4 Pills, the best strategy gives thrice as many points for pills/Ghosts as for dots alone. Our implementation reflected this by defining 5 states requiring distinct utility calculations, which are described in Table 1.

Table 1. Utility Calculation States.

State	Description
1. Hunt	Pacman hunting Ghosts (occurs after Pacman Eats a Pill).
2. Cherry	Pacman Moving to Eat the Fruit (Ghosts can eat Fruit 1st)
3. Pill	Pacman in proximity to Ghosts and a Pill, and thus Moving toward the Pill.
4. Flee	Pacman in proximity to Ghosts but not a Pill, thus Moving away from Ghosts.
5. Dots	The default state, where Pacman looks for the largest number of dots locally and avoids proximity to Ghosts.

We conducted concurrent case studies on four subjects, with different levels of gaming experience. In each case 6 consecutive games of Pacman were played. Play stats were logged in real-time, and players not interrupted during the session which allowed a more natural style of gaming. Although it was a small sample size, we obtained results that reflect both variations in playing styles and the strengths and weaknesses of our method. Across all the case studies 51.095% of predictions were correct (that is: correct predictions x 100 / total predictions = 51.095). In half the cases, this correctness ratio was within 0.1% of 50%. The higher and lower ratios of correct prediction corresponded to the more and less experienced players, respectively. The range of correct predictions over cases was 48.134 – 53.125%.

Table 2 presents results for each of the states from Table 1. This includes how many of the predictions in each state were correct, the minimum and maximum number correct for each case study in each state, and the standard deviation among case studies in terms of correctness. The result 'Weighted Value' is the state's total number of predictions (i.e. relative importance in gameplay) times the difference between its ratio and the overall ratio of 51.095% (i.e. value as a discriminator).

Table 2. Results Overall (from 4 Case Studies) by Utility Calculation State.

State	% Correct	Weighted Value	Range: % Correct in Cases Studies	StdDev σ of Cases
1. Hunt	47.707 %	6355.888	33.3 - 56.9%	10.275 %
2. Cherry	60.109 %	1649.562	50 - 62.5%	5.748 %
3. Pill	53.658 %	105.083	41.2 - 100 %	24.913 %
4. Flee	59.887 %	3112.368	54.4 - 62.4%	3.535 %
5. Dots	51.375 %	370.16	50.4 - 53.2%	1.362 %

The number of predictions in each of the utility states (1-5, see Table 1), was significantly higher for the default state number 5, which had a percentage correct very close to 50% and didn't vary significantly across cases. More variation was seen in the other states, suggesting they would discriminate playing styles more effectively.

State.1 offers players the highest potential utility, along with high associated risk. Given the skill required to maximise utility in this state, it is interesting to see that the ratio of correct predictions in each case study increased along with the experience of the players studied. We concluded that although the prediction function is not that accurate, it does offer a metric of player skill and discriminates players well.

State.2 had the most correct predictions of all, which indicates that our method is most accurate when considering a well-defined fixed-location objective (in our implementation the cherry is constantly in one spot). It remained consistent for all cases also, indicating players generally saw the cherry as high utility. But with a low standard deviation, this state seems less valuable to illustrate individual player style.

State.3 was the least frequently entered state, indicating that the players liked to keep a healthy distance from the Ghosts even when a Pill is close by. The most points can be scored when there are Ghosts clustered near a Pill and Pacman can eat the Pill and chase them down. Not seeing this behaviour, we posit that this caution may be linked to the Ghost's erratic movement, which was simply based on generating pseudo-random numbers (between 0 and 3) for direction and thus may have instilled a false (i.e. not their natural play style) hesitancy in players.

State.4 saw a relatively high ratio of correct predictions, and a low standard deviation. However, given the simplicity of the state utility ("move away from Ghosts"), the predictions are still quite low. This may suggest that this state can't be judged outside the context of possible higher-level strategies, since it's unusual for players to go near Ghosts unless they have good reason to do so. Their strategy is what will maximise their utility, not simply running away in the opposite direction.

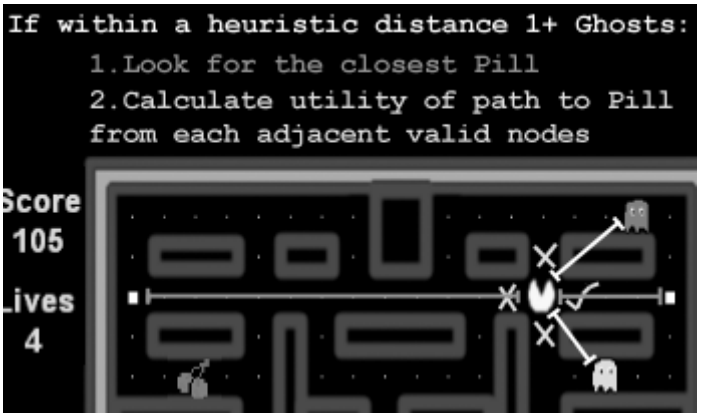


Fig.3 Pacman screenshot, overlaid with a schematic of the state.3 utility calculation algorithm.

Results showed that players varied greatly in how fast they played (i.e. ratio of their moves to game cycles). We believe that when this ratio is low it indicates pausing, and pausing corresponds to player experience when matched to high scores, as waiting for the optimum moment makes it easier to maximise their utility. In support, the average cycles and scores for experienced players were both around 1.5 times that of less experienced players. Yet their average number of moves was only 1.2 times greater, indicating longer game time was not due to a lot of extra movement. Since we know who was experienced and who less so, we can thus say that experience prompts players to play longer games to get higher scores.

In addition to the small sample size skewing results and non-consideration of high-level playing strategies, predictions had poor accuracy for two main reasons:

1. Pacman is constrained by walls so that he will always have either two, three or four directional options for movement. This means our algorithm, which only predicts one move ahead, has a 25% to 50% chance of being right *no matter how it arrived at that prediction.* A better approach would be to calculate the utility for several moves ahead at each game loop, weight move predictions inversely to their distance from the current position, and predict the move beginning the sequence which generated the highest (weighted) utility.
2. In each state except for State 2, the calculations of utility were not refined or precise enough. For instance, in the default state it was assumed the player would look for the greatest number of local Dots – however in tests players would turn away from large numbers of Dots to eat one or two isolated Dots, because they "wanted to clean up an area before moving on". Of course refining our method is the purpose of running these initial tests.

Observing the player differences in the results of the case studies provides a valuable tool for post-play analysis. However the results from this early implementation are not sufficiently accurate to be used for in-game adaptation. We believe that refinement of the utility function in future work would provide the accuracy necessary.

The method we have described in this paper provides a simple metric of player performance. Adaptation based on this measure would involve altering low-level game entities which affect or represent player utility – for example one could impose dynamic constraints on ghost movement that would decrease, or increase, Pacman's ability to score highly by eating them, depending on whether the method judges the player to be struggling or becoming bored respectively. Thus, the player will be forced to employ more insight and skill as their experience grows, generating more variety in the patterns of play. Variety of play, both intuitively and according to [21], should generate a more enjoyable game for an experienced Pacman player.

Initial results suggest that while we can specify a plan as a short series of moves ending in an objective, often the player will be considering a much wider picture, involving more moves than it is practical to calculate a utility for. Plans of this nature, which we would call *strategies*, would require a higher level of abstraction of representation in code. This means they go beyond the scope of the current method, but might be susceptible to predictions using something like Case Based Plan Recognition (CBPR) [22]. If the player's reaction to our adaptation could be formulated as a plan, then we can envision another level of adaptation using CBPR, where the observation of such a plan would imply a high skilled player, and thus prompt a higher level of difficulty, either in the current level or in subsequent ones.

