What's in a Game? The Effect of Game Complexity on Deep Reinforcement Learning

Erdem Emekligil(✉) and Ethem Alpaydın

Department of Computer Engineering, Boğaziçi University, 34342 Istanbul, Turkey
{erdem.emekligil,alpaydin}@boun.edu.tr

Abstract. Deep Reinforcement Learning (DRL) combines deep neural networks with reinforcement learning. These methods, unlike their predecessors, learn end-to-end by extracting high-dimensional representations from raw sensory data to directly predict the actions. DRL methods were shown to master most of the ATARI games, beating humans in a good number of them, using the same algorithm, network architecture and hyper-parameters. However, why DRL works on some games better than others has not been fully investigated. In this paper, we propose that the complexity of each game is defined by a number of factors (the size of the search space, existence/absence of enemies, existence/absence of intermediate reward, and so on) and we posit that how fast and well a game is learned by DRL depends on these factors. Towards this aim, we use simplified Maze and Pacman environments and we conduct experiments to see the effect of such factors on the convergence of DRL. Our results provide a first step in a better understanding of how DRL works and as such will be informative in the future in determining scenarios where DRL can be applied effectively e.g., outside of games.

Keywords: Deep reinforcement learning · Computer games

1 Introduction

In their seminal work, Mnih et al. [10] show that a Deep Q-learning Network (DQN) can learn to play ATARI 2600 games end-to-end. The neural network takes the raw screen image as input, which it processes through a number of layers (first convolutional, then fully-connected), and its output units directly control the actions of the joystick. They show that DQN needs no fine-tuning for each task and that using the same learning algorithm, network architecture and hyper-parameters, any one of the 49 games can be learned.

They report that DQN "outperforms competing methods in almost all the games, and performs at a level that is broadly compatible with or superior to a professional human games tester in the majority of games." They also note that "the games in which DQN excels are extremely varied in their nature," but that

T. Cazenave et al. (Eds.): CGW 2018, CCIS 1017, pp. 147–163, 2019.
https://doi.org/10.1007/978-3-030-24337-1_8

"games demanding more temporally extended planning strategies still constitute a major challenge."

Our main idea in this paper is that there are factors that define the complexity of a game and that games that may appear different at first glance may actually be similar in terms of such abstract factors, or vice versa. For instance, whether the player is the only agent or if there are other agents, possibly hostile, that can act is such a factor. We assume not only that there are such factors but also that the speed a game is learned, e.g., by DQN, depends on these factors.

In this work, we are going to take the DQN as it is and test it on a number of settings where we vary such factors. It should be noted here that our aim is *not* to understand how DQN learns a particular task (game) or what its hidden units or layers are doing to handle that task, but rather we take the totality of DQN (the network, learning algorithm, and hyper-parameters) as a black-box and want to see what type of task attributes affect DQN's performance and how. The settings we use is simple by design so that we can easily observe the effect of changes on DQN's convergence. We believe that such a study is informative in understanding where and how DQN, or similar approaches, can best be used, and finding out such abstract factors that define a learning task and how such factors effect learning will especially be useful when we want to use models like DQN outside of the game-playing domain.

2 Background

The DQN takes four consecutive screen images as input which it processes by three convolutional and then two fully-connected layers with a final output layer where there is one output for each valid joystick action. Q-learning is used to update the network weights, with experience replay to randomize over data. We do not discuss DQN any further here; the interested reader is referred to [10].

Since then, the related literature can be divided into two, as work where DQN is generalized for other tasks, and works that strive to improve the performance of DQN.

As examples of the first, Levine et al. [7] use supervised learning before reinforcement learning for more complex tasks. Similarly in AlphaGo, Silver et al. [14] take advantage of supervised data to initially train a network that evaluates the Go board. They combine Monte Carlo tree search with a DQN variant to create an agent that is able to beat the human player with the second best Elo rating at the time. AlphaGo Zero [16] goes one step further and is trained without any supervised data, and is able to beat AlphaGo. Alpha Zero [15], which is the generalized version of AlphaGo Zero, learns to play Shogi and Chess better than the best computer programs for each game. DQN is also used outside of the domain of game-playing; for example, a version with 1D convolution is used for simulating animal movements [12]. It is also altered for control tasks such as cart-pole, locomotion and car driving problems by using deep function approximators [8].

As improvements to DQN, Double-DQN (DDQN) uses a backup network as the target action value function [5]. The actual network is used for selecting the action with the maximum value whereas the target network is used for estimating that action's value. Instead of random picks in experience replay, Schaul et al. [13] define different methods to prioritize the picks from replay memory and show that prioritizing achieve major improvements. Wang et al. [17] introduce a new architecture that combines a state value function and an action advantage function. DQN is also adapted to work on distributed [11] and multi-core CPU systems [9] using actor-critic (A3C) methods with LSTMs. Unlike previous methods, Distributional DQN [2] can distinguish risky actions and the Rainbow method [6] unites six previous developments over DQN and show that these different methods achieve better results when they are combined.

3 Factors that Define a Game

It is our contention in this paper that games that seem very different at first glance may be very similar at a more abstract level and beyond their immediate facade, games can be defined in terms of a number of factors, and furthermore it is these factors that define the complexity of a game and the best strategy to play it well.

There is previous work on the various characteristics of games and how they effect playability, by humans and AI programs; see the book by Elias et al. [4]. Anderson et al. [1] compare performances of several tree-based search algorithms such as MinMax, MCTS and A* on seven different games that they have created. Yannakakis et al. [18] define five different factors based on [4] and explain their effects on AI methods:

- Number of players. Is the game played by a single player or multiple players, or does a single player play with/against computer controlled enemy units?
- Stochasticity. Does the outcome of the game determined only by the player?
- Time granularity. Is the game turn-based or real-time?
- Observability. Is the game partially observable or the player has perfect information?
- Action space size. The number of the actions the player can take.

In our case, we take DQN as the game-learner and consider games that can be learned by DQN, similar to the ATARI 2600 games, where we can define and test the effect of such factors.

We start by clustering the ATARI 2600 games according to how DQN learns them. We run DQN on 45 games and for each we record the convergence of DQN in terms average game scores. We then use dynamic time-warping (DTW) [3] to measure the distance between vectors of different lengths, each normalized between 0 and 1, since each game runs for a different number of epochs. The dendrogram achieved using average-linkage hierarchical clustering is shwown in Fig. 1, with convergence plots of some example games.

