SETTLING THE UNKNOWN: DECISION-MAKING IN THE UNCERTAIN WORLD OF CATAN

Pauline Arnoud
Department of Computer Science
Stanford University
Stanford, CA 94301
parnoud@stanford.edu

Arjun Jain
Department of Computer Science
Stanford University
Stanford, CA 94301
arjunj@stanford.edu

Rishi Verma
Department of Computer Science
Stanford University
Stanford, CA 94301
rishirv@stanford.edu

ABSTRACT

The Settlers of Catan is a popular multiplayer strategic board game. Effective play requires developing long-term strategies and evaluating complex, multi-phased gameplay. This paper proposes Deep Q-Network (DQN) and Monte-Carlo Tree Search (MCTS) methods for an AI agent to play Catan. When benchmarked against a random player, we demonstrate the effectiveness of MCTS and the difficulties of training DQN.

1 Introduction

The Settlers of Catan, also known simply as Catan, is a classic multiplayer strategy board game that has captivated players worldwide since its creation in 1995. Designed by Klaus Teuber, the game immerses participants in a world of settlement-building, resource management, and strategic trading, with the ultimate goal of accumulating victory points. Each player must make decisions under uncertainty, with unpredictable dice rolls determining resource acquisition and limited information on opponents' strategies. Catan compels players to make strategic choices amidst incomplete knowledge and uncertain outcomes. The goal of our project is to create an intelligent AI Catan agent that will navigate the uncertainty inherent in the game. We benchmark our results against a single opponent with random gameplay. This paper explores the effectiveness of Deep Q-Learning (DQL) and Monte Carlo Tree Search (MCTS) for playing Catan.

2 Settlers of Catan: The Board Game

2.1 Overview

Settlers of Catan is a 2-4 player board game set on the fictional island of Catan. Players aim to establish settlements, gather resources, and expand their territories to earn victory points. The player who first reaches 10 victory points wins the game. A game board consists of 19 connected hexagonal tiles, each one numbered from 2-12 and associated with one of 5 resources (wood, brick, wheat, ore, and wool). Players can place settlements at the intersections of these tiles and roads on the edges of these tiles. At the start of the game, the arrangement of the tiles is randomized.

Each player can place two settlements and two roads at the start of the game. The order of placement is determined by rolling a dice (player with highest dice roll goes first). On each turn, a player rolls two six-sided dice. The sum of the rolls determines which hex produce resources. If the player has a building on one of the hex's corners, they collect resources corresponding to the hex (one resource for settlement, two resources for city). After collecting resources, players can buy and place settlements and roads, trade resources with other players, and buy/play development cards. Development cards provides special abilities and advantages and additionally give one victory point when played. Players can also earn victory points by building settlements (1 point) or cities (2 points), among other mechanics.

2.2 Uncertainty in Catan

There are several sources of uncertainty in Settlers of Catan that impact player's decision making strategies. These include:

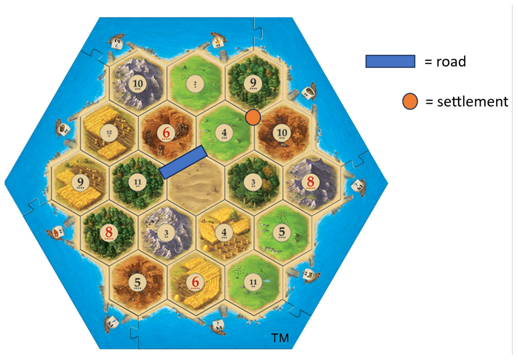

Figure 1: An example of an initial board setup for Catan, with a road built by Player Blue and a settlement built by Player Orange.

- Initial setup: The random board layout at the start of each game and the random order in which players place initial roads and settlements affects resource availability throughout the game.
- Resource gains: Each turn, dice rolls determine which tiles produce resources for a player, impacting resource collection and what a player is able to purchase.
- Development Card Draws: The development card deck is shuffled and placed face-down. Players don't know the specific cards they'll draw, introducing uncertainty about the reward for purchasing a card.
- Opponent Strategies: Predicting opponents' strategies and intentions can be uncertain, influencing decisions regarding settlement placements, road building, resource prioritization, or trade and negotiations.