5 Conclusion

We have described a method for in-game real-time analysis of player performance, which is based on formal methods derived from Decision Theory. It has been implemented using the Pacman game as testbed, and tested on players with a range of experience of playing Pacman. Though the implementation is at an early stage, results indicate that the approach holds promise. It is very useful for post-game analysis of play. Player models could be based on the real-time in-game analysis, with the aim of

enabling adaptivity. But currently, any model would be solely a data gathering exercise, as adaptivity requires a much higher accuracy of results.

In future work we hope to refine the utility prediction functions to give more accurate results in longer experiments with more players. We aim to use this method for player modelling contributing directly to real-time gameplay adaptivity. Such adaptivity is a key part of the future of videogame artificial intelligence (AI), as players look for more realistic and challenging opposition from their games.

References

1. Huizinga J, Homo Ludens: a Study of the Play Element in Culture, Temple Smith, 1970.
2. Crawford C, Chris Crawford on Game Design, Prentice Hall PTR, 2002.
3. Salen K, Zimmermen E, Rules of Play : Game Design Fundamentals, The MIT Press, 2003.
4. Gmytrasiewicz P J, Lisetti C L, Modeling Users' Emotions During Interactive Entertainment Sessions, Proceedings of AAAI 2000 Spring Symposium Series. 20-22 March 2000.
5. Gilleade K M, Dix A, Using Frustration in the Design of Adaptive Videogames. ACE '04: Proceedings of the ACM SIGCHI, vol. 74, ACM Press, New York, NY, 228-232, 2004.
6. Lee W, Decision Theory and Human Behaviour, New York, (Chichester) : Wiley, 1971
7. Rauterburg M, About a Framework for Information and Information Processing of Learning Systems, Proceedings: Conference on Information System Concepts, 1995.
8. Bateman C M, Boon R, 21st Century Game Design, Charles River Media, 2005.
9. Caillois R, Barash M (translation), Man, Play and Games, London: Thames & Hudson, 1962.
10. Cowley B, Charles D, Black M, Hickey R, User-System-Experience Model for User Centred Design in Computer Games, 2nd Proceedings of The International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems, Dublin, Ireland, June 2006.
11. Csikszentmihalyi M, Flow: The Psychology of Optimal Experience, Harper & Row Publishers Inc., New York, NY, USA, 1990.
12. Rabin S (Ed.), AI game programming wisdom 2, Charles River Media, 2004.
13. Miller S, Auto-dynamic Difficulty, weblog, last accessed 22/02/2005, http://dukenukem.typepad.com/game_matters/2004/01/autoadjusting_g.html
14. Hunicke R, Chapman V, AI for Dynamic Difficult Adjustment in Games, Proceedings of the Challenges in Game AI Workshop, 19th National Conference on AI, 2004.
15. Ekman P, Emotion in the Human Face, New York: Cambridge University Press 1982.
16. Sykes J, Brown S, Affective Gaming: Measuring Emotion through the Gamepad, CHI '03 extended abstracts on Human Factors in Computing Systems, ACM Press 2003.
17. Ahn H, Picard R W, Affective Cognitive Learning and Decision Making: The Role of Emotions, 18th European Meeting on Cybernetics and Systems Research, Apr 18-19, 2006.
18. Ravaja N, Saari T, Laarni J, Kallinen K, Salminen M, The Psychophysiology of Video Gaming: Phasic Emotional Responses to Game Events, DIGRA, Canada, 2005.
19. Yannakakis G N, Lund H H, Hallam J, Modeling Children's Entertainment in the Playware Playground, in Proceedings of the IEEE Symposium on Computational Intelligence and Games, Reno, USA, May, 2006.
20. Hunicke R, LeBlanc M, Zubek R, MDA: A Formal Approach to Game Design and Game Research, Proceedings of the Game AI Workshop, 19th National Conference on AI, 2004.
21. Koster R, A Theory of Fun for Game Design, Scottsdale, AZ : Paraglyph Press, 2005.
22. Fagan M, Cunningham P, Case-Based Plan Recognition in Computer Games, 5th International Conference on Case-Based Reasoning, Trondheim, Norway, 2003.
23. Malone T W, What Makes Things Fun to Learn? Heuristics for Designing Instructional Computer Games. SIGSMALL '80. ACM Press, New York, NY, 162-169, 1980.

Where am I? – On Providing Gamebots with a Sense of Location Using Spectral Clustering of Waypoints

Christian Bauckhage[1], Martin Roth[1], and Verena V. Hafner[2]

[1] Deutsche Telekom AG, Laboratories
10587 Berlin, Germany
http://www.telekom.de/laboratories
[2] Technical University Berlin, DAI Labor
10587 Berlin, Germany
http://www.dai-labor.de

Abstract. Most computer games that are set in simulated 3D worlds require the player to act according to his or her current location. For the programming of convincing artificial game agents this poses the problem of providing them with a sense of where they are. However, due to the growing size and complexity of games, manual annotations that capture all essential areas and aspects may soon become infeasible. In this paper, we therefore propose a data-driven approach to the semantic clustering of in-game locations. Following the paradigm of gamebot programming by human demonstration, we apply the technique of *spectral clustering* to automatically derived waypoint maps. First results underline that this approach indeed provides spatial partitions of game maps that make sense from a gamers point of view.

1 Motivation and Background

Modern computing hardware and the technical sophistication of state of the art computer games apparently promise an exciting and fascinating gaming experience. Yet, from looking at prospects of the future of computer games, one is led to believe that consumers' craving for more immersive game play is far from being satisfied.

Alas, it is not entirely clear what makes an immersive game. While there are many concepts and opinions, immersiveness still eludes definition. Many experts, though, agree that game AI is an important factor and nobody seriously disputes that current game AI could need improvement. Consider, for instance, popular genres such as massively multiplayer online role playing games, tactical combat games, or team sports simulations. If the *gamebots* that inhabit these games were smarter and would act more plausibly or life-like, game play would be less mechanic. Non player characters with rudimentary cognitive capabilities would obviously allow for more dynamic interaction and thus enhance the gaming experience beyond what we are used to today.

Therefore, it is not surprising that industrial and scholarly interest in gamebot programming has rekindled. Game programmers and researchers alike have realized that the increasing complexity of modern games will soon make it impossible to manually program and maintain agent behaviors that would be compelling enough to guarantee a satisfactory playing experience. Consequently, research on novel approaches to game

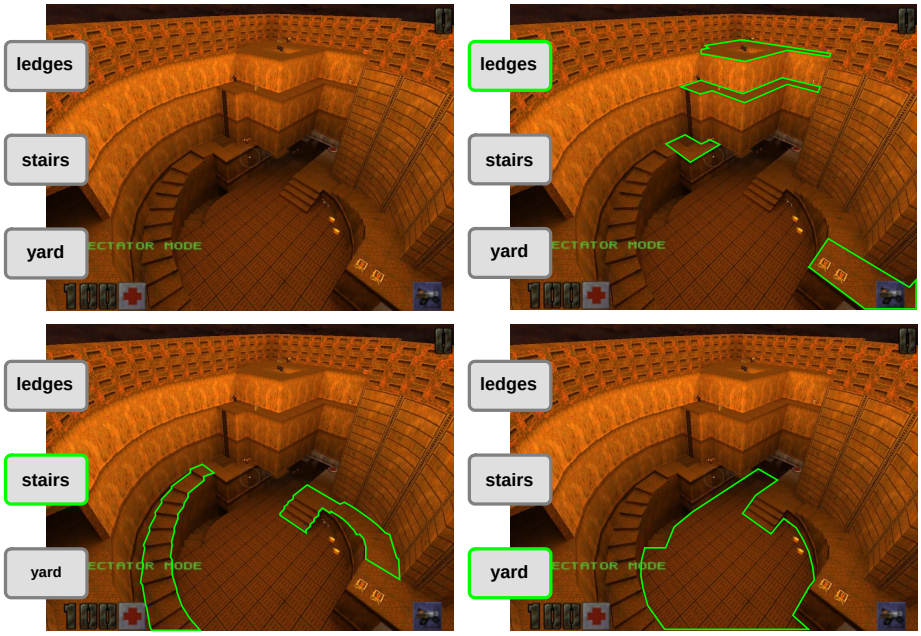


Fig. 1. Screenshots showing a prominent part of the popular QUAKE II® map *q2dm1*. During a match, each of the highlighted areas may require the player to move and behave differently.

