CPSC 490 Project Report

Daniel Kim
Advisor: Prof. James Glenn

May 4, 2023

1 Abstract

This paper attempts to optimize a set of simplistic heuristics for the board game Settlers of Catan, in an attempt to create an agent that distills the complexity of a

the work done here provides a framework for future explorations of heuristic-based algorithms for computer games.

2 Game Description

Settlers of Catan is a three or four-player turn-based stochastic/Markov game with limited information that pits players against each other to gather resources and build a series of outposts on the fictional landmass of Catan.

Catan is represented as a series of randomly-generated interlocking hexagons; each hexagon is associated with one of five differing resources (wood, brick, sheep, stone, wheat), and a number from 2-12 (excluding 7). Players can place settlements on the vertices of these hexagons to gather resources; on a player's turn, they roll two standard six-sided dice, add up the total, and then all players gather resources for every settlement-hexagon connection that they have created. Afterwards, the player can choose to spend these resources on a number of various improvements to their colony, including:

- Roads, which are placed on the edges of a hexagon, and are required to be connected to a player's existing roads or settlements/cities. New settlements must be placed next to a player's existing roads, and a given hexagon's edge can only contain one player's road, making road-building (especially early in the game) essential for expanding one's colony and achieving victory. Roads are cheap, costing only one wood and brick, counterbalanced by the large number of roads players will inevitably build over the course of the game.
- Additional settlements, which cost one wood, brick, sheep, and wheat. Settlements must be placed at least two hexagon edges away from all other settlements as well as the road connection restriction mentioned above, restricting the number of potential settlement spots available to players and adding strategic complexity to the game. Players can build up to five of these.
- Cities, which are upgrades to settlements, doubling the resources per city-hexagon connection that players can gather. They cost three stone and two wheat, and replace an already-existing settlement on the game board. Players can build up to four of these.
- Development Cards, which can contain a variety of useful effects, such as stealing all of one particular resource from every other player or allowing the player to instantly place two roads without paying the normal cost for them. They cost one sheep, wheat, and stone.

If players roll a 7, instead of gathering resources, the robber is activated. First, every player with 8 or more resource cards must discard half of them, rounding down. Then, the player who rolled a 7 can place the robber on any hex in the game; afterwards, that player can choose another player with a settlement/city connected to that hex and steal one resource from their hand at random, after which the building phase continues as normal. The "soldier" development card, which can be activated on one's turn, also has the same effect of the robber, minus the card-discard portion.

Many of the colonies on the outskirts of the game board are connected to harbors, which facilitate easier trading for resource. Initially, players are able to trade 4 of any resource to the bank for 1 of any other resource, but harbors are able to make this more efficient. There are two types of harbors: resource harbors and general harbors (denoted with a "?" on the game board). Resource harbors change the exchange rate for that resource to a 2:1 ratio, meaning that you can exchange 2 of that resource for 1 of any other resource. General harbors change all resource exchange rates to 3:1.

The aim of the game is to earn a total of 10 Victory Points ("VPs"); settlements are worth 1 VP, cities 2 VPs, having the longest road (at least a length of 5) 2 VPs, and having the largest army (made up of soldiers gained from drawing development cards, with at least a size of 3) 2 VPs. Players are initially given two settlements and a road for each of them, and play does not stop until a player achieves the 10 VP goalpost.

Note that inter-player trading was removed from this implementation as a major simplification to the game, as including it would introduce a level of complexity unconducive to the creation of working agents and the completion of the project.

3 Previous Work

Catan is a very popular game among board game enthusiasts, so it is not surprising that multiple different projects and papers have analyzed the game using multiple different methods. One of the earliest was Thomas's JSettlers, a real-time, Java-based implementation of Settlers of Catan based off of a plan-based heuristic method (2003). JSettlers and its various updates and upgrades have become a sort of gold standard for Catan, being used in other studies as a benchmark for their own agents.