3 Prior Work

In the realm of applying AI to Settlers of Catan, the leading model is "JSettlers", a Java-based model that uses a comprehensive game model combined with hardcoded rules and heuristics to make decisions. Despite various attempts, no other AI methodologies have surpassed JSettlers' efficacy [4]. Beyond "JSettlers," Deep Q-Learning (DQN) has gained prominence as a favored model for AI Catan agents [2, 7]. DQN extends Q-Learning by utilizing a neural network to approximate Q-values, allowing it to handle larger state spaces inherent to complex games like Catan. However, the DQN-based Catan agents focus only a subset of actions (trading) because of the substantial complexities and challenges associated with training a comprehensive DQN model to cover all facets of the game simultaneously.

Applying Monte Carlo Tree Search (MCTS) to Settlers of Catan is much less explored than applying Deep Q-Learning to this game. Szita et al. used MCTS in a perfect-information variation of the game that removes elements of imperfect information (e.g., buying of development cards) and gets rid of trading with other players [6]. When playing against a "JSettlers" agent, this MCTS agent achieved a 27% win rate with 1000 simulations and 49% win rate with 10000 simulations.

4 Methods

4.1 Simulator

To evaluate our agent's performance, we need a controlled Catan simulator meeting specific criteria: it must be implemented in Python, accurately replicate gameplay, seamlessly integrate with an agent for action and reward interactions, feature graphical representation, and sustain stable simulation across multiple games without crashing. After testing numerous Catan simulators against these criteria, we chose the "Catanatron" simulator, which has the advantage of representing the full set of states and actions possible with no simplifying assumptions [1]. To evaluate our models, we trained against a random baseline. This agent chooses any of the available actions at the each turn uniformly at random. We further tested a HeuristicPlayer, which built a settlement if the action was available, and otherwise chose randomly. This player beat random 55% of the time.

4.2 Deep Q-Learning

Our first approach leveraged Deep Q-Learning. Given the high-dimensional nature of Catan and complex gameplay with multiple phases, explicitly representing the transition function is difficult. Instead, we choose a model-free reinforcement learning method. Q-learning is a standard approach that learns the action-value function Q by incrementally updating the estimate $Q(s, a)$ as we explore the space:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left(R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right)$$

Here, s' is the next state after taking action a from state s, α is our learning rate, and γ is the discount factor. However, maintaining this lookup table is intractable given the billions of possible states that exist in Catan. Deep Q-learning, introduced in [3], parametrizes the action-value function using a neural network. We train over a limited number of games and explores a small subset of the possible state space, but the neural network will be able to generalize to handle unseen states. This approach has been successfully used for many games with high-dimensional state spaces, and it was first developed for the Atari 2600 game suite. The original paper noted its success in games with long-term strategies and delayed rewards, like Breakout. Catan has similar delayed rewards, as we only receive a reward for winning or losing the game on the very last action.

Our first step was vectorizing the state space. Our representation drastically simplified the state to only consider information we considered most valuable:

- 1 dimension: The player's current victory points
- 5 dimensions: The number of each of the 5 resources that the agent possesses
- 54 dimensions: For each of the 54 intersections, whether there is a settlement there.
- 54 dimensions: For each of the 54 intersections, whether there is a city there.
- 72 dimensions: For each of the 72 edges, whether there is a road there.

This resulted in a 186-dimensional input. There are 289 possible actions in Catan, only a subset of which are valid on each turn. Our DQN model was a multilayer neural network, which input a state $s \in \mathbb{R}^{186}$ and output a vector $q \in \mathbb{R}^{289}$, where q_i is the prediction for $Q(s, a_i)$. This predicts the value of invalid actions - however, we only consider valid actions when making a move, and do not train or consider the predictions for invalid actions.

The DQN approach has several nuances, which are briefly described here, but are explained in greater detail in [3]. Note that when training over a game, the observations are highly correlated, which can pose difficulties when training a neural network. By a mechanism known as *experience replay*, we store a buffer of memory of past states and actions, and then randomly sample batches of training data, reducing the correlations between training points. Further, Q-learning incrementally updates Q, requiring the network to learn a constantly moving target. To provide stability while training, we use two networks. We freeze one neural network as our baseline (the *target net* which predicts $Q(s', a')$) while training our predictions $Q(s, a)$ using a *policy net*, and then update the target net using the policy net once per epoch.

