Stirring the Pot - Teaching Reinforcement Learning Agents a ”Push-Your-Luck” board game

Maximilian Hünemörder
Christian-Albrechts-University
Kiel, Germany
mah@informatik.uni-kiel.de

Mirjam Bayer
Christian-Albrechts-University
Kiel, Germany
miba@informatik.uni-kiel.de

Nadine-Sarah Schüler
Ludwig-Maximilians University
Munich, Germany
n.schueler@lmu.de

Peer Kröger
Christian-Albrechts-University
Kiel, Germany
pkr@informatik.uni-kiel.de

***Abstract*—Recent successes in AI research concerning traditional games like GO, have led to increased interest in the field of reinforcement learning. Modern board game design, however, has risen in complexity. This paper introduces a novel task for reinforcement learning: ”Quacks of Quedlinburg”. A modern board game with risk management, deck building, and the option to choose a specific rule set out of thousands of possible combinations for every game. We provide an environment based on the game and perform initial experiments. In these, we found that Deep Q-Learning agents can significantly outperform simple heuristics.**

***Index Terms*—Risk management, Board Game Environment, Reinforcement Learning, DQN, DDQN**

I. Introduction

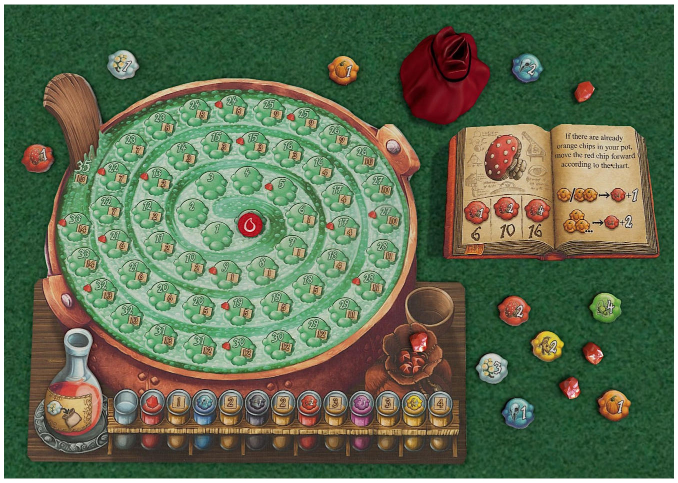
Board games have played a part in human history for thousands of years. Since games often serve as simple simulations for real life decision making processes, they provide interesting tasks for modern artificial intelligence research. For example, Chess holds a special place in the history of AI research [1], presumably because it is internationally played, has perfect information, is completely deterministic and reasonably brute forcible [2]. However, modern board games, especially so-called ”German Board Games”, e.g. ”Settlers of Catan”, have become much more complex. They incorporate tasks like resource or risk management, can involve social interaction and generally complex strategic decision-making.

A popular subdomain of artificial intelligence is reinforcement learning (RL). RL research and board games are linked since early work was based on learning to solve games like Tic Tac Toe, Checkers, and Blackjack [3]. While early RL agents performed worse than traditional algorithms on such games, the addition of deep learning techniques, e.g. Deep Q-Learning [4], lead to recent success stories for more complex games like GO [5]. These developments show that RL can even tackle problems that were thought to be very hard by using traditional AI.

Therefore, in addition to many modern board (and card) games, that have already been researched in the context of RL [6]–[11], we introduce the game ”Quacks of Quedlinburg” as a novel and interesting RL task. This game from 2018 has a risk management component akin to Blackjack and a deckbuilding mechanic that strongly rewards planning multiple rounds ahead. For example, a riskier behavior in the beginning might lead to a higher reward in the later rounds of the game. The main reason why we believe this game in particular to be of interest to the RL research community is that it has a set of interchangeable rules, that allow for over 45.000 different rule sets. In this preliminary work, however, we only focus on a single set of rules. Our contributions can be summarized as follows:

- A novel application for RL based on the popular board game ”Quacks of Quedlinburg”
- An implementation of this game in Python and a corresponding environment to be used by RL agents
- Preliminary experiments using simple heuristic AIs and the established RL algorithms Deep Q Network (DQN) and Double Deep Q-Network (DDQN)