Methods that utilize an algorithmic method have become relatively commonplace. Pfeiffer opted for an approach that used hierarchical reinforcement learning to create an effective agent to play Catan, subdividing a strategy into individually-learned behaviors and then learning independent, high-level policies for those behaviors (2004). MCTS (Monte Carlo Tree Search) has also been used for the goal of agent creation, with Szita, Chaslot, and Spronck writing an MCTS agent that compared favorably against JSettlers; with 1000 games simulated, it performed about as well, while with 10000 games simulated, it performs significantly better than the JSettlers agent (2010).

Other researchers have also focused on Catan from heuristics-based standpoints, such as Guhe and Lascarides, which created an empirical framework for testing strategies in Catan, focusing mainly on planning and symbolic models with heuristic strategies. This work also made heavy usage of JSettlers, improving the JSettlers protocol and also creating new agents that focused on different aspects of the game, such as focusing on development cards (2014). Finally, researchers at Yale have also worked on Catan; Omar Ashraf and Christopher Kim's Senior Project in 2018 came up with the idea of creating a computational intelligence for Catan, where they utilized deep Q-learning to effectively build an agent that could play Catan and compared that Q-learning agent with heuristic agents that they had created (2018).

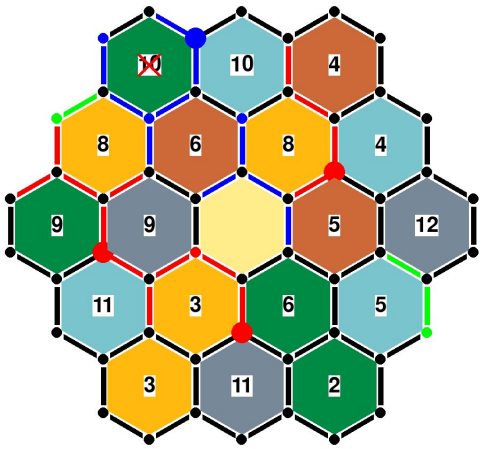
4 Game Implementation

Before writing any agents or heuristics, the actual game of Catan must be built first. This implementation of Catan was built in Python with an OOP-based implementation, with a number of different components that, together, create a working Catan simulator. OOP was chosen to effectively compartmentalize the disparate parts of the Catan game, for ease of development. The main components are:

- Catan class, which handles all of the game logic (rolling dice, building settlements, calculating the longest army, etc.)
- Position class, which represents a certain position in a Catan game.
- Player class, owned by the Position, class, which represents a player's position in a Catan game (resources owned, size of army, etc.)
- Board class and its components the Road, Colony, Hex, and Harbor classes, owned by the Position, which represents the board's position in a Catan game (board configuration, which roads, settlements, and cities have been built, placement of harbors, etc.)
- Display class, which utilizes pygame to create a visual display of the Board and any changes.
- Policy class, an abstract class outlining an agent to play Catan.

An example of a completed Catan game with the pygame visual aspect is shown below. (The red X denotes the hex where the robber is located, preventing resources from being generated from there.)

Figure 1: A completed game of Catan.



5 Heuristics Discussion

Given the wide variety of work already available on Catan and board games of similar complexity, a focus on heuristic simplicity was decided on. If it is possible to create a simple, heuristics-based

agent for Catan that performs at a high or at least a medium level, then it may be possible to reduce other games (or problems in general) to simplistic algorithms, providing good results when time is an extreme issue and the computational power to run a more complex algorithm such as Q-learning or MCTS does not exist or is prohibitively expensive.

Therefore, the decision was made to attempt to optimize a simplistic heuristic-based agent. As to what heuristics to implement and optimize for, a number of options present themselves, including but not limited to:

- What resources to prioritize for initial settlement placing, and how much to prioritize them over a lower-priority resource with a higher rate of generation
- What resources to prioritize for later settlement placing, and how that should interact with the type and number of resources already being generated by the player
- Infrastructural concerns - whether to prioritize upgrading settlements to cities or placing new settlements first
- Where to place new roads, and whether to build new roads or prioritize other infrastructure, such as development cards.
- What cards to discard when forced to discard by a robber attack.
- When to trade your stored resources, and what to trade those resources for.
- How much to prioritize harbors when placing new settlements.
- Where to place the robber, when prompted to.

