Reinforcement learning as a tool to solve Zhed games

1st Carlos Albuquerque
Computer Engineering Student
FEUP
Porto, Portugal
up201706735@fe.up.pt

2nd Maria Ferreira
Computer Engineering Student
FEUP
Porto, Portugal
up201704508@fe.up.pt

3rd Pedro Mendes
Computer Engineering Student
FEUP
Porto, Portugal
up201704219@fe.up.pt

***Abstract*—The advantages of stimulating games, classified as NP-complete, have been increasingly proven in our daily life. This paper tackles a game of this kind, ZHED, by presenting different strategies to solving it using different Reinforcement Learning algorithms, and analysing the efficiency of each approach. The algorithm's hyper-parameters were consecutively altered with the aim of achieving the fastest learning speeds possible for most levels.**

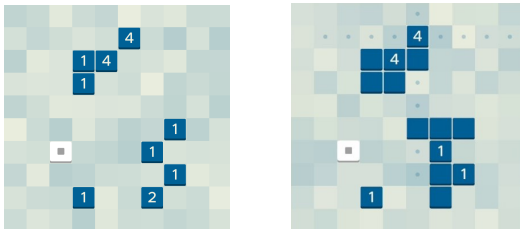
***Index Terms*—Reinforcement Learning, Games, Puzzles, NP-complete**

I. Introduction

Classic puzzles that are enjoyable to play have been acquiring considerable attention from the scientific community, specially the computer science people that work on find the solution of hard and little known stimulating single-player games. This paper tackles a NP-complete problem of that kind, ZHED. The countless benefits of solving puzzles are very well known and depend on the puzzles played, therefore the need to solve puzzles with different goals, strategies and lines of thought. The resolution of puzzles can improve the memory and mood, refining the problem-solving skills and visual-spatial reasoning. In addition, this practice among the elderly has even been found to assist in delaying dementia and Alzheimer's.

The resolution of ZHED puzzles can be very interesting and challenging to people who want to develop varied skill-sets. Nevertheless, fast and precise solutions are required for players that want to check their answers. This article presents different strategies to solve the ZHED puzzles, displaying the tendencies found in more effective and efficient approaches for the problem and dealing with the lagging task of checking all of its possible solutions.

This article has the following structure: firstly the problem under analysis is presented with a detailed description, being followed by section III, containing a description of all approaches used to model the problem as a reinforcement learning (RL) problem. Section IV describes the setup used in the experiments and shows the obtained results, along with an analysis of said results. Finally, section V presents the conclusions that can be drawn from this work, the limitations of the proposed solution, and aspects for improvement in future developments.



(a) Unsolved (b) Partially solved

Fig. 1: Zhed game

II. Description of the problem

ZHED is a fascinating puzzle with simple rules. It is played in a grid board with a variable size and variable number of numbered playable squares, but only one white square (goal square). The goal is to fill the only non playable white tile on the board by using the tiles with numbers on them. These playable numbered titles can generates squares in the four orthogonal directions: left, right, down and up, but never diagonally. The number represents exactly how many squares get filled in the selected direction, and can only be played once in one direction. If a tile (numbered or unnumbered) is already filed along the path, another tile is added in the chosen direction [1]. In sum, to solve each level, it is necessary to find the right sequence of moves for every board. By opposition to most games, is possible to solve some of the challenging levels in more than one way, which means the oneness of the solution isn't guaranteed.

Figure 1 shows an example of a Zhed game (left) and the partially corresponding solution (right).

At first glance, the Zhed puzzle seems to be simple enough, specially when the number of squares is low. However, upon further examination it is not as trivial as it might initially appear. The difficulty of Zhed games can be measured and depends on the number of squares and it's dispersion. Problems where there are more than one numbered square in line with the goal square can tease greater difficulty, as the last play is always a number square in front of the white square. Problems with more squares in the same row and column are usually easier, as they promote lower interaction between each other.