AI has gained momentum. Recent work on more traditional gamebot programming, for instance, has investigated the use of inexpensive AI techniques or reusable behavior patterns [1, 2]. Pursuing less traditional approaches, other recent contributors have emphasized incorporating machine learning into game AI. Examples are the use of Bayesian- and Markovian decision processes [3, 4], statistical- and reinforcement learning [5–8], evolutionary computing [9, 10], as well as neurophysiologically inspired behavior synthesis [11].

Having investigated purely data-driven machine learning for gamebot design ourselves, we came to realize that there is a fundamental problem looming at the bottom of corresponding approaches to behavior synthesis. If artificial game agents are supposed to be more than merely *reactive* units, behavior learning and behavior synthesis must grapple with the fact that –at least in real life– behaviors reside on different layers of abstraction and happen on different time scales. While reactive behaviors are triggered according to a current situation and the immediate surroundings, *tactical* and *strategic* behaviors depend on larger spatio-temporal contexts, goals, desires, and needs of an agent. In contrast to reactive behaviors, tactical and strategic behaviors are therefore much harder to synthesize from what can be learned statistically. Most of the above cited works hence assume a catalog of basic behaviors and focus on training parameterized higher-level models that then generate complex behaviors from elementary ones. However, our experiences with the bottom-up paradigm are too promising to give up on

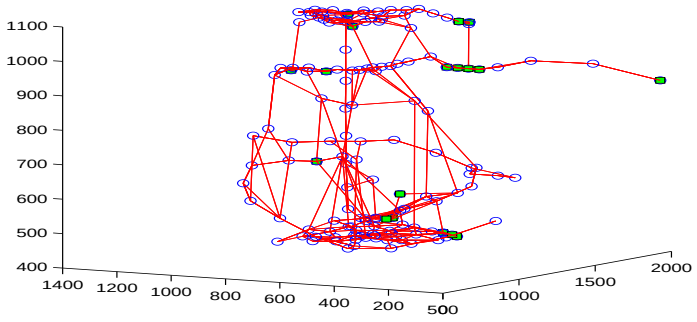
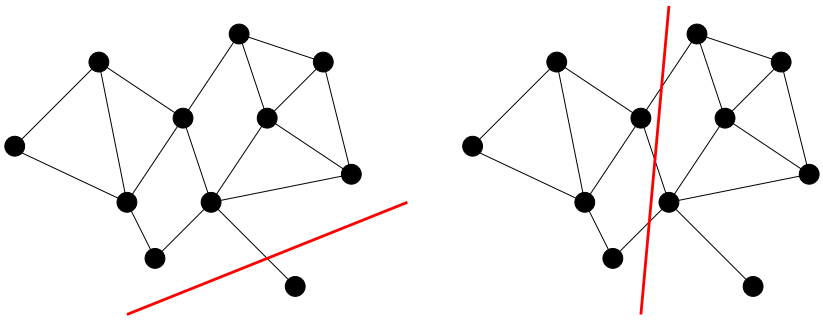


Fig. 2. Waypoint map of 150 nodes resulting from k-means clustering of player trajectories recorded during a duel on the map in Fig. 1. The upper ledges and the yard clearly pop out; green boxes indicate locations of item pickups. The recording player was trying to guard higher grounds. Expert maneuvers such as strafe jumping from the upper ledge to reach the health packs across the yard caused some waypoints to be situated 'in midair'

it. Of course, our goal, too, are gamebots that behave believably across a wide range of (possibly unforseen) situations. But instead of manually providing basic actions, we decided to explore how far the data-driven approach will carry and have therefore turned to *behavior mining* from recordings of human generated game data.

In this paper, we propose an approach towards behavior mining with respect to spatial contexts. Similar to our previous work on gamebots, we focus on the genre of first person shooters and consider learning from demo file recordings of the network traffic of QUAKE II® . In particular, we discuss automatically partitioning 3D game maps into semantical units. This was motivated by the expert notion that players will act differently depending on where they are in the game world. Consider, for example, the aerial view of a part of the QUAKE II® map *q2dm1* which is shown in Fig. 1. The highlighting in three of the panels indicates several topologically distinct areas. Even without a background in gaming but from common sense alone, one may suspect that these areas demand different behaviors. Moving along the ledges, for instance, will require more prudence than moving in the yard. However, while being in the yard, it is advisable to stay covered whereas standing on the ledges requires less care for cover but allows for easy map control. Therefore, if we had a means of identifying crucial areas automatically, we might train gamebots which would have a sense of location and would behave accordingly.

The idea we focus on in this paper is to apply *spectral clustering* for determining important areas. Since spectral clustering assumes the input data to be structured by a graph whose labeled edges indicate similarities of data elements, it almost naturally applies to *waypoint maps* so commonly used in games. Since we follow the paradigm of data-driven learning, we assume automatically derived waypoint maps. Indeed, using the k-means clustering procedure which is provided with the QASE programming in-



(a) $cut(A, B) \rightarrow$ min (b) $cut(A, B) \rightarrow$ min, where $|A| \approx |B|$

Fig. 3. A *cut* partitions a graph G into two disjoint subgraphs A and B. Its cost $cut(A, B)$ corresponds to the sum of the weights of edges that have to be removed. Simply looking for minimal cut sizes often leads to partitions where only single vertex is split off. More sophisticated cut criteria impose constraints such as balancing the sizes of the resulting subgraphs.

terface [7] for QUAKE II®, the computation of maps as in Fig. 2 is simple and straightforward.

Next, we briefly summarize the theory of spectral clustering and then discuss some exemplary results.

2 Synopsis of Spectral Clustering

Since too detailed a discussion of different techniques of spectral clustering would take us too far afield from the scope of this paper, we just briefly present the basic idea where we focus on providing an intuitive understanding of the method. For more thorough but still readable treatments, the reader may refer to [12–14].

In spectral clustering, the data that ought to be clustered are supposed to form the set of vertices V of a graph G. Weighted edges between any two such vertices indicate similarities. In practice, the weights are gathered in a matrix $\boldsymbol{S}$ where $s_{ij} = s(v_i, v_j)$ denotes the similarity of the data in vertices v_i and v_j.

This closely resembles a *waypoint map* used for path planning in computer games. The QASE API, for instance, computes waypoint maps by k-means clustering of demo files. This results in k prototypical player positions $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\}$ which are assigned to the vertices $v_1, \ldots, v_k$ of a graph. Edges are drawn, if the recording contains transitions between the corresponding waypoints. Finally, Floyd's algorithm is used to compute distances d_{ij} (given in terms of shortest path lengths) between any two vertices. The only difference between a similarity graph and a waypoint map thus lies in the use of distances and similarities. For our purpose, we therefore normalize distances $d_{ij} \in \mathbb{R}^+$ to distances $\tilde{d}_{ij} \in [0, 1]$ and define similarities as $s_{ij} = 1 - \tilde{d}_{ij}$.

A *cut* partitions a graph G into two disjoined subgraphs A and B by removing those edges that connect the subgraphs. From Fig. 3 we see that there may be numerous possible cuts; With respect to clustering, this raises the question of how to determine a suitable one?

