Conquering Catan: A Comparative Study of Reinforcement Learning Algorithms for Strategic Gameplay

Gyde Lund
Taylor Kalajian

1 Abstract

Catan is a highly complex and strategic board game featuring multiple players, stochastic behavior, a complex state environment of hexes, intersections, and paths, and a large hierarchical action space. This makes it a challenging environment for an RL agent to learn within without highly complex networks and long training times. In this paper we implement two learning algorithms, Actor Critic (AC) and Deep Q Network (DQN) using a feature representation of the environment and heuristics to perform certain actions. We show that there is room for improvement in this implementation as well as show evidence that this approach could be applied to the entire game eventually providing a fully trained Catan agent.

2 Introduction

Reinforcement Learning, a sub-field of machine learning, focuses on enabling agents to learn optimal decision-making strategies by interacting with an environment, receiving feedback in the form of rewards or penalties, and adjusting their actions accordingly. RL has demonstrated remarkable success in a variety of domains, including robotics, finance, and gaming. We can see these successes in its application to strategy board games as well. Specifically we are hoping to test two popular learning methods to find the best possible policies for the complex, strategy game, Settlers of Catan.

The mechanics of the board game Catan consist of 2-4 players whose goal is to earn 10 victory points through building settlements, cities, roads, and purchasing development cards. A typical game board can be seen in Figure 1 showing the 19 hexagonal land pieces that are randomly placed at the start of each game with each piece characterized by an associated resource and probability of production. The edge of a piece is called a path and a corner an intersection with a board featuring 72 paths, and 54 intersections. On a player's turn they roll two 6-sided dice and any settlements on the matching tiles receive those resources. Because a player needs certain combinations of resources to build with and each player may not have access to each resource, trading between players is often necessary.

This proposal aims to use RL to develop an AI system capable of mastering the elements of the Catan game to showcase how these methods could be expanded to fully learning the game. We have selected Deep Q-Learning (DQN) as well as Advantage Actor Critic (AC) as the learning methods for analysis based on previous research and personal interest. Additionally both DQN and AC are well suited for working in massive state action spaces and comparing their results to one another as well as standard heuristic bots will offer new insight into both methods. Once the agents are

Figure 1: Board Game Settlers of Catan [1]

capable of playing this game using their different learning methods, we can compare their efficiency and work to further optimize our approach to this problem.

3 Game Mechanics And Motivation

There are a few motivations behind this problem. For one it is a uniquely challenging environment for an RL agent and will test the limits of AC and DQN as well as test our choices in how we limit the state-action space. Additionally we have interest in the game recreationally with some background strategic knowledge so any strategy the agent learns will be interesting to analyze. In addition, Catan is a game with an aspect of random chance meaning there is not one deterministic best strategy that can win in every game or in all game states.

4 Background and Related Work

Games remain a popular testbed for various RL techniques and AI research as AlphaZero's general reinforcement learning method successfully mastered games like Chess, and Go without human heuristics. These are deterministic games with perfect information, however, while Catan has imperfect information (other players hands/dev cards), is non-deterministic (random chance of resource production), and offers a high dimensional state-action space. For this reason along with its popularity, Catan has been the subject of many studies using various reinforcement learning methods that we hope to explore and compare here.

Early attempts to learn in Catan such as those by Michael Pfeiffer used Hierarchical RL [6] alongside heuristics, in which a high-level policy first selects a behavior and that behaviors sub-

policy can choose several lower level actions. They used model trees to select actions, training on a simplified version of the game and had a agent capable of occasionally beating a human. Future research delved into the use of an Actor-Critic method such as A2C used by Gabriel van der Kooij [9] and a modified version of A3C used by Gendre and Kaneko [2]. In particular the model used by Gendre and Kaneko saw success using a unique Advantage Actor Critic (AC) using Cross-Dimensional Neural Networks (CDNN). The implementation of this complicated of a method was beyond the scope of the project, however and a more tranditional actor/critic method is used not dissimilar from the A2C approach used by van der Kooij [9]. Another approach with promising results is using Deep Q-Learning (DQN) such as the experiments done by Kim and Li in 2021 [4] and by Qsettlers online, also in 2021 [5] . Both approaches were able to play a modified game and beat some bots, but never any humans or the full game. Our goal will be to look at the two learning approaches of AC as well as DQN along within Pfeiffers hierarchical and heurisitc framework to compare methods and create an agent capable of challenging the purely heuristic bot.

5 AI Methods

5.1 DQN

Deep Q-Learning (DQN) is a value based method that has been a popular tool for reinforcement learning for some time now using neural networks to estimate Q-values. In instances such as Catan where the state-action space is so large it isn't possible to learn Q values tabularly, value-function approximation is required where a measure of loss will be minimized over time by adjusting weight vectors within the neural network. This is what gives DQN its 'deep' moniker, the use of neural networks to estimate the Q-values by updating weights between nodes. The formula for loss is often measured using mean square error (MSE) though in our implementation L1 loss or mean absolute error is used and whose formula is shown. The agent also has an experience replay as well as a

$$\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Figure 2: L1 Loss Formula

target model which technically makes our implementation a Double Deep Q Network (DDQN, often just called DQN here). Experience Replay collects a memory of (state, action next state, reward, done) experiences and when training occurs will randomly sample a buffer from the experience to update model weights from. This is done to break bias of concurrent events and help prevent the main DQN model from adjusting too aggressively one way or another [4]. Thus the agent will interact with the environment, collect experiences, and update its Q-network by using the loss function. The weights are being optimized using the adam optimization technique. Adam, short for Adaptive Moment Estimation, is an extension of the more basic Stochastic Gradient Descent and should help our agents learn faster [3].

The pseudo-code generally followed for this project can be seen in Figure 3 with a few adjustments. The target network was updated every 10 training episodes (C in the code), loss was calculated using L1, SGD was swapped out for Adam, the memory D is 500 transitions, and the mini-batch size was experimented with, but ultimately was 64 transitions.

