Ninety-Nine Card Game

Skyler Atchison • Professor Michael Soltys • COMP 499

Channel Islands
CALIFORNIA STATE UNIVERSITY

Introduction

Ninety-Nine is an innovative trick-taking card game created by David Parlett in 1967. While it never achieved the popularity of Bridge or Hearts, it is an excellent game. I wanted to bring this game to a wider audience by implementing it as a computer game, which has never been done before. I also wanted to use this opportunity to experiment with an AI decision-making algorithm called the Monte Carlo Tree Search.

The Game

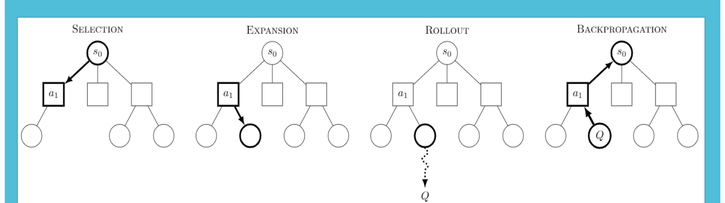
The rules of Ninety-Nine are a variation on trick-taking card games. The game is split up into nine "tricks," where each player plays one card from their hand and the winning card captures the trick. This is standard for trick-taking games like Bridge or Spades, with the added twist that players must predict exactly how many tricks they will take before the round begins. Every hand, each player discards three cards from their hand to represent this prediction and earns points if they correctly match their bid.

First Goal: The User Interface

I wanted my implementation of Ninety-Nine to be fun and easy to play, so I decided to create a graphics interface with the Python library Pygame, a classic tool that provides developers with precise control over graphics and player interactions. I wanted the player to be able to select a game mode, click on cards to play them, and see a running total of their current bid and score.

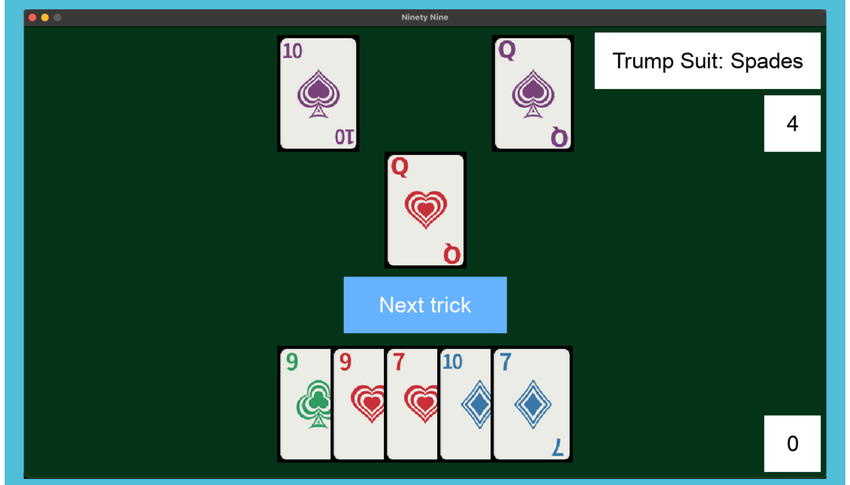
Second Goal: The Computer AI

The algorithm I chose to use for the computer players is called the Monte Carlo Tree Search algorithm, which works by creating a tree representing the space of all possible game decisions, where each node is a possible game state and each edge is a possible move. The algorithm then searches a subset of this tree according to probabilistic criteria and uses the results of this search to make an informed decision of which move to make. I saw in my research that this algorithm has been applied to many other games, including other card games, with great success.

