Artificial Intelligence Techniques for the Puerto Rico Strategy Game

Rafał Dreżewski[1] and Maciej Klęczar[2]

[1] AGH University of Science and Technology, Department of Computer Science, Kraków, Poland
drezew@agh.edu.pl

[2] The School of Banking and Management, Faculty of Management, Finance and Computer Science, Kraków, Poland
maciekkl@o2.pl

Abstract. It was always a challenging task to create artificial opponents for strategy video games. It is usually quite easy to discover and exploit their weaknesses because their tactics usually do not adapt to changing conditions and to human opponent tactics. In this paper two artificial intelligence techniques for well known Puerto Rico strategy game are proposed. One of them does not rely on any precoded tactics, but tries to dynamically learn and adapt to the changing game environment. Both techniques were compared on the basis of results of games played against each other and also against human expert players.

Keywords: strategy games, artificial intelligence techniques, computer games

1 Introduction

Since the invention of computers, Artificial Intelligence (AI) was always a topic of high interest. Research on AI was not only related to computer games—in fact it was rather conducted as a part of academic studies or military development. In the recent decades, along with the rapid growth of video games industry, AI for games started to become a necessary part of games development. Almost every published game has some simple—or more sophisticated—AI algorithms inside, which are responsible for actions performed by various gaming agents.

In games that require rather agility skills (e.g. shooting games, hack and slash games), AI development is focused on creating more human-like agents, that are competitive, but not too difficult to compete with, allowing human players to eventually win. If the task is to calculate some number based on the given variables (e.g. hit the target accurately) AI player will outplay human player quite easily, so in most cases the main difficulty is to properly handicap AI player (make it less perfect), in order to achieve more human-like behavior.

The situation becomes completely different in the case of games that require thinking, inventing new tactics, adapting to environment, properly reacting to enemy's moves. While the above skills would be appreciated in many types of games, strategy games definitely require these skills. That is why it was always hard to create a good AI technique for strategy games [8]. AI techniques for enterprise video games usually are based on instructions what to do in particular situations. This is usually achieved by finite state algorithms or scripts. Skill of such AI depends on how many possible situations the AI's creator would predict. However, in most strategy games it is impossible to predict all situations or combinations, or even get close to it. Such an approach leads to weak AI technique for games. Usually, AI would behave correctly in some situations and much worse in the others. Even bigger problem is that such AI will often perform the same actions in the same or similar situations and its moves would be easy to predict, while prediction of enemy moves is a key of every successful tactic. That is why a good player will sooner or later find such weaknesses and exploit them in order to easily win the game.

The goal of this paper is to present a different approach—AI player that can quickly adapt and change its tactics dynamically on the basis of changes of the game environment. This is achieved by using tree search technique, with such adjustments, that would minimize search time, while keeping results at the satisfying level. Such AI player is capable of competing with an advanced human player and of making its decisions within a reasonable time slot, which is very important for a good game experience. In order to test different proposed AI techniques the computer game based on Puerto Rico board game was implemented. Also two AI algorithms for that game were proposed and implemented in order to illustrate the difference between more common approach (based on precoded decisions) and the proposed adaptive approach.

2 Related Work

Due to the complexity of the most of strategy games, creating AI for them was always very difficult—that is why AI techniques for most enterprise games are quite simple, usually based on scripts. Such an AI will not be a serious challenger for any advanced player, therefore the common technique of improving its skills is providing it a number of advantages (more money, more units, less penalties, etc). However such an approach is not ideal—basically it is cheating.

Of course, everyone can point to chess game, where AI have achievements of beating the best human players. But there are some significant differences between chess game and enterprise strategy games. Chess AI has been a subject of university studies for almost 70 years. There are a lot of strategies available and that strategies can be hard-coded into chess AI algorithms. The lower bound of the game-tree complexity of chess is 10^{120}, which makes it possible to search all states for next few moves in a reasonable time, or even prepare a database with some precalculated moves. In most of the enterprise strategy games environment is totally new and the number of possible moves is almost infinite. Any brute-force-like methods will fail or require a lot of computational time to obtain any viable results, while common human player requires such an AI technique to perform actions almost instantly.