Initialize network Q
Initialize target network $\hat{Q}$
Initialize experience replay memory D
Initialize the *Agent* to interact with the Environment
while *not converged* **do**
  `/* Sample phase`
  $\epsilon \leftarrow$ setting new epsilon with ϵ-decay
  Choose an action a from state s using policy ϵ-greedy(Q)
  Agent takes action a, observe reward r, and next state s'
  Store transition $(s, a, r, s', done)$ in the experience replay memory D
  if *enough experiences in D* **then**
    `/* Learn phase`
    Sample a random *minibatch* of N transitions from D
    for *every transition* $(s_i, a_i, r_i, s'_i, done_i)$ ***in*** *minibatch* **do**
      if $done_i$ **then**
        $y_i = r_i$
      else
        $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
      end
    end
    Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2$
    Update Q using the SGD algorithm by minimizing the loss $\mathcal{L}$
    Every C steps, copy weights from Q to $\hat{Q}$
  end
end

Figure 3: Pseudocode for a DDQN Network [8]

5.2 Actor/Critic

Actor-critic learning is a popular reinforcement learning technique used in machine learning. It combines value-based (critic) and policy-based (actor) methods to enhance the efficiency of learning in complex environments. Similar to DQN, Actor-Critic also has difficulty handling the large state-action space that is present in Catan. Several methods are utilized to overcome this limitation. A value-function approximation, along with a feature extraction, is used to approximate the state-action space and gives the agent the ability to capture important patterns from the observed states. In addition we are utilizing a Policy Gradient Method, or policy approximation which optimizes the policy parameters using gradient ascent, and ensures exploration by preventing the policy from becoming deterministic [7]. The formula for this calculation is shown below: The Actor-Critic agent

$$\pi(a|s, \boldsymbol{\theta}) \doteq \frac{e^{h(s,a,\boldsymbol{\theta})}}{\sum_b e^{h(s,b,\boldsymbol{\theta})}},$$

Figure 4: Calculation for policy approximation [7]

interacts with the environment, taking actions based on its policy with the purpose of maximizing its rewards. The critic evaluates these actions using an estimated value function or an expected return and determines how good the actions take by the actor for a given state were. Since our state-action space is defined by a number of features, the agent will apply weights based on the features to each action that it can take and to the policy as the critic makes evaluations. The pseudo-code in the figure below is the basis of what was used to setup this agent.

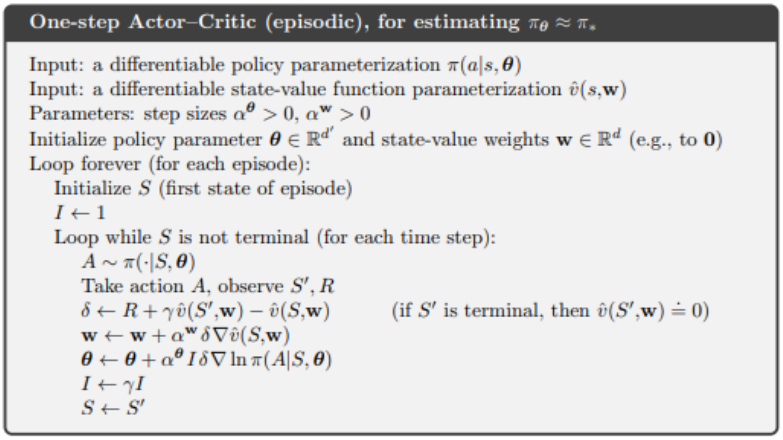


Figure 5: Calculation for policy approximation [7]

6 Environment

The environment turned out to be a larger portion of the problem than originally anticipated. Our plan had been to use the "industry standard" JSettlers version of the game which runs on Java using a local server. The goal was for JSettlers to emit the state each time it was the agents turn to choose an action which would go to python where the agents respective learning algorithm could process it and return an action. This integration proved to be very difficult and as such we began to investigate a number of different implementations of the game within Python finally settling on a version called PyCatan2–a very bare-bones method of running Catan entirely within python [10]. The consequences of this meant we had lost the ability to run our agent against automated jsettlers bots and the implementation of the game was not complete. A number of methods within the game had to be adjusted or created from scratch to follow the actual rule structure of the game and limit issues human players would run into less often than random agents (implementing the 15 road placements per player limit for example).

After a lot of debugging, limiting infinite loop situations, and applying more efficient decision heuristics, we had a Catan environment built within Python that could loop through a 5000 games in an hour on a modern CPU and GPU for DQN learning. This meant that a 2000 episode training run with 10 iterations could be accomplished in around 4 hours. Other mechanisms that were implemented included saving agent policies in pickle files that could be called later for more training as well as allowing the eventual implementation of a fully trained agent playing against a human opponent.

The nature of Catan also drove how episodes and steps were considered. An episode consisted of an entire game of catan going from start to finish while a step for the agent was them taking a turn. This introduced some further challenges to the agents as games ended without agents taking turns and could be seen as elements of stochasicity to the agent as the environment changes without their intervention. The final game state was always returned to the agent for learning, however, regardless of which players action caused it so that final game states could be recognized by the agent.