4.3 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) is a decision-making algorithm used for strategic games and scenarios involving incomplete information and uncertain outcomes. This approach is characterized by four key steps: Selection, Expansion, Simulation (or playout), and Backpropagation. MCTS iteratively builds a search tree based on the game state and uses random simulations to evaluate potential moves. The balance between exploration of new moves and exploitation of known successful strategies is a critical aspect of MCTS, making it an ideal choice for complex, strategy-based games like Catan.

4.3.1 Assumptions

In contrast to DQN, MCTS can handle large, complex state spaces without explicit simplification. We do however assume a fully observable and deterministic environment for simulating future states based on known actions, as well as consistent transition dynamics for reliable predictions during simulation and backpropagation. Unlike DQL, MCTS explicitly explores the state space through tree traversal, assuming that sampled future states during simulations accurately represent the true underlying environment dynamics.

4.3.2 Our implementation

Our implementation follows the 4 classic MCTS phases. Here are some of the decisions we made for each of them:

1. **Selection**: In this step, nodes are selected based on a balance of exploration and exploitation. We used the Upper Confidence Bound (UCB) to guide this balance. The UCB formula, defined in equation 9.1 of the textbook as
$$\text{UCB} = \overline{X}_j + \text{EXP\_C}\sqrt{\frac{2\ln N}{n_j}}$$
combines the win rate of the node (exploitation) with a term dependent on the exploration parameter (`EXP_C`) and the number of visits to the node (exploration). This balance ensures that the algorithm adequately explores less-visited yet potentially rewarding paths while also exploiting known advantageous strategies.
2. **Expansion**: The next step is to generate new children for potential actions, expanding the search tree. Due to Catan's complexity and the variety of possible actions at every point, we introduced a pruning mechanism to reduce the set of possible actions based on game specific heuristics, such as prioritizing resource acquisition or settlement expansion, using the game heuristics provided to us by the Catanatron simulator.
3. **Simulation**: In this phase, a rollout from the current state to a game's conclusion is simulated to estimate the potential of a move. The number of simulations is determined by the `num_simulations` constant which we had to finetune to maximize the win rate.
4. **Backpropagation**: Finally, we backpropagate the results of the simulations through the tree, updating the statistics of nodes along the path taken during the selection phase. Each node's win count and visit count are updated based on the simulation's outcome.

To make all of these operations possible, we implemented a StateNode class, drawing inspiration from the Catanatron sample code on their GitHub [1]. Each instance of the class represents a possible state in the game, encapsulating information such as the current player's color, the game state, and the tree structure including parent and child nodes.

4.3.3 Hyperparameter Tuning

To fine-tune our two hyperparameters, `num_simulations` and `EXP_C`, we followed the hyperparameter tuning strategies proposed in [5]. To reduce the computation cost, we parallelized the simulation of multiple games at once across all available cores.

For the number of simulations (`num_simulations`), we adopted a methodical and iterative testing approach to balance decision-making quality with computational efficiency. Guided by the paper's recommendations, we explored varying simulation counts: on the lower end, for scenarios with limited computational resources or under time constraints, we scaled down to as few as 10-20 simulations. Conversely, our upper range extended to 100-150 simulations. While a higher count of 500-1000 simulations is often suggested for complex games like Catan to fully capture their dynamics, we found this range impractical due to computational resource contraints.

While tuning the exploration parameter `EXP_C`, we tested different exploration-exploitation balances. We experimented with lower values between 0.5 and 1 to assess if emphasizing exploitation, or favoring known successful moves, would improve performance in scenarios suited to conservative strategies. Conversely, in scenarios requiring innovative or unpredictable strategies, or where the game presented a wide array of potential strategic paths, we tested higher values of `EXP_C` going up to 2. These higher values were intended to encourage the agent to explore less explored paths, potentially uncovering new and better strategies. We tested these adjustments in small increments of 0.1, following the recommendations from the referenced paper.

5 Results

5.1 Deep Q-Learning

We trained our DQN model over 500 games using an ϵ-greedy exploration strategy against a random player. We attempted to vary multiple hyperparameters, including the learning rate, model architecture, and target net update rate, but our model consistently performed worse than random. We observed a large amount of oscillation in the performance of our model, as seen in Figure 2. This explained by an incredibly large amount of variance in the performance of our model, rather than our model truly learning the action-value function of Catan.

