Reinforcement Learning in Card Game Environments Using Monte Carlo Methods and Artificial Neural Networks

Conference Paper · September 2019

DOI: 10.1109/UBMK.2019.8907235

2 authors:

Ömer Baykal
Middle East Technical University
2 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Ferda Nur Alpaslan
Middle East Technical University
72 PUBLICATIONS **752** CITATIONS

SEE PROFILE

Reinforcement Learning in Card Game Environments Using Monte Carlo Methods and Artificial Neural Networks

Ömer Baykal
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
obaykal@ceng.metu.edu.tr

***Abstract*—Games are one of the areas that commonly use artificial intelligence. Today, it is an important feature for most of the game environments to have intelligent agents to offer more entertainment and challenge to players. Since it gets harder to develop good agents as games become more complex, machine learning methods have started to be used to shorten the development process of agents and to improve their quality. Popularity of machine learning applications in game environments has increased in the last decades. Reinforcement learning methods can be applied to develop artificial intelligence agents that learn to play a game by themselves without any supervision and can play it at a high level of expertize. Supervised learning methods, on the other hand, can be applied to develop agents that play a game by imitating the supervisor players. The purpose of this study is to develop self-learning agents agents for a card game, namely Batak, using reinforcement learning combined with supervised learning. Batak is a trick-taking card game popular in Turkey.**

***Index Terms*—artificial intelligence, learning systems, reinforcement learning, state-value functions, monte carlo methods, supervised learning, neural networks, games**

I. Introduction

Artificial intelligence has many application areas and games are one of the important ones. There are many applications of artificial intelligence methods in game environments. It is very common for game environments to include intelligent agents. Having intelligent agents makes a game more entertaining and challenging for its players.

Batak is a trick-taking card game which is played with four players and a standard deck of 52 playing cards. It is one of the most popular card games in Turkey. Rules of the game and some other details about it are given in following section. In this study, self-learning agents for the Batak card game were developed using Monte Carlo estimation of state-values method, which is a reinforcement learning method, combined with artificial neural networks for supervised learning and function approximation. Reinforcement learning can be defined as learning what to do so as to maximize a numerical reward signal [1]. Reinforcement learning agents are self learning agents. They are not supervised by external teachers. They learn to play a game by exploring different possible gameplays and checking the consequences. Supervised learning can be defined as learning from a set of example data labeled by a supervisor (teacher).

II. Related Work

Artificial intelligence methods are being used in game environments since 1950s. Card and board games are popular application areas since the beginning. There are many successful studies in literature for developing game playing agents for card and board games like checkers, chess, backgammon, and poker. Main reason of these studies were to develop agents that play at highest level and to compete against best human opponents. The resulting agents were evaluated by comparing them against world-wide champion human opponents. The first and the pioneering study in this area is the checkers playing program developed by Arthur Samuel [2], [3]. His work forms the basis of reinforcement learning.

Reinforcement learning is a major area of machine learning. Reinforcement learning agents learn by exploring their environments on their own. The environment gives rewards for actions of the agents and the concern of the agents is to maximize the total rewards. Game environments are appropriate for reinforcement learning. Card and board game environments are appropriate for reinforcement learning. They respond to the actions of reinforcement learning agents by checking if the agents satisfy the objectives of the game or determining how close they are. There are several studies in literature in which reinforcement learning techniques are used to develop game playing agents. As stated before, Arthur Samuels checkers playing program [2], [3] is one of the earliest work that uses reinforcement learning. TD-Gammon [4], [5] is one of the most successful applications of reinforcement learning for games. It is a world class backgammon playing program developed using temporal difference learning [6] technique. He successfully combined it with artificial neural networks to solve the state space size related issues of the backgammon environment. AlphaGo is one of the most recent and successful applications of reinforcement learning in games released by Google DeepMind [7]. It learned to play Go from scratch and achieved to beat a world champion for the first time.

Reinforcement learning methods were also used in commercial video games. Main issue for applying reinforcement learning on video games is the size of the state-action spaces. Although standard reinforcement learning techniques give successful results for problems with smaller spaces, solving problems with standard techniques gets harder as spaces get bigger. In order to solve huge and complex problems like digital games to develop self-learning agents, hierarchical reinforcement learning techniques are used. These techniques decompose complex problems into simpler subproblems to make them solvable by standard reinforcement learning approaches.