Parts of this work has been done in the MARISPACE-X project funded by German Ministry of Economy (BMWi)


If there are already orange chips in your pot, move the red chip forward according to the chart.
6
10
16
Fig. 1: A player game board (cauldron) and a rule card for the red ingredients. The drop marker in the cauldron indicates the starting place each turn. The bag of ingredients in the top-right corner, exemplary ingredients and some rubies are placed around the game board. (Screenshot of Tabletop Simulator)

II. The Game

”Quacks of Quedlinburg” is an award-winning German board game by Wolfgang Warsch. In this game, each player embodies a quack doctor, who is mixing up their own brew given various ingredients by successively drawing from an opaque bag. One of the ingredients however is spoiling the mixture and when a cumulative value of 7 is exceeded, this player's cauldron explodes and they are restricted in their following actions for this turn. Every turn of the game is played in two phases, a brewing and a scoring phase. While brewing, each player blindly pulls ingredients out of their own bag and places them in their cauldron in order, advancing on the drawn circular path, see Fig. 1. Each ingredient is marked by a color and a value, indicating additional rules to be applied throughout the game. In this process, the player can decide to stop drawing from their bag at any time, given their cauldron did not explode yet. After all players have either stopped or exploded, the second phase begins. In case of an explosion, the affected player has to choose to either forgo his earned VP for this turn or waive his chance to buy new chips. In the scoring phase, all advances on each players cauldron are scored as *victory points (VP)* and marked on a common score board. Further ingredient rules are applied, then each player can use his advances from the previous phase to buy new ingredients (two distinct ones per turn) and add them to their bag. With every bought ingredient, the risk of pulling a white ingredient out of the bag is reduced. This is repeated for nine turns, after which the game ends and the player with most VP wins the game. Note, that in this preliminary work we use a simplified single-player version of the original game, as it contains the main mechanics and ideas but was reduced in complexity and some game features were omitted for now. (Full instruction sheet[1])

We programmed the above described game in Python using a virtual representation of the score board (the cauldron) and the white, orange, green, red, blue, yellow and purple ingredients (black was omitted here as it only works in a multiplayer game). Each ingredient color is associated with a specific rule. The full game contains multiple different rules per color to choose from, as mentioned above, we will limit ourselves to one rule per color. Our selection can be found in Table I. One further resource of the game we implemented are *rubies*. A player obtains a ruby by either earning one from an ingredient (blue and green) or by stopping on a certain position. Pairs of rubies can be traded for advancing a *drop* placed in the middle of the cauldron, that corresponds to the starting position each turn. The rubies can further be saved up and count as additional VP at the end of the game, making collecting them a secondary objective.

All information of the game, such as the scoring board and each player's belongings, is saved in a collection of variables called `gamestate`. Each player's cauldron and ingredient bag are represented by lists of ingredients. Each ingredient is represented as a tuple containing the color and value of that chip (i.e. `['white', 2]`). For each of the nine turns, we store each player's temporary advances, such as preliminary victory points, earned money as well as whether that player exploded or voluntarily stopped in the current turn. The number of rubies, the assured victory points and the starting position marked by the drop conclude the gamestate.

III. Experiment Setup and Compared Methods

To study this strategic decision task, we observed different AI agents playing the game. The implementation can be found in our public git repository[2].

A. Environment

In order to train and evaluate the RL agents, we build an environment using the TensorFlow PyEnvironment class, that wraps around the implementation of the game itself. This allows us to return rewards to the RL agents based on the game state. The environment performs the agents' chosen action, yielding the game state update. A description of the exact observable game state and action space can be found in our repository.

1) Reward: We chose to use the earned victory points (VP) as the reward measurement for the agent. This guarantees that the reward directly corresponds to the main objective of the game. Additionally, we avoid skewing the learned strategies by rewarding actions that might be artificially enforced and do not lead to direct VP gain. Consequently, the task becomes more laborious as not every action results in victory point, i.e. an immediate reward. However the following section shows that this reward function is sufficient for the agents to learn that an action without an immediate gain can still result in a higher overall result.