III. Approach

After the remarkable success of AlphaZero in Go, chess and other complex games, reinforcement learning approaches have garnered tremendous popularity in the field of self-play games [4] like Zhed, being the chosen approach for our study case. In this type of learning, the agent takes action and interacts with the environment, learning via this interaction and following feedback [2]. The game can be classified as an episodic task, seeing we have a starting point, when all squares haven't been played, and an ending point, when we can evaluate if the goal was reached or not, achieving a terminal state, independently of the success. The algorithms used were Q-learning, State-Action-Reward-State-Action(SARSA) and Proximial Policy Optimization (PPO). In all approaches the feedback to the agent is given via reward, which serves as the measure for the success or failure of an agent's actions on a given state. In the environment created, the agent is rewarded if the moves played resolve the puzzle and is penalized when the end of the game is reached, but the goal isn't. In games like ours where the success of the agent can only be measure in the end of the level, and depends only if the level was passed our not, the Monte Carlo learning method is optimal. In this method, the total cumulative reward is received by the agent only in the end of each episode.

A. Level by level learning

The policy, which is the strategy that the agent employs to determine the next move based on the current state, was initially random. This means that in the first environment the agent chooses a position and a square from the possible non-played ones. The achievement of the solution is reached with a high number of episodes, that increases proportionally with the number of playable squares, leading to a complete exploration of the environment. The agent is not forbidden to play a square that has been played before, it just gets a negative reward and the play is ignored, learning, by itself, that playing a previously played square should not be done and doesn't help to achieve the solution.

In this environment the agent has to learn level by level, which implies each time a new challenge is presented, the knowledge acquired during the resolution of the previous levels is lost, as it only learns by attaching feedback to the squares' specific positions on the grid. The game represents a deterministic and discrete environment, where the state is given by the position of the squares in the grid board. In a game with N playable squares there are $4N$ possible actions and state space with size $N! * 4^N$.

B. Continuous learning

In a second approach, a new environment was created where the agent learns, in each level, a general knowledge that allows him to solve the problem with less exploration of the new environment. In this approach the agent is initially trained with a high number of different levels and is able to resolve a new puzzle level with less prior concrete learning of said level taking into account all of its past experience, much like humans. As the reinforcement learning approach is the closer to the human learning process, we analyse the way we resolve the puzzles through this approach as classifying the squares into types of squares we can find in any level of the game. This way, a square isn't seen as it being in certain coordinates, but instead as being that particular type of square based on its general and relative location. A particular type of square tends to call for a certain type of action, for example, we know that the last square to be played and used to fill the goal will stem from the squares which belong to the type of "being aligned with the goal".

After a careful analysis of the game, we conclude that in the initial phase it is more gainful to play the more distant squares in the direction where their interaction with surrounding squares is largest. In this game, a square interacts with another if the squares it generates can align with the other square. For example, when there are two number squares, one inline with the goal square and with a number that can not resolve the puzzle and other that can generate squares in a way the square inline of the goal square resolves the puzzle, the not inline square interacts with the other number square. In an intermediate phase, the square(s) adjacent with the square in line with the goal square are the ones we are interested in playing. Finally, the last square play is always the square that can reach the goal square with its generated squares, so it is in a position lined up with the white goal square. Therefore, we conclude that the squares can be comparable between each other if analysed based on ten criteria:

1) Is the square in line with the goal square?
2) Is it adjacent with the square in line with the goal square?
3) Is the square on the edge of the board?
4) Is the square the more distant one to the goal square?
5) Is the square the closer one to the goal square?
6) Is the square the one with highest value?
7) Is the square the one with lowest value?
8) The direction chose creates squares in the goal square direction?
9) Does the direction of the play interacts with the greater number of other squares?
10) Does the direction of the play interacts with the lower number of other squares?

As each item is a yes or no question, a binary value is assigned to a list of flags with the response, that allows us to evaluate the squares. The list of flags that represent the answer to the questions above is the one used to determinate the choice of the square and direction to play. In regard to the second criteria, the evaluation takes into account if the position of the square is between the goal square and the square inline with the goal square in the dimension they are not seen as inline. The action taken by the agent is always based on the progress of the game. The distance is measured based on the Manhattan distance, that measures distance as the sum of absolute differences of positions, therefore the distance (a, b), (x, y) is calculated by $abs(a - x) + abs(by)$ [5]. This

criteria is appropriate because the board is a square grid and the positions can be expressed as integer values.

