Reinforcement Learning for Bubble Crush

Huajun Wu, Yu Zhu
Viterbi school of Engineering, Department of Computer Science
huajunwu@usc.edu, zhuyu@usc.edu

Abstract—Match-3 game has simple game-play but it is hard for AI to learning specific strategies. Here, we present our self designed match-3 game bubble crush. The game mimics the situation that players make moves to earn scores to win different levels on the Unity and ML-agent platform. Traditional studies in Match 3 games have attempted to utilize Monte Carlo tree search (MCTS) and convolutional neural networks (CNNs), but this articles demonstrated how Reinforcement Learning (RL) method based on Proximal Policy Optimization(PPO) outperforms random policy AI and provides metrics for evaluation of agents and corresponding baselines in different scenarios.

I. INTRODUCTION

One of the most challenging issues in game development is to design "levels" to grab users and keep their interest alive. To create game levels, testing is needed to measure difficulty in different level. It is a difficult task in terms of human testing. [1] First, it is time-consuming for game designers to test multiple times for one level. Second, the limitation of human operation speed will slow down the process of game development. To reduce the need of human resources and improve testing efficiency, machine learning is employed in game testing. [6]

In this paper, the focus is on a match-3 game, called "Bubble Crush". A classic match3 game is look like figure 1 In "Bubble Crush", there is a grid with bubbles of various colors. Matching three or more bubbles in a column or a row will cause these matched bubbles to disappear. The disappearance of different colored bubbles will get different scores. There are also some special bubbles like bomb and lightning. Matching these special items will trigger a wider range of bubbles disappearing. To pass the game, players need to obtain a certain score by using a predetermined number of moves. The player's score on the level is the cumulative score of the individual scores for each moves the player makes. [2]

Reinforcement learning is one of the most suitable machine learning paradigms that can be adopted to testing difficulty levels. In this study, reinforcement learning is applied in testing "Bubble Crush". The learning process involves an environment with an agent. The agent can interact with the environment. The agent takes the current state of the game board and chose the possible moves. A reward or punishment are given according to the action. The agent tries to avoid punishment and obtained reward by changing its weights to get a good structure of the task. Finally, the agent predicts the best action to get the highest reward and achieve the goal. The difficulty of a level is measured by the score obtained in the predefined moves.

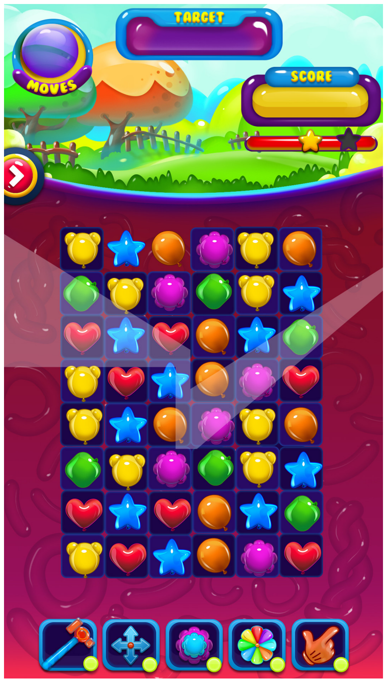


Fig. 1. A classic match3 game

Therefore, this article defines strategies that can be applied in Bubble Crush games. Reinforcement learning is used to establish a system that can automatically test Bubble Crush based on these strategies. The score obtained by the self-play agent can help determine the difficulty of the level. The rest of this paper is organized as follows. In Section 2, we present an overview of the related works. In Section 3, we propose our methods. In Section 4, we explain the experimental results. Finally, we conclude our work and discuss future avenues of research in Section 5.

II. Related Work

Level design is one of the most important problems in the game development because the difficulty of a level is usually tested by humans many times for one level to ensure the difficulty is appropriate for players.It is a heavy and time-consuming process in terms of human testing. To improve the efficiency of play testing, automatic testing method has been investigated by researchers for a long time.

Algorithms such as finite-state machines (FSM), convolutional neural networks (CNN), and Monte Carlo tree search (MCTS) are widely used in calculating the difficulty of different game levels.

Convolutional neural networks (CNN) has become an important tool to solve many computer vision tasks of today. The technique is though costly, and training a network from scratch requires both a large dataset and adequate hardware. A solution to these shortcomings is to instead use a pre-trained network, an approach called transfer learning. Several studies have shown promising results applying transfer learning, but the technique requires further studies. This thesis explores the capabilities of transfer learning when applied to the task of filtering out offensive cartoon drawings in the game of Battlefield 1. GoogLeNet was pre-trained on ImageNet, and then the last layers were fine-tuned towards the target task and domain. The model achieved an accuracy of 96.71% when evaluated on the binary classification task of predicting non-offensive or swastika/penis content in Battlefield ”emblems”. The results indicate that a CNN trained on ImageNet is applicable, even when the target domain is very different from the pre-trained networks domain. The work of [8] and [9] introduced CNN-based method to achieve the prediction accuracy of 44.4% and 42% respectively on a Go data set.