Encouraged by the performance of the heuristic player, we tried to “jump-start" our model by providing small, positive rewards for known good actions, like building cities and settlements, while maintaining a large positive or negative reward for winning or losing, respectively. However, this did not yield any improvements. This can be explained by the incredibly large state and action spaces for Catan. Even over 500 games, we are only able to visit a small subset of the possible states, and it may not even be able to learn which intersections connect to which edges from the input. The

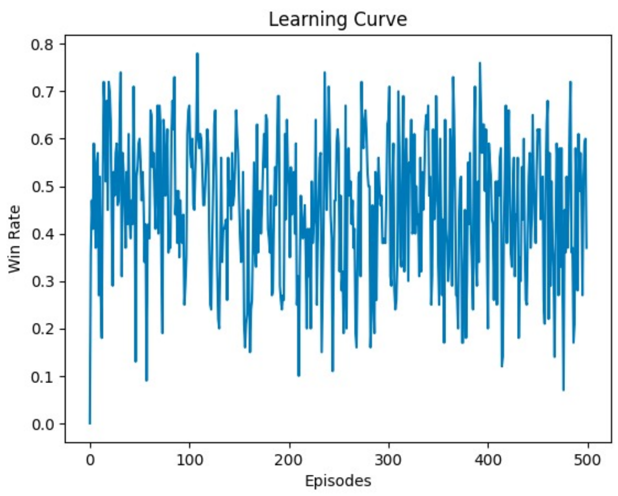

Figure 2: Win rate of a DQN model per epoch

model is further hampered by the fact that the majority of actions are illegal for any given state, making training and backpropogation difficult. Catan has numerous phases of gameplay, including initial set-up decisions, trading, building, and moving the robber, which may be too complex for a single model to learn. Given our computational constraints, we turned to an online planning approach.

5.2 Monte Carlo Tree Search

To evaluate the efficiency of our MCTS agent, we conducted a series of 100 games. The choice of this sample size was a compromise between computational feasibility and the need for a sufficient number of games to ensure statistical significance. 100 games allow for a diversity of scenarios and outcomes to be observed, providing a reasonable approximation of the agent's performance while keeping the computational time within manageable limits. This approach allowed us to have a controlled environment to assess our agent's performance and to fine-tune the key hyperparameters, namely the exploration parameter (`EXP_C`) and the number of simulations per move (`num_simulations`), with the objective of maximizing the win rate of the MCTS agent defined as

$$\text{win rate } = \frac{\text{number of wins by MCTS agent}}{\text{total number of games}}$$

The highest win rate achieved by our MCTS agent in these tests was 0.92, with `EXP_C` set to $\sqrt{2}$ and the number of simulations set to 50.

5.2.1 Impact of the Exploration Parameter

The results of the simulations are presented in Figure 3, which illustrates the win rate evolution as a function of the `EXP_C` value. The simulations were executed with `EXP_C` values ranging from 0.5 to 2.0, incremented by 0.1, while keeping the number of simulations constant at 50 per move.

Across the tested range, the win rates did not exhibit a consistent trend that would suggest a particular `EXP_C` value outperforms others significantly. The win rates fluctuated within a relatively narrow band, with no clear pattern of increase or decrease correlating with the `EXP_C` values. This suggests that, within the range of `EXP_C` values tested, the performance of our MCTS agent is relatively robust to changes in the exploration parameter. Thus, we decided to set `EXP_C` to $\sqrt{2}$, aligning with theoretical recommendations for a balanced exploration-exploitation trade-off.

5.2.2 Impact of the Number of Simulations

The performance of our MCTS agent was also evaluated against varying numbers of simulations per move to determine the optimal setting for maximizing the win rate. Figure 4 displays the relationship between the number of simulations and both the win rate and runtime of the agent. The simulations were executed with numbers of simulations set to [1, 15, 50, 100, 150], keeping other parameters constant.

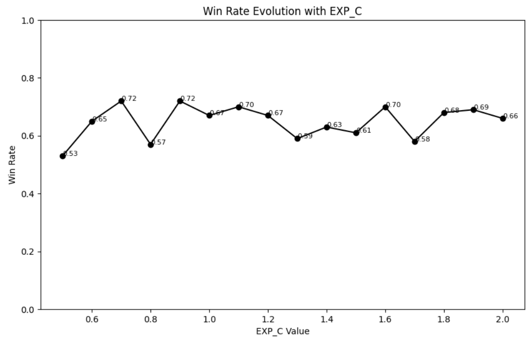

Figure 3: Win Rate Evolution with EXP_C values.