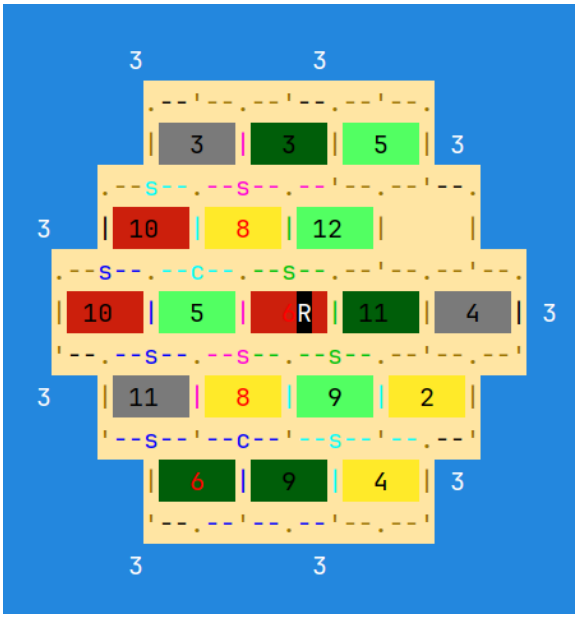


Figure 6: Sample PyCatan Board

7 Actions Setup

Settlers of Catan is a very complicated game for RL tasks and part of that complexity comes from the kinds of actions available to agents at different points in the game. From an RL perspective, there can be considered four action types in Catan:

- Top Level actions actions amount to the 7 possible actions a player can choose following a dice roll such as building a city or choosing to trade. These usually lead to one of the three other action types as subsequent actions.
- Placement actions typically follow a top level action and could be placing the robber on a tile (19 possible hex tile actions), placing a settlement or city (54 possible intersection placement actions each), or placing a road (72 possible path placements).
- Resource choice actions are choosing which of the 5 resources to receive following some other action like playing the Year of Plenty development card.
- Trade actions would be order to simplify the actions as much as possible while not limiting our agent to only making settlement placement decisions

Due to the complexity and hierarchy of all these action types, the choice was made to focus solely on the 7 top level actions with heuristics controlling the other actions. The goal with this setup is to train an agent to learn when to select certain actions. Additionally other simplifications were made:

1. All ports are 3:1 ports with no resource specific ports

2. No Year of Plenty, Monopoly, or Road Building development cards
3. No trading between players. Bank and ports only.
4. Development Cards can be played same turn as their purchase

It was experimented playing with only two players and on smaller boards, but the agents ran well on the full 19 tile board while 4 player games tended to complete faster and get stuck in difficult to play states less often. Shown below are the 7 actions available to our agents each turn and the integer encoding done to represent them:

Top Level Actions:

- 0: Build Settlement
- 1: Build City
- 2: Build Road
- 3: Build Development Card
- 4: Make Trade
- 5: Play Development Card
- 6: Pass

Some of these, such as Build City, are sparse actions available only occasionally. To avoid the agent wasting a lot of training time learning when actions are available, action masking was done for both the AC and DQN agents where only actions valid at that time were available for final action selection.

7.1 Heuristic Agent

The action space and environment necessitated the creation of a heuristic agent that plays the game according to set rules. The heuristics were chosen based on the researchers knowledge of the game mechanics and general strategies to win the game. The heuristic agents top level policy is always choosing to act if it can on top level actions and in this order: Play Knight (only dev card to choose), trade, build city, build settlement, build road, build dev card, pass. The logic behind this ordering is to focus on resource availability since trading between players is disabled. There is room to beat this agent just by knowing what order to prioritize these actions given different game states and long term goals.

The other heuristics used by the heuristic agent such as where to place a settlement and what resource to trade for are shared by the DQN and A2C agents as well. This does mean that our agents ability to exceed the heuristic agents performance is capped to an extent as the decision making behind some of these rules can be stochastic or not actually helpful (for example trading for an ore in a certain situation might be smart, but the heuristic instead returns wheat as it's the players lowest yield resource). So in a way the agent will be learning to optimize this heuristic set of rules to the game, but in theory both A2C and DQN agents can be trained on all the action types either by creating sufficiently large networks and running enough games, or by creating numerous agents that handle the different hierarchies of actions.

8 State Space and Feature Selection

With 19 hexes randomly containing resource types and yield, 54 intersections containing building types and owners, 72 paths containing roads and owners, 9 port locations, and one roaming robber location Catan has a very large state space and representing it is its own challenge. This is even before considering how connected the information can be such as the hexes adjacent to an intersection, which player owns what roads or resources, or more. As a result we decided to simplify the state space down into 44 features:

- Player's Current Victory Points
- Opponents Current Victory Points (3x)
- Players Number of Resource in Hand (5x, one per resource)
- Players Board Yield per Resource (5x)
- Opponents Resources in Hand (15x, 5 per opponent)
- Opponents Board Yield per Resource (15x, 5 per opponent)

These features were chosen because we felt they offered a good mixture of the players current state if choosing a top level action is all the agent is doing. It lets the agent see how the opponents are doing and they should see direct changes to their own yields and victory points with certain actions that they can associate with good or bad outcomes. During the experimentation phase an additional 54 input features were added for a total of 98 to compare performance in the DQN agent. These represented each intersection and could have a -1 if an opponent building was there, a 0 if it was empty, or a 1 if it was occupied by the agent. This was done to see whether including a simple board state like that could create stronger connections between building settlements/cities, victory points, yields, and winning the game.

9 Reward Structure

In order to encourage the agent to learn good behavior a series of rewards needed to be implemented. Many researchers simply used the final victory point standing for rewards, however we felt that the sparseness of those rewards (1 reward signal every 100 turns for example) meant adding additional rewards for subgoals would be smart. Our initial reward structure is as follows:

1. +100 for 1st place
2. +2 per victory point earned on a turn
3. +0.1 per resource earned on a turn
4. -0.2 per card discarded

The motivation for these goals was to try and stay somewhat flexible and generally encourage the player to see some positive and negative rewards periodically while also getting a big reward boost for winning a game. These ended up being adjusted individually during A2C and DQN experimentation with results discussed there.