Monte Carlo Tree Search (MCTS) is a tree search algorithm that has had an important impact in Game AI since it was introduced in 2006 by several researchers. MCTS estimates the average value of rewards by iteratively sampling actions in the environment, building an asymmetric tree that leans towards the most promising portions of the search space. Each node in the tree holds certain statistics about how often a move is played from that state (N(s, a)), how many times that node is reached (N (s)) and the average reward (Q(s, a)) obtained after applying a move a in the states. On each iteration, or play-out, actions are simulated from the root until either the end of the game or a maximum simulation depth is reached. One of the main advantages of MCTS is that it is considered to be an anytime algorithm. This means that the algorithm may stop at any number of iterations and provide a reasonable, valid, next action to take. This makes MCTS a particularly good choice for real-time games, where the time budget to decide the next move is severely limited. A variant of MCTS is used to build synthetic game testers.

Although these methods have a good performance in playtesting, there are several limitations. FSM and MCTS methods are developed according to a set of specific game rules. [6] When the rules of the game change, the model needs to be modified accordingly. It is time-consuming because existing learning is cannot be reused and new learning is required every time the rules changes. CNN needs a large amount of gameplay data. However, due to the funding constraint,it is often difficult to collect large amounts of data.

In order to overcome these limitations, researchers have proposed reinforcement learning to automatically testing difficulty level in games. Unlike other methods that require large amounts of data or need to be modified when the game rules changes, reinforcement learning shows good performance and adaptability without being limited by the amount of training data.

The learning process is in an environment with an agent which is part of the learning process and the agent makes an action in the environment to finish a goal. After that, there is a reward or punishment system according to the action or depending on the algorithm used. the agent tries to get more rewards and avoids being punished by changing its weights to get a great construction of the task. At the end, the agent predicts the best action to get the highest reward and achieve the objective. Reinforcement learning algorithms are goal-oriented algorithms. They have a goal to achieve or a value to maximize at the end of an episode. Those algorithms aim to teach a task to an agent by penalizing them if the agent makes a wrong choice, and giving reward if its decision is a good one. These algorithms surpass humans and also champions in games. The success of reinforcement algorithms increased rapidly and they are now able to play Atari games and also some 3D games that compete with humans. Reinforcement learning has been proven to have a good ability to surpass humans in various games, such as in card games, sports games, chess, racing games and arcade games, etc. In work of [10], deep reinforcement learning approach is employed to learn approximate Nash equilibrium of imperfect-information games form self-play. In work of [11], the authors used deep reinforcement learning method to generate fighting game agent. It demonstrated that the agent outplay other AIs in the game with 94.4% win rate.

III. BUBBLE CRUSH

Our bubble crush game is a classic match3 game which includes these key concepts: level, state, action space, trajectory, reward, and objectives.

But we will start from the description of a level, which is specific to the environment.

A. Level

The main element of the game is the level, we defined our own board template. The template is a map which allows determining height and width of the board and coordinates of each bubble. We can use the Level Editor to edit each level such as the board size, the frequency that each bubble occurs, the special type of bubbles(bomb, lighting bubble). After filling the template with different bubbles and determined the type of bubbles we may encounter, we get a final board. Obviously, the bigger

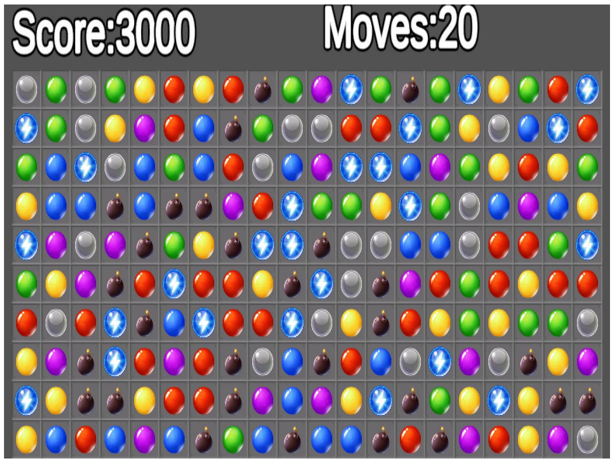


Fig. 2. Bubble crush

size of the board and more types of the bubble will make our game more difficult and complicated so that our agent will require more samples to learn.

B. State

The current state of the game in bubble crush is a game board, which consists of movable bubbles. The size of the board is adjustable and the number of bubbles can be defined using Level Editor. So we can easily manage the frequency of each bubble and also the difficulty of the game for our agent.

C. Action space

The core mechanic of the bubble crush is to swap two neighboring bubbles on the board, and only adjacent bubbles are allowed to be swapped(not diagonal). Bubble crush environments can be classified as model-based if the agent knows valid moves which lead to deleting bubbles. In this paper, we only consider the model-based agent because the agent will only focus on learning how to choose between different bubbles rather than try to figure out what is a valid move.