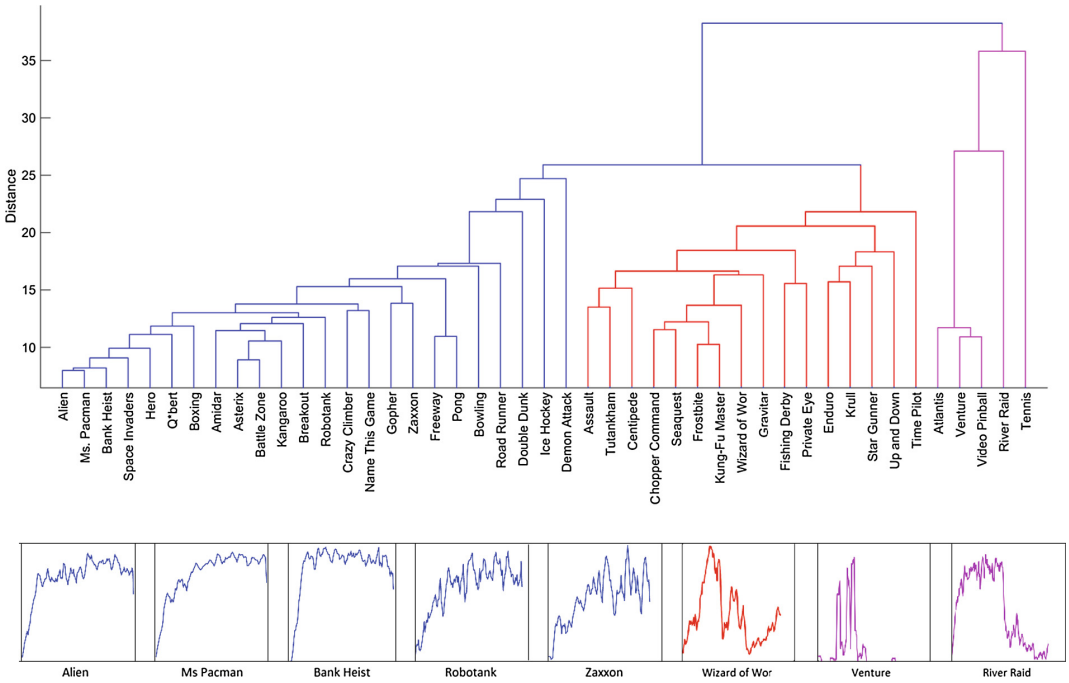
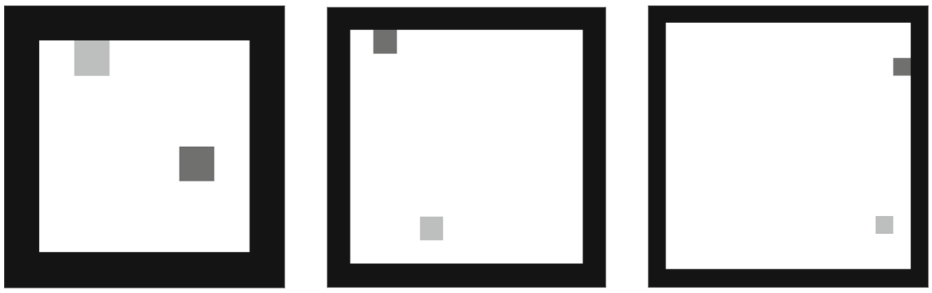


Fig. 1. Hierarchical clustering results of ATARI games, and the convergence of DQN on some example games.

Games whose convergence plots are similar are placed nearby in the tree and for certain cases, we can see that actually they correlate with similarities between the games.

For example, Ms Pacman and Alien, connected early on, are very similar both in terms of how DQN learns them and also in terms of how they are played. They are similar in many aspects such as actions, objective, rewards and so on; in both games, the agent tries to collect as many reward (food items) as possible without getting caught by the enemy and both have power-up units that temporarily give the ability to destroy opponents. Interestingly, Bank Heist looks similar to these but in Bank Heist, instead of power-up, the agent can counter enemies by bombs which changes the strategy. This difference changes the convergence of DQN and this explains the distance of Bank Heist to Ms Pacman and Alien on the tree.

Wizard of Wor and Alien are games that look very similar visually, but they are played differently and hence the convergence behavior of DQN is different and they are very distant in the dendrogram despite their visual similarity. River Raid and Venture are two games on which DQN fails and that is why they are on the same cluster. In Zaxxon and Robotank, the agent controls an airplane, a tank and the main objective is survival whether from an airplane or a tank crash. The games look similar, are played similarly and DQN convergence curves on them are similar and that is why they are not far from each other on the dendrogram.



(a) 8 × 8 maze (b) 12 × 12 maze (c) 16 × 16 maze

Fig. 2. Example mazes of different sizes with outer walls (black), the target (dark gray) and the agent (light gray).

4 Maze Experiments

In this simple Maze task, the objective of the agent is to get to the stationary target by moving one square, horizontally or vertically, in each step. The position of the agent and the target are randomly chosen in each episode and a score of 100 is awarded when the agent reaches the target before the maximum number of allowed moves ((Width + Height) ∗ 10).

In our experiments, we keep the original DQN network, learning algorithm, and hyper-parameters of [10][1] and test it first on a maze. The only parameter changed is the decay rate ϵ, which is adapted to the complexity of the maze by setting it to its minimal value that allows convergence. Since our generated mazes are much smaller, they are stretched to fit 84×84; the mazes contain the outer walls so the actual playable area is one less on all four sides. Each agent is evaluated after every 250,000 training frames for 125,000 test frames and the average episode score is plotted.

4.1 The Effect of the Size of the Search Space

We start by testing the effect of the maze size, which is an indication of the search space: A larger maze requires longer sequences of actions. We use mazes of 8×8, 12×12 and 16×16. A maze contains only the outer walls, the agent and the target. The walls are colored white, the target is dark gray, the agent is light gray. The background is black (encoded as 0) to help the network to learn faster (see Fig. 2; the colors are inverted to save from ink). We did five runs with different random seeds and plot the one that best represents the average behavior.