Tao Feng: Fist of the Lotus is a fighting video game which was published by Microsoft in 2003. Computer controlled fighter agents embedded in this game were developed by using offline Sarsa [8], [9] algorithm. In this application of reinforcement learning, neural networks were used to compute value functions. Gameplaying performances of the agents were stated to be nearly optimal in the study [10]. *Pac-Man* is an arcade video game whose first version was released in 1980. It has been very popular all over the world from its first release to this day. In his study, Galway has created computer agents for a variation of this game. Both Sarsa and Sarsa(λ) algorithms were used with linear value function representations in this study [11]. Another study with similar approach was done by Lucas and Togelius in a simulated environment of a car racing video game and it was given satisfying results [12]. *Battleground* is a real time strategy video game series, which was developed and published by TalonSoft between 1995 and 1999. This game has hierarchical reinforcement learning agents that learn by playing against a rule based opponent agent [13]. Learning here is hierarchical because of the complexity of the game. Sarsa(λ) algorithm is used in this study as reinforcement learning method. In a continuing study [14], they have also tried neural network as value function representations. They have achieved to create agents that are comparable to human players. So, the results show that reinforcement learning is suitable for also complex strategy games. *Settlers of Catan* is a popular board game which also has a video game version. The video game version has intelligent agents that use hierarchical reinforcement learning approach. The game is decomposed into lower level behavioral decisions and reinforcement learning is applied to find out the policies for behavior selections. In this study, model trees have been used for representing state-action space [15]. *Battle of Survival* is a real time strategy video game. This game has intelligent agents that are developed with reinforcement learning with hierarchical approaches. Agents have been developed using Q-learning [16] algorithm and its modified versions [17]. *Civilizations IV* is the fourth game of the popular turn based strategy video game series, which is published by 2K Games in 2005. In a study by Wender and Watson [18], Q-learning algorithm is used to develop reinforcement learning agents for city placement selection task, which is a part of the gameplay. They compared their agents with the built-in intelligent agents of the game and reported that the results are encouraging.

III. BATAK

Batak is a trick-taking card game. It is one of the most popular card games in Turkey. It is played with four players. There are no groupings among players, they are all individuals. It is played with a standard deck of 52 playing cards with 4 different suits (clubs, diamonds, hearts, spades) and 13 different ranks (lower to higher: 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K, A). There are 13 rounds in each game. All 52 cards are shuffled and distributed to players at the beginning of each round. Each player is given 13 cards. The objective of the players is to get the highest score at the end of the game. A players score at the end of the game is the sum of the scores received by the player for each round of the game.

Each round consists of two stages. The first stage is the bidding stage. At the beginning of a round, players freely declare the number of tricks they expect to win (from 1 to 13) one by one. There is no need for any bid to be higher or lower than the other bids. Players can also declare that they expect to win no tricks. In Batak, the trump is always spades. Cards with a suit other than spades (or trump) have no value over the cards with any other suit. The only exception is that the cards with the suit that leads a trick beat cards of any other suit except spades.

After the bidding stage, the gameplaying stage begins. There are 13 tricks in each round of the game. Players must obey a couple of rules when choosing a card to play for a trick. First of all, they have to choose a card with the suit that leads the current trick and also it must be a card with rank higher than the highest ranked card in the current trick with the same suit. In other words, players have to follow the leading suit and raise. However, if they have cards with the leading suit but they do not have cards with a higher rank, any card with that suit can be played. If they even do not have any card with the leading suit, they have to choose a trump (spades) card. Raising the trump cards in the current trick is also a must for players. If they do not have such a card that raises the trumps, any trump card can be played. If players do not have trump cards too, they can choose any card without suit and rank restrictions. Moreover, if the current trick is not led by trump but there are trump cards in the current trick, players do not have to raise the leading suit. Finally, a player cannot start to a trick with a trump card unless one of the players compulsorily played a trump card before. The player that wins a trick is determined at the end of the trick according to the 4 played cards. If there are trump cards in the trick, the player who played the highest ranked trump card wins the trick. Otherwise, the player who played the highest card with the leading suit wins the trick. The player who wins a trick starts to the next trick.