D. Trajectory

A trajectory is a sequence of the game board states and moves of bubbles on the board. [2] The environment in bubble crush game is stochastic because each next state generated randomly from previous state and deleted bubbles. When there are no valid moves in the current situation, the game will automatically end.

E. Reward

For the current state, the action just is taken and the next state of the board reward is the number of deleted bubbles. In the blue bubble mode, destroying a blue bubble will get 100 rewards, while destroying other bubbles will only get 1 reward. The game mechanics do not allow other actions so every action of the agent should result in a reward. For model-free settings, it's more reasonable to allow the agent to try to make the wrong move and figure out how the valid move looks like. But in this paper, we generated a valid moves list and only allowed agents to choose moves from the list. In special bubble mode, some bubbles are marked with a white circle. The goal is to destroy those special bubbles. Destroying the special bubbles will get 1 reward, and 0 for other bubbles.

F. Objectives

In the blue bubble mode of our bubble crush game, our objective is to collect N elements of particular bubbles (in this case, the blue bubble) or collect Z bubbles in 30 seconds (N steps). The current implementation of the environment is the sum of reward collected after 20 steps. Each level in bubble crush is associated with an objective. Players win a level if they reach its objective within the level specific move limits. Here we set the objective to infinite and move limits to 20, so our goal is to collect as many blue bubbles as possible within 20 moves.

G. ML Agents

Unity Machine Learning Agents (ML-Agents) is an open source which allows us to train intelligent agents in game environments and simulation environments. Mlagents-learn is the main training utility provided by the ML-Agents Toolkit. It accepts a number of CLI options in addition to a YAML configuration file that contains all the configurations and hyperparameters to be used during training. Here we use the default configuration file and try different parameters in order to obtain the best results and PPO [12] and SAC algorithm is used to trained agents to interactive with the environment in bubble crush.

H. Level Editor

Default levels: In the blue bubble mode, we will test the obtained agents on the whole set of default levels. The table size in all levels is 10x20.

IV. PROPOSED APPROACH

A. Game state

Bubble crush game board is similar to other board games like chess, Go and Tic-tac-toe. Each game has N shapes on the board, so we created different type of tensor with its own dimensions based on the shape on the board. All levels are resized to the maximum board size if each level has its own size.

B. RL Architecture

We used the PPO and SAC algorithms in ML-Agent to train our agent. The PPO algorithm is called Proximal Policy Optimization, it is a new type of Policy Gradient algorithm for reinforcement learning. The motivation of PPO is to ensure data efficiency and reliable performance of TRPO, while using only first-order optimization. PPO proposes a new objective function that can be updated in multiple training steps in small batches, which solves the

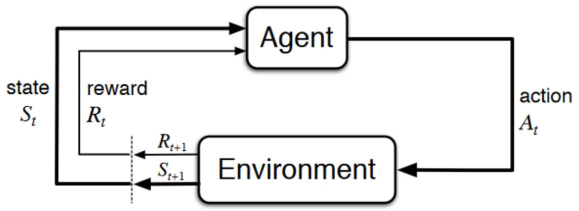


Fig. 3. rl model

problem that the step size is difficult to determine in the Policy Gradient algorithm and it is now the most applied benchmark algorithm used by OpenAI.

1) PPO: In the process of the development of reinforcement learning, PPO can be regarded as one of the most classic algorithms, which is mainly due to its 3 very obvious advantages: simple and easy to understand, adaptable, and excellent in efficiency.

PPO is an on-policy algorithm, and it performs relatively poorly in sampling efficiency because they directly optimize the goal. The on-policy algorithms trades sampling efficiency for reliability.

There are two versions of the ppo algorithm in OPENAI's baseline. According to their comments, PPO2 should be the official version. In fact, the difference between the two is not big, which is mainly reflected in the optimization of the algorithm. Here, we only discuss PPO-Clip (the main form used by OpenAI).

PPO-Clip: There is no KL divergence term in the target, and no constraints at all. Instead, it relies on tailoring the objective function to reduce the difference between the old and new strategies.

$$\theta_{k+1} = \arg\max_{\theta} \underset{s,a\sim\pi_{\theta_k}}{\mathrm{E}} \left[L(s,a,\theta_k,\theta)\right], \tag{1}$$

We can use SGD to maximize the objective. Here L is given by

$$\begin{aligned} L(s,a,\theta_k,\theta) = \min\Bigg(&\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s,a) \\ &, \operatorname{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}, 1-\epsilon, 1+\epsilon\right) A^{\pi_{\theta_k}}(s,a), \end{aligned} \tag{2}$$

PPO will explore through sampling operations based on the latest version of its random strategy. The randomness of action selection depends on initial conditions and training procedures. During the training process, the strategy usually becomes less and less random because the updated rules encourage the strategy to take advantage of the rewards that have been discovered.

