Reinforcement Learning in Drop7

Adam Zaffram
College of Interactive Games and Media, Rochester Institute of Technology
Rochester, NY 14623 USA
acz1391@rit.edu

Abstract

This paper goes into detail about the application of Q-Learning to the game Drop7 created by Area/Code Entertainment. This document outlines the intended implementation of the algorithm and how the results will be evaluated. This algorithm is compared to an artificial player that makes choices at random to determine if it can achieve higher performance.

Introduction

Games like Chess and Go have already been beaten by advanced forms of artificial intelligence. Companies like DeepMind are the driving force behind this movement and have been putting well known games to the test. With a combination of Q-Learning and Neural Networks, the baseline for human performance can be exceeded without a previous model. This has been proven in almost all of the early popular games such as Space Invaders and Pong (Mnih et al. 2015). Unfortunately, with the recent publishing of games being beaten by similar algorithms, Area/Code Entertainment's game, Drop7, has not received much attention.

I will be taking part in changing this notion by working on my application of the Q-Learning reinforcement learning method to Drop7 using C#. In applying this algorithm, I am looking to determine if an agent driven by this algorithm can achieve a human-level performance.

Background and Gameplay

Drop7 was released in 2009 by Area/Code Entertainment, which was acquired by Zynga Inc. in 2011 and renamed Zynga New York (Wikipedia 2022). Drop7 can be generically described as a puzzle game with a finite number of actions and items (Wikipedia 2022). The main interactable object in the game is the disc. There are a total of eight types of discs in the game. This includes discs numbered from one to seven and a blank disc. Discs can only be dropped into the Drop7 playing field, a 7x7 grid. During a game of Drop7, the player is greeted with a disc at the start of every turn and has the option of choosing one of the seven columns to drop it. These actions are the same across all game modes, however the starting playing field and provided disc on each turn might differ. According to Wikipedia, Drop7 has three game modes, comprised of "Normal Mode", "Blitz Mode", and "Sequence Mode" (2022). I will be implementing the Q-Learning algorithm into the "Normal Mode".

"Normal Mode" starts the player off with a blank 7x7 grid and follows the traditional ruleset. The player is provided with a disc at the start of each turn, which can then be dropped into any of the seven columns unless the column is full. The disc provided in "Normal Mode" can be any of the available options which consist of a blank disc, and discs numbered from one to seven. To receive points, the player must break a disc. To break a disc, the player must drop a disc such that the number of continuously touching discs in either the same column or row match the disc's number. This can invoke a chain reaction that exponentially increases the multiplier. Blank discs take three hits to break while the others take one. The blank disc turns into a randomly numbered disc when it has one life left. After a certain number of turns, the board is shifted up by one row with the bottom row being filled with blank discs. I chose to have the shift occur after ten turns. Turns are repeated until no more columns can be chosen or a disc is pushed above the top row after a shift occurs.

Related Work

Reaching number two in IGN's top twenty-five iPhone Games list in 2009 (Wikipedia 2022), Drop7 was among the most popular games at the time. Unfortunately, with poor game maintenance and company controversy, it wasn't long until the game was unobtainable in all smartphone app stores. This could be the possible cause as to why there is a lack of available information relevant to Drop7 and little to no artificial intelligence research conducted on it. Fortunately, I was able to find one paper that directly involves applying machine learning to the game Drop7. While I do not know the validity of this paper, it has substantial data

regarding their experiments and uses a similar method of implementation. According to Ben Friedmann and Erez Klein (2018), they chose to just use Q-Learning due to their time constraints with the addition of linear function approximation. Due to Drop7's massive possible state space, using Q-Learning without a Neural Network limited their data collection. Because of this, they used linear function approximation and limited themselves to 50,000 iterations for the reinforcement learning to complete. Friedmann and Klein were then able to play 10,000 games using this policy.

For their evaluation, their baseline was based on an agent that would choose an action to take at random, as well as a set of human scores. This random agent would have data collected on 5,000 games while human scores would be collected on thirty. The two researchers concluded that their reinforcement learning agent outperformed the random agent, however, it did not outperform the human players (Friedmann and Klein, 2018).