2) Legal Actions: A challenge this game presents is that some game states entail that an agent cannot perform certain actions. The legal actions vary based on the current phase of the game, the chosen rule set and the game's progress. We therefore add a mask of currently legal actions to the observations the agents receive.

B. Random Agent

In order to be certain, that *"Quacks"* is not only based on pure chance we programmed a random agent, which randomly picks an action at each step of the environment. Even using the same method to bound the legal moves mentioned above, this agent is expected to perform poorly, because in the brewing phase the agent is faced with a string of binary decisions to draw an ingredient from the bag or stop drawing (similar to Blackjack). When randomly choosing at this point, the agent usually stops very early, resulting in little reward and no possibility to expand by buying more ingredients.

C. Heuristic Agents

For a more advanced baseline, we implemented a number of heuristic agents, that act in simple, predetermined ways.

[1]https://cdn.1j1ju.com/medias/ba/73/db-the-quacks-of-quedlinburg-rulebook.pdf, Visited: 13.05.2022

[2]https://github.com/huenemoerder/quacks-rl

Rule Color	Rule Text	Time to apply the rule	Costs (1, 2, 4)
Red	"If there are already orange chips in your pot, move the red chip up 1 or 2 places"	Instantly when drawn	6, 10, 16
Blue	"If this chip is on a ruby space, you IMMEDIATELY receive 1 ruby."	Instantly when drawn	4, 8, 14
Green	"If the last or second-to-last chip in your pot is a green chip, gain one ruby"	At the end of each turn	4, 8, 14
Yellow	"Your first placed yellow chip is moved 1 extra space, the 2nd yellow chip 2 extra spaces and the 3rd yellow chip 3 extra spaces"	Instantly when drawn (available from turn 2)	8, 12, 18
Purple	"For 1, 2 or 3 purple chips you receive the indicated bonus. 1: 1 VP - 2: 1 VP + 1 ruby - 3: 2 VP + 1 drop"	At the end of each turn (available from turn 3)	9

TABLE I: Overview of exemplary rules in the game (white and orange have no special rules)

1) "Explosive" Agent: The explosive agent is programmed to always draw as many chips as possible until its cauldron explodes so it is not allowed to draw chips anymore. Afterwards the agent always chooses to take the earned VP instead of buying any ingredients. Consequently, the contents of its bag stays the same and it can never advance far on the board, not gaining many VP.

2) "Single-Color" Agent: For each color ingredient (red, green, blue, yellow and purple) we designed different agents, that would draw chips until the risk to explode is higher than 70%. In the case of an explosion, the agent will buy chips in the first 6 turns and choose VP starting from turn 7. When buying, the agent is programmed to buy the most expensive chip available of its favored color and if there is enough money left, additionally buy one orange chip. A weakness of this type of agent is that it will not use its money efficiently especially in the later turns.

3) "Expensive" Agent: The expensive agent uses a strategy many first time (human) players choose. Analogously to the single-color agent, it draws chips from the bag until the risk of exploding is high ($> 70\%$), however during the buying phase it buys chips that deliberately utilize all of its available money, i.e. it buys the most expensive and the second most expensive chip. This leads to less wasted money but more variation in the color choice.

D. DQN

The Deep Q-Network (DQN) approach was proposed in 2015 by DeepMind, [4] and was first applied on "Atari 2600 Games" [12]. Exemplary for our RL algorithms we trained two off-policy agents on the environment described in Section III-A. Each trained agent consists of a learned Q function which approximates the expected return at a given state (s) for a single action (a). This function is the agent's policy which enables it to select the best action to take given an input state. Due to our complex game that has an abundance of game state options, simple q-learning based on value iteration to fill a q-value table is not feasible. Instead the optimal Q function is approximating by training a neural network using a loss $L(\theta)$, that is computed during each training step by computing the difference of the predicted q values $Q(s, a, \theta^{pred})$ to the target values $Q(s, a, \theta^{target})$ of the Bellman Equation factored with γ added to the reward r.

