Online Planning Methodologies in Settlers of Catan

Martin L. Altenburg
Department of Electrical Engineering
Stanford University
Stanford, CA 94305
altenbur@stanford.edu

Sandy Li
Department of Computer Science
Stanford University
Stanford, CA 94305
sandyli@stanford.edu

Benjamin I. Rocklin
Department of Computer Science
Stanford University
Stanford, CA 94305
brocklin@cs.stanford.edu

Abstract

Settlers of Catan is a game wrought with uncertainty, owing to its randomized resource payouts, multiplayer nature, and randomized starting space. Each game plays out entirely differently, and no set policy works well for all games and opponents. We sought to create an agent that could perform effectively in a game of Settlers. We followed an online planning approach and achieved remarkable results against baseline opponents.

1 Problem

Machine learning and game theory are two concepts that have been intrinsically intertwined since AI's inception. Owing to the fact that games are relatively easy to model and rely on little data, they provide great insights into AI's capabilities and limitations. However, many of these games are not deterministic and rely on multiple agents, creating additional uncertainty that standard supervised algorithms are unable to find consistent solutions to. One such example of a complex game is Settlers of Catan. Owing to the various sources of uncertainty in Settlers, we decided to create an agent utilizing online planning to choose strategic actions.

At every stage of the game, there are several different decisions a player must make. They must decide what, if anything, that they will build, and where they intend to build it. Most of the game's infrastructure relies on serious resource investment that can be foiled by another player at any moment. Players also have access to "development cards" that can change the course of the game when played at the right time or have no impact whatsoever if played poorly. They may also make trades at any point with other players, which can very quickly lead to power plays or set other players up for failure due to poor rolls.

Compounding the number of difficult decisions is the amount of uncertainty. In Settlers, resources have tiles with numbers above them that provide resource cards adjacent players can use if they roll the number on the tile using dice. Therefore, a great deal of uncertainty stems simply from which numbers are rolled by each player, as a single roll can vastly change the course of the game. Because players choose to roll the dice as an action, this is a form of outcome uncertainty. Similarly, all other players on the board are adversarial and may choose to cut other players off and play hostile development cards to others' detriment. As a consequence, there is much interaction uncertainty. While all players may see the resources others gain, a human player cannot possibly keep track of all cards gained and played due to the rapid nature of the game, resulting in state uncertainty. Finally, the locations of resources and tiles are not known until the game is being played, meaning that the agent cannot pick a consistently good set of locations that will work for all games and creating model uncertainty.

Given the vast array of actions, sources of uncertainty, and our project's limited timespan, we decided to simplify Settlers of Catan's mechanics. To do so, we made several simplifying assumptions. We first limited the game to two players. Additionally, we decided to eliminate trading between players, placing more emphasis on a player's starting locations; however, we did still allow 4 to 1 resource trading into the bank to prevent agents from getting stuck in initial states impossible to win from. Furthermore, we eventually eliminated the usage of development cards, as we found these extremely difficult to model and had difficulties making them work properly in our Catan simulator (more on this in Section 3). Thus, the main goal of our bot thus became making smart building and resource management decisions, and our bot performed outstandingly in this regard.

2 Related Works

To begin, we investigated related work in automated Catan agents to draw inspiration. While there aren't many agents that work particularly well at all aspects of the game, we found several interesting reinforcement learning approaches towards solving the problem of optimal training and heuristic approaches aimed towards building and resource management.

The first approach that we discovered was a Deep Q-Learning agent by McAughan et al. dubbed "QSettlers" [1]. This approach is very similar to Q-learning; however, rather than building an explicit table of state-action pairs, deep Q-learning networks, or DQN, approximate Q-values for new states by comparing them to known states. This allows it to perform more strongly on problems that have extremely large state spaces; given the high number of variables that compose a Catan state, this performs much better than Q-learning.

The QSettlers team initially intended to implement two DQNs: one for building and one for trade negotiations. Ultimately, the team was unable to implement a DQN for building due to time and infrastructure constraints, as training a network over all board features and buildings is an extremely expensive task. They did, however, create a prototype of what such a network might look like, though it is limited to predicting the value of settling on a given tile, resulting in it not being a complete solution for deciding between building cities, settlements, and roads and development card usage. They did, however, create another DQN that, given information about a trade, yields the utilities of accepting or rejecting the given trade. This allows it to determine what trades are more favorable for it or an opponent, allowing it to decide whether or not it should propose or accept a given trade with an opponent. For building decisions, QSettlers used JSettlers agent logic (discussed more below). The team noted that the agent's overall performance tended to be hindered by poor building decisions and board control, while its higher trading ability resulted in consistent second-place finishes, suggesting that a building network might be able to yield even higher performance than the base JSettlers agent.