Players are given scores at the end of each round based on their bids and the number of tricks they win. In other words, the round score a player depends on the number of tricks he declared to win (i.e. declared_number) and the number of tricks he actually won (i.e. actual_number). If the

TABLE I
STATISTICS ABOUT THE SIMULATED GAMES TO TEST PERFORMANCE OF REINFORCEMENT LEARNING AGENT TRAINED TO WIN AS MANY TRICKS AS POSSIBLE AGAINST RULE BASED AGENTS

		RL Agent	Rule Base Agent	Rule Based Agent	Rule Based Agent
Initial state-value function	Number of Wins	159	265	303	342
	Number of Tricks	40222	42334	42681	43763
After 50000 games	Number of Wins	446	157	203	251
	Number of Tricks	45539	40363	40812	42286
After 100000 games	Number of Wins	490	155	199	235
	Number of Tricks	46255	40092	40485	42168
After 150000 games	Number of Wins	509	138	177	234
	Number of Tricks	46363	40036	40476	42125
After 200000 games	Number of Wins	525	151	167	231
	Number of Tricks	46509	40127	40374	41990

TABLE II
STATISTICS ABOUT THE SIMULATED GAMES TO TEST PERFORMANCE OF REINFORCEMENT LEARNING AGENT TRAINED TO WIN AS FEW TRICKS AS POSSIBLE AGAINST RULE BASED AGENTS

		RL Agent	Rule Base Agent	Rule Based Agent	Rule Based Agent
Initial state-value function	Number of Wins	147	281	311	341
	Number of Tricks	39770	42524	42833	43873
After 50000 games	Number of Wins	22	379	332	322
	Number of Tricks	34635	45210	44620	44535
After 100000 games	Number of Wins	24	378	344	323
	Number of Tricks	34139	45475	44771	44615
After 150000 games	Number of Wins	19	375	350	312
	Number of Tricks	33794	45440	44970	44796
After 200000 games	Number of Wins	14	371	337	337
	Number of Tricks	33444	45577	44928	45051

player declares to win tricks and he wins tricks at least as much as he declares, he receives $(10 * (declared\_number) + (actual\_number - declared\_number))$ points. If the player declares to win tricks but he wins no tricks or tricks less than he declares, he is punished with $(10 * (declared\_number))$ points. If the player declares to win no tricks, he receives 50 points for success or he is punished with 50 points for failure.

IV. METHODS

In this study, deep reinforcement learning agents were developed for Batak using the policy gradient approach. In order to create policy networks, fully connected multilayer feedforward neural networks were utilized and they were trained by backpropagation algorithm. Although the general structure of all networks created during this project was same, they had various configurations (e.g. number of hidden layers or number of nodes in hidden layers). Configuration details are given in following sections.

All policy networks in this study consist of an input layer, an output layer, and a number of hidden layers. Input and output layers are same among all networks. Input layer takes the state as input and output layer returns the preferability of that state as output. Number of hidden layers and number of nodes in each layer differ among networks. Throughout this study, nodes with sigmoid activation function were used in all networks. Moreover, for training the networks, stochastic gradient descent (SGD) were used as backpropagation strategy and mean squared error (MSE) function were used as loss function.

V. EXPERIMENTS

Batak consists of two main stages: the bidding stage and the gameplaying stage. So, a Batak playing intelligent agent should have both capabilities of bidding well at the beginning of rounds and choosing appropriate cards to play during rounds. The agents developed in scope of this study have both capabilities. In order to develop a Batak playing intelligent agent, three subagents were developed: two gameplaying agents and a bidding agent. The real agent uses all three

TABLE III
STATISTICS ABOUT THE SIMULATED GAMES TO TEST PERFORMANCE OF SUPERVISED LEARNING AGENT TRAINED TO LEARN BIDDING AGAINST RULE BASED AGENTS

		RL Agent	Rule Base Agent	Rule Based Agent	Rule Based Agent
Initial network	Number of Wins	523	146	183	218
	Number of Tricks	46729	40066	40424	41781
	Number of Bids	77122	39059	38936	39250
After 10000 games	Number of Wins	523	146	183	218
	Number of Tricks	46729	40066	40424	41781
	Number of Bids	40315	39059	38936	39250
After 30000 games	Number of Wins	523	146	183	218
	Number of Tricks	46729	40066	40424	41781
	Number of Bids	42172	39059	38936	39250
After 50000 games	Number of Wins	523	146	183	218
	Number of Tricks	46729	40066	40424	41781
	Number of Bids	43044	39059	38936	39250
After 70000 games	Number of Wins	523	146	183	218
	Number of Tricks	46729	40066	40424	41781
	Number of Bids	44231	39059	38936	39250

subagents to make decisions of bidding and choosing cards to play. Although the bidding stage precedes the gameplaying stage, the gameplaying agents were developed before the bidding agent.