10 DQN Experiments and Results

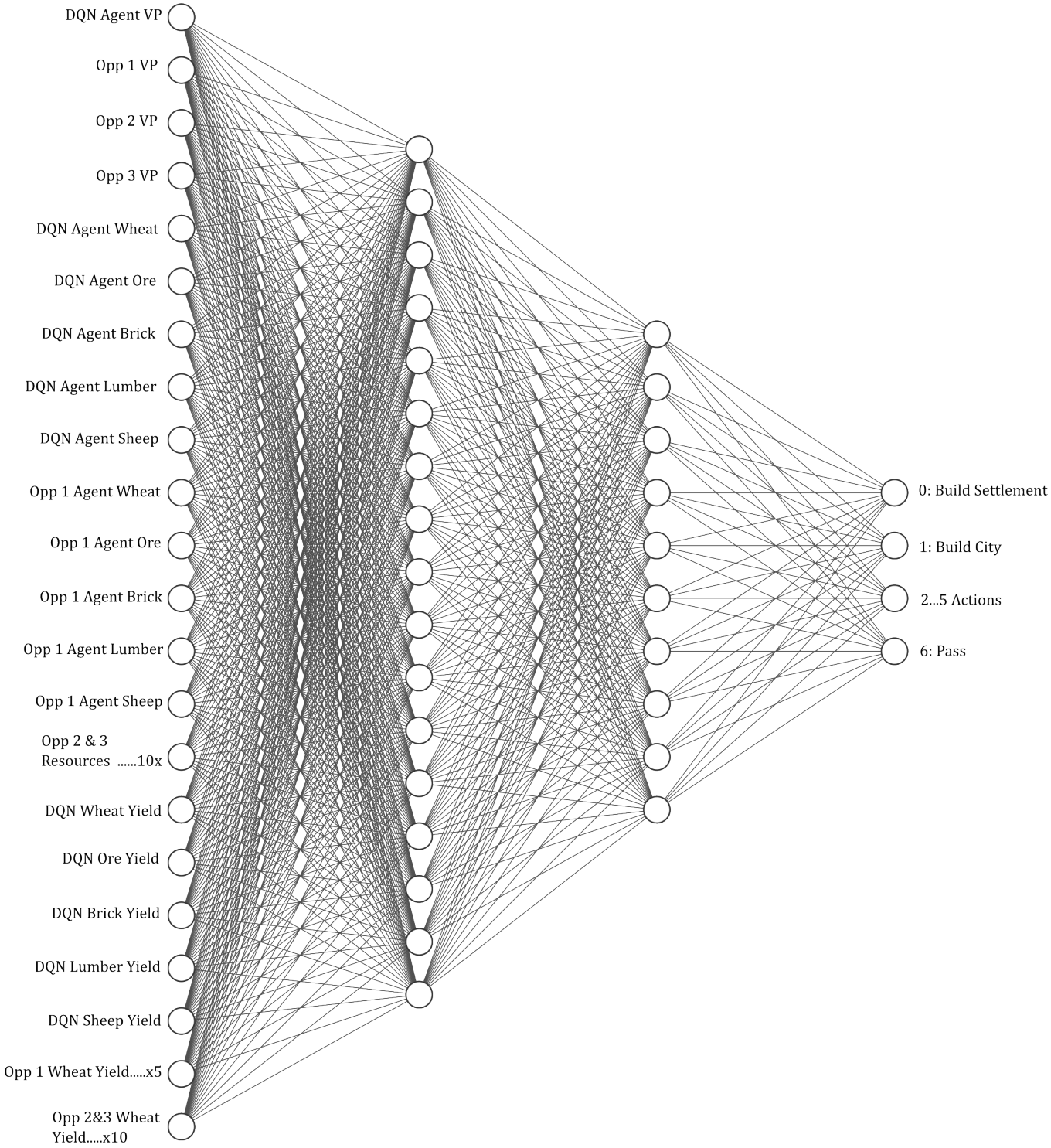
The development of the DQN agent involved adjusting many parameters during the experimentation phase as the agent simply wasn't learning well in the base environment. Starting with 44 input features my initial network was set up with 2 hidden layers with 30 and 18 input neurons respectively. When my agent was struggling to learn I added an additional 54 input features representing each intersection on the game board representing building ownership on those spaces. While theoretically this should give the agent more information to make decisions on, too many features can sometimes over-complicate the model or lead to too much correlation as certain inputs may contain the same information.

The Experience Replay hyperparameters were also adjusted a number of times to try and improve performance and the ultimate settings of a memory size of 500 transitions, a batch size of 64 transitions, and a learning rate of 0.001 was used for learning. The idea was for there to be a wide range of experiences to choose from an 500 player turns would be a number of games worth of decisions while 64 decisions often would constitute one or two games worth of choices.

The results were not promising. The agent did the best when its epsilon was nearly 1 and performed steadily worse as the epsilon decay method was called. The agent was passing turns constantly even with other actions available and eventually largely only passed. To address this a few changes were made. Beyond adjusting the feature representation, I also adjusted the reward structure outside of just the values for rewards already being distributed. Across many iterations these different rewards were selected to try and address various poor behavior by the agent:

- -5 for passing with another action available
- +5 for building a settlement
- +5 for building a city
- +2 for building a road
- disabled rewards for resources

The idea behind these adjustments was to push the agent to stop passing and play their cards. Rewards were disabled for gaining a resource in case that was being associated with the pass action and a heavy negative was applied if the agent passed with other actions available to it.