As my methods of implementation and evaluation are similar, the section where I start to stray apart from Friedmann and Klein is in their strategy of awarding the player. While there are many possible methods of providing awards, Friedmann and Kline chose to award the artificial player one point "as long as the game did not end" (2018). While I think this is a fair system to have in place, the player will not have any motivation to make moves that would accumulate a better score and benefit the future of the game. I will break down how this method differs from my system and how it will ultimately provide a different set of data for evaluations. The last thing I differed from their method is that they have the discount rate set to one which will prioritize expected future rewards. I wanted more exploration in my case.

In a paper published by Arthur L Samuel (1959), Arthur studied the way machine learning would play checkers in hopes to conclude that the implemented artificial intelligence can play better than the person who made the program. While Samuel found success in a short amount of time, he also made a decision that was counterintuitive at first thought. He ignored all the values if the terminal states reached. This differed from how Friedmann and Kline and I updated weights. I will be taking those data sets and feeding them back into the agent, so it knows the outcome of each instance.

Method

As previously stated, I chose Q-Learning, a method of reinforcement learning, to achieve a human-level performance. With any form of machine learning, a solid winning strategy is necessary. For the agent to learn its environment without an initial model, I used a Markov Decision Process.

Per the Markov Decision Process, there must be a set of states (S), a set of actions (A), the probability of S' given an action, A, in state S ($P_a(s,s')$) and an award for every completion of an action ($R_a(s,s')$). The state space, S, that I utilize for the algorithm includes each cell in the 7x7 grid and the next disc to be dropped. The action space consists of each column in the grid which can be mapped onto numbers one to seven. Neither probability nor reward can be calculated or initialized before the start of the game due to the massive state space, so they are both calculated as the game progresses.

As for the reward policy, I utilized the current scoring mechanic built into the game with the consequences of losing the game. If the agent destroys any discs, it will receive the final score calculation of that chain. With every sequential disc destroyed in a chain, the exponential multiplier goes up. Any discs destroyed on the first chain seven to the first. Any on the second chain is seven to the second. This repeats for however many chains. Needless to say, the agent will prioritize destroying discs with the hopes of improved final scores. The agent will also avoid any game-ending actions as it will be rewarded negative ten points if that happens.

To calculate the Q-value, or quality, associated with each state, I utilized the Bellmen Optimality Equation:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha(r + \gamma max(Q(s',a'))$$

This equation provides me with the Q value for the current state, S, and action, A. α is the learning rate. I have this set at 0.4 because I want it to prioritize the current state while also giving the expected future reward a decent amount of weight. γ is the discount rate which determines how much we should prioritize future rewards. Because rewards can be drastic in size, I have this value set at 0.90 so it gets the majority of the future reward. This decision helps with computational efficiency since the algorithm is doing less exploring and grabbing the more important data points.

Evaluation

To make this evaluation, I have two versions of players in the game for comparison. The first player will be my baseline comparison for the game while the other will include the applied algorithm. Like Friedmann and Kline (2018), the first player will play the game by dropping discs randomly until they lose. The second player will have the Q-Learning algorithm built into it. What makes this algorithm theoretically smarter than randomly picking a spot to place a disc is its parameters. It will be able to see a snapshot of the game's status and be able to make decision from this information.

To determine which agent outperformed the other, I will be running both algorithms 1,000 times each and observing the level and score of each game. From this, I will compare

the frequency distribution and averages of the level and scores reached. The smart agent will be learning as the games play out. I set the algorithm to make 20 iterations before every turn in one of the 1,000 games. The level and score data from each game are stored into an excel document which is used for my analysis.