The observation space, which is a list of all possible states according to level completion, can take five values, lower or equal to 20%, 40%, 60%, 80% and 100%. This value is the ratio of the number of squares played so far out of the number of initially number squares, measuring the state of the game for the current level.

At first glance, it can appear that the size of the action space is bigger than in the first approach, however when the number of squares is five or more (which represents the vast majority of levels), the action space becomes smaller than the one in the initial approach. The new environment has a action space of size 2^{10}, because the number of criteria is ten and each one is a binary value.

Although available time has cut short further efforts in developing this approach and led us to fail to achieve any concrete results, we found it to be an enlightening approach and interesting food for thought.

IV. Experimental evaluation

Studies have shown that different hyper-parameter settings can lead to very different results [3]. Therefore, the run time of the Q-learning algorithm was measured varying the various hyper-parameters, which were the number of episodes of the learning phase and the maximum number of steps and represent the maximum number of action that the agent can perform. The value of gamma, that represents the discount rate is always a value close to one, but never equal because it will give a lower discount to your agent that aims to the long term reward and not any immediate reward. Another hyper-parameter varied was the learning rate and the exploration/exploitation ratio. The maximum value of epsilon, which represents the exploration/exploitation factor, was 1.0 in all tests

For the development of the game environment the OpenAI Gym, a python library, was chosen. A new environment was created where the levels were extracted from the files, where the different positions of the squares are presented. All tests were performed using python version 3.7 in a computer with an i7-5820K processor with 16GB of RAM and a 84GB SSD disk.

A. *Effects of the number of episodes and steps*

The rewards given to the agent were evaluated based on the graphics of his performance. After a detailed analysis, a positive reward was attributed to valid plays and a penalisation was assigned when the agent tries, without success, to play an already played square. These additional rewards enable us to lower the number of steps, as the exact number of step used to play would be quickly closer to the number of number squares.

The Figure 2 shows the rewards obtained in each level during the training process of the agent to resolve the fifth level, using 50 000 episode and a maximum number of steps of 1000. The value of the hyper-parameters was 0.01 for the learning rate, 0.99 for gamma, a decay rate of 0.001 and

Fig. 2: Rewards obtained in episode 5 during the training

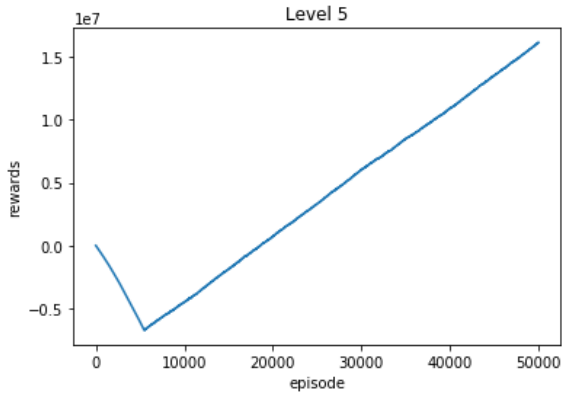
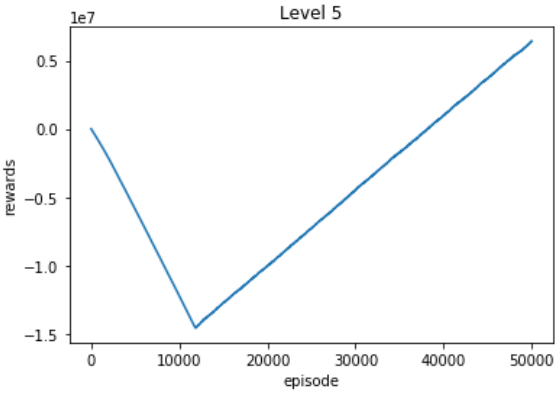


Fig. 3: Rewards obtained in episode 5 during the training with 50 000 episodes and 100 steps



a minimum epsilon of 0.1. It can be observe that initially, due to the unfamiliarity of the agent in the environment, the rewards obtained were negative. Nevertheless, after a considerable number of episodes, the agent stops exploring and starts exploiting, meaning the agent already knows the correct answer for the puzzle and so performs the best combination that brings about the bigger reward all the time, and doesn't try to find another. The agent starts obtaining better rewards around the episode 6700 and, after that, the results were always positive meaning two things, the agents start to exploit more and is already able to solve the level.