Additionally, we read a very similar approach for using Deep Reinforcement Learning network for trading decisions. Namely, Xenou et al. [2] use JSettler heuristics for building decisions and a series of parallel LSTMs and Q-learning in order to make trading decisions. This is similar in the goals of the prior approach, but differs in that rather than using a DQN, it uses RNNs. Additionally, unlike the prior approach, this approach trains itself in real time while playing. This approach wins roughly half the games it plays, but it still relies on jSettlers for building.

jSettlers agents are a bot implementation included with a Java Settlers of Catan simulator [3]. While there is no formal paper written about their exact implementation, via the code we may see that they make decisions using heuristics manually-tuned towards Settlers of Catan, where more cards and more structures are favorable but victory points are prioritized. This approach tends to work well, but as we see from the above two approaches, it is outclassed when even a fraction of its logic is replaced by reinforcement learning algorithms. Moreover, the heuristics require domain knowledge, whereas other approaches might be game-agnostic and thus more generalizable.

Consequently, the prior successes of other authors in trading decisions and domain-reliance in building decisions motivated us to create an agent that performs well in building and resource management.

3 Approach

We began our approach by finding a suitable simulator to run Catan on. We wanted to create the project in Python, so we began by trying several different existing Catan simulators and attempted to complete a human vs. human round on these. Despite the large number of these libraries, many would either not launch, crash shortly after game start, or did not have any exposed code that would allow us to create a visualization of the game board or allow the bot to fetch the game state. We eventually found PyCatan [4], a mostly functional simulator that allowed us to finish a game of Catan. However, as we rapidly discovered, the simulator had a number of bugs. Roughly 20% of the time, the board generated by PyCatan would be invalid and create out-of-bounds crashes farther into the program. The dice-rolling method would not work properly and, rather than returning a number in the range 2-12 as a sum of two random 6-sided dice should, returned a number from 0-12. Additionally, adding the yields for dice rolls often caused out-of-bounds crashes due to the function for fetching adjacent hexes supplying invalid hexes. These are a small glimpse of many bugs that we ran into using the simulator, and these significantly impeded our progress.

Eventually, we were able to build agents despite these bugs and even remedy these issues locally over time by reproducing these methods in a bug-free manner ourselves. We began by implementing a human agent, which takes user input to make decisions, alongside a baseline random agent. We then attempted unsuccessfully to create a reinforcement learning agent and later successfully built an online planning agent. We limit all agents to the same actions and knowledge as a human agent in the interest of fairness; this also means that our agent did not factor the cards of other players into its decision making, despite the small advantage this may have given it.

In parallel, we also developed a tool to visualize a PyCatan instance in parallel with implementing our agents. This tool allowed us to debug our agents much easier and view the decisions that they made more concretely than pure console output. Section 4 provides examples of this.

3.1 Baseline

For our baseline, we created a random agent. This baseline was our main target-to-beat for our simulations later on. The agent begins by choosing initial settlement and road locations at random. Quite often, this created incredibly difficult starting locations that no skilled player would be able to succeed with (see Section 4 for details). For the rest of play, our random agent would check if it had rolled. Given that we have no development cards in our simplified model, any agent may only roll at this stage of the game. Otherwise, we build an actions array for the building phase, as after we roll we may build given that trading is barred. The possible actions in a given post-roll state are as follows:

- An agent may choose to end their turn after rolling at any time.
- An agent may build a road adjacent to any structure or road owned by the same agent in an empty location.
- An agent may build a settlement if it is adjacent to a road owned by the agent and no other settlements or cities are directly adjacent.
- An agent may build a city over an existing settlement owned by the agent.
- An agent may trade any four resources in for a different resource card from the bank. We ignore harbors for the interest of simplicity.

We append all of the above actions to the action array if it is valid. For building, we append all possible sites where the agent can build to the array if the agent possesses the necessary resources. For trading resources, we append all possible resource card trades the agent can make given that they have four cards of at least one resource. We then pick an item at random from this list as the action-to-take.

3.2 Reinforcement Learning

We also tried to create an agent via Q-learning. However, we ran into several problems that prevented us from being able to do so. We chose to represent an agent's state as whether or not the agent had rolled, the number of each resource card the agent has, the building in each possible slot on the board (or lack thereof), the presence of a road in each possible location, the agent's victory points, and the resource for each hex on the board, and the number corresponding to each hex dictating resource payouts. This resulted in a prohibitively large state space for Q-learning. Even if we cut the hex information, it would be nearly impossible to find an optimal action for each possible arrangement of player buildings and roads on the board, and if we were to omit this, it would be impossible to represent the possible actions in a state, as this information is needed to determine what actions may be taken in a given state. Consequently, after much time coding this, we eventually had to give up on this approach, as we were nearing the project deadline and had no functional implementation to show for it.