Results

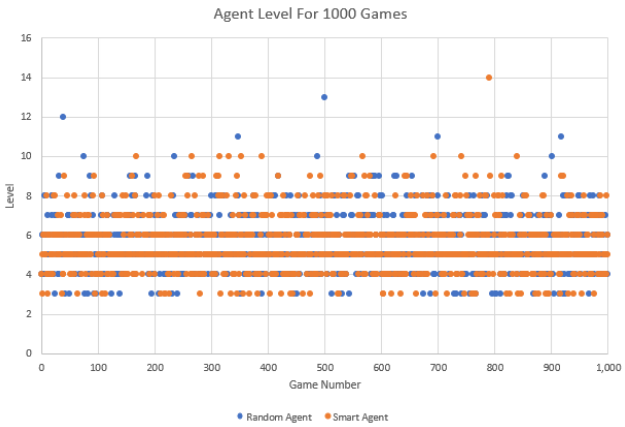

Table 1: The level reached for the random and smart agent over one-thousand games.

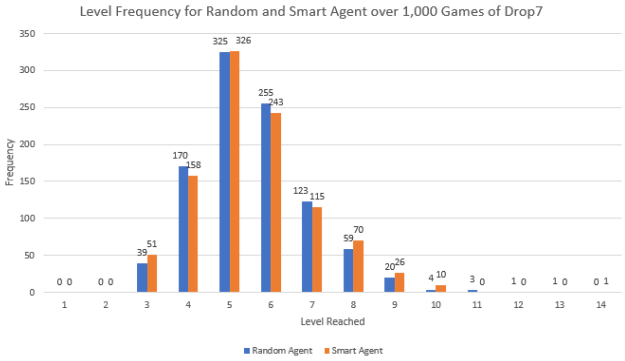

Table 2: The distribution of reached level of the random and smart agent over one-thousand games.

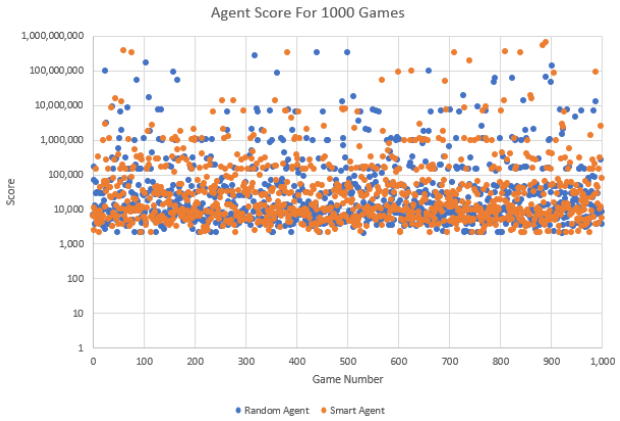

Table 3: The final scores of the random and smart agent over one-thousand games.

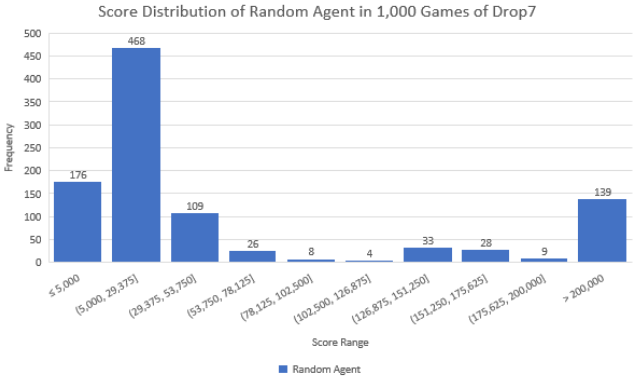

Table 4: The distribution of final scores of the random agent over one-thousand games.

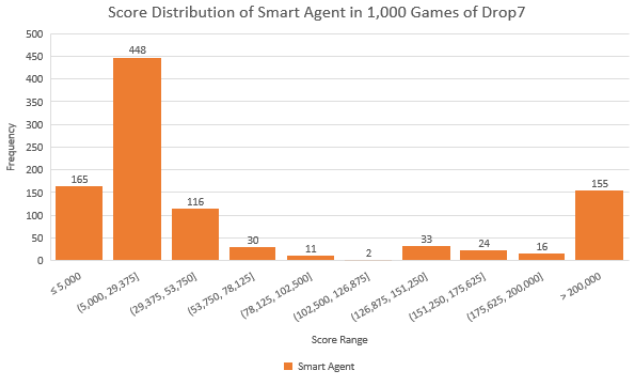

Table 5: The distribution of final scores of the smart agent over one-thousand games.