An alternative test was done with the same 50 000 episodes but only 100 steps at most, shown in the Figure 3. In this test the agent was able to solve the episode in final test, where it just exploits. Nevertheless, the results obtained were worse than the ones obtained with a bigger step. Thus, there must be a compromise between the number of episodes and the number of steps, whereas when the number of steps is lower, the learning needs longer to be complete.

The number of episodes needed to resolve each puzzle level is dependent on the difficulty of the level. In harder games, the agent was to explore more iterations in order to reach a valid solution. As the number of different actions is also dependent

Fig. 4: Variation of hyper parameters in level 1

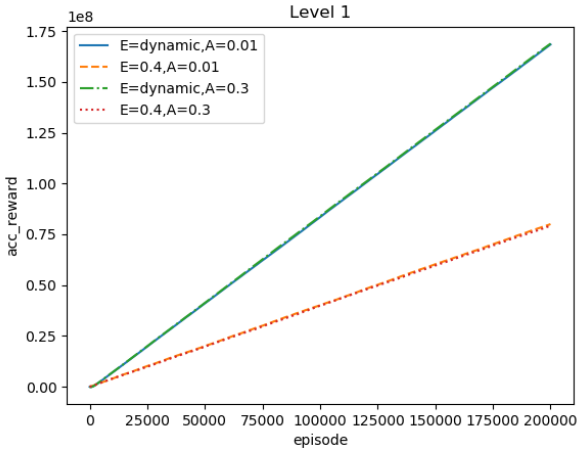


Fig. 5: Variation of hyper parameters in level 5

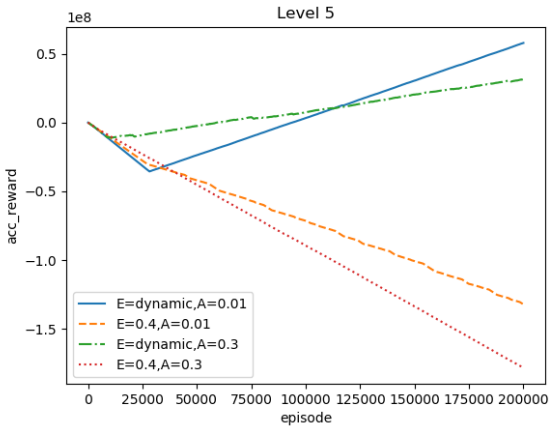


Fig. 6: Variation of hyper parameters in level 6

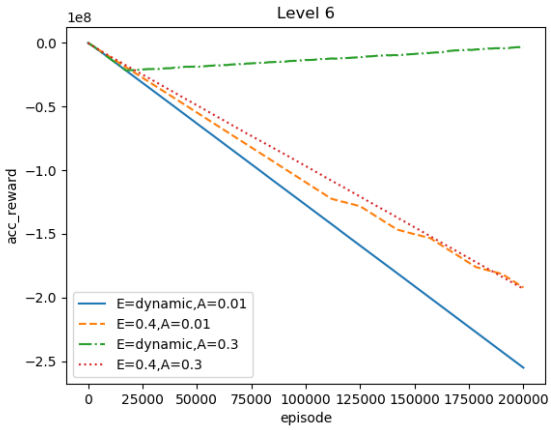
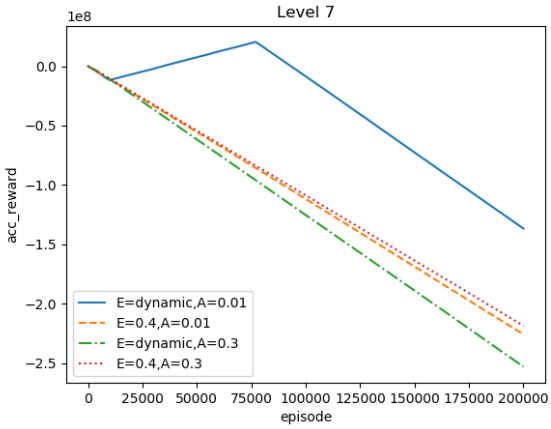


Fig. 7: Variation of hyper parameters in level 7



on the number of numbered squares, the maximum number of steps must increase proportionally to the number of squares.