The research is now conducted on methods that would improve static, scripted AI, without significant effect on the speed of decision making. One of the popular ways seems to be "Reinforcement learning" [10]. This method can be used to optimize single decision made during the game using historical data. In [11] usage of this method to optimize algorithm for city placement in popular game *Civilization IV* is described, while in [3] optimization of high-level strategies for that game is presented. Those methods help improve AI efficiency a bit, but it is obvious that this is only tip of the iceberg. Both of the above mentioned research works focus on optimization of only one aspect of the game. Although the second approach allows for focusing on global strategy that impacts the entire game, it is still just choosing from four scripted tactics.

Another method is called "Dynamic Scripting" [9]. It is quite similar to reinforcement learning—it is focused on adjusting factors of scripts on the basis of historical data collected from previous runs of those scripts and thus improving their efficiency.

Such methods allow for AI improvement with the use of data coming from competition against other AI players, however the most important data is collected when playing against human opponents. In order to improve the learning process during competition with human players "online learning" can be used. It allows to collect the data generated directly by human players, accumulate it and adjust strategy of individual AI players.

While those methods are having their successes, the main problem is balance between exploration and exploitation. Exploitation of collected knowledge is needed to perform the best actions, while exploration is necessary to collect more data. So such approach always requires significant number of tries, before the results can be visible—and this is the situation when only one aspect of the game is taken into consideration. The more aspects are automated by these algorithms, the more tries are needed to produce a reasonable version of the AI technique for the given game. This is the main factor why the game producers do not like such approach. Usually everybody wants a product which is immediately ready for usage—not after some time (which is even hard to predict) needed for learning.

The key difference with the algorithm proposed in this paper is that in our technique adjustment of the strategy is done not on the basis of historical data, but it is based on simulations performed during the game, while trying to keep AI responses as quick as possible. Such an approach can be used in a strategy game to adjust several important decisions in the case of complex games, or even all of them in the case of less complex ones.

Algorithms like Monte-Carlo Tree Search (MCTS) are also used in games, however usually either to simulate simple actions, or in games that do not require quick response times, like Chess or Go. There are three main differences between the approach proposed in this paper, which is based on game tree search, and standard MCTS algorithm:

- Simulations of the moves are performed with the use of scripted AI algorithm, instead of using random movements.
- Simulations are not performed till the end of the game, but usually end after few rounds. This requires special formulas to calculate value of moves.
- Exploration versus exploitation balance is shifted in order to minimize the number of simulations needed to achieve reasonable outcome.

The above differences have one main purpose—minimize the time needed for performing simulations.

3 Puerto Rico Game

Puerto Rico is a German-style strategy board game, created by Andreas Seyfarth in 2002 and published by Alea [2]. The game quickly reached top ranks in BoardGameGeek (world's most popular board game site) and still stays in top10, becoming the oldest game at the top ranks [1].

There are three main reasons why we decided to focus our research on this game, and not on any other: **lack of randomness**, **level of rivalry** and **easy rules**. From all the board games we have played, Puerto Rico is less dependent on randomness, like throwing a dice or random events. It requires no luck, but pure tactical skills to win a game. That makes it a good test-bed for AI techniques. The level of rivalry in this game makes it impossible to create some unique "always good" strategy. Players fight for control over common pool of various kind of resources and performing good moves highly depends on opponent's moves. That additionally increases the difficulty level of creating AI technique properly reacting to environment changes (which always is a hard task). Easy rules decrease time spent on creating game engine and coding all possible situations on the board, allowing to focus on creating AI mechanisms.

In Puerto Rico game we can perform the following actions:

- Raise plantations, which later allows us to produce goods of 5 kinds: corn, indigo, sugar, tobacco and coffee.
- Build production buildings, which combined with plantations, let us produce goods.
- Build utility buildings, which gives us several bonuses.
- Grant colonists to work on plantations or buildings.
- Produced goods can either be sold on the Market, granting doubloons, or shipped to Europe, granting victory points.
- Every turn we pick one of the 6–7 roles. Then every player performs action connected with this role, however the player who picked it is granted additional bonus and also opportunity to perform the given role's action first. Possible roles are the following: Settler (raising plantations), Mayor (getting new colonists and transfer them to plantations/buildings), Builder (builds buildings for the doubloons), Craftsman (produces goods), Trader (sells goods for the doubloons), Captain (ships goods for victory points) and Prospector (no action).