A popular criterion was proposed by Shi and Malik [12]. In order to understand their idea of a *normalized cut* of G, we need to consider the *degree* of a vertex v_i

$$deg(v_i) = \sum_{v_j \in G} s_{ij} \tag{1}$$

and the *volume* of a subgraph $A \subseteq G$

$$vol(A) = \sum_{v_i \in A} deg(v_i). \tag{2}$$

While the cost of a simple cut can be characterized by

$$cut(A, B) = \sum_{v_i \in A, v_j \in B} s(v_i, v_j) \tag{3}$$

for the normalized cut, we have

$$Ncut(A, B) = \Big(\frac{1}{vol(A)} + \frac{1}{vol(B)}\Big) \sum_{v_i \in A, v_j \in B} s(v_i, v_j). \tag{4}$$

Thus, while the costs of simple cuts correspond to the combined weights of the removed edges, costs of normalized cuts are normalized by the total weights of the resulting subgraphs. Clustering according to the normalized cut criterion then simply requires finding a partition of minimal costs.

Although minimizing $Ncut$ is NP hard, a good approximation can be found efficiently. In [12], Shi and Malik show that by introducing an *indicator vector* $\boldsymbol{y}$ where

$$y_i = \begin{cases} > 0, & v_i \in A \\ < 0, & v_i \in B \end{cases} \tag{5}$$

an approximate minimum of (4) results from minimizing the *Rayleigh quotient*

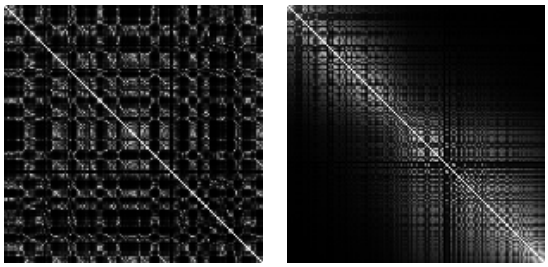
$$J(\boldsymbol{y}) = \frac{\boldsymbol{y}^T(\boldsymbol{D} - \boldsymbol{S})\boldsymbol{y}}{\boldsymbol{y}^T\boldsymbol{D}\boldsymbol{y}} \tag{6}$$

where $\boldsymbol{D}$ is a diagonal matrix formed by the degrees of the vertices. Minimizing $J(\boldsymbol{y})$, however, is tantamount to solving the generalized eigenvalue problem

$$(\boldsymbol{D} - \boldsymbol{S})\boldsymbol{z} = \lambda \boldsymbol{D}\boldsymbol{z}. \tag{7}$$

The eigenvector $\boldsymbol{z}^F$ belonging to the second smallest eigenvalue λ^F is called the *Fiedler vector*. It allows for a bipartition of the graph, since there exist $\alpha, \beta \in \mathbb{R}$ such that

$$z_i^F = \begin{cases} \approx \alpha, & v_i \in A \\ \approx \beta, & v_i \in B. \end{cases} \tag{8}$$



(a) unordered waypoints (b) ordered waypoints

Fig. 4. Two similarity matrices resulting from the waypoints in Fig. 2.

If more than two clusters are sought for, spectral clustering may recurse on the resulting subgraphs A and B. The recursion can be repeated until the desired number of clusters has been found or another suitable convergence criterion is met.

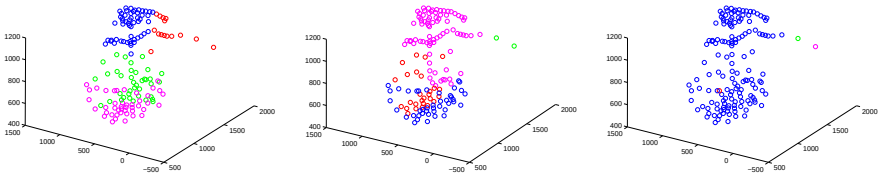
In essence, we see that, in each recursive step, the method is looking for implicit block structures in the similarity matrix $\boldsymbol{S}$ and clusters the graph accordingly. Figure 4 exemplifies this. It shows two similarity matrices computed from the waypoints in Fig. 2; the higher the similarity of two waypoints $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, the brighter is the corresponding matrix entry s_{ij}. Although the matrices appear different, they are in fact equivalent. The difference is due to didactic purposes: before computing the right matrix, we sorted the waypoints with respect to their distance to the origin. Thus, in contrast to the unordered set of waypoints yielded by the QASE API, topologically similar waypoints now more likely had similar indices, too. Consequently, for the right matrix, the block structure inherent to the data becomes apparent. However, since the right matrix is a mere permutation of the left one, both matrices have the same eigenvalues; for the task of spectral clustering it is therefore irrelevant if the waypoints are sorted or not.

3 Practical Performance and Discussion

In this section, we will present and discuss results of applying spectral clustering to waypoint maps. In doing so, we will concentrate on a particular example. This is not because we found that the proposed method did not work on other data, but because we feel that this example is well suited to point out the advantages spectral clustering offers for automatic area identification. Not surprisingly, our example is the central yard area of *q2dm1* which was introduced in Fig. 1.

In all experiments reported below, we considered the waypoint map of 150 nodes (see Fig. 2) which resulted from a recording of a duel between two players.

Let us first have a look at how other popular clustering algorithms perform for the task at hand. Figure 5 depicts results obtained from k-means, ward-linkage, and single-linkage clustering where each method was parameterized to produce 4 distinct clusters. In terms of a reasonable partition of the map into strategic or tactical areas, the quality of results obviously decreases from left to right.



(a) k-means (b) ward-linkage (c) single-linkage

Fig. 5. Results of clustering the waypoint map in Fig. 2 into 4 areas using the k-means, the ward-linkage and the single-linkage algorithm, respectively. From a gamers point of view, ward- and single-linkage fail to produce a reasonable partition of the map. k-means clustering does a much better job. However, its way of splitting the upper ledges does not reflect the map's topology.

k-means clustering aims at producing a Voronoi tessellation of the data space, i.e. it tries to place prototypes into the data such that the average distance to one of these becomes small for all data points. In general, and for our example, too, this leads to a partition of the data into more or less equally sized Voronoi cells. In our example this does not produce a semantic partition of the waypoints. The upper ledges, for instance, were horizontally split into two clusters and not along the more logical vertical direction. Also, the waypoints in midair, some of the points in the yard and some points on the stairs were fused into an amorphus cluster.

Ward's linkage is a hierarchical approach that tries to minimize the increase of variance after merging two clusters. For our example, we see that this leads to blob-like structures similar to the k-means result but of even less semantic quality.

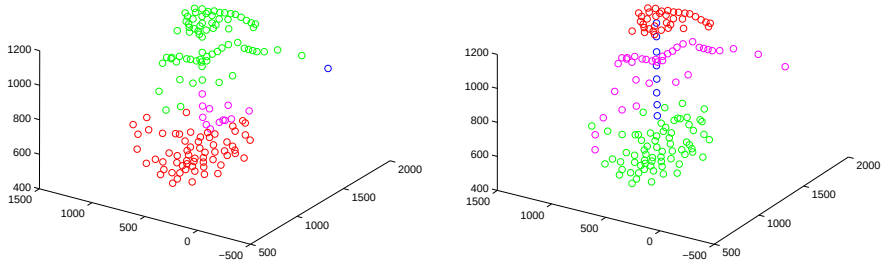
Single linkage clustering tries to minimize the minimum distance between the elements of a cluster. As we can see, for our waypoint map this leads to almost no clustering at all. Apparently, the criterion was fulfilled by producing a 'giant' containing all but three points and three 'dwarfs' consisting of a just single point each.

Note that all three methods only consider the coordinates of data points and do not care about similarities or distances among them. This is different for spectral clustering. Because of this, it is able to detect intrinsic structures and hence may be more appropriate for our purpose. And indeed, our next experiments will show that, in contrast to the above methods, spectral clustering leads to much more meaningful partitions of waypoint maps and game worlds.