DQN Agent VP
Opp 1 VP
Opp 2 VP
Opp 3 VP
DQN Agent Wheat
DQN Agent Ore
DQN Agent Brick
DQN Agent Lumber
DQN Agent Sheep
Opp 1 Agent Wheat
Opp 1 Agent Ore
Opp 1 Agent Brick
Opp 1 Agent Lumber
Opp 1 Agent Sheep
Opp 2 & 3 Resources 10x
DQN Wheat Yield
DQN Ore Yield
DQN Brick Yield
DQN Lumber Yield
DQN Sheep Yield
Opp 1 Wheat Yield.....x5
Opp 2&3 Wheat Yield.....x10
0: Build Settlement
1: Build City
2...5 Actions
6: Pass
44 Input Neurons
30 H1 Neurons
18 H2 Neurons
7 Output Neurons

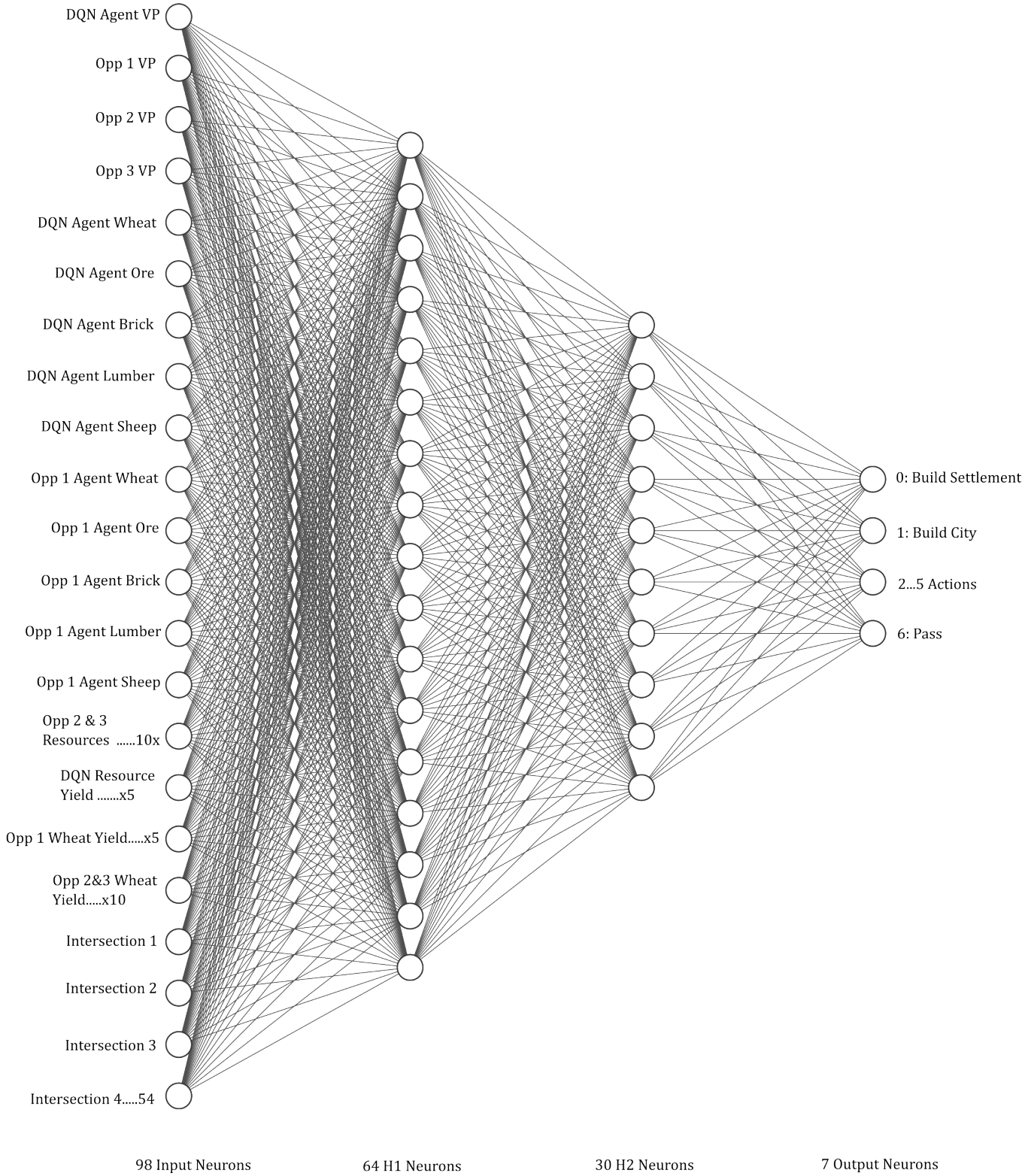

DQN Agent VP
Opp 1 VP
Opp 2 VP
Opp 3 VP
DQN Agent Wheat
DQN Agent Ore
DQN Agent Brick
DQN Agent Lumber
DQN Agent Sheep
Opp 1 Agent Wheat
Opp 1 Agent Ore
Opp 1 Agent Brick
Opp 1 Agent Lumber
Opp 1 Agent Sheep
Opp 2 & 3
Resources 10x
DQN Resource
Yield x5
Opp 1 Wheat Yield.....x5
Opp 2&3 Wheat
Yield.....x10
Intersection 1
Intersection 2
Intersection 3
Intersection 4.....54
0: Build Settlement
1: Build City
2...5 Actions
6: Pass
98 Input Neurons
64 H1 Neurons
30 H2 Neurons
7 Output Neurons