Because increasing the maze size increases the average number of actions required to get to the goal, as expected, and as we see in Fig. 3, the number of training epochs it takes for DQN to converge also increases.

[1] We expand and use Nathan Sprague's replication: https://github.com/spragunr/deep_q_rl.

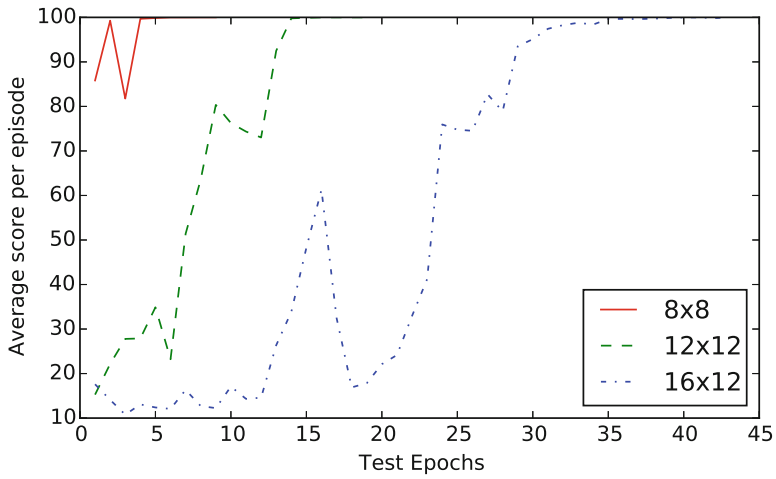


Fig. 3. Convergence of DQN as a function of maze size. As expected, larger mazes take longer to learn.

(a) Maze with a horizontal wall (b) Maze with a vertical wall (c) Maze with two walls

Fig. 4. Example 12×12 mazes.

4.2 The Effect of Obstacles

The complexity of the path to solve the maze can be increased by adding obstacles. The agent cannot just take any path to the goal but needs to recognize and avoid the obstacles. In our experiments, we simulate this by a wall with a single gate. The positions of agent and target, as well as the location of the gate and the wall orientation are also randomly assigned in each episode. To make the task more complex, we also experimented with two intersecting walls that divide the maze into four playable areas connected by three randomly located gates. Randomly generated examples are shown in Fig. 4.

We trained DQN on these three setups with different obstacle structures (no wall, one wall, two walls) and three different sizes just like in the previous experiment. In Fig. 5, we see that because adding walls increases the path complexity and consequently the number of actions to achieve the goal, convergence of DQN is much slower needing more training iterations; with larger mazes, the differences get larger.

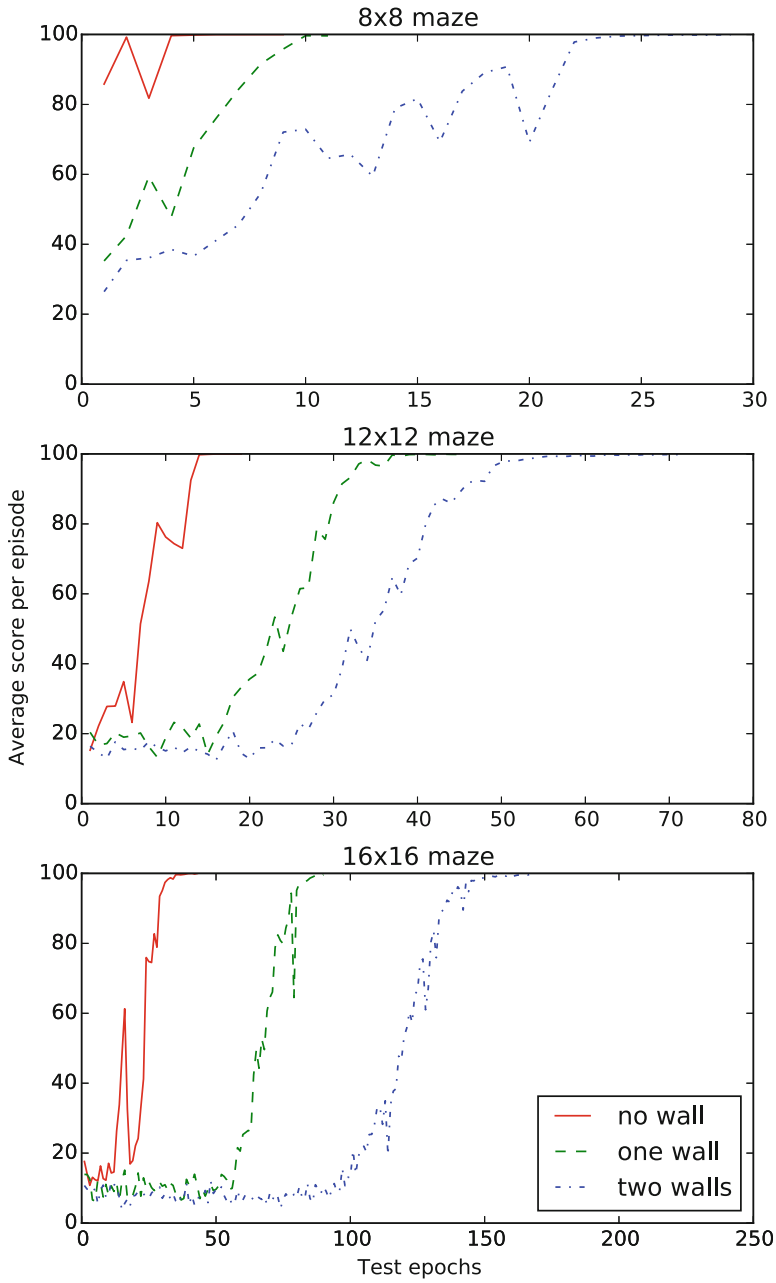


Fig. 5. Convergence of DQN as a function of maze sizes and wall structures. With more obstacles, learning gets slower.

4.3 The Effect of Hostile Agents

If the game-player is not the only agent that can change the environment, the presence and actions of other agents make the task harder. In our maze experiments, we added stationary unit-sized enemies that ends the game on contact (without any reward), to test if the network can learn to recognize and avoid them efficiently. As shown in Fig. 6, a different gray level is chosen to encode

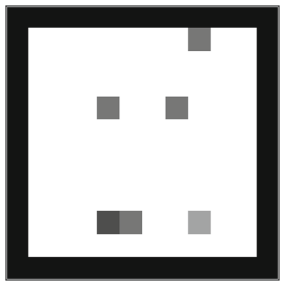


Fig. 6. A 12×12 sample maze with outer walls (black), a target (dark gray), an agent (light gray) and four enemies (medium gray).