4 Game Engine

The game engine used in this research was created in C# using MS Visual Studio, with .Net 4.5 and WPF frameworks. WPF allows to create a nice graphical interface for the game, while .Net features, especially generic, allows to create a clean object oriented code. The engine of the game consists of 3 main modules (classes): Game, GamePage and Player. First two ones are responsible for general game rules, while Player class holds all user actions. By default these are human player actions performed with the use of mouse. Every AI player class derives from Player class, overriding its virtual functions, which are responsible for making choices (there are 10 such functions). Such architecture makes it quite easy to add a new AI technique.

5 AI Techniques

Low complexity level of the game rules leads to only 10 different possible choices to be made. Of course these choices always appear within the context of different situation—giving almost infinite number of possible combinations—but it allows for easier grouping of all AI actions. The set of choices include:

- Choosing a role;
- Choosing a plantation from available plantations stack;
- Choosing a building to build from the list of available buildings;
- Choosing how to re-allocate colonists over plantations and buildings;
- Choosing an additional production (as a bonus to craftsman role);
- Choosing which good to sell on the Market;
- Choosing which type of goods should be loaded onto ship;
- Choosing a ship onto which the goods should be loaded;
- Choosing which goods will be spoiled (due to the lack of storage);
- Choosing whether to use a Hacienda (an utility building).

The above grouping allowed for creating 10 virtual functions, one for each choice. Initially these functions were created for a human player and later they were overridden in AI classes, which provided specific implementation.

For a human player, a specific GUI is launched, which allows for making a choice with a mouse click. For AI players it is required to create a specific algorithm for each of those functions.

Two kinds (generations) of AI mechanisms were created for the Puerto Rico game. First one is similar to the already mentioned scripted AI (with some small adjustments implemented). The second one does not have any hard-coded strategies—instead it tries to perform a simulation for choosing best moves.

5.1 Scripted AI

First generation of AI was based mostly on scripts and Bayesian networks, with a little addition of random algorithms.

Most of the 10 possible actions, mentioned above, have a very similar algorithm:

1. Prepare a list of all possible choices (e.g. all buildings that are possible to build taking into account the amount of money and placement limitations).
2. Calculate which choice is potentially the best one by adding specific weights to each of them and then select one of the possibilities on the basis of these weights.
   - Add an a priori weight, which is based on some values predefined for each of the choices.
   - Consider multiple cases that can modify weights accordingly to the current situation on the board.
3. Randomly choose final option, on the basis of weighted list (choice with highest weight is the most probable).

Slight changes had to be made for re-allocating colonists action. Due to the fact the we have to place multiple colonists, the above algorithm actually runs in a loop and the choice is done multiple times, until all colonists are placed.

It should be noticed that there is a slight difference when we compare our approach to the standard scripted approach—it is randomness factor. It was added to achieve the lack of predictability, so AI player not always acts in the same way in a similar situation (e.g. at the beginning of the game). Puerto Rico game offers multiple, almost equally valuable, choices at many stages of the game, so randomness factor is used in order to avoid following the same path in the case when there are several reasonable possibilities. In order to avoid the selection of bad moves, precalculated weights were added, so it is very unlikely that really bad (low value) moves would be chosen.

Such AI approach is extremely fast, relatively easy to code and slight randomness factor gives lower rates of predictability. The above approach has also some limitations. It does not "think"—all the decisions are defined by its creator (so basically it acts like him in some limited way). Quick (not very sophisticated) implementation of such approach would result in quite low strategy skills, which would not be really challenging for advanced human player. This approach needs a lot of work when trying to implement large number of cases, resulting in performing better moves in different situations and generally being more challenging for advanced players. Randomness factor, which protects against the predictability, unfortunately sometimes lowers general skills of AI player, so it should be used with caution.

5.2 Thinking AI

The second generation of AI is based on game tree searching approach. During the tree search, simulations of moves are performed with the use of first generation (scripted) AI. In short, the algorithm works as follows:

1. When the choice is to be made, start a simulation. To save some time, start it in the background as soon as possible (e.g. we can start thinking about which building should be built, as soon as other player picks the Builder role).
2. Like previously, create a list of all possible choices.
3. Start a simulation:
   - Create a copy of the game with the current state.
   - Replace all the players with Scripted AI.
   - Make a choice (pick one of possible choices from the list).
   - Let the game run for a turn or few turns (depending on the type of choice).
   - Estimate the final result. Estimation is different depending on the choice. For example if it is about choosing a resource to be sold on the Market then the estimation is quite simple—the amount of money earned is compared to the amount of money earned by other players. If it is about a role to choose then player's virtual score after few rounds of the game is estimated. This estimation is made on the basis of goods, money and holdings, as well as on current victory points.
4. Repeat a simulation multiple times for each choice and prepare the statistics.
5. When it is time to make a choice, stop the simulation and choose statistically best action.