Figure 9: 44 Input Network AVG Reward

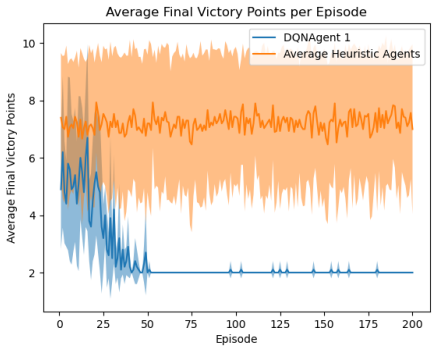


Figure 10: 44 Input Network Final VP

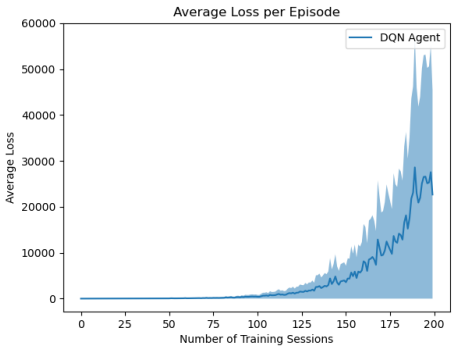


Figure 11: 44 Input Network Loss

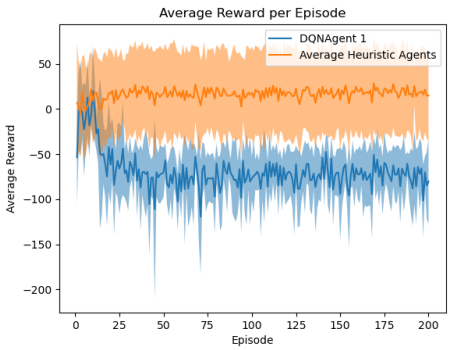


Figure 12: 98 Input Network AVG Reward

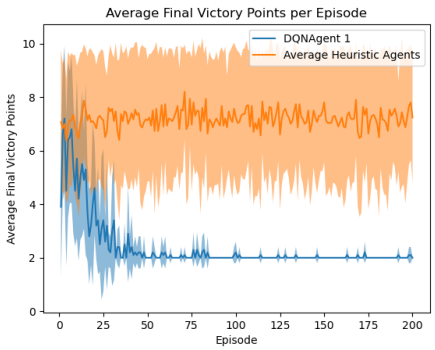


Figure 13: 98 Input Network Final VP

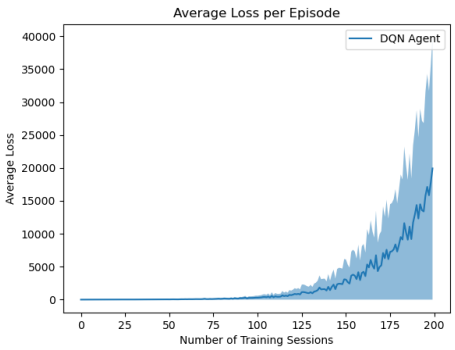


Figure 14: 98 Input Network Loss

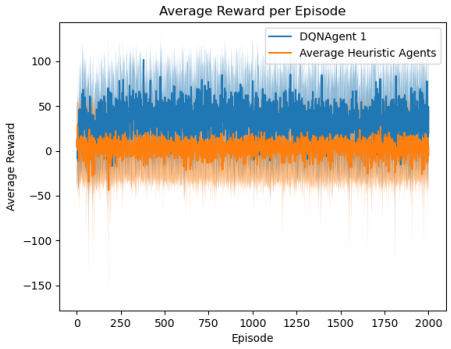


Figure 15: 98 Input Network Reward, no pass

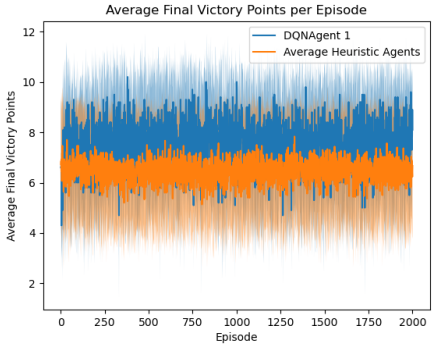


Figure 16: 98 Input Network VPs no pass

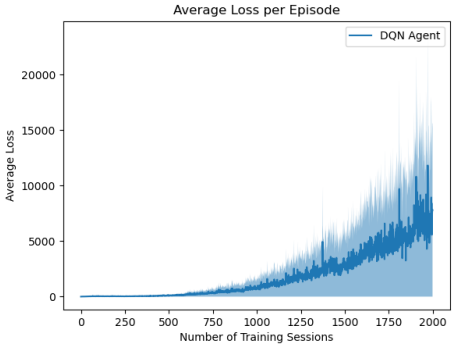


Figure 17: 98 Input Network Loss, no pass

None of it was ultimately successful. The agent was still passing nearly constantly and even though these were short 200 episode training runs it was clear the agent had stalled and was stuck in poor behavior. Another adjustment was made to implement prioritized experience replay. The idea was the passing is such a normal part of the game that the agent was maybe selecting batch sizes filled with mostly transitions filled with passing and so couldn't learn the other actions. With transitions that weren't passing being selected with higher preference than those that were a pass action. This also did little to improve the agent and it's final results with the initial action setup are a failure.

To try and force the issue the action masking was adjusted so that passing was only available if the agent had no other actions to perform. This is very limiting to the agent and isn't entirely dissimilar from how the heuristic agent would function some of the time (situations with only one action besides pass). Across 2000 episodes, the agent was shown here to select actions better than the heuristic agents policy finishing with 8VPs on average or so and higher reward indicating that there is some learning occurring. The loss still looks unreasonable with its exponential growth, however, so further work must be done to tune this approach to having success outside of this limited environment.

11 Actor Critic Experiments and Results

The Actor-Critic agent ran through several adjustments while learning on this environment. Using the main features that had been established, the agent focused on observing player victory points, resources, board yield, and resources in hand. While we felt that these features were a reasonable representation of the board state we had also included some other more complex features. Originally, we also included location data for resources as well as for player settlements. Though this information seemed informative, it also resulted in extremely slow run times. The agent needed to recalculate some of these locations every turn to accurately represent the game-state. The advantage of including this information did not seem to be significant enough to justify the run times. Due to this, we cut these features down.