$$L(\theta) = Q(s, a|\theta^{pred}) - (r + \gamma \max_a Q(s, a|\theta^{target}))$$

The used Q network architecture contains two dense hidden layers, the first layer with 150 and the second with 75 neurons. Both layers use ReLu activation and are initialized using an truncated norm distribution. The output layer contains as many elements as the environment allows actions. The TensorFlow library, [13], provides a variety of pre-implemented agents including a DQN agent, which we initialized using this Q network. The policy was updated using the Adam optimizer and we used epsilon greedy exploration with a probability of 0.1. The agents were equipped with a TensorFlow Uniform replay buffer with a capacity of maximum 100,000 trajectories. Because all parties, i.e. the agent and the environment are Python based, integration was seamless. The serial interface is defined by the structure of the observation and action tensors, the two components which are passed in between the agent and the game. As mentioned in section III-A2, in order to communicate the legal actions at any state, the agent is given a mask that encodes this information. The mask is applied to the network's q values for a given state. The q values of any illegal action are set to the minimum of the q values, ensuring only legal actions are chosen. The training was performed using a batch size of 64 and a learning rate of 1e-4. The displayed victory points in Fig. 2a were achieved by the DQN agent after 267.000 training steps.

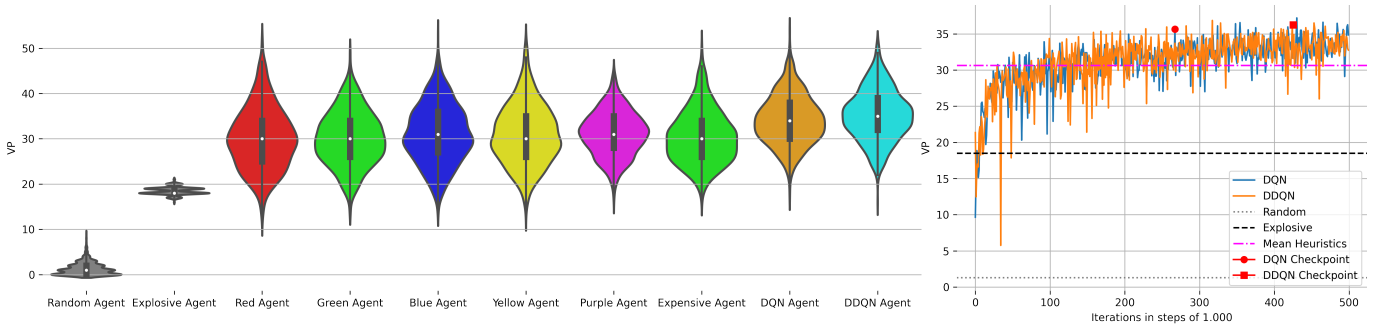
E. DDQN

In addition to the DQN agent a Double Deep Q-Network (DDQN) was trained. DDQN is an amplification of DQN proposed in 2015 by Hado van Hasselt et al. [14]. A DDQN agent takes advantage of fixed Q targets using an additional Q network, the target network. Additionally, this algorithm solves the overestimation that is to be expected from the DQN agent by using two networks. The local network will be used to calculate the single q target for the current state but the second network is used to calculate the q values in the loss objective. The same agent parameters and implementation base as for the DQN agent were chosen. The victory points shown in Fig. 2a were achieved by the DDQN agent after 425.000 training steps.

IV. RESULTS

We evaluated each of the compared agents by calculating the average victory points over the same 1,000 seeded games. The seed ensures a fair comparison since the played games are deterministic. The results are shown in Fig 2a.