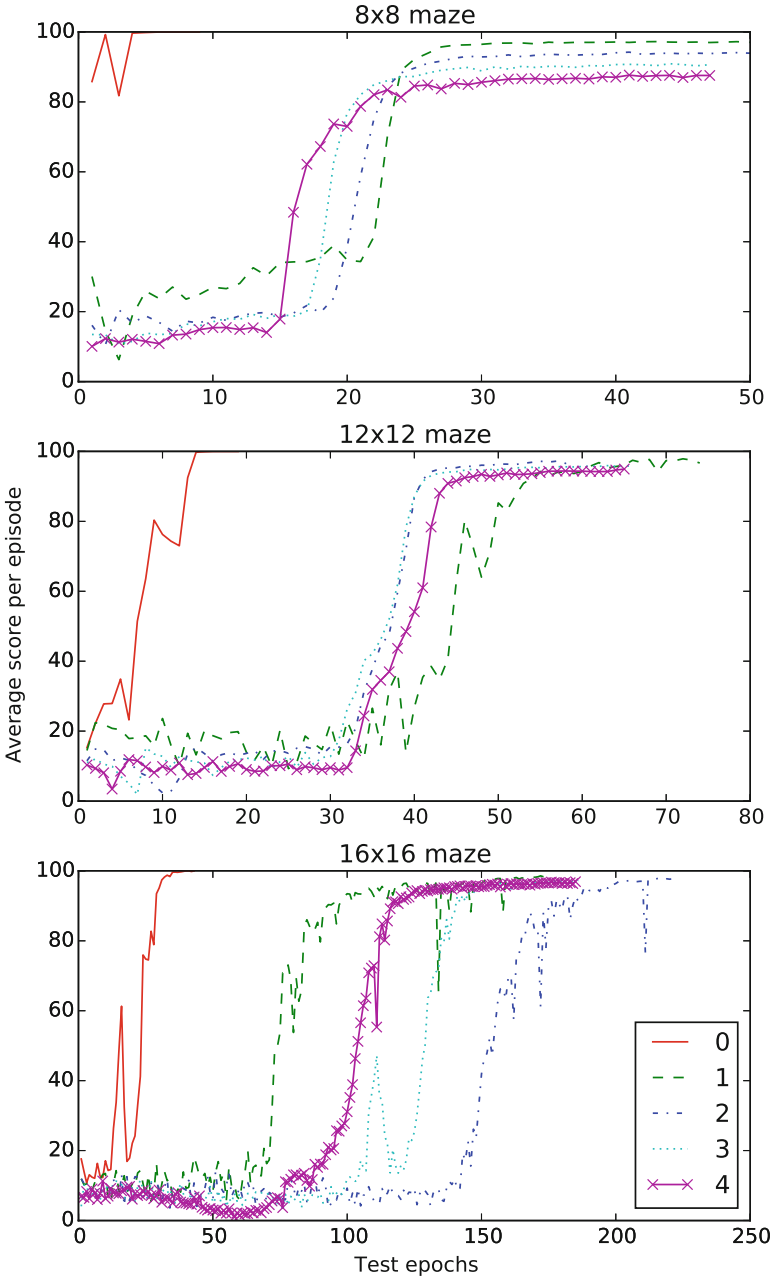
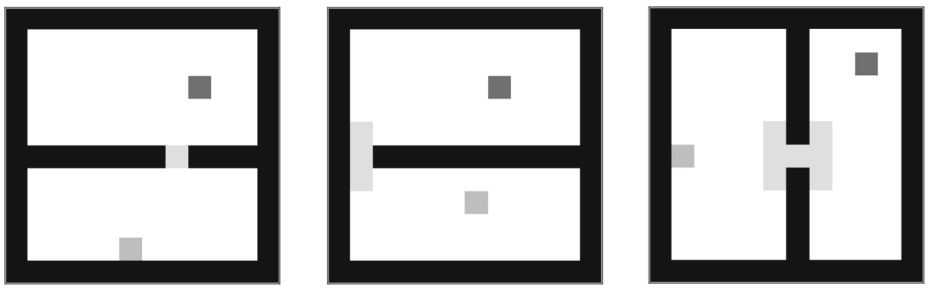


Fig. 7. Convergence of DQN as a function of maze size and the number of enemies. It is whether there is any enemy or not, rather than the number of enemies, that slows down learning.



(a) Maze with 1×1 intermediate reward

(b) Maze with 1×3 intermediate reward

(c) Maze with 3×3 intermediate reward

Fig. 8. 12×12 mazes with different sizes of intermediate rewards.

the enemy, and the locations of these enemy units are chosen randomly in each episode.

Our results are given in Fig. 7, where we see that the presence of an enemy makes the task harder to learn, regardless of maze size. The number of enemies, as long as it is nonzero, does not seem to have a drastic effect. Plots with 2, 3, 4 enemies seem to be clustered together for mazes of 8×8 and 12×12; for the maze of 16×16, we believe that the variability is due to chance. Once DQN learns to recognize an enemy and how to avoid it, and it is enough to do enough episodes with a single enemy for that, DQN can then recognize and avoid any number of enemies that it later encounters.

4.4 The Effect of Intermediate Reward

In most games, the reward is given not only at the end but also at some special intermediate state, such as destroying an enemy unit or passing through a checkpoint. Such an intermediate reward is useful in hinting the learning agent that it is on the correct path to the goal. In our maze experiments, we implement this in the case with one wall with a gate and by giving a reward of 10 as an intermediate reward upon reaching the gate. Afterwards, if the agent achieves its goal an additional 90 points is given to get the same total of 100.

In this experiment, we use different sized mazes all having one wall. The target and agent locations are forced to be on different sides of the wall (this was not forced in previous experiments) and the intermediate reward area is given in a different color, close to white. We tested using different sizes of intermediate reward areas to check if extending the reward area increases the learning speed (see Fig. 8).