Again, there is a slight difference for re-allocating colonists action. To save some time, most of the colonists are placed with the use of scripted algorithm (because usually it is an obvious choice), only for few last colonists simulation is made.

Choosing a Role may be implemented as follows. First, for every possible choice 5 rounds of the game are played. Next the score = $\sum_{P_i}$ Difference(P_i) is calculated, where *P(i)* are all other players in the game.

Difference is calculated as follows:

$$\text{Difference} = (Pr+20)\cdot(VP0-VP_i)+(80-Pr)\cdot(Pos0-Pos_i) \tag{1}$$

where Pr is a game progress valued from 0 to 100, $VP0$ are Victory Points of AI, VP_i are Victory Points of i-th player, $Pos0$ is Virtual Score (based on possession state) of AI and Pos_i is Virtual Score (based on possession state) of i-th player.

Possession state is calculated in the following way:

$$\begin{aligned}\text{Pos} = (100\cdot D+50\cdot C+\sum_{R_i}\text{Quan}(R_i)(100+50\cdot\text{Val}(R_i))+\text{Prod}(R_i)\cdot(200+\\ 100\cdot\text{Val}(R_i))+450\cdot Q+\sum_{Pl_i}50+25\cdot\text{Val}(Pl_i)+\sum_{B_i}150\cdot\text{Prize}(B_i)\end{aligned} \tag{2}$$

where D is Quantity of doubloons, C is Quantity of colonists, R_i is type of resource, Quan is Quantity of resource, Val is Value of type of the resource (Corn = 1, Indigo = 2, Sugar = 3, Tobacco = 4, Coffee = 5), Prod is Production capacity of the resource, Q is Quantity of Quarries, Pl_i is i-th plantation in possession, B_i is i-th building in possession and Prize is prize for building.

It is worth to notice (Eq. (1)) that game progress is very important. In early stages of the game it is more important to have high Virtual Score (resources that will allow to produce Victory Points later), while later it is more important to have Victory Points.

There are of course some pros and cons of such AI technique. Firstly, we can say that this AI is "thinking", choices are not coded by the creator—instead they are the results of simulations. Implementation of this algorithm gives good results and do not require a lot of work. This approach possesses a highly challenging strategy skills, capable of competing with experienced human player. It can dynamically react to the changes taking place on the board and adjust accordingly its tactics. Its actions and tactics are very hard to predict.

The biggest disadvantage of the second generation AI is that it is dependent on other (scripted) AI, it simply can not work without it. And actually its skills also depend on how precisely the scripted AI was developed. But even with quite basic implementation of the first generation AI, the second generation is working very well. The time needed for performing moves is noticeable, but not annoying for human players. This technique, when given more time for performing a move, gives slightly better results, but on any modern PC quite short time is enough for very good performance (it is usually between one and several seconds for a single action). Implementing estimation functions—which are most important for making a proper choice—can be quite difficult. The functions can be obtained by a series of additional simulations or by applying reinforcement learning technique, which seems to be a next step to be made in order to further improve AI's efficiency.

6 Experiments and Results

6.1 Comparison of Both AI Mechanisms

Assumption of the research was that Thinking AI should be a perfect opponent, able to win with Scripted AI—at least if given enough time for thinking. “Thinking time” is a maximum time dedicated for performing simulations by Thinking AI. Some of the functions use the whole time, some only 1/2 or 1/4 of a given time. A series of experiments were carried out in order to test that. The configuration details of the machine used during experiments are as follows: Intel i5-2500k CPU, 3.30GHz, 16GB RAM running MS Windows 7. Rules of the experiments were as follows:

- Run tests for 4 possible settings of thinking time: 100ms, 2.5s, 5s,10s.
- Run 100 games for each series, with 4 players, 2 of each AI types.
- Check the game winner and prepare statistics based on that.