Starting from the left of the figure, as expected the random agent performs poorly, receiving 1.4 VP on average. The explosive agent performs better and is able to obtain an average of 18.5 VP. The single-color agents as well as the expensive



(a) Average victory points achieved for the same 1,000 seeded games

(b) Training progress for the RL agents

Fig. 2: Overview and comparison of all agents

agent all perform similarly, reaching an average VP score of around 30, (Red: 30.1, Green: 30.1, Blue: 31.0, Yellow: 30.9, Purple: 31.2, Expensive: 30.5). This implies that our chosen set of color rules is balanced and no color is significantly better on its own. The DQN and DDQN agents outperform the base line agents. Both RL agents yield a higher mean of VP, DQN with an average of 34.2 VP and DDQN with 35.2 VP respectively. Hypothetically, if pitted against each other the DQN agent would win 35.3 % of all seeded games against the heuristic agents, while the DDQN would win 42.6 %. When compared directly against each other the DDQN performs slightly better with a winrate of 53.1 %. We trained both agents for 500.000 iterations, we evaluated 20 random games each 1.000 iterations and picked the iterations where they performed the best. In Figure 2b these returned results are plotted and the checkpoints are marked. The average returns of the other agents are marked as horizontal lines. The agents were initialized with 500 games taken from a random policy, but almost instantly overtake the random agent and after 10.000 consistently reach a higher score than the explosive agent. After 100.000 iterations the agents have roughly found their optimum and outperform all the heuristic agents. The almost equal performance of DQN and DDQN might point to there not being a better strategy at least for the exact rules combination. We will explore this further in future work.

V. Conclusion and Future Work

Our hitherto research concludes that RL agents can learn the simplified version of the game and detect optimal strategies, despite the strong luck component. Following up on this, the next steps would entail extending our environment to support multiplayer and adding the corresponding rules and features. Besides that, we aim to train more advanced RL agents. So far, we only used one rule during training. In the future, we aim to extend the game to incorporate the 6 different rules every ingredient color can be assigned. In the full game, i.e. using all six ingredients, this leads to 6^6 different possible rule combinations. We would like to perform more studies on how agents would adapt to the application of the high variation of rules available in the full game and how a generalized strategy for the enhanced game could look.

References

[1] D. Heath and D. Allum, "The historical development of computer chess and its impact on artificial intelligence," in *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence*, 1997.

[2] C. Koch. (2016) How the computer beat the go master. [Online]. Available: https://www.scientificamerican.com/article/how-the-computer-beat-the-go-master/

[3] R. Sutton. (1997) History of reinforcement learning. [Online]. Available: http://www.incompleteideas.net/book/1/node7.html

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[5] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, 01 2016.

[6] M. Pfeiffer, "Reinforcement learning of strategies for settlers of catan," 05 2022.

[7] K. Xenou, G. Chalkiadakis, and S. Afantenos, "Deep reinforcement learning in strategic board game environments," in *Multi-Agent Systems*, M. Slavkovik, Ed. Cham: Springer International Publishing, 2019, pp. 233–248.

[8] Q. Gendre and T. Kaneko, "Playing catan with cross-dimensional neural network," *CoRR*, vol. abs/2008.07079, 2020. [Online]. Available: https://arxiv.org/abs/2008.07079

[9] R. Canaan, X. Gao, Y. Chung, J. Togelius, A. Nealen, and S. Menzel, "Evaluating the rainbow DQN agent in hanabi with unseen partners," *CoRR*, vol. abs/2004.13291, 2020. [Online]. Available: https://arxiv.org/abs/2004.13291

[10] L. Perez, "Mastering terra mystica: Applying self-play to multi-agent cooperative board games," *CoRR*, vol. abs/2102.10540, 2021. [Online]. Available: https://arxiv.org/abs/2102.10540

[11] D. Zha, K. Lai, Y. Cao, S. Huang, R. Wei, J. Guo, and X. Hu, "Rlcard: A toolkit for reinforcement learning in card games," *CoRR*, vol. abs/1910.04376, 2019. [Online]. Available: http://arxiv.org/abs/1910.04376

[12] M. G. Bellemare, J. Veness, and M. Bowling, "Investigating contingency awareness using atari 2600 games," in *Twenty-Sixth AAAI Conference on Artificial Intelligence*, 2012.

[13] M. Abadi and A. A. et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[14] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015. [Online]. Available: http://arxiv.org/abs/1509.06461