As we can see in Fig. 9, adding an intermediate reward increases the learning speed as the maze size gets larger. In the 8×8 setting, the search space is already so small that no intermediate reward seems necessary. But especially in the 16×16 maze with its larger search space it helps and is more helpful when the intermediate reward area gets larger.

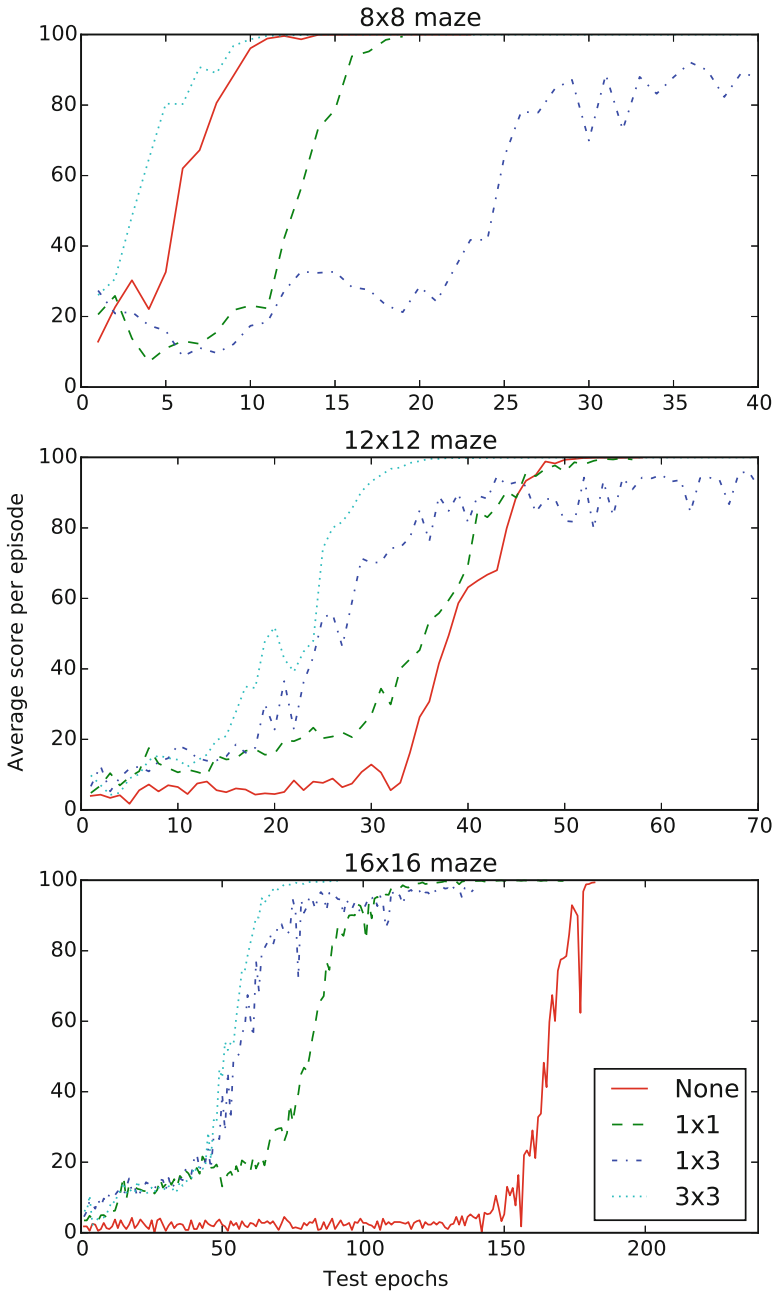
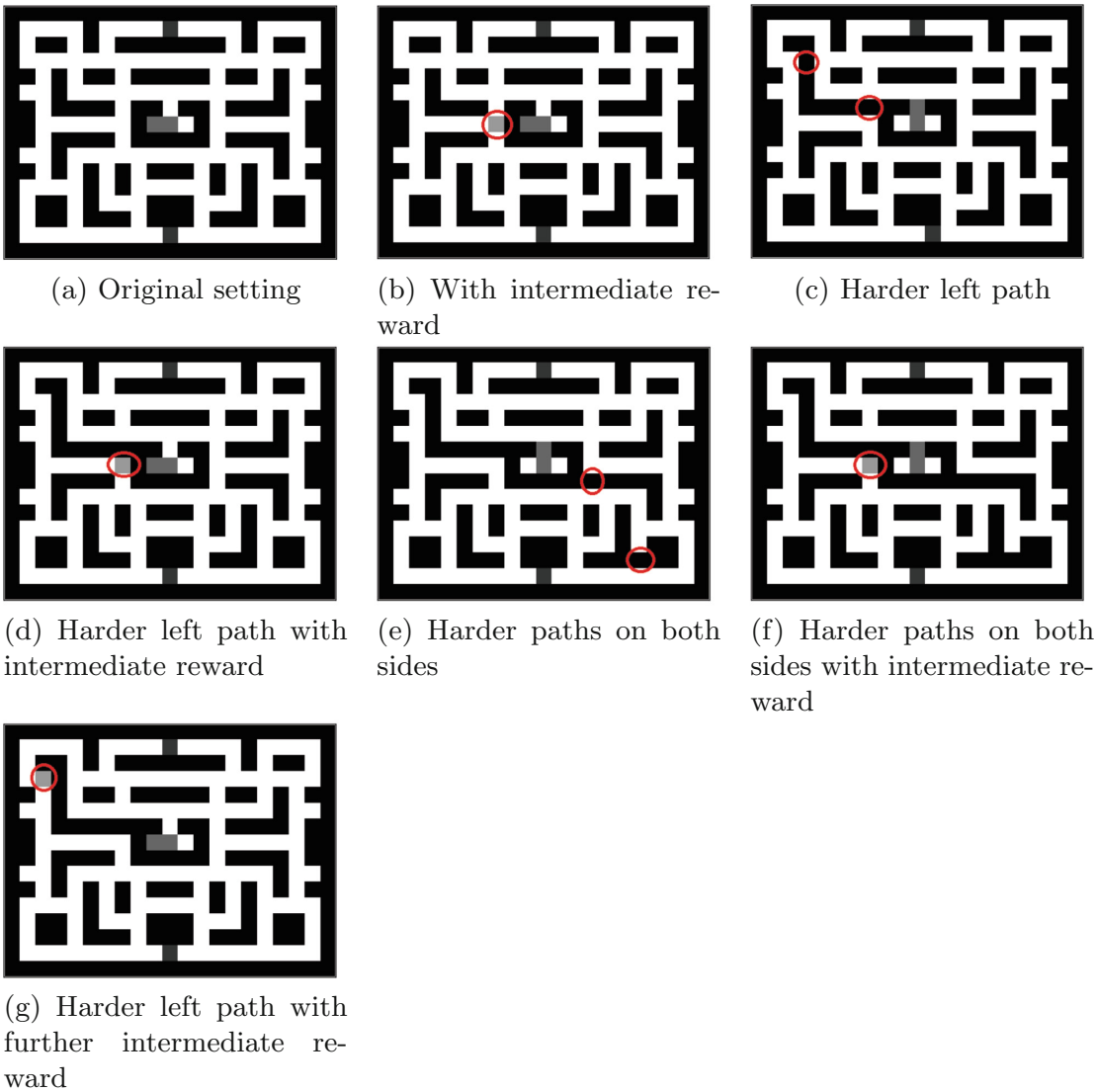