Figure 6(a) shows 4 areas resulting from spectral clustering based on point similarities which were computed from the points' Euclidean distances:

$$s_{ij} = e^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}; \quad \sigma = 10. \tag{9}$$

Here, the upper ledges and the yard form prominent clusters. Still there are artifacts such as the amorphous cluster in the center and the solitaire point on the far right. Not until we consider similarities according to topological distances does a clearer picture appear.



(a) euclidian similarities (b) topological similarities

Fig. 6. Results of using spectral clustering for partitioning the waypoint map in Fig. 2 into 4 areas.

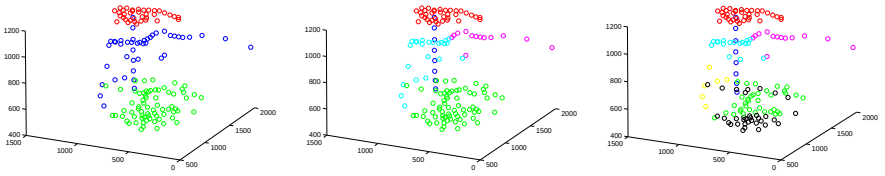


Fig. 7. Results for partitioning the waypoint map in Fig. 2 into 3,5 and 7 areas.

The similarity matrix used to produce the result in Fig. 6(b) was derived from path lengths between waypoints. In this case, the upper two ledges were correctly clustered into two vertically separated parts; the yard and the surrounding stairs form an area and even the elevator which leads from the yard to the uppermost ledge pops out. Especially the latter underlines and exemplifies the power of the approach: The recording player only used the elevator in the upward direction. Since on *q2dm1* this elevator only works in that direction, this behavior could be expected from an experienced player. Going down the elevator chute would mean to fall down several virtual meters of altitude and therefore would cause considerable self-damage. Since the player moved (more or less) freely in the yard and on the uppermost ledge and there is not much space for moving in the elevator itself, topologically, the elevator forms a thin bridge between two larger, highly connected chunks of waypoints. With spectral clustering such semantically grounded clusters as well as bridges in between can be identified. Topology also explains why the waypoints in midair are clustered with the second highest ledge. First of all, the recording player mostly jumped from there. Second of all, as it is impossible to reach the midair points from the ground level their similarity to the points on the ledge is much higher than to the points in the yard even if some of latter might be closer geometrically.

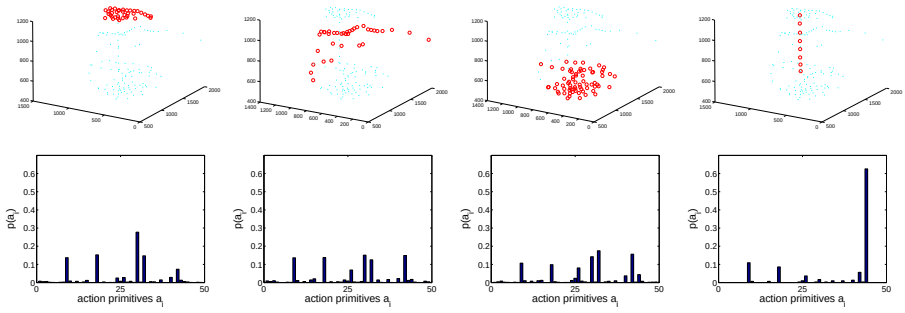


Fig. 8. Distributions of action primitives for the clusters in Fig. 6(b). Although the distributions do not disclose anything about temporal sequencing of primitives, it is obvious that different behaviors were executed in each area. While the distributions for the two clusters in the middle of this figure are somewhat similar, clearly different action primitives governed the movements on the uppermost ledge and in the elevator leading there.

Finally, Fig. 7 indicates that this favorable characteristic of spectral clustering is not an artifact of the particular choice of 4 desired clusters. From clustering into fewer areas (e.g. 3) as well as from clustering into more segments (e.g. 5 and 7) we obtained meaningful structures, too.

4 Possible Impact on Behavior Learning

Given the above findings, it remains to train gamebots that show behaviors which are adapted to certain areas of a map. Unfortunately, as of this writing, we have not developed a corresponding gamebot which we could present in the remainder of this paper. Instead, we will briefly sketch first results which prove the intuition that expert human players behave differently on different parts of a map and also indicate that location-based behavior learning will be possible soon.

The results are based on the idea of *action primitives*. Evidence from neurobiology hints that, in mammals, motion planning and execution happen by superimposing and sequencing elements from a set of prototypic motion patterns most of which were learned during infancy. In [11], we already demonstrated that the concept of action primitives may also be applied to the problem of life-like motion synthesis for computer game characters. Based on that experience, we computed a set of 50 action primitives from recorded demo files and examined to what extend these were active for the 4 different clusters in Fig. 6(b).

Figure 8 plots the distributions of the 50 primitives for each cluster. Although this kind of plot neglects temporal sequencing and superposition of action primitives during behavior execution, the distributions clearly prove that the recording player behaved differently in each of the 4 automatically determined areas. While the distributions for the second highest ledge and the yard are different but do not vary too much, considerably

different action primitives were assembled into (more complex) behaviors on the uppermost ledge and in the elevator leading there. Apart from supporting the claim we made in the introduction, these findings readily point out an avenue towards location-based behavior synthesis: one of the approaches we are currently investigating integrates slack variables that indicate map areas into the Bayesian model introduced in [11].

5 Summary

We proposed a data-driven approach to the partitioning of game worlds into topologically and semantically distinct areas. Since *spectral clustering* assumes a data structure closely related to waypoint maps used in computer games, we were not surprised to find it yielding useful results. First investigations towards behavior synthesis hint that gamebots with corresponding location specific behavior should be available soon.

References

1. Khoo, A., Zubek, R.: Applying Inexpensive AI Techniques to Computer Games. IEEE Intelligent Systems **17**(4) (2002) 48–53
2. Onuczko, C., Cutumisu, M., Szafron, D., Schaeffer, J., McNaughton, M., T. Roy, K.W., Carbonaro, M., Siege, J.: A Pattern Catalog For Computer Role Playing Games. In: Proc. GameON-NA. (2005) 33–38
3. Le Hy, R., Arrigioni, A., Bessière, P., Lebeltel, O.: Teaching Bayesian behaviours to video game characters. Robotics and Autonomous Systems **47**(2–3) (2004) 177–185
4. Arvey, A., Aaron, E.: Online Markov Decision Processes for Learning Movement in Games. In: Proc. Int. Conf. on Computer Games, AI and Mobile Systems. (2005) 48–52
5. Bauckhage, C., Thurau, C., Sagerer, G.: Learning Human-like Opponent Behavior for Interactive Computer Games. In: Pattern Recognition. Volume 2781 of LNCS., Springer (2003) 148–155
6. Bauckhage, C., Thurau, C.: Towards a Fair 'n Square Aimbot – Using Mixtures of Experts to Learn Context Aware Weapon Handling. In: Proc. GAME-ON. (2004) 20–24
7. Gorman, B., Fredriksson, M., Humphrys, M.: QASE: An Integrated API for Imitation an General AI Research in Commercial Computer Games. In: Proc. Int. Conf. CGAMES. (2005) 207–214
8. Yannakakis, G.N., Hallam, J.: Evolving opponents for interesting interactive computer games. In: Proc. Int. Conf. on the Simulation of Adaptive Behavior. (2004) 499–508
9. Priesterjahn, S., Kramer, O., Weimer, A., Goebels, A.: Evolution of Reactive Rules in Multi Player Computer Games Based on Imitation. In: Proc. Int. Conf. on Natural Computation. (2005) 744–755
10. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Online Adaptation of Game Opponent AI in Simulation and in Practice. In: Proc. GAME-ON. (2003) 93–100
11. Thurau, C., Bauckhage, C., Sagerer, G.: Synthesizing movements for computer game characters. In: Pattern Recognition. Volume 3175 of LNCS., Springer (2004) 179–186
12. Shi, J., Malik, J.: Normalized Cuts and Image Segmentation. IEEE Trans. Pattern Anal. Machine Intell. **22**(8) (2000) 888–905
13. Meila, M., Verma, D.: A Comparison of Spectral Clustering Algorithms. Technical Report UW-CSE-03-05-01, Computer Science and Engineering, Univ. of Washington (2003)
14. Kannan, R., Vempala, S., Vetta, A.: On Clusterings: Good, Bad and Spectral. J. of the ACM **51**(3) (2004) 497–515