Now with our features established, it seemed that the agent was running into situations where it would incorrectly favor an action that did not lead to a positive board state. In particular the agent seemed to favor passing the turn despite having other available actions. In addition to not taking game relevant actions, the agent was also being forced to discard cards often leaving it unable to take other actions moving forward. To try and change this behavior, several updates to the reward structure were made. First, when an agent took an action other than passing it was rewarded +1, but if it passed with other actions available it was rewarded -2. These rewards were given per action and were added to any gain in victory points. Unfortunately, this seemed to result in instability when approximating the policy as the delta being calculated seemed to stop converging. The additional penalties per action were removed, with the only reward between episodes being the gain in victory points obtained by the agent's actions. Despite these changes, the agent still seemed to struggle with passing the turn on occasion despite having other better actions to take. One adjustment that was tested, was only utilizing the victory point comparison feature when adjusting the action weights, but keeping all of the features when adjusting the policy weights. This seemed to allow the agent to avoid prioritizing holding too many resources during the game and it seemed to start spending them more frequently. In addition to these rewards, the agent also received a large reward for winning a game, which was composed of the total reward from the initial rewards structure.

When training against the heuristic agents there were two types of games being played. Two player and four player games. Although the agent was capable of playing against the heuristic agent one on one, the win-rate was not close enough over large numbers of games. The agent seems to perform slightly better in games with more players, and despite its fluctuations it seemed to perform similarly to the heuristic agent as seen below:

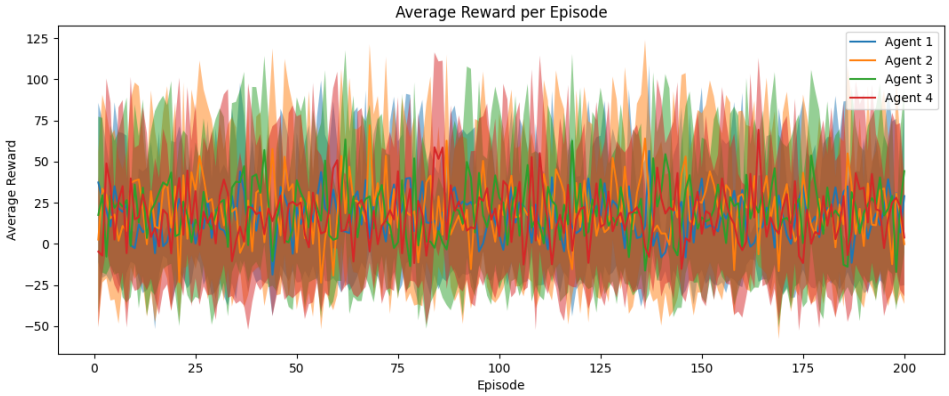


Figure 18: AC agent 1 vs Heuristic agents average rewards over 10 iterations of 200 games. Agents 2, 3, and 4 are all Heuristic agents.

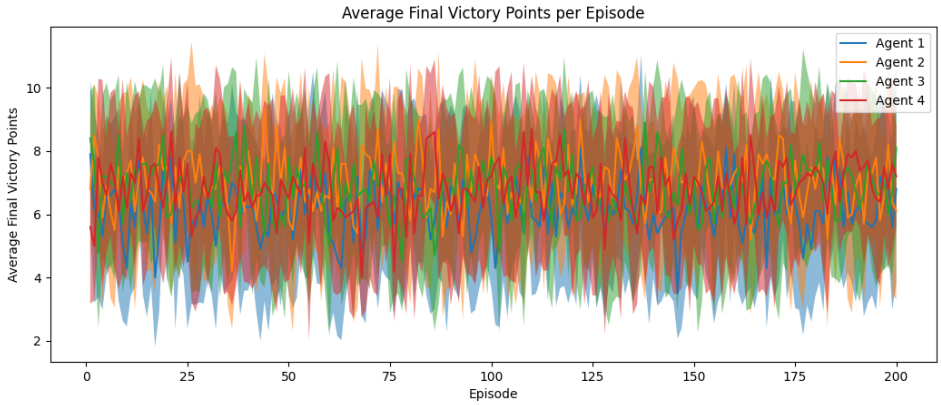


Figure 19: AC agent 1 vs Heuristic agents average final VP over 10 iterations of 200 games. Agents 2, 3, and 4 are all Heuristic agents.

Despite a large number of iterations, the agent did not show significant improvements and still seems to suffer from a fluctuating score from game to game. This lack of consistency in results suggests that there is a lot of room for improvement.

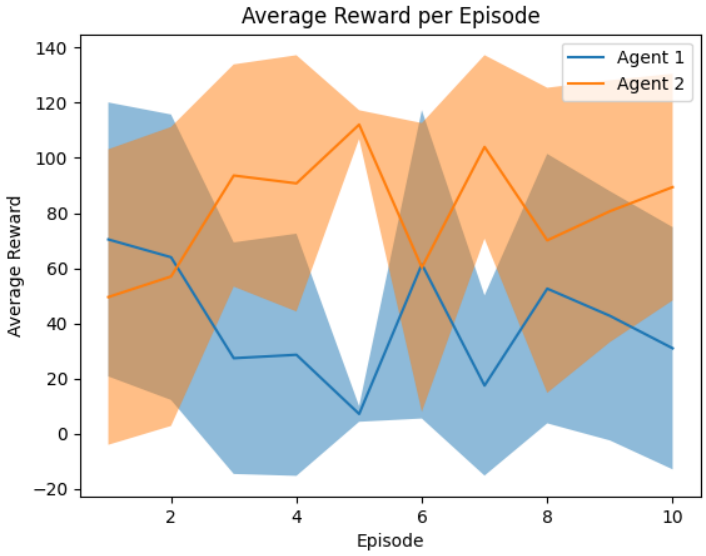


Figure 20: AC agent 1 vs one Heuristic agent average rewards over 100 games. Agent 2 is a Heuristic agent.