3.3 Online Planning

Noting the above problem with the large state space, we considered online planning as our secondary approach. We were drawn towards this as the reachable state space for a player is much smaller than the total state space, as a player's existing settlements and resources largely limit which states it can transition to after any action. We encoded our state space identically to the above and decided to do simple Forward Search to depth 5 to create a policy shown below.

$$\pi(s) = \arg\max_{action} (R(s,a) * (0.75 * \sum_{s'} T(s'|s,a) * U_{\theta'}(s'))$$

Our transition function probabilities are agnostic to our previous states given that they are decided by the result of our two dice being rolled between rounds. Additionally, s' could only differ from s by the number of resources accrued during an optimal dice roll. We tried several different depths and found that the depth of 5 offered the best trade off between speed and strong decision-making. We hoped to add on additional improvements but did not have time to do so; these will be discussed in section 5. Given that we had not learned anything about multiagent learning at the time of the project, we opted to ignore the other agent entirely in our Forward Search with the exception of the locations of structures that they had already built; this had some slight repercussions later on that will be discussed in section 4.

4 Results

Overall, our online planning agent performed admirably. Both agents were given random starting locations and had to make strategic building decisions to expand to better hexes and win the game. Figure 4 shows a sample output of our Catan visualization tool in a game between a random agent and our online planning agent. In all examples below, our online planning agent is assigned the color red and our random agent is assigned the color blue. The color of the hexes denote their associated resource (light red for brick, green for wood, chartreuse for sheep, gold for wheat, gray for ore, and white for the desert/wasteland/no resource). The number on each hex denotes its associated dice roll to drop resources, where the desert has none as it has no associated resource. Squares denote settlements and triangles denote cities, and colored lines between hex vertices represent roads.

Initially, we evaluated our agent without the ability to trade resources into the bank for another resource. This severely limited an agent's ability to play if it started without all resources, as it might not be able to build any roads or settlements if it did not start with wood and brick. Consequently, our initial evaluations were skewed due to some starts being better than others; however, we found that even in this scenario, given two equal starting positions, our online planning agent outperformed the baseline. A large reason for this is that the baseline, due to picking a random action, would often greedily build roads resulting in it not having the resources to build new settlements for points. Figure 4 showcases this behavior. We see that Red and Blue have equal starting positions, where Blue, the baseline, has a slight advantage. Despite this, Red has a point advantage, and Blue has a sprawl of roads on the right side of the map that lead nowhere. The reason for this is that the resources to build a settlement are a superset of those needed to build a road, and given that more road locations are available to build on than settlement locations in an average game, a random agent will be more likely to choose a road location at random given that all possible actions are weighted equally. We see that Red, our online planning agent, did not build any wasteful roads with the exception of a few towards the top and one at the bottom. Initially, we encountered an issue where our reward function had our agent greedily adding settlements whenever possible - sometimes in sub-optimal locations. However, we implemented a small reward for building a road which helped us balance exploration

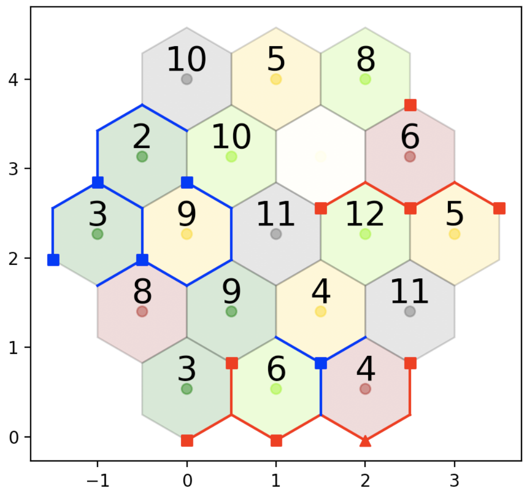

Figure 1: Sample Catan view.

and development. In the figure below, the bottom road is an instance of where our agent does not consider the opposing agent's ability to expand, as this road was quickly cut off at random by blue, something which may have been caught had we used a form of multiagent reasoning in our online planning search.