Making Racing Fun Through Player Modeling and Track Evolution

Julian Togelius, Renzo De Nardi, and Simon M. Lucas

Department of Computer Science
University of Essex
Colchester CO43SQ, United Kingdom
{jtogel, rdenar, sml}@essex.ac.uk

Abstract. This paper addresses the problem of automatically constructing tracks tailor-made to maximize the enjoyment of individual players in a simple car racing game. To this end, some approaches to player modeling are investigated, and a method of using evolutionary algorithms to construct racing tracks is presented. A simple player-dependent metric of entertainment is proposed and used as the fitness function when evolving tracks. We conclude that accurate player modeling poses some significant challenges, but track evolution works well given the right track representation.

1 Introduction

The video game genre of racing games is almost as old as video games, and shows no signs of losing popularity; with every generation of new gaming hardware new racing games come out, with more complex physics, graphics and network connectivity. Whether any real progress is being made in racing game AI or track design is another matter. We have previously studied the application of evolutionary robotics techniques to developing controllers for cars in a simple racing game [1][2][3]. We have two distinct motivations: to create more adaptable, believable and entertaining games through using AI, and to explore the suitability of video games as environments for studying the emergence of general intelligence. In this paper, we explore the use of the framework developed in our previous work to create racing tracks tailor-made to maximize the enjoyment of individual human players.

While the use of logic-based AI in games has a history as long as artificial intelligence itself, and computational intelligence in games is now an established field[4], the use of AI techniques to enhance player satisfaction is a relatively new endeavour[5]. As far as we know, no-one has yet attempted this for racing games.

This paper is organized as follows: first we comment on the computational car racing problem in general and our racing game in particular. We then present the problem of optimizing racing game entertainment, and present our approach to this: player modeling and track evolution. Next, we briefly describe the car

controllers and evolutionary algorithm used. The following section is on player modeling, for which three different approaches are compared. Once a player is modeled, we want to evolve entertaining tracks for this player, so the final section presents our representation for evolvable racing tracks, our fitness functions and some results. We conclude with an outlook on possible future work.

With this paper, we are merely dipping our toes in a new and fascinating area, and both ideas and results are to be considered as preliminary. Due to space concerns, some technical details have been omitted, but these can be found in our earlier papers on evolutionary car racing.

2 The car racing problem

The focus of the earlier papers has been the following. We have one or several tracks, containing free space, non-permeable walls, starting positions of the car, and a chain of waypoints which the car must pass sufficiently closely in the correct sequence. We also have either one or two cars, each equipped with a sensor model that permits it to sense the immediate environment from its own perspective, and a discrete set of motor and steering commands. The physics of the simulation are reasonably realistic, accounting for inertia, drift, and semi-elastic collision between cars, and between cars and walls. The objective of the game is for the car to progress as far as possible along the track in 700 timesteps, which corresponds to 35 seconds when run in real-time. Progress is judged by the number of waypoints passed; for most tracks it is possible to complete several laps within the allotted time. At every timestep, the task of the controller is to choose, on the basis of available sensor data, one of three possible motor commands (forward, neutral or backward) and one of three possible steering commands (left, center or right). Given the momentum of the car and its turning radius, the action taken at any single timestep contributes little to the overall course of the car. The non-holonomic nature of the car and the solidity of the walls also make the car racing problem considerably harder than many simple mobile robotics problems, such as wall-following with a typical differential-drive robot.

In previous work, we compared a number of different controller architectures and sensor representations for this problem, and concluded that the configuration described below is the best performer overall[1]. In our experiments, the best evolved controllers outperformed all human drivers on the tracks tested. We also proposed methods for scaling up to several tracks[2] and cars[3] simultaneously.

3 What makes racing fun?

We have been unable to find any prior research on what makes it fun to play a particular racing game, or to drive on a particular track in that game. The initial hypotheses offered here are therefore based on our own experiences and on opinions gathered from an unstructured selection of non-experts.

– The sensation of speed is clearly a factor - people like to drive fast. A track should allow a high maximum speed.
– Driving on an endless straight track is not fun, even when driving at high speed. Tracks should be challenging to drive.
– Crashing all the time is not fun either. Tracks should have *the right amount of challenge.*
– People also tend to like a variety of challenges, so tracks should vary in character and not repeat the same kind of challenge all the time.
– Drifting or skidding in turns seems to be a major fun factor.

The renowned game designer Raph Koster, writing about video games in general, offers a different perspective[6]. According to Koster, playing and learning are intimately connected, and a fun game is one where the player is continually and successfully learning. One way to interpret this in the context of car racing would be that a good racing track is one on which the player does pretty poorly the first time he plays, but quickly and reliably improves in subsequent races.

4 Technicalities: sensors, neural networks and evolutionary algorithms

4.1 Sensors

For ease of comparison we use the same sensor setup as in earlier papers. All the control methods used in this paper use information from eight simulated sensors: the speed of the car, the angle to the next waypoint, and six wall sensors. Each wall sensor is located at the center of the car, and has two parameters under evolutionary control: its direction (expressed as the angle it makes with the axis of the car) and its maximum range, which may be anything between 0 and 200 pixels. A wall sensor returns a value between 0 and 1, depending on whether it detects a wall within its maximum range, and on the distance of the wall as a proportion of the maximum range.

4.2 Neural networks

Most of the car control methods are based on neural networks. These networks are all standard fully connected feedforward nets (MLPs) with the *tanh* transfer function. Only the weights of the networks are changed by evolution or backpropagation, the topology remaining fixed. Nine inputs (sensors and a bias input), six interneurons, and two outputs are used. The first output is interpreted as driving command (less than -0.3 means backward, more than 0.3 means forward and in between means neutral) and the second as steering command.

4.3 Evolutionary algorithm

Variations on an evolutionary algorithm we call *Cascading Elitism* (inspired by [7]) is used to evolve both tracks and controllers, problems which have several conflicting fitness measures. The algorithm is essentially an evolution strategy without self-adaptation, but with a simple modification in order to handle

multivariate problems. At the start of the first generation, a population of 100 genomes is created. In each generation, all genomes are evaluated according to the first fitness criterion, and the 50 best individuals (the first elite) are retained while the rest are thrown away. If there is a second fitness measure, the first elite are then sorted according to the new fitness measure and the best 30 individuals (the second elite) are retained; the same principle is used for the third fitness measure where the 15 best individuals of the second elite are kept. Mutated copies of the surviving individuals are then used to fill the population up to the initial level of 100. To eliminate the risk of disruptive effects such as ''competing conventions", crossover is not used. We have not studied the effect of varying certain parameters, such as population size, selection pressure, and order of the fitness functions. When evolving a car controller, the mutation operator applies perturbations drawn from a gaussian distribution with standard deviation 0.1 to all weights and parameters. For track mutation methods, see section 6.

5 Player modeling

In order to use evolution to automatically create an entertaining track for a particular human player, we need to be able to test the generated tracks against the relevant human player thousands of times. Subjecting any real human to such treatment would surely remove any entertainment value, so we need a way of reproducing the player's driving style on tracks on which he has not yet driven.