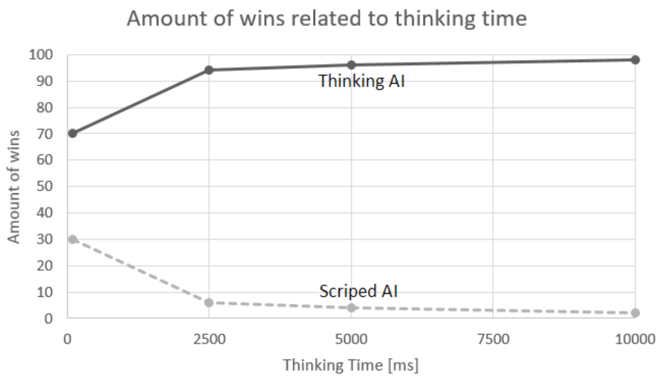


Fig. 1: Comparison of AI mechanisms

Results (see Fig. 1) were a bit surprising. It could be expected that with minimal thinking time Scripted AI would win because Thinking AI would perform quite random moves, but in reality it was opposite. Apparently 100ms was enough to perform such a number of simulations that Scripted AI could be outplayed. With 2.5s thinking time the second generation AI almost always won.

6.2 AI vs Human Players

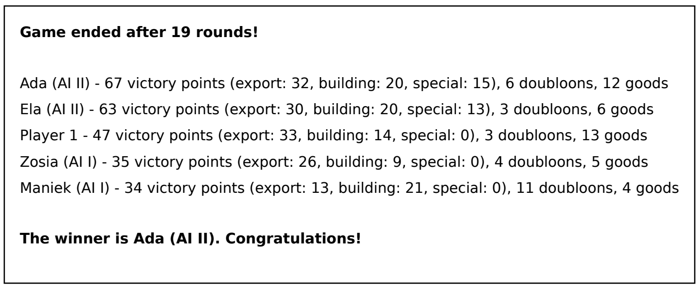
Experiments in which AI competed with human players were much more difficult to carry out. During those experiments, one of the authors of this paper and few other players with various range of strategy skills played about 50 games.

Most of the games were played by an expert player against 2 AIs of both types. In these games each player received points on the basis of his position at the end of the

Table 1: Results of competition between AIs and expert player

Player	**Score in consecutive games**																				**Average score**
AI Scripted 1	1	2	1	1	2	1	2	3	1	2	2	2	1	3	1	1	2	1	1	1	1.55
AI Scripted 2	2	1	2	2	1	2	1	1	2	1	3	1	2	1	2	2	1	2	2	2	1.65
AI Thinking 1	4	5	4	3	4	5	4	2	3	5	5	5	3	4	3	4	3	5	4	5	4
AI Thinking 2	3	3	5	5	3	4	3	4	5	4	4	3	4	2	5	5	5	4	3	4	3.9
Expert Player	5	4	3	4	5	3	5	5	4	3	1	4	5	5	4	3	4	3	5	3	3.9

game (5 points for winning, 4 for second place, etc.) Then statistical data was computed on the basis of the results of 20 games in order to compare performance of human expert player during competition against different AIs (see Table 1). As we can see, Scripted AI had no chance at all, while Thinking AI scored almost the same as expert player. Statistics from the games played against beginner or intermediate human player are currently not available because the focus of this research was to create the best possible AI player capable of competing with an expert human player. However a few games were played also against less advanced human players and typical results of one of such games are presented in the Fig. 2.



```
Game ended after 19 rounds!

Ada (AI II) - 67 victory points (export: 32, building: 20, special: 15), 6 doubloons, 12 goods
Ela (AI II) - 63 victory points (export: 30, building: 20, special: 13), 3 doubloons, 6 goods
Player 1 - 47 victory points (export: 33, building: 14, special: 0), 3 doubloons, 13 goods
Zosia (AI I) - 35 victory points (export: 26, building: 9, special: 0), 4 doubloons, 5 goods
Maniek (AI I) - 34 victory points (export: 13, building: 21, special: 0), 11 doubloons, 4 goods

The winner is Ada (AI II). Congratulations!
```

Fig. 2: Typical results of competition between intermediate human player and various AIs (*AI I* is Scripted AI, *AI II* is Thinking AI)

The following observations may be formulated on the basis of the obtained results. Scripted AI is a good opponent for beginning player or a player with lower strategy skills. Intermediate player will often win with the Scripted AI, but winning against Thinking AI will be almost impossible. Human player needs to possess good strategy skills to be able to win with it at all. Advanced player is usually capable of beating the first generation AI and sometimes the second generation AI. Only very good player would win with Thinking AI more often, but still not always. On the basis of the performed experiments it is safe to assume that Thinking AI's skills are similar to such an experienced human player.