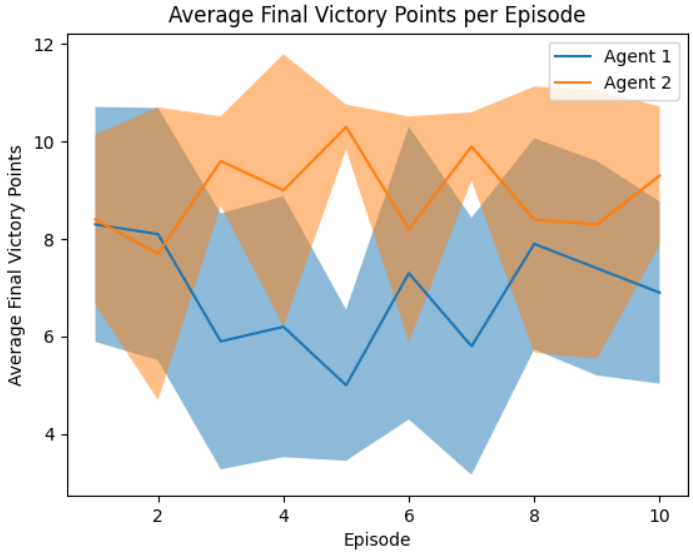


Figure 21: AC agent 1 vs one Heuristic agent average final VP over 100 games. Agent 2 is a Heuristic agent.

12 Comparing Agent Behavior

Although the DQN agent struggled to learn while passing was enabled, it did manage to perform on average better than the heuristic agents with passing disabled. In comparison this is better than how the AC agent performed, but which a massive restriction to its available actions. The AC agent was able to perform better when passing was allowed and managed an average score that was only a few points lower than the Heuristic agents. A large number of runs was used to train each agent, but over the course of this training neither of them showed significant improvements. There is a significant amount of future work that can be down to enhance the capability of each agent and to expand on their interaction with the environment.

13 Future Work

13.1 DQN

There is substantial future work to be done on the DQN agent if it is to succeed even at this more rudimentary version of Catan. As of now the agent is unable to learn the action structure and without essentially elimination passing, the agent is unable to show meaningful improvement over the heuristic agent. The rewards can still be adjusted further though I think the final structure has covered most aspects of the game. Likely the network architecture needs the most optimizing. Changing the number of hidden layers further, changing the feature representation, changing how the layers are structured, all were adjusted in experimentation, but not exhaustively so. Additionally the target network could be updating to often, though the every 10 training episode threshold was adjusted a few times with little adjustment. Lastly self-play could be implemented where the agent learns and trains against previous versions of itself. This approach is taken by some researchers and it does show some success but wasn't implemented due to time constraints.

13.2 AC

The Actor-Critic agent has a lot of potential changes that could be made to improve it significantly. Most importantly I think adjusting and testing different features and cross-multiplying some of the existing ones would help to establish more relationships between actions and states that may lead to better decision making. Some example features that may be beneficial to the analysis of the action-space might be tracking the number of resources spent during a turn, excluding ones discarded. Finding a way to track player location information without heavily impacting runtime could also lead to improvements. In addition, this would require a more complex value function to be used by the agent. As mentioned above, the concept of self-play could be beneficial to the learning of the agent, but I believe this would be more helpful once it can perform closer to the heuristic agents or better than them. At this time the AC agent does not perform well enough that it would gain benefit from self-play. There are also other variations of actor-critic that could be utilized, meaning the current one would be reworked. These variations may allow for some improvement to the ability for the agent to learn and altering the structure of the policy.

13.3 Final Agent

Ultimately the goal of this kind of work would be to create an RL agent that is capable of playing the entire game of Catan without any limitations or heuristics applied. The approach of AC as well is shown to be capable of learning the top level actions to an extent and could be extended to learning placement, resource, and trade decision types as well with more actor/critics and more time. The same could theoretically be said of DQN though the results from this experiment were not promising for DQN learning or require substantial adjustment to the underlying architecture of the agent. A fully capable RL agent is shown to be possible though and future work could connect all the disparate elements of the game into a single working agent.

14 Conclusion

Although there is room for improvement, these agents still prove that a complex game such as Catan can be broken down into smaller action-state spaces. This process could potentially allow us to create a combination of policies and agents that could learn and successfully play out the entire game, making decisions through-out the entirety of the action-state space. As we built on these agents and improve them this proof of concept will become clearer and we will be able to apply this method to play full games of Catan without any assistance from heuristic decision making. Each of our agents hit some setbacks, but through adjustments and analysis there is a path to making huge improvements.

15 Individual Responsibilities on Project

Gyde Lund was responsible for setting up the PyCatan environment as well as the training loop file the agents learned within. He also completed the heuristic agent that served as a baseline and also contained many functions the agents needed as well. Finally, he did work on the DQN agent though this did not succeed in the end. Taylor was responsible for setting up the actor-critic agent, and the features being used as well as normalizing them. We both spent time testing and adjusting the code to try and get everything working to the best of our abilities.

References

[1] Ullstein Bild. *Jubilee edition of the board game "Siedler von Catan" (settler of Catan)*. Ullstein Bild, Oct 2007.

[2] Quentin Gendre and Tomoyuki Kaneko. Playing catan with cross-dimensional neural network. *Neural Information Processing*, page 580–592, Aug 2020.

[3] Ayush Gupta. A comprehensive guide on optimizers in deep learning, Sep 2023.

[4] Chris C Kim and Aaron Y Li. [pdf] re-l catan: Evaluation of deep reinforcement learning for ..., 2021.

[5] Peter McAughan, Arvind Krishnakumar, James Hahn, and Shreeshaa Kulkarni. Qsettlers: Deep reinforcement learning for settlers of catan.

[6] Michael Pfeiffer. Reinforcement learning of strategies for settlers of catan - researchgate, 2004.

[7] Richard S. Sutton, Francis Bach, and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press Ltd, 2 edition, 2018.

[8] Jordi Torres. Deep q-network (dqn)-ii, May 2021.

[9] Gabriel Van der Kooij. Actor-critic catan: Reinforcement learning in high-strategic environments, Nov 2020.

[10] Josef Waller. Pycatan2, Nov 2021.