Charles and Black[8] survey player modeling in general, and Yannakakis and Maragoudakis[9] use player modeling for optimising satisfaction in a version of the Pac-Man predator-prey game. Much closer to our problem domain, the popular racing game *Forza Motorsport* for the XBox allows its players to train a "drivatar" to drive just like them. This is accomplished by dividing all tracks into segments, and recording the precise path the player takes through each segment[10]. When the drivatar subsequently drives a track it has not been trained on, this track is analyzed into segments and a new path is calculated that the car has to adhere to. Crash recovery is a hard-coded mechanism. While the *Forza* approach has proved very successful, it imposes severe constraints on the track designs, and requires information about the preferred path to be available to the controller, something we wish to avoid.

Other relevant prior research include the ALVINN experiments, in which a real car learned to drive on real roads using backpropagation, visual inputs and human driving, although it should be noted that the objective was to learn driving in general rather than a particular driving style[11].

5.1 Learning behaviour

Backpropagation Our first attempt at player modeling was very straightforward: learning a car controller from a human example. A human player drove a number of laps around a track, while the inputs from sensors and actions taken by the human were logged at each timestep. This log was then used to train a

neural network controller to associate sensor inputs with actions using a standard backpropagation algorithm. Several variations on this idea (such as more sensors, larger networks, and selective editing of the log file before training) were tried with very little success. Although the training often achieved low error rates (typically 0.05), none of the trained networks managed to complete even half a lap. Usually the car just drove straight into a wall, though in some cases it did not move at all.

Nearest-neighbour classification We also tried controlling the car with a simple nearest-neghbour classifier. In these experiments, we used the same training data as in the experiments above, and let the controller choose the outputs of the training data point that closest matched the current sensor inputs. The small amount of noise that is applied to sensors guarantees that the car does not simply replay the human action. The resulting behaviour looked very "human-like", indeed very like the driving style of the particular human being modeled. It was quite good as well, sometimes taking a lap or two at decent speed. But almost invariably it ended up taking a turn just a bit too early or too late and therefore crashed into a wall. After that, it was unable to back away from the wall. Another problem with this control method was that it had serious problems with generalization.

5.2 Learning characteristics of behaviour

Given the limited success of learning player behaviour directly, we decided to try an indirect approach, namely identifying measurable characteristics of the human player's behaviour, and adapting an evolved controller to reproduce these characteristics. We decided to use total progress along the track, the variance of progress along the track between different trials, and the number of action changes (changing from one driving command to another), as these characteristics are easily quantifiable and vary considerably with driving styles. In the evolutionary runs, we started from good pre-evolved controllers, which the multi-stage evolutionary algorithm further evolve based on logs of human driving.

Results were mixed. While it was not very hard to adapt the controller to exhibit the same progress as the human on the same tracks, the other characteristics were harder to learn. It was possible to evolve a low variance in progress, but only occasionally did we manage to evolve a higher variance. Trying to adapt the controller to produce a given number of action changes was hopeless. One of the authors changed driving command 69 times in 700 timesteps, while an evolved network made about 250 changes, a difference which could not be evolved away.

5.3 Learning performance profiles

We then reasoned that if we could not capture the actual driving style of a human player with the methods at our disposal, we could at least capture the unique performance profile of that driving style. For example, assume player

A drives rather recklessly, and so does well on straights and broad curves but often crashes into walls in narrow passages and turns, and player B drives very carefully and so maneuvers through narrow passages well but doesn't exploit his vehicle's full potential in straights. We would then want the model of player A to make better progress than B on tracks rich in straights and broad curves, and the model of B to do better than A on tracks requiring lots of precise maneuvering, even if the micro-features of the models' driving look nothing like those of the corresponding human players. To accomplish this, we designed three test tracks

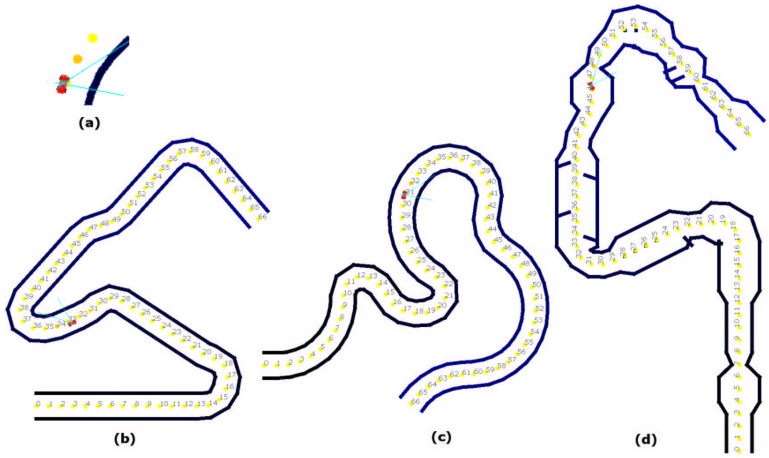


Fig. 1. Handcrafted test tracks

(see figure 1) to emphasize different aspects of racing skill. The first track consists of continuous gentle curves, the second track has long straights and sharp turns, and the third track has lots of narrow passages and obstacles. The progress of two of the authors on these tracks within 700 timesteps was measured (0.94, 0.86, 0.81, and 1.49, 1.49, 1.46 laps respectively). Controllers that approximated these performances were found within 100 generations.

5.4 A non-trivial problem

We see our attempts at player modeling as having been a partial success. On the one hand we have been able to produce controllers that closely match the performance profile of human players, but on the other hand we failed to model human driving behaviour more directly. The controllers modelled on performance profiles differ from humans in their micro-behaviour both qualitatively, in that their driving looks a bit artificial, and quantitatively, e.g. when counting action changes. We believe that these problems are due both to the reactive nature of all the controllers examined so far, and to the limited sensory inputs; the

sensor data we humans use to decide on the car's movement are much richer, and our complex neural networks allow such strategies as the construction and exploitation of forward models of the car. Much work remains to be done here.

6 Track evolution

In this section we define concrete fitness measures based on the considerations in section 3, and propose an evolutionary approach for track design. Many different representations of the racing track are possible; here, we propose one that emphasizes smoothness of the fitness landscapes. The track is encoded as a fixed length sequence of segments, all with the same midline length (the length of the track is therefore fixed). Each of the segments can represent a short part of a straight path, or of a curve (three different curvature radii are possible). In order to allow for more flexible tracks, each segment can have its own final width (three different widths are possible), subject to the constraint that the initial width has to match the width of the previous segment. Segments containing obstacles (in the center or in one side of the track) were also available as part of the genetic material. The mutation operator used by the evolutionary algorithm works simply by changing a segment into one of a different type with uniform probability 0.1. It is clear that the fitness obtained by a car driving on the mutated track will be similar to its fitness on the original track since, from the car's perspective, a large part of the track has not changed at all. This concept directly translates into a smooth fitness space by minimizing disruptive mutations. The chosen representation theoretically still allows for the track to turn back on itself and overlap, but no attempt was made to prevent this; in the unlikely event of this happening, the fitness obtained by the track would be very low and so the track would not survive the selection process. To ensure a set of viable individuals in the starting population, the initial elite consists of tracks using only straight segments. The tracks produced by this method will not in general form closed circuits, and so we artificially close the tracks by means of 'teleportation'. As soon as the car reaches the end of a track it is moved to the start, keeping the same position, velocity, and orientation relative to the track.

Fitness measures Three different fitnesses have been chosen to drive the evolution process: target final fitness, maximum speed, and maximum fitness variance. The first and most important fitness used, is given by $f_1 = 1 - \mid p_f - p_t \mid$, where p_f is the final progress (fractional number of laps) and p_t is the desired target progress. In order to smooth the effect of the sensor noise, the progress fitness of the single track is averaged over 10 repetitions. The final progress is intimately related to the average speed achieved during the test, therefore setting the target final progress corresponds to setting the type of track (slow or fast) that we are trying to produce. The second fitness measure is the maximum speed achieved within the trial; the aim of this fitness is to force the inclusion in the track of at least one section on which the player could possibly reach a very high speed. To drive the search towards challenging tracks, the third fitness selects for tracks

with a high variance in the final progress. Other fitness measures considered include speed variance and number of turns taken.