7 Summary and Conclusions

The AI players proposed in this paper were, more or less, performing in accordance with our expectations. Two types of AI, with different playing styles and skills, can ensure a good game experience both for beginner and advanced human players. It is really hard to get bored when playing against such AI players—actions of both of them are rather hard to predict. Especially Thinking AI is very promising technique because it can adapt and dynamically react to other players' actions.

It is worth to notice that there are still possibilities to improve both AI mechanisms. Improving Scripted AI relies on tests and observation. We can try to improve conditional statements and weights calculations used during decision making. This technique can be also greatly improved by adding more conditional statements, taking into account additional situations. Possibly, also dynamic scripting could be used here, however it would require much more programming effort.

Improving Thinking AI can be done automatically when improving Scripted AI. Improvement of Scripted AI would impact simulation results because moves would be performed against stronger opponents. Another way to improve Thinking AI is to create better formulas for approximation of the best moves. The most important thing would be more precise approximation of a real value of opponent player's assets. It could be done by performing additional simulations or by applying reinforcement learning technique. The goal would be to find better ways of valuing different kind of opponent players' properties, for example how to value buildings, plantations and colonists as compared to doubloons and victory points. Creating better formula could eventually lead to choosing much better moves from the simulated ones.

The proposed Thinking AI technique, of course when correctly implemented, seems to be highly effective and significantly improving skills of AI player. Due to the fact that it is based on the standard scripted AI, it could be used along with other known techniques (that improve efficiency of the scripted AI) and thus made even more challenging for human players.

In the future research we plan also to adapt and use the techniques proposed in this paper in agent-based strategy and tactical games, where agents are individuals interacting independently with each other and with an environment. In such a case AI mechanisms are encapsulated within each agent and thus many different AI techniques can be coherently combined together (for example evolutionary algorithms, neural nets, multi-objective evolutionary optimization techniques, etc.)—we have already used such approach with great success in computational systems [6,5,7,4] and it seems that it can also bring many advantages to the construction of AI for computer games. The simulation technique proposed in this paper could be used by each agent in order to assess its future moves and to select the best one.

Acknowledgments. This research was partially supported by Polish Ministry of Science and Higher Education under AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications statutory project no. 11.11.230.124.

References

1. BoardGameGeek. https://boardgamegeek.com/browse/boardgame
2. Puerto Rico (board game). https://en.wikipedia.org/wiki/Puerto_Rico_(board_game)
3. Amato, C., Shani, G.: High-level reinforcement learning in strategy games. In: W. van der Hoek et al. (ed.) Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010). pp. 75–82. IFAAMAS (2010)
4. Cetnarowicz, K., Dreżewski, R.: Maintaining functional integrity in multi-agent systems for resource allocation. Computing and Informatics 29(6), 947–973 (2010)
5. Dreżewski, R., Sepielak, J.: Evolutionary system for generating investment strategies. In: M. Giacobini, et al. (ed.) Applications of Evolutionary Computing, EvoWorkshops 2008. LNCS, vol. 4974, pp. 83–92. Springer-Verlag, Berlin, Heidelberg (2008)
6. Dreżewski, R., Siwik, L.: Multi-objective optimization technique based on co-evolutionary interactions in multi-agent system. In: M. Giacobini, et al. (ed.) Applications of Evolutionary Computing, EvoWorkshops 2007. LNCS, vol. 4448, pp. 179–188. Springer-Verlag, Berlin, Heidelberg (2007)
7. Dreżewski, R., Siwik, L.: Co-evolutionary multi-agent system for portfolio optimization. In: Brabazon, A., O'Neill, M. (eds.) Natural Computing in Computational Finance, vol. 1, pp. 271–299. Springer-Verlag, Berlin, Heidelberg (2008)
8. Millington, I., Funge, J.: Artificial Intelligence for Games. CRC Press (2009)
9. Spronck, P., Ponsen, M.J.V., Sprinkhuizen-Kuyper, I.G., Postma, E.O.: Adaptive game AI with dynamic scripting. Machine Learning 63(3), 217–248 (2006)
10. Szepesvári, C.: Algorithms for Reinforcement Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool Publishers (2010)
11. Wender, S., Watson, I.: Using reinforcement learning for city site selection in the turn-based strategy game Civilization IV. In: Hingston, P., Barone, L. (eds.) CIG. pp. 372–377. IEEE (2008)