Fig. 9. Convergence of DQN as a function of maze size and the intermediate reward area. With small mazes intermediate reward does not help, but as the maze gets larger, larger areas of intermediate reward help more.

5 Pacman Experiments

For our next set of experiments, we use the game of Pacman, which is a more interesting maze game[2]. This environment provides customizable mazes and

[2] We use the Pacman environment prepared for the course UC Berkeley CS188 Introduction to AI, available at http://ai.berkeley.edu/reinforcement.html.



(a) Original setting

(b) With intermediate reward

(c) Harder left path

(d) Harder left path with intermediate reward

(e) Harder paths on both sides

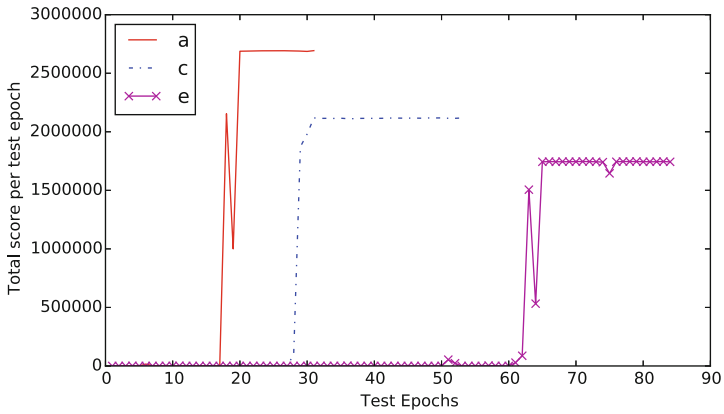
(f) Harder paths on both sides with intermediate reward

(g) Harder left path with further intermediate reward

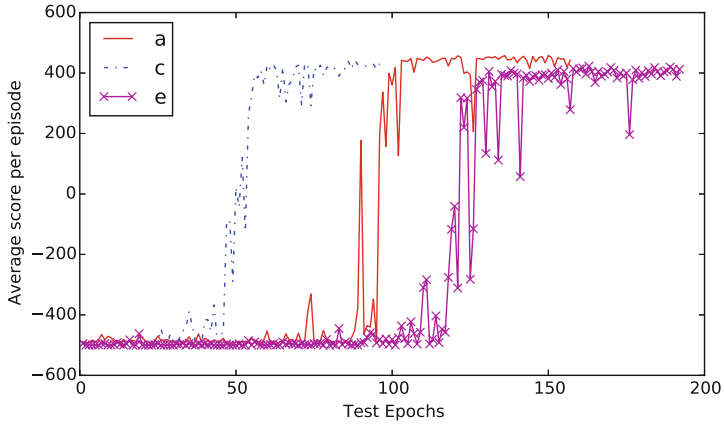
Fig. 10. The different setups of Ms. Pacman environment we tested DQN on. The agent starts from the bottom and tries to get the goal at the top. Two enemy units are shown at their spawn location in the center. Intermediate reward is colored in brighter gray. Changes between setups are denoted with red circles

basic AI options for enemy units. We adapted the maze style of the Ms. Pacman ATARI game to this environment and selected enemy units with random movement capabilities. These enemies pursue their path until they reach the end and select their next path randomly when they are at a junction point. A contact with an enemy ends the game with -500 reward points. Intermediate reward gives $+10$ points whereas the goal gives $+500$.

Just like in the maze experiments, we test for factors that change the complexity of the game to see their effect on the convergence of DQN. There are three factors: (a) There are two paths from the initial position at the bottom to the goal at the top, and one can block either of them or not, (b) There may be



(1) No enemies

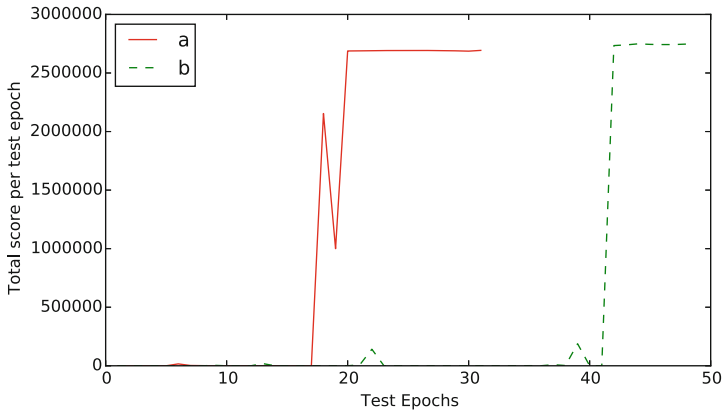


(2) Two enemies

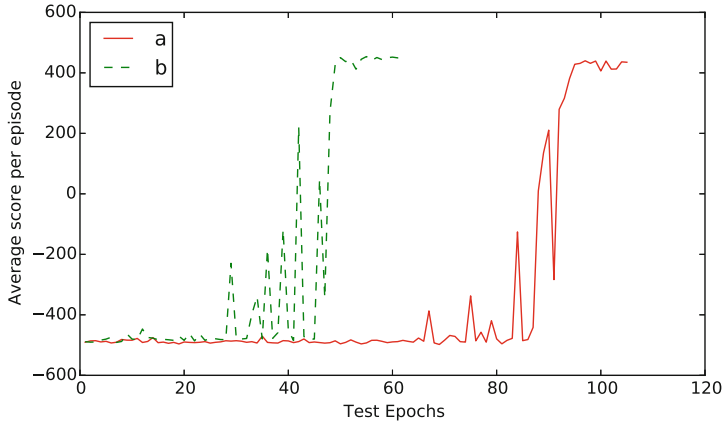
Fig. 11. Comparison of DQN convergences on original Pacman setup (a), harder left path (c) and harder paths on both sides (e)

enemy units or not to avoid contact with, and (c) There may be intermediate rewards on the correct path. We also try combinations of those factors and in Fig. 10, we show the seven different setups. By training DQN in each of these setups with zero or two enemy units, we experiment with a total of fourteen different setups.