Given the number and complexity of the heuristics outlined above, a decision was made to port Ashraf and Kim's "smart heuristic agent" to my Catan implementation as a starting point, to which further improvements could be made. Then, a "builder" heuristic was created loosely based on the "smart heuristic player" and my own personal playstyle, which eschews building the longest road, development cards, or the largest army in favor of building up a large base of resource generation infrastructure (settlements and cities) and gaining the required 10 VPs from settlement and city building.

An initial testing module was created to run the smart heuristic, the builder, and a random policy against each other to determine a winning heuristic that would serve as a starting point for further analysis. Various parameters of the winning heuristic was tweaked through the hill-climbing module that I created, in order to ascertain the existence of possible optimal hyperparameters. This then was compared to a random policy and the original policy in order to ascertain the quality of the heuristic.

6 Results

The results of the initial tests between the smart agent, the builder agent, and the random agent (tester.py) are shown below. These agents were tested 100,000 times in a 3-player game against each other with random player order.

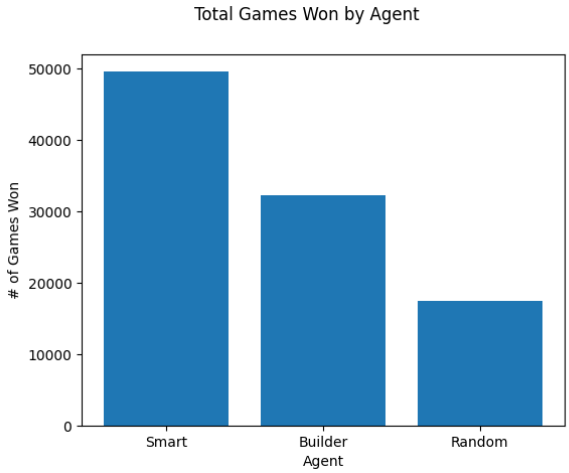


Figure 2: Victories, by agent.

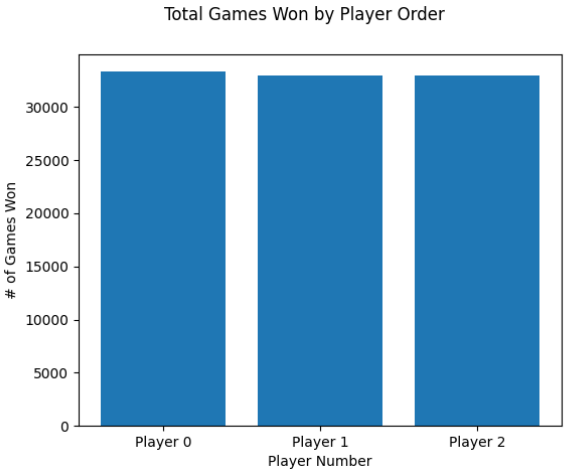


Figure 3: Victories, by player order.

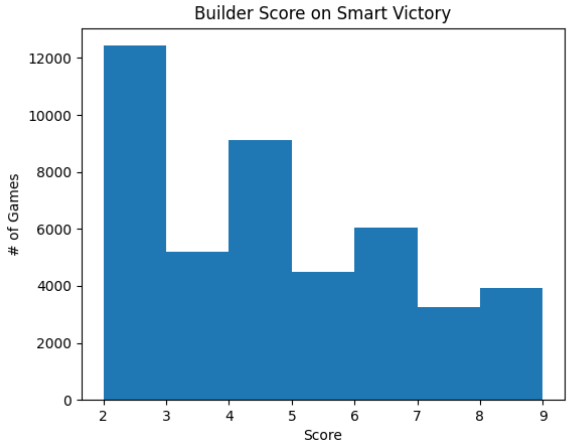


Figure 4: Builder agent score distribution when Smart agent wins.