2) SAC: Before SAC was proposed, deep reinforcement learning (DRL) algorithms had achieved significant results in continuous action space control tasks, but each had some shortcomings. The following first introduces the classic achievements of DeepMind and OpenAI in the field of continuous control.

The SAC algorithm is an Off-policy algorithm. The sample efficiency has been improved comparing to the on-policy algorithm such as PPO. SAC is a random policy algorithm when compared with DDPG and its variant D4PG.

The SAC algorithm is constructed under the framework of Maximum Entropy Reinforcement Learning. The purpose is to make the strategy randomize. The advantage is that it is very friendly to robot control problems and can even be used in a real environment.

The maximum entropy of the strategy also means that the exploration of the strategy space and the trajectory space is more sufficient than the deterministic algorithm. For the state with more than one optimal action, the SAC can output the probability distribution of an action instead of a certain action. .

In summary, SAC has three features:

- The learned policy can be used as an initialization for more complex and specific tasks.
- It is obvious that SAC has stronger exploration capabilities, and it is easier to find better modes under multi-modal reward . For example, the robot is not only required to walk, but also to save energy.
- SAC is designed to be more robust and can explore various optimal possibilities in different ways, it is easier to make adjustments in the face of interference.

The Q-functions are learned by MSBE minimization, using a target value network to form the Bellman backups. They both use the same target, like in TD3, and have loss functions:

$$\begin{aligned} L(\phi_i,\mathcal{D}) = \underset{(s,a,r,s',d)\sim\mathcal{D}}{\mathrm{E}} \Bigg[\Bigg(&Q_{\phi_i}(s,a) \\ &- \left(r+\gamma(1-d)V_{\psi_{\text{targ}}}(s')\right)\Bigg)^2\Bigg]. \end{aligned} \tag{3}$$

The RL model is shown as figure 3

Here, we give the specific situations of environment, agent,reward, state, and action.

- Environment: The environment contains the 2-dimension board filled with bubbles. Agent should be able to make valid move and destroy bubbles and earn high scores.
- State: Each agent has 4 states in the bubble crush: 2D coordinate position(x, y), the current bubble it

chooses, the current score it earns and the remaining moves it has.

- Action: Each agent has 4 actions that could be operated: Choose a bubble and then move up, down, left or right.
- Reward: Here, we set the total scores S as the reward. When S is higher than the required scores to pass the current level, it will be given huge rewards. When making a move, it will be given a -50 penalty. Whenever a bubble is destroyed, the agent will be give reward based on the bubble type.

C. Observation Type

The match3 sensor in ML Agent supports three observation type, the vector, uncompressed visual and compressed visual. The visual one will include a convolution with 5x5 filter and ReLU activation. This type of convolution is designed for 2-dimension board games, which is very good for Match-3 games. Note that using the match3 CNN with very large visual input might result in a huge observation encoding and thus potentially slow down training or cause memory issues. So, the compressed visual observation type is a better choice to save training time.

V. EXPERIMENTS

To investigate the performance of our agent, this study measured the performances of the agents in our special designed game levels. The levels serviced by bubble crush were evaluated by comparing the average score earned by random policy AI and the RL Agent to complete the level within 20 moves. The default game board is 10x20 for all training and testing levels.

The training curve is shown as figure 4. The only difference between the blue and orange curve is the observation type used during training. The blue curve uses the vector type while the orange curve uses the compressed visual type in match 3 sensor. When using a compressed visual type, the agent will learn faster and converge faster than the vector type. In the 25k episode, the orange agent is able to achieve nearly 4k reward while the blue agent can only get 3k. This will help us save a lot of training time by using the compressed visual setting.

Based on the evaluation, we confirmed how our agent outperformed the random policy AI.

A. Blue bubble mode

TABLE I
Average score in blue bubble mode

Level	Random AI	PPO
Level 1	8	25
Level 2	15	39
Level 3	10	28
Level 4	18	43

An agent trained by PPO algorithm outperforms random policy twice in average score shown in table I. In test level 1, 2, 3, 4, our PPO algorithm will choose three blue bubbles to destroy in 2 moves, while the random policy AI will require 5 moves or more to choose a blue bubble. This result shows how our agent learns the importance of blue bubbles which will give the agent much higher rewards.

In the blue bubble mode, we can clearly see that the PPO and SAC outperforms random AI three times in average score and PPO performs slightly better than SAC. The larger the map size, the better score that PPO and SAC can get.

B. Special bubble mode

In special bubble mode, some bubbles are marked with a white circle. The goal is to destroy as many special bubbles as you can. Note that the MLAgent allows AI to clearly identify which bubbles are special, and all special bubbles are randomly generated.

In the special bubble mode, figure 7 and figure 8 show the PPO and SAC outperforms random AI twice in average score and PPO also performs slightly better than SAC. Although SAC requires continuous action space, an alternate version of SAC, which slightly changes the policy update rule, can be implemented to handle discrete action spaces.