Results In general terms the track evolution process was considered successful; the evolutionary algorithm is indeed able to produce a track on which the player-modeled controller achieves the desired progress fitness. Changing the final progress target allows the production of a low speed or high speed track. We have already commented on the weaknesses of our player modeling approach, particularly in replicating the driving style of the human player, and so we were surprised to find that its performances in these experiments were close to those of its human counterparts. In several instances in which one of the authors was testing a track tailored to the model of his own driving, he would obtain a final progress quite close to the target for which the track had been evolved. This typically happened only on the first trial on the new track; in subsequent attempts on the same track, the player would generally improve on his first trial by exploiting his accumulated knowledge of the track, an impossibility for a reactive controller.

In figure 2(a) we can see plotted the evolution of the three distinct fitnesses of the best individual throughout the evolutionary run (10 evolutionary runs are averaged) when only the final progress fitness is in the selection mechanism; figure 2(b) shows instead the evolution of the fitnesses when all of the fitnesses are employed in the selection. As can be seen, in figure 2(a) the progress fitness quickly converges to 1 in the first tens of generations. Top speed and final progress standard deviation instead reduce in favour of the final progress optimization. When optimization for top speed and progress variance is also enforced

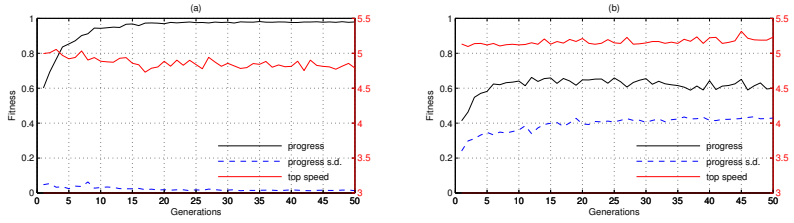


Fig. 2. Fitnesses of the best individual (average of 10 evolutionary runs): (a) selection based only on track progress finess, (b) selection based on all the three fitnesses.

(figure 2(b)), it seems clear that these fitness measures are in direct competition. The evolutionary algorithm drives the progress fitness toward the requirement, settling at fitness 0.6, a reasonable value since the final progress standard deviation is equal to 0.4. The initial population is constituted by straight tracks, the top speed is therefore close to the maximum since the early generations; the

evolutionary selection can be seen to guarantee the top speed not to be dropped in favour of the other fitnesses.

In figure 3 are displayed three tracks that evolution tailored on the player model of two of the authors; track ((*a*)) is evolved for a final progress of 1.1 (since the respective human player was not very skilled), track (*b*) and (*c*) are instead evolved on a model of a much skilled player for final progress [1] 1.5. For track (*a*) and (*b*) all the three fitness measure were used, while for track (*c*) only progress fitness was used.

The main difference between tracks (*a*) and (*b*) is that track (*a*) is broader and has fewer tricky passages, which makes sense as the player model used to evolve (*a*) drives slower. Both contain straight paths that allow the controller to achieve high speeds. In track (*b*) we can definitely notice the presence of narrow passages and sharp turns, elements that force the controller to reduce speed but only sometimes causes the car to collide. Those elements are believed to be the main source of final progress variability. These features are also notably absent from track *c*, on which the good player model has very low variability. The progress of the controller is instead limited by many broad curves.

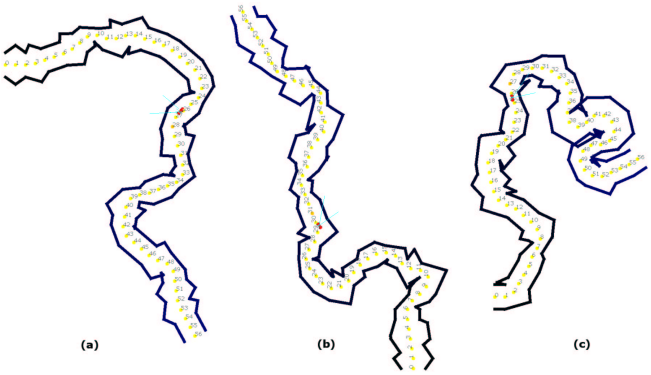


Fig. 3. Three evolved tracks: ((*a*)) evolved for a bad player with target progress 1.1, (*b*) evolved for a good player with target fitness 1.5, (*c*) evolved for a good player with target progress 1.5 using only progress fitness.

7 Conclusions

We have shown that we can evolve tracks that, for a given controller, will yield a predefined progress for the car in a given time, while maximizing the maximum

[1] The target progress is set between 50 and 75 percent of the progress achievable by the specific controller in a straight path. As a comparison, in Formula 1 races this ratio (calculated as ratio between average speed and top speed) is about 70 percent, and for the latest *Need for Speed* game it is between 50 and 60 percent.

speed and the standard deviation of the progress. The tracks produced are all drivable for a human player, and appear quite well designed, to our eyes at least. We do not yet know whether we have succeeded in automatically creating entertaining tracks for particular players, for the simple reason that we have not done any empirical studies on actual human players (apart from ourselves) to find out what they like. This is something that definitely needs to be done.

Our approach using player modeling works less well, but we can at least reliably produce controllers that make the same progress as the corresponding human player on a number of tracks, even though the driving styles differ from human driving. We found this level of modelling to be sufficient for track evolution, though of course we would like to be able to model human driving more closely. The failure of our supervised learning approaches to player modelling is probably due to the simplicity of the rangefinder sensors compared to human vision, and of the feedforward neural networks compared to human cognitive capacities, but additional experiments need to be done using recurrent neural networks and detailed visual inputs for the controllers to establish this.

Some interesting and relatively simple extensions of this research would be to evolve appropriate opponents in a multi-car race, and also to evolve tracks for learnability, following Koster's theory of what makes a game fun.

Acknowledgement Thanks to Owen Holland for stimulating discussions.

References

1. Togelius, J., Lucas, S.M.: Evolving controllers for simulated car racing. In: Proceedings of the Congress on Evolutionary Computation. (2005) 1906–1913
2. Togelius, J., Lucas, S.M.: Evolving robust and specialized car racing skills. In: Proceedings of the IEEE Congress on Evolutionary Computation. (2006)
3. Togelius, J., Lucas, S.M.: Arms races and car races. In: Submitted. (2006)
4. Kendall, G., Lucas, S.M., eds.: Proceedings of the First IEEE Symposium on Computational Intelligence and Games. IEEE Press (2005)
5. Yannakakis, G.N.: AI in Computer Games: Generating Interesting Interactive Opponents by the use of Evolutionary Computation. PhD thesis, School of Informatics, University of Edinburgh (2005)
6. Koster, R.: A Theory of Fun for Game Design. Paraglyph Press (2004)
7. Jirenhed, D.A., Hesslow, G., Ziemke, T.: Exploring internal simulation of perception in mobile robots. In: Proceedings of the Fourth European Workshop on Advanced Mobile Robots. (2001) 107–113
8. Charles, D., Black, M.: Dynamic player modelling: A framework for player-centred digital games. In: Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education. (2004) 29–35
9. Yannakakis, G., Maragoudakis, M.: Player modeling impact on players entertainment in computer games. In: Proceedings of the 10th International Conference on User Modeling. (2005) 74–78
10. Herbrich, R.: personal communication (2006)
11. Pomerleau, D.A.: Neural network vision for robot driving. In: The Handbook of Brain Theory and Neural Networks. (1995)

Author Index