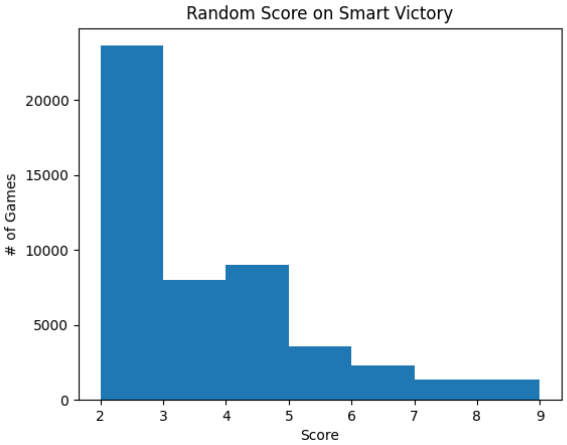


Figure 5: Random agent score distribution when Smart agent wins.

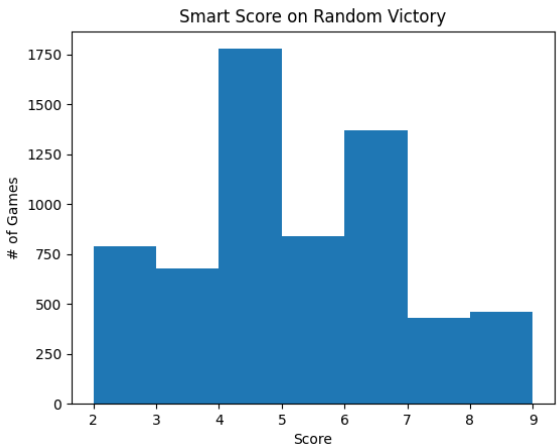


Figure 6: Smart agent score distribution when Random agent wins.

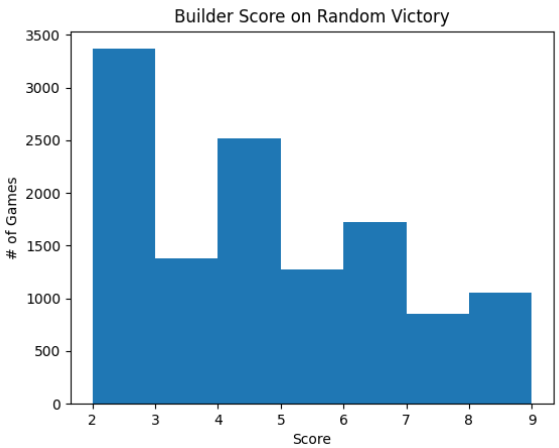


Figure 7: Builder agent score distribution when Random agent wins.

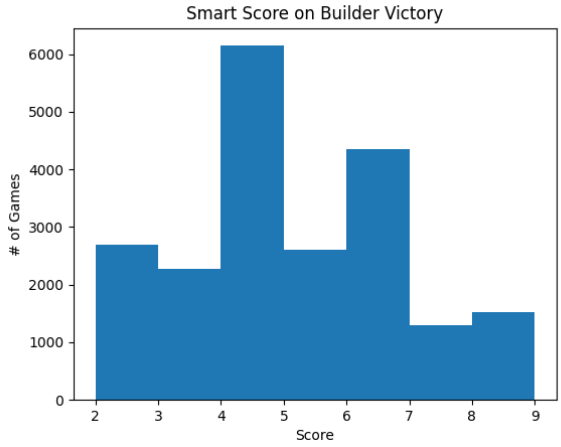


Figure 8: Smart agent score distribution when Builder agent wins.

Figure 9: Random agent score distribution when Builder agent wins.

The results here show the primacy of the smart agent, so this was the agent chosen for optimization. The agent was then put through a hill-climb module made to identify the most optimal combination of hyperparameters. The hill climb module (hillclimb.py) does this by choosing a random hyperparameter starting point between specified minimums and maximums for all hyperparameters. Then, the algorithm iterates. For each iteration, the algorithm finds all neighbors of the current best hyperparameter configuration, defined as every configuration where each parameter is 1, 0, or -1 steps away from the current configuration (step size defined as an input). Note that the hill climb algorithm includes functionality for halving the step size if the algorithm does not show a preset level of improvement.

Then, for each neighbor configuration that has not been viewed yet, the module plays 1,000 games with the neighbor configuration, the initial configuration, and the builder agent. Finally, the neighbor configuration with the best win rate is chosen as the new current best configuration. If no neighbor configuration has a win rate better than the current best configuration's win rate, the algorithm returns the current best configuration as its result.