C. Sample Efficiency

Sample Efficiency is very important in deep learning algorithms and learning skills in the match3 game can take a substantial amount of time. The total time required to learn a new game skill quickly adds up when trying different game scenarios. SAC is an off-policy algorithm so we can reuse data collected for another task. In a typical scenario, we need to adjust parameters and shape the reward function when prototyping a new task, and use of an off-policy algorithm allows reusing the already collected data.

The figure 9 and figure 10 shows that the SAC requires 50% less training samples to achieve the same performance with PPO in both blue bubble and special bubble mode. Also, since the SAC uses fewer samples, the training time is also better than PPO. However, PPO can usually get better results when the reward function is carefully designed.

VI. CONCLUSION AND FUTURE WORK

In a word, this article presented a specially designed Match-3 game using Unity ML agent to study deep reinforcement learning similar to the one played by millions of people every day. The article described the bubble crush game in terms of reinforcement learning and carried out experiments for different test levels. The experiments have shown that our reinforcement learning agent achieved nearly 2 times better results than our simple random policy AI. Currently, when added bomb and lighting bubbles, both PPO and SAC do not know how to take advantage of those bubbles, and the results are nearly

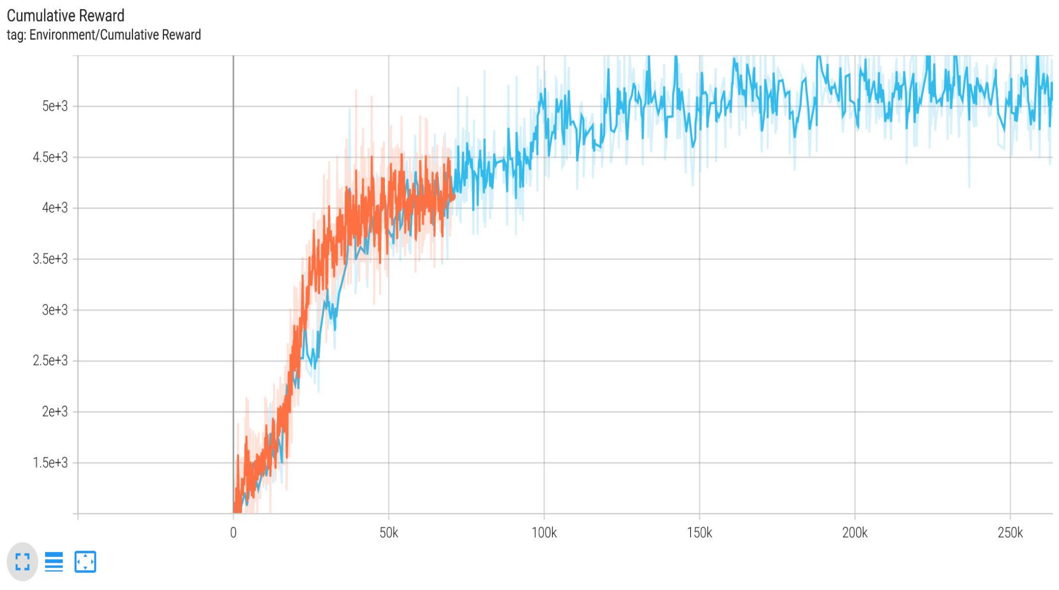


Fig. 4. training results for blue bubble mode

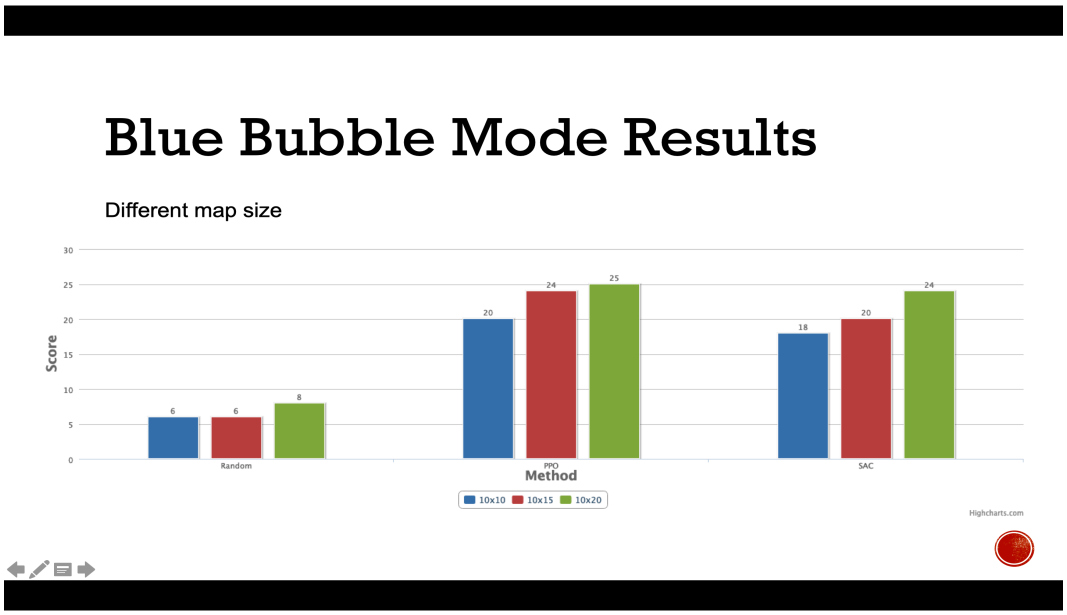


Fig. 5. Training results for blue bubble mode with different map size

the same with random policy AI. Future improvements of the bubble crush environment are creating different sizes of levels, adjusting reward functions to teach AI to use bomb and lighting bubbles, and providing hints for human players.

References

[1] E. R. Poromaa, Crushing Candy Crush: Predicting Human Success Rate in a Mobile Game using Monte-Carlo Tree Search', Dissertation, 2017.

[2] I. Kamaldinov and I. Makarov, "Deep Reinforcement Learning in Match-3 Game," 2019 IEEE Conference on Games (CoG), 2019, pp. 1-4, doi: 10.1109/CIG.2019.8848003.

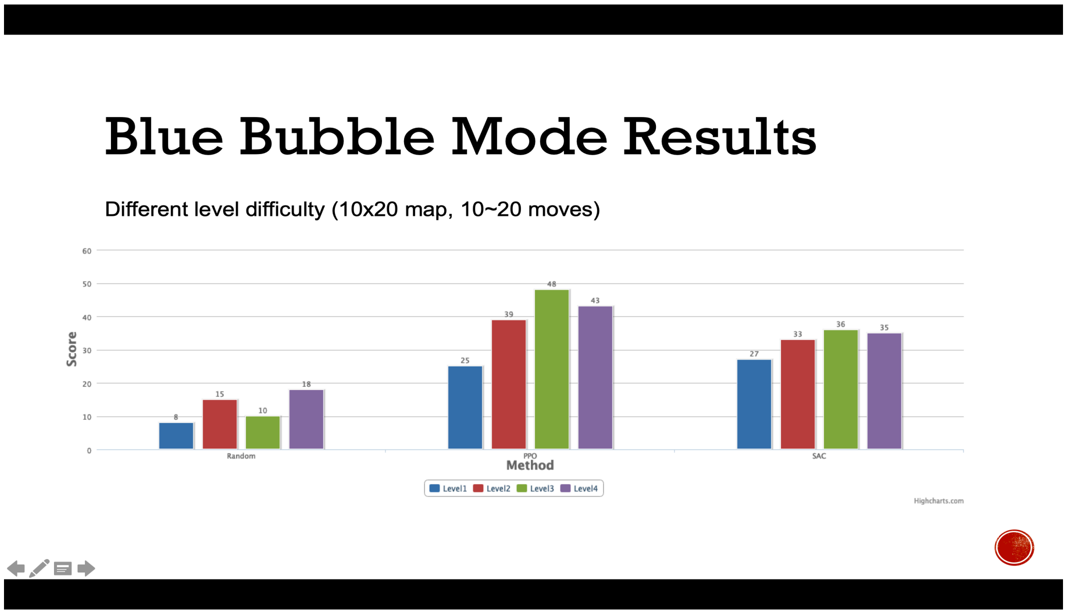
[3] Lorenzo, F. V., Asadi, S., Karnsund, A., Wang, T., Payberah, A. H. Generalized Reinforcement Learning for Gameplay.

[4] Shin, Y., Kim, J., Jin, K., Kim, Y. (2020). Playtesting in Match 3 Game Using Strategic Plays via Reinforcement Learning. IEEE Access, PP(99), 1–1. doi: 10.1109/ACCESS.2020.2980380

[5] Kristensen, J. T., and Burelli, P. (2020). Strategies for Using Proximal Policy Optimization in Mobile Puzzle Games. arXiv, 2007.01542. Retrieved from https://arxiv.org/abs/2007.01542v1