B. Effects of varying the Hyper-parameters

Figure 4 shows the results of changing the value of epsilon to a fixed value instead of a dynamic one, while also changing the value of alpha, the learning rate. The results suggest that a dynamic value of epsilon produces better results in training in easy levels like the first one. This is easily explained by the fact that the agent will find the only right play with a small number of episodes and after the use of exploration is no longer found to be very useful, as there is other way of achieving the solution. The changing of the learning rate was not significant in the training of this level.

In opposition to the graphic 4, in another level like the fifth, that contains four numbered squares, the value of alpha was significant to the cumulative reward in the training phase. This can be observed in figure 5. It is still clear that a dynamic value of epsilon produces better cumulative rewards than a fixed one. However, it is an impediment to finding all possible moves when levels start to get harder so the uniqueness is not guaranteed. With a high number of episodes, the learning rate of 0.01 starts to produce better results than that with a value of 0.3, with a dynamic epsilon.

The level 6 contains has also four squares but that are also more interactive than the ones in the previous level. In this level, interestingly enough, the combination of a dynamic value of epsilon and a alpha 0.01 was the one to obtain less rewards. This is completely different compared to the previous level, where this combination produced the best results.

Finally, we analyse the cumulative rewards in level 7, where the training phase produces a negative cumulative reward in all tests. The combination of hyper-parameters that result in better accumulative reward was a dynamic epsilon with a learning of 0.01. On one hand, the value of the learning rate was not significant when the epsilon had always value 0.4. On the other, the effect of the alpha was crucial when the epsilon was dynamic.

C. PPO results

In order to evaluate the performance of the PPO algorithm, the value of the clipping range was inputted to vary between

Fig. 8: Variation of hyper parameters in level 2 with PPO algorithm

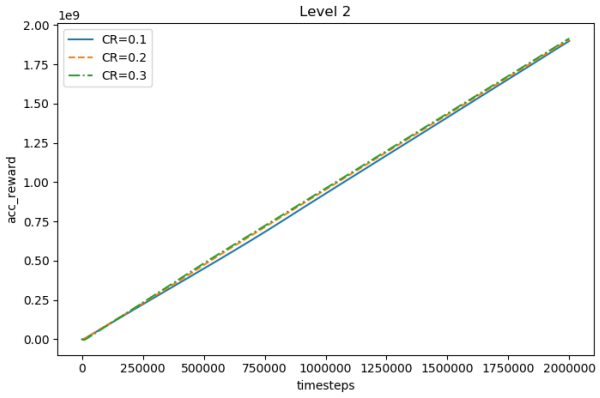
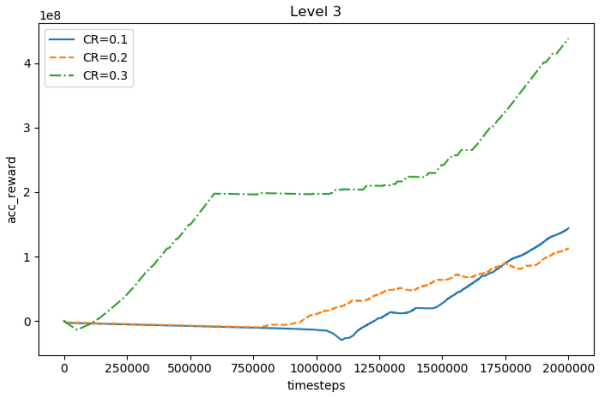


Fig. 9: Variation of hyper parameters in level 3 with PPO algorithm



0.1 and 0.3.

Figure 8 shows the results of varying the Hyper-parameters using the PPO algorithm. The results obtained for the 1st level were analogous to the ones observed in 2nd level. The cumulative reward obtained was almost the same for the three approaches. The value that produced the worst results was the highest, of 0.3.