Interpretation

Comparing the resulting data of the random agent to the data of the smart agent, it is clear that there is no significant difference between their performances. This conclusion did not

come as a surprise. The similarity between the agents is likely due to a collection of limitations, many of which Friedmann and Kline had also encountered. In Table 1 and Table 3, there is no significant increase in scores or levels for the smart agent as it progressed through the games. Additionally, looking at the distribution of the final level reached, and final score obtained for both agents, they are almost identical. However, it should be noted that the smart agent had 16 more games more than the random agent that surpassed a score of 200,000. Furthermore, the smart agent had eleven less games than the random agent that scored less than 5,000 points. To my initial surprise, when looking into the averages across all the games played, the smart agent had an average final score approximately 1.8 times the average final score of the random agent. The smart agent had an average of about 2,410,254 points while the random agent had an average of 5,324,567 points. Looking into the average final level data for both agents, they are almost identical. The random agent accumulated an average of 5.56 as the smart agent accumulating an average of 5.59. This difference is insignificant.

Conclusions

Given that the reinforcement learning algorithm had a higher score than the random algorithm, I think it can be explained through chance. The smart agent has a few outstanding outliers that push the overall average higher. After logging another 1,000 games for the smart agent, I obtained an average of 2,230,919, which is much closer to the random agent results.

The root of the poor results is likely a result of the enormous state space that Drop7 has. With such a big state space, the possibility that the algorithm will encounter the same grid state with the same next disc is incredibly small. In order to maintain a state that can outperform the random agent, the smart agent would need to simulate many more games such that the probability of running into stored data is higher. Unfortunately, doing this would only lead us to another limitation, memory storage and computation time.

With the conclusion that the two agents are almost indistinguishable from each other, it is understood that this algorithm will not be effective without the addition of a Neural Network or some way to represent data much more efficiently. Without these, many factors are added to the equation with just the use of reinforcement learning.

Future Work

In future work, I think a better form of evaluation on something like this would be to run many more tests on the game to get more accurate results. It also might be effective to define a new reward policy. A more effective and readable reward policy would require uniform reward distribution for particular actions. This would provide data that has less outliers and is overall closer together. In addition to a better reward policy, I would like to introduce the comparison against human data. This would better help gage if the reinforcement learning is actually performing to a human level. I could compare it to a hundred games played by me to determine what human-level performance data looks like. It might also be helpful to compare my algorithms performance to the Friedmann and Kline's algorithm for better judgement.

Based on the conclusion of the project and lessons learned, there is undoubtedly room for improvement. I would like to continue working on this project to find a more optimal method of data collection, retrieval, and updating. I am determined to advance my knowledge in this area of reinforcement learning and achieve observable enhancements.

References

Area/Code Entertainment, 2009. Drop7

Arthur L Samuel. 1959. Some studies in machine learning using the game of checkers. IBM Journal of research and development 3, 3 (1959), 210–229.

deeplizard, 2018. Q-Learning Explained – A Reinforcement Learning Technique, In *YouTube.*

deeplizard, 2018. Exploration vs. Exploitation – Learning the Optimal Reinforcement Learning Policy, In *YouTube.*

Friedmann, B. and Klein, E. 2018. Final Report – Drop7

Kolodko, A. 2017. Drop7 Browser Game

Wikipedia Contributors, 2022. Drop7, In *Wikipedia*

Wikipedia Contributors, 2022. Zynga, In *Wikipedia*

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, et al. 2015. "Human-Level Control through Deep Reinforcement Learning." Nature 518 (7540): 529–33. https://doi.org/10.1038/nature14236.