We start by closing some paths by adding obstacles (see Fig. 11). In setup (c), we make one of the two possible routes longer by closing some paths and in setup (e), both the leftmost and the rightmost solution paths are extended by adding obstacles. We can see in Fig. 11(1) that extending the leftmost path (c) decreases learning with respect to (a). However, if we examine Fig. 11(2), we can see that adding some enemy units to (c) increases the learning speed, since most of the time the agent gets destroyed in the leftmost path by the enemies, thus the agent learns that it should not dwell on the leftmost path concentrating on the correct path which is the rightmost path. If we compare setup (e) with (a) and (c), we see that making all possible paths longer decreases the learning speed, as expected.



(1) No enemies

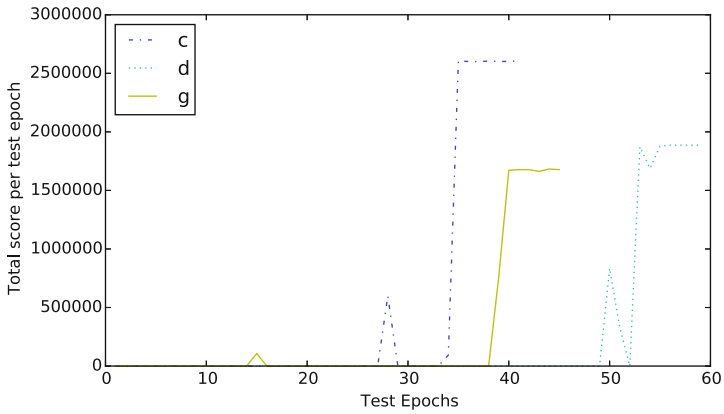


(2) Two enemies

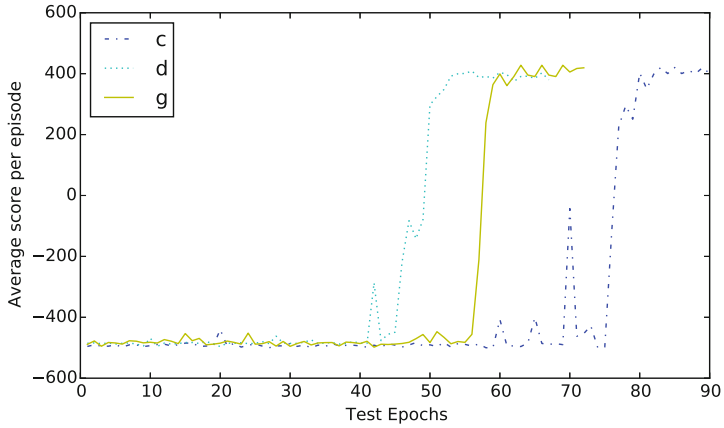
Fig. 12. Comparison of DQN convergences on original Pacman setup (a) and intermediate reward (b)

If we examine setups (a) and (b) (see Fig. 12), we see that adding an intermediate reward increases DQN's learning speed considerably in the two-enemy setup. This is expected since the intermediate reward helps algorithm to focus on one of the two paths. In the no-enemy setup however, the agent learns getting the intermediate reward quickly, but it tends to stay near the intermediate reward for many epochs to come since ϵ value gets its lower bound in four epochs and the agent cannot explore the target quickly. Thus, it decreases the learning speed.

The intermediate reward in setup (d) was actually designed to hinder learning since it is en route to a longer path. In the no-enemy case, the results show that this reward actually hinders learning as can be seen in Fig. 13. After the agent takes this reward, instead of choosing the leftmost path, it uses the rightmost path which is the closer one to the target. However, it results with even better outcomes than setup (c) when there are enemies. The agent in this case is able to learn to choose between left or right paths according to the closeness of the enemies to those paths. In setup (g), we move the intermediate reward to a further position on the left path and saw that it decreases learning speed in



(1) No enemies

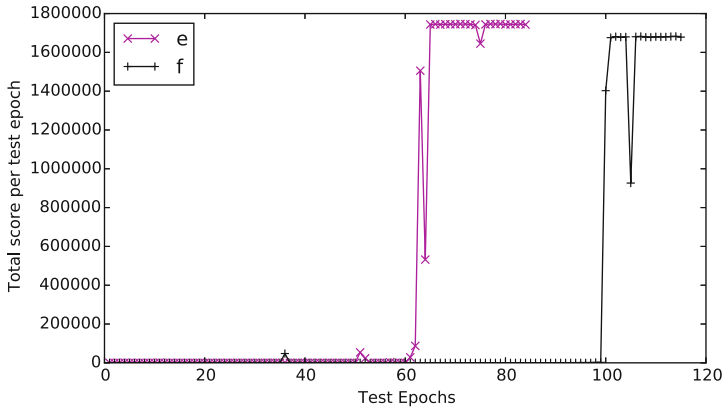


(2) Two enemies

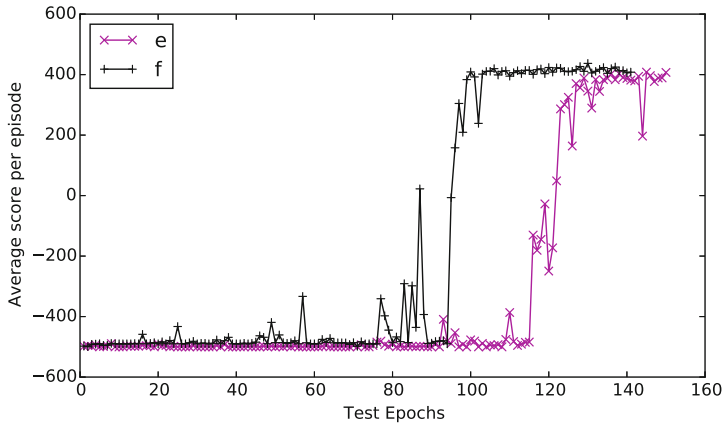
Fig. 13. Comparison of DQN convergences on Pacman setup with harder left path (c), early intermediate reward (d) and hard intermediate reward (g)

no-enemy setup, better than (d). This also helps learning when there are two enemies and provides worse results than (d), because the risk of death is higher in the path with the intermediate reward. Nevertheless, when the algorithm converges, the agent is able to wait for risk of the opponents to pass and go back if it is necessary.