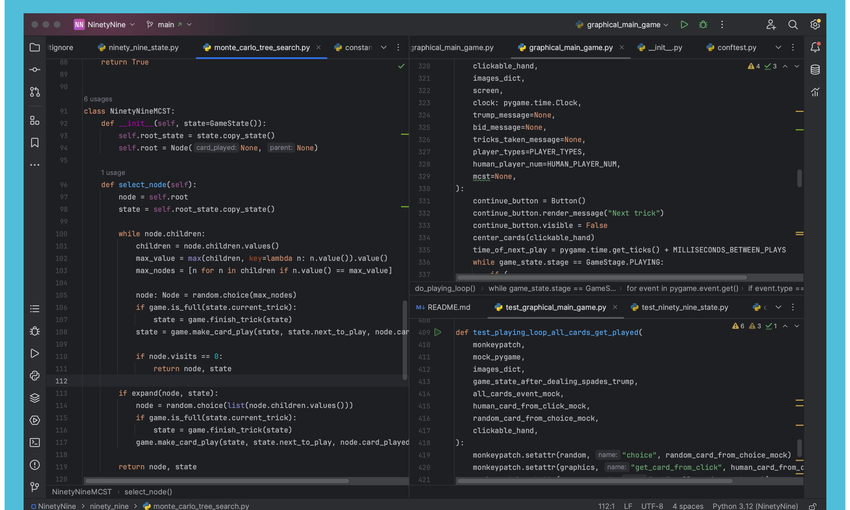


Results

The User Interface turned out well, resulting in a fun computer implementation of Ninety-Nine. Players can see their hand displayed in the bottom of the screen, the current trick in the center, and their score, bid, and trump suit on the right side. The cards also have animations when they are played to the table to simulate an actual tabletop game.



After learning about test-driven development in my Software Engineering course, I decided to use it for my development process to reduce the need for debugging. I utilized the library Pytest to accomplish this, resulting in a robust testing suite that covers all modules in the code. Thanks to this suite, it will be straightforward for me or others to add more features to the project in the future and ensure that the new features do not break previous functionality. I created my own fixtures and parameterized tests to streamline this testing process.
I also made use of many helpful features of the Python syntax, including f-strings, operator overloading, dataclasses, and list comprehension.



Conclusions

This was an enjoyable project, and I learned a lot about the theory of card games and game-playing engines,. I also gained practical skills in frontend game development.

My future goals would be to improve the AI, as currently its win rate is not significantly higher than the rate of the control group. I would either further develop the MCTS algorithm or try a different algorithm altogether. I would also want to create an interactive tutorial to help teach players who are new to card games. Finally, I would want to port my game to JavaScript as a web application to make it more convenient to install and play.

Research Sources

To learn about the Monte Carlo Tree Search algorithm and how researchers have applied it to card games, I read through this paper:

Kupferschmid, S., Helmert, M. (2007). A Skat Player Based on Monte-Carlo Simulation. In: van den Herik, H.J., Ciancarini, P., Donkers, H.H.L.M.(. (eds) Computers and Games. CG 2006. Lecture Notes in Computer Science, vol 4630. Springer, Berlin, Heidelberg.

Other resources I used for learning about the algorithm and required libraries include a tutorial from GitHub user ai-boson and the website GeeksForGeeks. The links may be found in my GitHub repository:
github.com/SkylerMime/NinetyNine

A Skat Player Based on Monte-Carlo Simulation

Abstract

We apply Monte-Carlo simulation and alpha-beta search to the card game of *Skat*, which is similar to Bridge, but sufficiently different to require some new algorithmic ideas besides the techniques developed for Bridge. Our Skat-playing program, called DDS (Double Dummy Solver), integrates well-known techniques such as *move ordering* with two new search enhancements. *Quasi-symmetry reduction* generalizes symmetry reductions, disseminated by Ginsberg's Partition Search algorithm, to search states which are "almost equivalent". *Adversarial heuristics* generalize ideas from single-agent search algorithms like A* to two-player games, leading to guaranteed lower and upper bounds for the score of a game position. Combining these techniques with state-of-the-art tree-search algorithms, our program determines the game-theoretical value of a typical Skat hand (with perfect information) in 10 milliseconds.

Acknowledgements

Thanks to Professor Michael Soltys (Capstone Project) and Professor Reza Abdolee (Capstone Prep) for advising me this semester. Thanks to all of my professors at CSUCI for helping me in my journey of learning coding and software development.

Thanks to my family for their support and for their testing of this project.