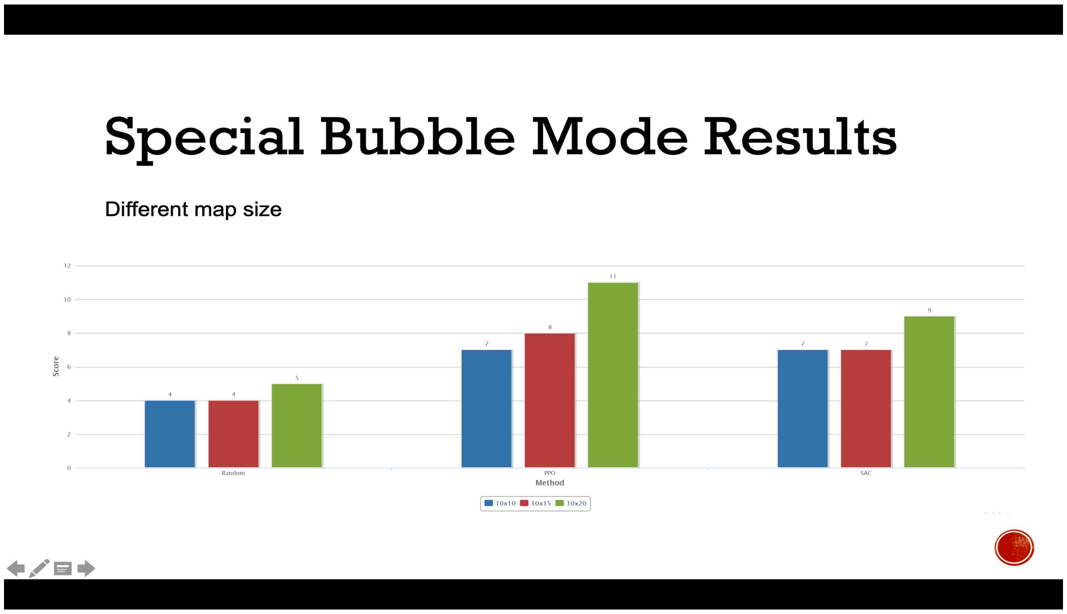
[6] Napolitano, N. (2020). Testing match-3 video games with Deep Reinforcement Learning. arXiv, 2007.01137. Retrieved from https://arxiv.org/abs/2007.01137v2

[7] Holmgård, C., Green, M. C., Liapis, A., and Togelius, J. (2018). Automated Playtesting with Procedural Personas through MCTS with Evolved Heuristics. arXiv, 1802.06881. Retrieved


Blue Bubble Mode Results

Different level difficulty (10x20 map, 10~20 moves)

Fig. 6. Training results for blue bubble mode with different level difficulty


Special Bubble Mode Results

Different map size

Fig. 7. Training results for special bubble mode with different map size

from https://arxiv.org/abs/1802.06881v1

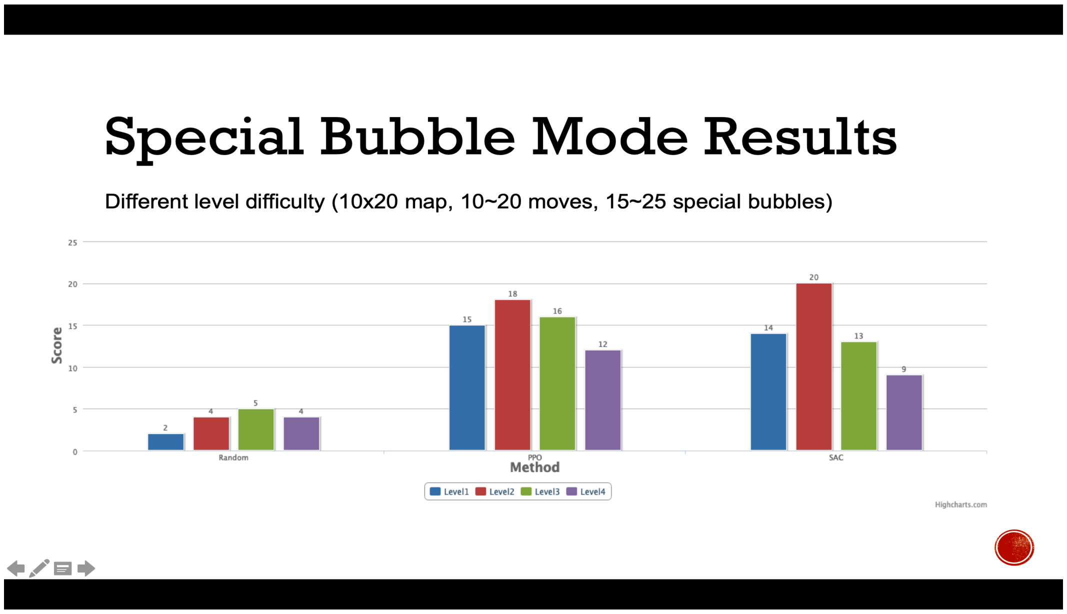
[8] Clark, C., and Storkey, A. (2014). Teaching Deep Convolutional Neural Networks to Play Go. arXiv, 1412.3409. Retrieved from https://arxiv.org/abs/1412.3409v2

[9] K. Shao, D. Zhao, Z. Tang and Y. Zhu, "Move prediction in Gomoku using deep learning," 2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC), 2016, pp. 292-297, doi: 10.1109/YAC.2016.7804906.

[10] Heinrich, J., and Silver, D. (2016). Deep Reinforcement Learning from Self-Play in Imperfect-Information Games. arXiv, 1603.01121. Retrieved from https://arxiv.org/abs/1603.01121v2

[11] D. -W. Kim, S. Park and S. -i. Yang, "Mastering Fighting Game Using Deep Reinforcement Learning With Self-play," 2020 IEEE Conference on Games (CoG), 2020, pp. 576-583, doi: 10.1109/CoG47356.2020.9231639.