The gameplaying agents were developed using nonstationary Monte Carlo state-value function estimation method. Monte Carlo methods are tabular reinforcement learning methods. Since the state space of Batak is too big to be represented as a table, Monte Carlo method could not be applied directly. To make it applicable, artificial neural networks were used. Neural networks can approximate highly nonlinear functions. In this study, neural networks that takes the state of the environment as the input and the value for that state as the output were used. Besides storing values, neural networks can also increase the learning rate because they can generalize for unseen data.

The bidding agent is a supervised learning agent. It was trained using the artificial neural networks. The bidding agent learns how to bid given the initial distribution of cards to players and the number of tricks each player wins at the end of the round with his cards. The neural networks used to train the bidding agent takes initial states as input and number of tricks as target output.

All neural networks in this study are fully connected multilayer feedforward neural networks and they were trained by backpropagation algorithm. Each of them consist of an input layer, an output layer, and a number of hidden layers. Input and output layers are same among all networks. Input layer takes the state as input and output layer returns the preferability (value) of that state as output. Throughout this study, nodes with sigmoid activation function were used in all networks. Moreover, for training the networks, stochastic gradient descent (SGD) were used as backpropagation strategy and mean squared error (MSE) function were used as loss function.

Batak is played with a standard deck of 52 playing cards. Cards are distributed to players at the beginning of each round. In this study, a simple player-centric state representation was used. Each game state is represented by 52 one hot vectors of length 5. Each vector corresponds to a card from the deck. If the player whom the state is being written based on owns the card, the vector is [1, 0, 0, 0, 0]. If another player owns the card, the vector is [0, 1, 0, 0, 0]. If the card is in current trick and it was played by the player, the vector is [0, 0, 1, 0, 0]. If the card is in current trick but it was played by another player, the vector is [0, 0, 0, 1, 0]. If the card was in previous tricks, the vector is [0, 0, 0, 0, 1]. These are all of the possibilities for a card. There is not any other possibility for a card that this game state representation does not cover. This state representation was used for creating both gameplaying agents and bidding agents.

A. Playing Agents

To estimate a state-value function, lots of games were played between four self-learning players. All players choose cards to play by using a common state-value function, which is an artificial neural network, and also they update the same state-value function (i.e. they train the same network) by their gameplay experience. All of the game states during a round that occur after players play cards (i.e. afterstates) are saved. At the end of the round, training is done by using that data. In this study, due to the simplifications made, players get rewarded according to the number of tricks they win.

Fig. 1. Plot of the number of games reinforcement learning agent trained to win as many tricks as possible wins in evaluations against rule based agents at each checkpoint. (Includes data from Table I)

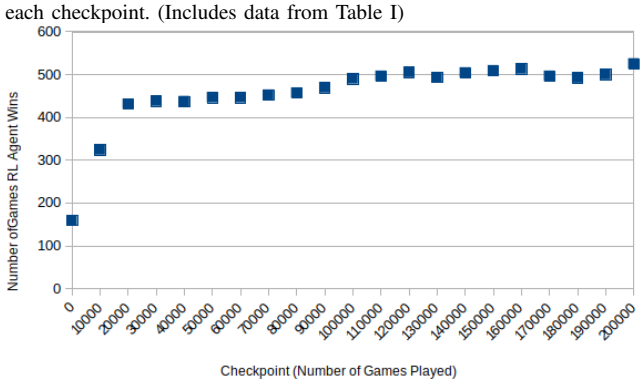
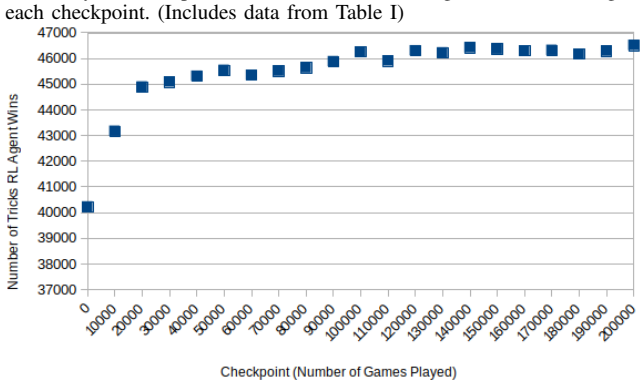