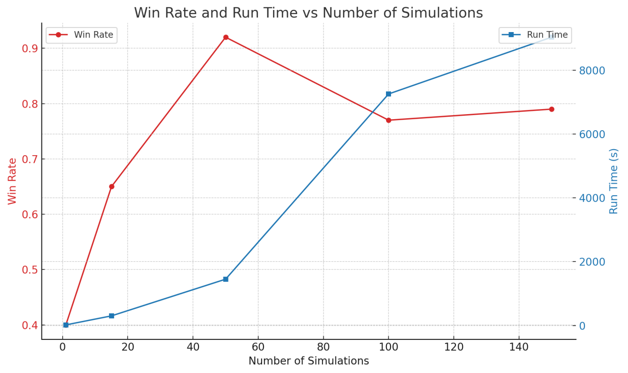

Figure 4: Impact of Number of Simulations on Win Rate and Runtime.

The data revealed a notable peak in performance at 50 simulations per move, where the win rate reached a maximum of 0.92. Interestingly, beyond this point, the win rate did not continue to increase with more simulations; rather, it showed a slight decrease. Additionally, the runtime increased substantially with more simulations, with the maximum runtime at 150 simulations being approximately 2 hours, 30 minutes, and 40 seconds.

Given these results, we selected 50 as the optimal number of simulations. This number not only provided the highest win rate but also kept the runtime within reasonable limits, ensuring a balance between computational efficiency and decision-making accuracy. This peak in performance aligns well with the expectations set by the theoretical framework, which posits that beyond a certain point, additional simulations yield diminishing returns in terms of win rate improvement.

However, it is important to contextualize these results within the limitations of our simulation approach. The win rate peak observed at 50 simulations per move, while being the most favorable outcome in our observations, is derived from a relatively small sample size of 100 games. There is a high amount of variance from our random agents, and it possible that agent was simply “lucky."

6 Conclusion and Future Work

In this paper, we proposed DQN and MCTS based models for playing Catan and benchmarked them against a random agent. The MCTS agent consistently beat random, peaking at a 92% win rate. Given additional time, there are a number of possible future directions for this work. We can attempt to train multiple DQN models for each phase of gameplay, developing separate models for trading, set-up, and the robber. We can further attempt to prune the Monte-Carlo search tree by pruning the possible actions we can take, only considering a subset chosen by some heuristic to be most impactful.

7 Contributions

All members contributed equally to the paper. AJ wrote the introduction, board game description, and prior work. RV wrote all sections related to DQN, and PA wrote all sections related to MCTS. RV and AJ investigated the Catanatron simulator and fixed several bugs, and co-wrote the DQN implementation. PA wrote the MCTS implementation. All members ran simulations to evaluate the models.

References

[1] Bryan Colazzo. Catanatron. https://github.com/bcollazo/catanatron, 2021.

[2] Peter McAughan, Arvind Krishnakumar, James Hahn, and Shreeshaa Kulkarni. Qsettlers: Deep reinforcement learning for settlers of catan," qsettlers: Deep reinforcement learning for settlers of catan. https://akrishna77.github.io/QSettlers/, 2019.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[4] Jeremy Monin. Java settlers. https://github.com/jdmonin/JSettlers2/tree/main, 2020.

[5] Chiara F Sironi and Mark HM Winands. On-line parameter tuning for monte-carlo tree search in general game playing. In *Communications in Computer and Information Science*, pages 75–95. Springer, 2018.

[6] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in settlers of catan. In *Advances in Computer Games: 12th International Conference, ACG 2009, Pamplona Spain, May 11-13, 2009. Revised Papers 12*, pages 21–32. Springer, 2010.

[7] Konstantia Xenou, Georgios Chalkiadakis, and Stergos Afantenos. Deep reinforcement learning in strategic board game environments. In *Multi-Agent Systems: 16th European Conference, EUMAS 2018, Bergen, Norway, December 6–7, 2018, Revised Selected Papers 16*, pages 233–248. Springer, 2019.