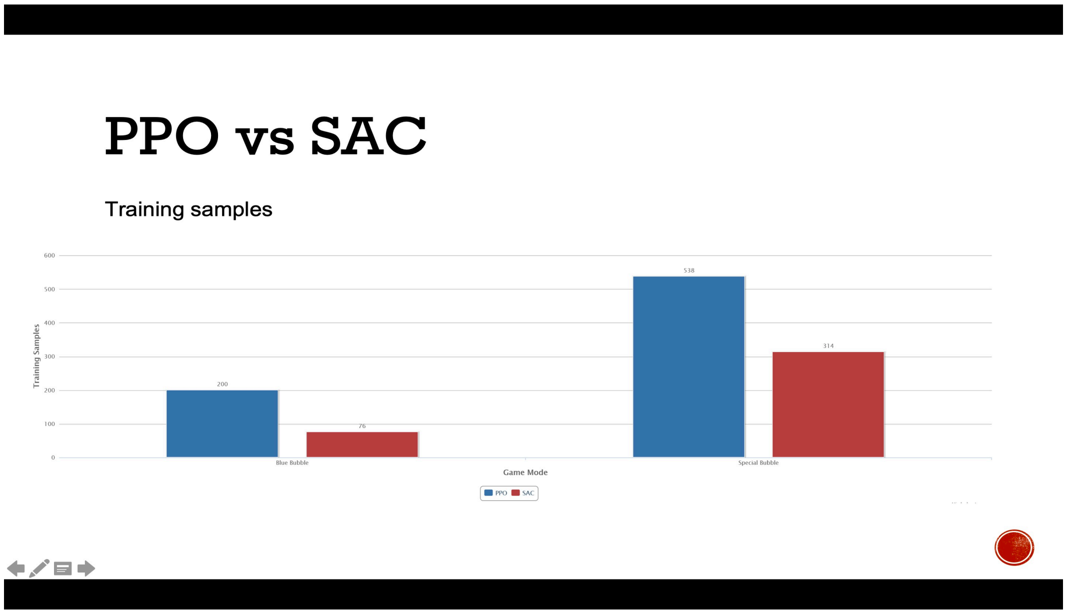
[12] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal Policy Optimization Algorithms. arXiv, 1707.06347. Retrieved from https://arxiv.org/abs/1707.06347v2



Special Bubble Mode Results

Different level difficulty (10x20 map, 10~20 moves, 15~25 special bubbles)

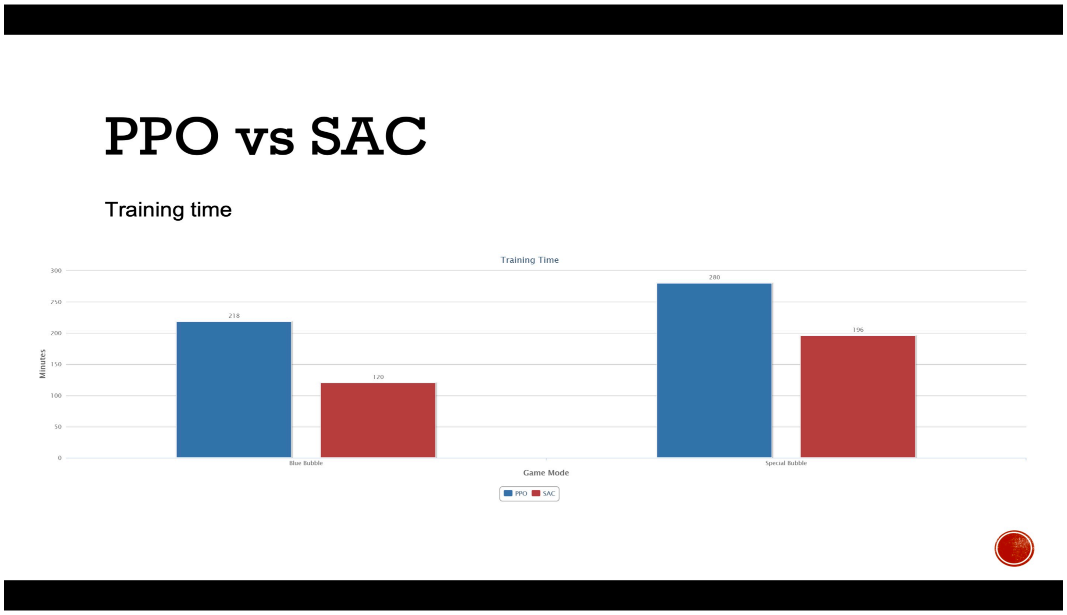
Fig. 8. Training results for special bubble mode with different level difficulty



PPO vs SAC

Training samples

Fig. 9. Training sample required by PPO and SAC



PPO vs SAC

Training time

Fig. 10. Total training time of PPO and SAC