Fig. 2. Plot of the number of tricks reinforcement learning agent trained to win as many tricks as possible wins in evaluations against rule based agents at each checkpoint. (Includes data from Table I)



For example, lets say a player won 9 tricks in a round. If the purpose of the player was to win as many tricks as possible, the recorded states of the player are rewarded by the value $9/13 = 0,69$. However, if the purpose of the player was to win as few tricks as possible or in other words not to win any tricks, they are rewarded by the value $1 - 9/13 = 4/13 = 0,31$. Moreover, in order to increase the training speed and performance, a trick was applied to generate extra gameplay experience data without actually playing. Since cards with a suit other than spades (or trump) have no value over the cards with any other suit in Batak, the training process was repeated six ($3! = 6$) times at the end of a round using the same data but different permutations of card suits. For example, treating that all diamonds cards are clubs cards, all clubs cards are hearts cards, and all hearts cards are diamonds cards, we can update the state-value function for those new game states by making only simple modifications to original game states.

In order to carry out this study and to get acceptable evaluation results, rule based Batak agents were also developed within the scope of this study. The rule based agents follow some hard coded rules to bid and to choose cards to play. They were the baseline players in this study. Performances of created Batak agents were evaluated according to their gameplay performances. Gameplay performances were measured by arranging games between created Batak agents and rule based Batak agents. Results of those matches were noted and overall performances of trained agents were determined according to their success against baseline agents.

Two types of gameplaying agents were developed in this study: agents that try to win tricks and agents that try not to win tricks. All 10000 games checkpoints belong to both types of agents were compared with rule based agents one by one by making them play against three rule based agents. For testing each checkpoint, 1000 games were simulated. Table I and II shows statistics about the results of the games arranged for testing some of the checkpoints. Furthermore, figures 1, 2, 3, and 4 shows the data for all checkpoints as plots for making it more understandable.

The number of games and tricks that the agent that learns to win tricks wins increases by time. This shows that the agent successfully learns to win tricks. Similarly, the number of games and tricks that the agent that learns not to win tricks wins decreases by time. This reveals that the agent successfully learns not to win tricks.

B. Bidding Agent

The bidding agent developed in this study is an agent that can predict how many tricks the player can win with the initial cards distributed to him. The bidding agent is a supervised learning agent. Training data needed was collected from the trained gameplaying agents. To generate training data, lots of games were simulated between four trained gameplaying agents that knows to win tricks. The initial cards (states) of the players are stored at the beginning of a round. Then, the artificial neural network is trained at the end of each round using the number of tricks players win.

All 10000 games checkpoints were compared with rule based agents one by one by making them play against three rule based agents. For testing each checkpoint, 1000 games were simulated. Table III shows statistics about the results of the games arranged for testing some of the checkpoints. In Batak, optimal bidding is bidding no more than number of possible tricks to win and also bidding not too less than possible. In the table, you can see that the total number of bids by trained agent gets closer to the number of tricks he wins by time. This shows that the agent successfully learns to bid.

VI. Conclusions and Future Work

The results of these experiments reveal that using Monte Carlo reinforcement learning methods together with artificial neural networks is an effective solution to develop self-learning Batak agents. The preferred methods succeeded to overcome the difficulties of Batak environment such as the big state space. Considering the encouraging results obtained in the evaluations, that would be appropriate to continue training of agents (especially the bidding agent). The developed agents can also be used to challenge against human players.

Fig. 3. Plot of the number of games reinforcement learning agent trained to win as few tricks as possible wins in evaluations against rule based agents at each checkpoint. (Includes data from Table II)