During the resolution of the 3rd level, it was clear that the value that produced the better performance was that of 0.3. Given the cumulative reward increase with the number of episodes, the agent learnt faster the correct solution of the puzzle.

The results obtained in the 4th level were close to the ones presented in level 5 and are presented in the figure 10. The most significant difference was in the performance of the algorithm with the CR equal to 0.2, significantly lower to than in level 3.

Finally, the algorithm starts obtaining a negative cumulative reward in the 5th level (see Fig. 11). In this level, for the first time, the cumulative reward obtained was increasingly worse as the number of episodes increased.

Fig. 10: Variation of hyper parameters in level 4 with PPO algorithm

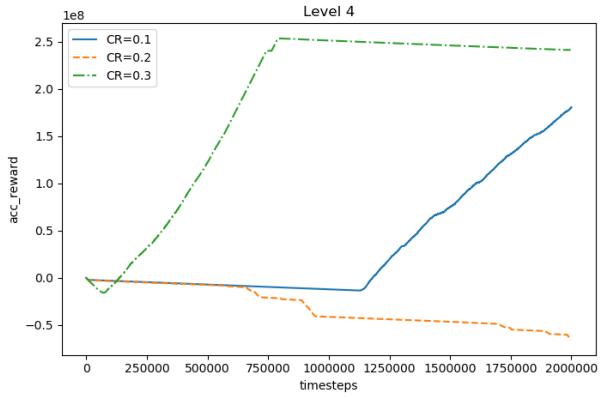
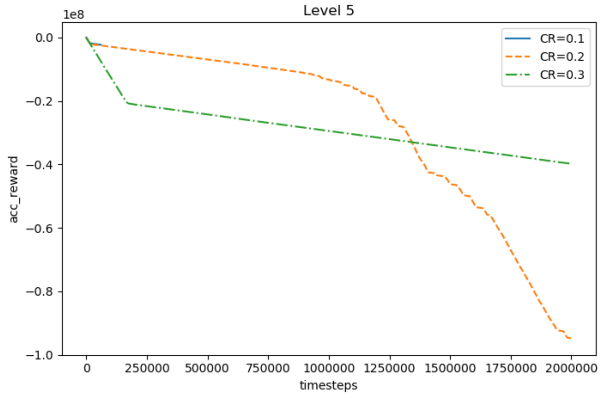


Fig. 11: Variation of hyper parameters in level 5 with PPO algorithm



V. Conclusions

In sum, we were able to solve Zhed puzzles until the level 7, after which point the memory started to limit any further progress. In a computer with more RAM the better combination of hyper-parameters would be a dynamic value of epsilon and a learning rate of 0.01. The algorithm PPO produced worse results than the Q-learning so to achieve the idyllic solution it would be interesting to keep changing the hyper-parameters in order to find the most optimal combination. In addition, we note that reinforcement learning is an interesting type of learning, however, presenting several limitations. In a future work, we would be more geared towards comparing our results to other types of approaches belonging to different fields of the Intelligence Artificial, seeking to achieve a solution that resolves mores levels in a more efficient way.

Acknowledgement

We thank professor Henrique Cardoso for the availability, attention and help provided during the realization of this paper.

References

[1] Gaming Boulevard page, https://www.gamingboulevard.com/2020/05/indie-corner-zhed/. Last accessed 20 of May of 2020

[2] Free code camp page, https://www.freecodecamp.org/news/an-introduction-to-reinforcement-learning-4339519de419/. Last accessed 24 of May of 2020
[3] Wang, Hanrou, Michael Emmerich, Mike Preuss and Aske Plaat. “Analysis of Hyper-Parameters for Small Games: Iterations or Epochs in Self-Play?” ArXiv abs/2003.05988 (2020): n. pag.
[4] Lee, Jay H.; Shin, Joohyun; Realff, Matthew J., Machine learning: Overview of the recent progresses and implications for the process systems engineering field
[5] Fred E. Szabo, The Linear Algebra Survival Guide (2015), Pages 219-233