This algorithm was run multiple times with different starting random points in order to determine if there truly was a best configuration, or if the configuration state space contained multiple local minima and maxima. However, the algorithm proved to be inconclusive, as the results varied widely based on random hyperparameter starting point.

7 Discussion

Testing 100,000 Catan games with the unoptimized Smart, Builder, and Random agents in a scenario where all three of these agents faced off against each other in a three-person game of Catan showed that the Smart agent performed the greatest, implying that, for simple strategies, a balanced focus that incorporates development cards into a strategy will perform better than a strategy focused more on infrastructure and resource generation above all else. Something of note is that the graph showing the builder score on smart victory has a noticeable leftward skew, while the graph showing the smart score on builder victory has a more balanced skew, further cementing the high performance of the smart heuristic agent.

Of course, limiting the analysis exclusively to simplistic strategies revealed a number of noticeable weaknesses, implying the relative impossibility of reducing a game with relatively complex components, such as Catan, to an easily digestible strategy and distilling that to a heuristics-based agent. Many of the games for both builder and smart agents even show a large number of games with only 2 points, implying that the agents failed to gain any points at all during the entire game. This suggests both the need for improvements in initial settlement logic and the need for a more focused strategy; the agents may have oscillated between multiple aims (such as gathering resources for building a settlement and building a city) instead of trading and building towards one exclusively.

Additionally, player order also was not very important when determining victory; as shown above, the first, second, and third players had around the same number of victories. Randomized player choice may have contributed to this, although not seeing a noticeable skew one way or the other is evidence for an order-agnostic analysis.

As for the results of the hill-climb algorithm, the inconclusive result of the algorithm implies that something simplistic, such as hill-climb, may not be adequate for finding the correct set of hyperparameters for the agent. More complex algorithms, such as a genetic algorithm, may be necessary to handle this optimization problem.

8 Future Work

Future work in this endeavor would definitely involve as a first step the porting of the JSettlers algorithm to this Python implementation of Catan - this set of more robust and complex heuristics utilized in many other studies of Catan would be a good starting point for further examination of Catan and heuristic agents. From there, the hill-climb or a similar hyperparameter algorithm could be used to effectively optimize the JSettlers algorithm, providing a stronger starting point for further analysis of Catan.

As for non-heuristic agents, one possibility could be the implementation of MCTS (Monte Carlo Tree Search), perhaps with more modern advances or domain knowledge, such as the inclusion of meta-actions. New advances regarding the MCTS algorithm, such as IS-MCTS-POM, may also be used here. Outlined in Cowling, Powley, and Whitehouse, this algorithm uses information sets as the primary nodes in the minimax tree, allowing MCTS to effectively work with incomplete information, among other improvements (2012). Combining this idea with the meta-action concept, the information sets that IS-MCTS considers could be related to the opponent's ability to build something (like a city or road) rather than something simplistic, such as how many cards the opponent has, with the latter being used to derive the former. In any case, Settlers of Catan remains a very rich avenue for studying computer games and strategies to optimize them.

References

[1] Omar Ashraf and Christopher Kim. Cpsc 490 project proposal: Computational intelligence for settlers of catan. 2018.

[2] Peter I. Cowling, Edward J. Powley, and Daniel Whitehouse. Information set monte carlo tree search. *IEEE Transactions on Computational Intelligentce and AI in Games*, 4(2), June 2012.

[3] Markus Guhe and Alex Lascarides. Game strategies for the settlers of catan. *2014 IEEE Conference on Computational Intelligence and Games*, 2014.

[4] Michael Pfeiffer. Reinforcement learning of strategies for settlers of catan. *International Conference on Computer Games: Artificial Intelligence, Design and Education 2004*, 2004.

[5] Istvan Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in settlers of catan. *Advances in Computer Games*, 2009.

[6] Robert Shaun Thomas. *Real-Time Decision Making for Adversarial Environments Using a Plan-Based Heuristic*. PhD thesis, 2003.