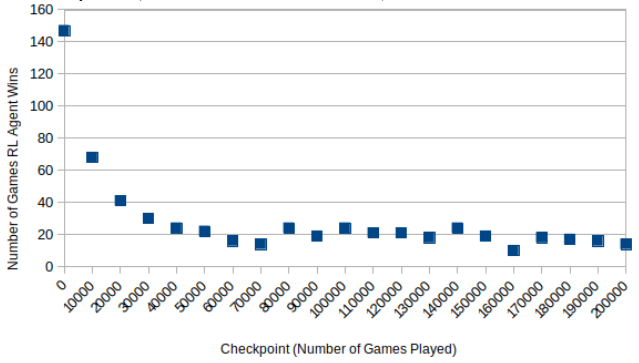
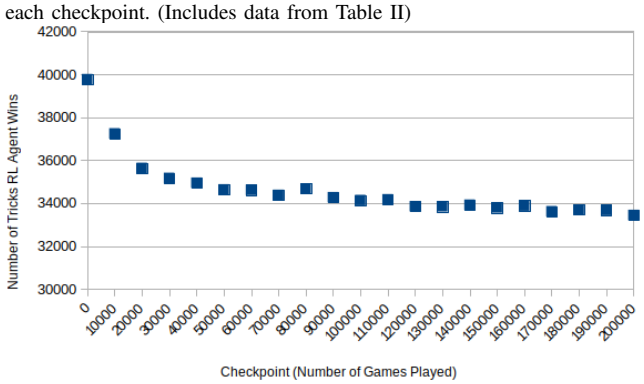


Fig. 4. Plot of the number of tricks reinforcement learning agent trained to win as many tricks as possible wins in evaluations against rule based agents at each checkpoint. (Includes data from Table II)



Although two different gameplaying agents were developed for both winning tricks and not winning tricks, only one bidding agent was developed and that was to predict how many tricks a player can win with his cards at the beginning of a round if he plays to win tricks. Considering the constantly increasing performance of the already trained bidding network, we can train another bidding network in future that predicts if declaring that he will win no tricks (nil bidding) for a player is appropriate or not. This network can be trained by collecting data for supervised learning from the trained self-learning agent that knows how to lose tricks.

The agents developed in this study were all tested individually. To test the overall performance of the full Batak intelligent agent, new tests can be arranged that uses both bidding agents to bid (bid nil if new bidding agent suggests to do so, otherwise bid the number that the other bidding agent suggests) and then uses corresponding gameplaying agent to choose cards to play to achieve the bid. Moreover, the full Batak agent can also be tested against human Batak players. Finally, the methods applied in this study can also be applied to other card games (e.g. Spades, Hearts, etc.) to develop intelligent agents for.

REFERENCES

[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[2] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal of research and development*, vol. 3, no. 3, pp. 210–229, 1959.

[3] ——, "Some studies in machine learning using the game of checkers. iirecent progress," *IBM Journal of research and development*, vol. 11, no. 6, pp. 601–617, 1967.

[4] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[5] ——, "Programming backgammon using self-teaching neural nets," *Artificial Intelligence*, vol. 134, no. 1-2, pp. 181–199, 2002.

[6] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[8] G. A. Rummery and M. Niranjan, "On-line q-learning using connectionist systems," 1994.

[9] R. S. Sutton, "Generalization in reinforcement learning: Successful examples using sparse coarse coding," *Advances in neural information processing systems*, pp. 1038–1044, 1996.

[10] T. Graepel, R. Herbrich, and J. Gold, "Learning to fight," in *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004, pp. 193–200.

[11] L. Galway, D. Charles, M. M. Black, and C. Fyfe, "Temporal difference control within a dynamic environment." in *GAMEON*, 2007, pp. 42–47.

[12] S. M. Lucas and J. Togelius, "Point-to-point car racing: an initial study of evolution versus temporal difference learning," in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*. IEEE, 2007, pp. 260–267.

[13] C. Madeira, V. Corruble, G. Ramalho, and B. Ratitch, "Bootstrapping the learning process for the semi-automated design of challenging game ai," in *Proceedings of the AAAI-04 Workshop on Challenges in Game Artificial Intelligence*, 2004, pp. 72–76.

[14] C. A. Madeira, V. Corruble, and G. Ramalho, "Designing a reinforcement learning-based adaptive ai for large-scale strategy games." in *AIIDE*, 2006, pp. 121–123.

[15] M. Pfeiffer, "Reinforcement learning of strategies for settlers of catan," in *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] M. Ponsen, P. Spronck, and K. Tuyls, "Hierarchical reinforcement learning with deictic representation in a computer game," in *Proceedings of the 18th Belgium-Netherlands Conference on Artificial Intelligence*, 2006, pp. 251–258.

[18] S. Wender and I. Watson, "Using reinforcement learning for city site selection in the turn-based strategy game civilization iv," in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*. IEEE, 2008, pp. 372–377.