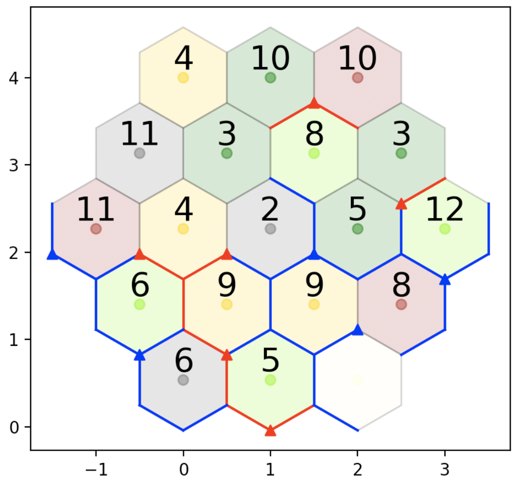

Figure 2: Wasteful road-building by the baseline

Eventually, we allowed agents to trade resources to the bank to avoid getting stuck for a four-card price as in the original game. After allowing this behavior, our online planning approach rapidly outperformed the baseline implementation. Figure 4 provides an example where the baseline is given a much larger advantage in starting position over online planning agent Red, who starts against the desert, on the edge of the board, and without three resource types. Despite this, Red still managed to win, as it traded resources to the bank to obtain wood and eventually expand across the board; it also uses its starting advantage of wheat and ore to build quick cities, a strategy many experienced Catan players use and newer players fail to take advantage of. Thus, our game agnostic approach managed to gain an insight into Catan many humans fail to achieve after reading the rulebook.

Overall, we noted our online planning agent consistently outperforming our baseline agent after adding this change to the game. In 50 separate games after this change, it managed to win 38, a win rate of 76%. Its resource management skills overall trended towards building infrastructure in key locations and expanding to strong locations to win the game, accomplishing our primary goal.

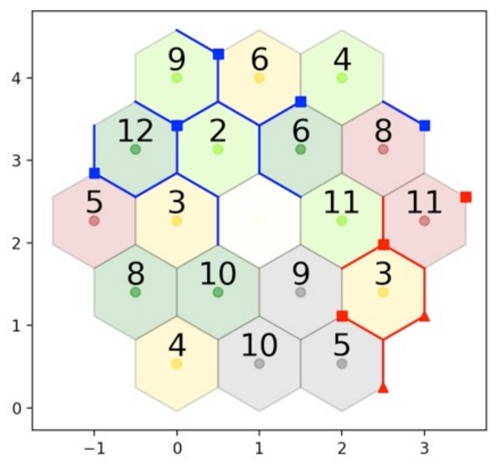

Figure 3: Online planning win despite severe disadvantage

5 Conclusion

Overall, our agent appeared to perform quite well. It managed to make strategic decisions regarding what to build and where to build, and it avoided over-spending and exceeded our expectations by trading resources into the bank to obtain other key resources that it would be otherwise unable to get through a dice roll.

Given more time, we had hoped to expand its behavior to use development cards given the lack of other projects in this domain. However, given time constraints, we were unable to do so. We had also hoped to add additional improvements to our model. One such improvement that we weren't quite able to finish was to modify our forward search to use Branch and Bound to estimate upper and lower utilities of different states; however, we did not have time to pursue this avenue and found that our existing implementation performed well-enough on this front. We also hoped to use online planning for initial settlement locations rather than picking one randomly, but we unfortunately did not have a chance to do this either; we did find it quite interesting that it still outperformed the baseline in many disadvantageous starting positions, however, and this effect would not have manifested had we made the change above.

Our code is in an open-source repository that may be used freely [5]. We hope to continue our work on this bot and perhaps hybridize it with one of the RL-based trading strategies above to build a strong agent that can outperform the current jSettlers state-of-the-art.

Member Contributions

- Martin and Sandy implemented the Online Planning bot, ran simulations, assisted with general bug-fixing, and created controller functions for state encoding.
- Ben built the logic for running a game of Catan, built the baseline random bot, created the Catan board visualizer, provided PyCatan bug-fixes and alternate methods, and constructed helper functions for extracting the state from a PyCatan instance.
- All three worked on high-level strategy for agent methodologies such as reinforcement learning and online planning.

References

[1] Peter, McAughan, Arvind Krishnakumar, James Hahn, Shreeshaa Kulkarni. QSettlers: Deep Reinforcement Learning for Settlers of Catan (2019), https://akrishna77.github.io/QSettlers/

[2] Konstantia Xenou, Georgios Chalkiadakis, Stergos Afantenos. Deep Reinforcement Learning in Strategic Board Game Environments. 16th European Conference on Multi-Agent Systems (EUMAS 2018), Dec 2018, Bergen, Norway. pp.233-248. ffhal-02124411f

[3] JSettlers. https://nand.net/jsettlers/

[4] Josef Waller. https://github.com/josefwaller/PyCatan/tree/master/pycatan

[5] https://github.com/BenRocklin/CatanBot