Comparison of setups (e) and (f) are given in Fig. 14. Similar to the previous results, since the ϵ value gets to its minimum value quickly, the intermediate reward in this case (f) slows down the learning process. Slow down rate is extremely high because most of the paths are blocked and the agent cannot get the target. It helps when there are enemies, because enemies force the agent to explore.



(1) No enemies



(2) Two enemies

Fig. 14. Comparison of DQN convergences on Pacman setup with harder paths on both sides (e) and intermediate reward (f)

6 Conclusions and Future Work

Deep reinforcement learning is a recent research area that combines deep neural networks with reinforcement learning. The Deep Q-Network learns to play Atari games end-to-end; but it learns some games better, some faster, and we do not know why. Our assumption is that the complexity of a game depends on some factors and these factors affect DQN's learning speed and quality.

To validate this claim, we clustered the DQN convergence curves of 45 ATARI 2600 games. We find that there seems to be indeed a dependence between game characteristics and DQN performance, that games that are played similarly are learned similarly by DQN and are placed nearby in the clustering dendrogram, whereas games that look the same visually but need different strategies are placed far apart.

We defined variants of a Maze task on which we defined a number of factors and tested their effect using DQN as it is, with no changes to the network architecture or learning algorithm. The four factors we tested are the maze size, presence/absence of walls, presence/absence of enemies, and presence/absence

of intermediate reward. We see that larger mazes and the presence of enemy units generally delay convergence since the environment becomes more complex. Intermediate rewards, however, increase the learning speed since they provide a hint for the main goal, dividing a long sequence into smaller sequences.

In the second set of experiments, we use a Pacman environment with similar factors with a total of 14 different setups. We find that the factors affect learning differently if there are enemies. Blocking a path with a wall usually decreases the learning speed, but has the opposite effect if it is blocking many of the possible paths thereby reducing possible actions.

Overall, most of the experiments led to expected results, but not all. These cases should be further investigated to see if they are due to chance, or if there is any dependence or interaction between factors that we cannot see immediately. Another future research direction is to add more factors that affect the difficulty. For example, in most games there is randomness. In our maze setup, the initial positions of units are randomly selected but the moves of agents are not random; it could certainly affect the learning performance if any random event should occur during the game.

The ultimate aim is to transfer what we learn from DQN's behavior on games to what deep reinforcement learning can do in real life. From these experiments, we would like to move a level up and define at a more abstract level, general tasks and general strategies to solve them, as well as how such strategies can be learned. Our work is one small step towards this aim.

Acknowledgements. We would like to express our gratitude to Yapı Kredi Teknoloji A.Ş. for supporting us in participating to IJCAI 2018 events.

References

1. Anderson, D., Stephenson, M., Togelius, J., Salge, C., Levine, J., Renz, J.: Deceptive games. In: Sim, K., Kaufmann, P. (eds.) EvoApplications 2018. LNCS, vol. 10784, pp. 376–391. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-77538-8_26
2. Bellemare, M.G., Dabney, W., Munos, R.: A distributional perspective on reinforcement learning. In: Proceedings of the 34th International Conference on Machine Learning-Volume 70, pp. 449–458. JMLR. org. (2017)
3. Berndt, D.J., Clifford, J.: Using dynamic time warping to find patterns in time series. In: Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining, AAAIWS 1994, pp. 359–370. AAAI Press (1994)
4. Elias, G.S., Garfield, R., Gutschera, K.R.: Characteristics of Games. The MIT Press, Cambridge (2012)
5. Hasselt, H.V., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, AAAI 2016, pp. 2094–2100. AAAI Press (2016)
6. Hessel, M., Modayil, J., van Hasselt, H., et al.: Rainbow: combining improvements in deep reinforcement learning. In: Thirty-Second AAAI Conference on Artificial Intelligence (2018)

7. Levine, S., Finn, C., Darrell, T., et al.: End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**(39), 1–40 (2016)
8. Lillicrap, T.P., Hunt, J.J., Pritzel, A., et al.: Continuous control with deep reinforcement learning. ArXiv e-prints, September 2015
9. Mnih, V., Badia, A.P., Mirza, M., et al.: Asynchronous methods for deep reinforcement learning. In: Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, 19–24 June 2016, pp. 1928–1937 (2016)
10. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
11. Nair, A., Srinivasan, P., Blackwell, S., et al.: Massively parallel methods for deep reinforcement learning. ArXiv e-prints, July 2015
12. Peng, X.B., Berseth, G., van de Panne, M.: Terrain-adaptive locomotion skills using deep reinforcement learning. ACM Trans. Graph. **35**(4), 81:1–81:12 (2016). https://doi.org/10.1145/2897824.2925881
13. Schaul, T., Quan, J., Antonoglou, I., et al.: Prioritized experience replay. ArXiv e-prints, November 2015
14. Silver, D., Huang, A., Maddison, C.J., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484–489 (2016)
15. Silver, D., Hubert, T., Schrittwieser, J., et al.: Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. ArXiv e-prints, December 2017
16. Silver, D., Schrittwieser, J., Simonyan, K., et al.: Mastering the game of go without human knowledge. Nature **550**, 354–359 (2017)
17. Wang, Z., Schaul, T., Hessel, M., et al.: Dueling network architectures for deep reinforcement learning. In: Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML 2016, vol. 48, pp. 1995–2003. JMLR.org (2016)
18. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer, Heidelberg (2018). http://gameaibook.org