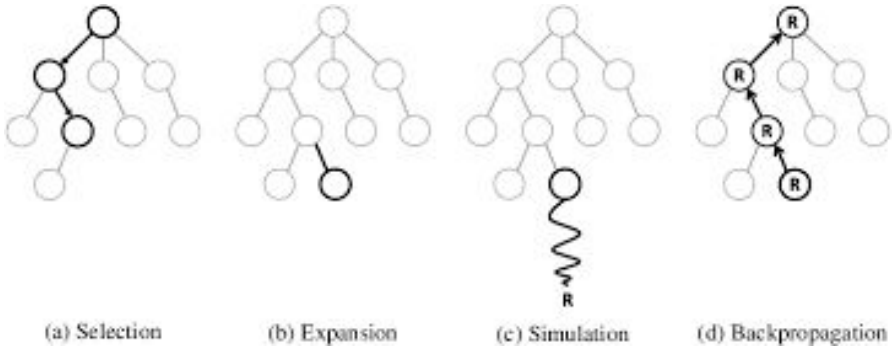
Settlers of Catan AI

Background

Monte Carlo Tree Search

- Randomly “rolling out” selected moves in order to statistically find the best path
- Different ways to select a move produce different results



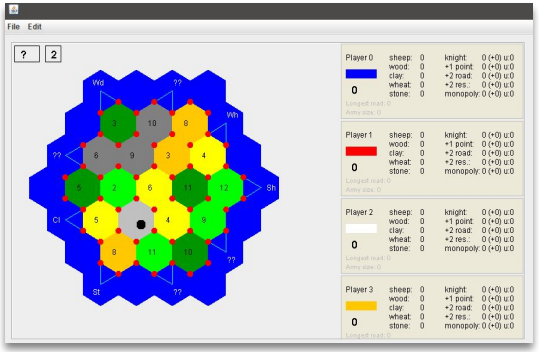
Settlers of Catan

- Turn based board game
- Goal: Garner resources to build settlements and roads in order to earn the most points

Settlers of Catan MCTS

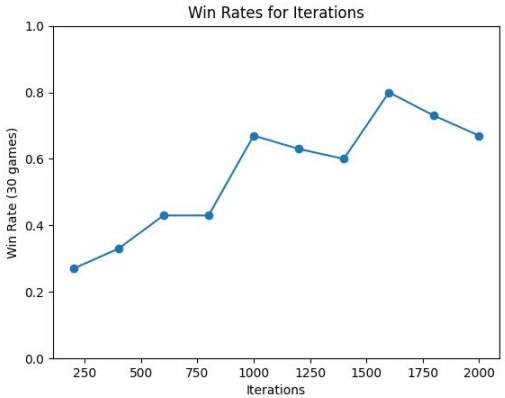
Settlers of Catan MCTS Simulation

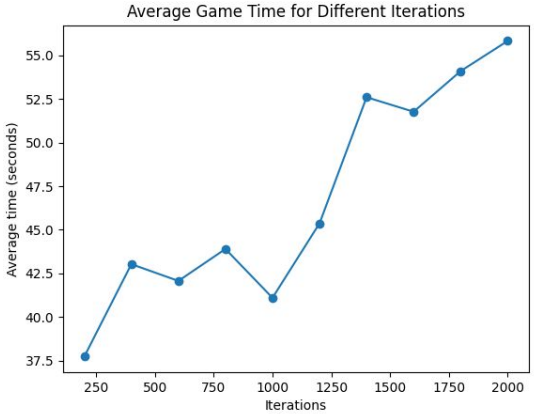
- Used a Java Catan MCTS implementation as a baseline
- Used a Java Catan simulator to compare different MCTS options
- Ran multiple simulations against a random choice opponent to see how MCTS changes affect different game stats

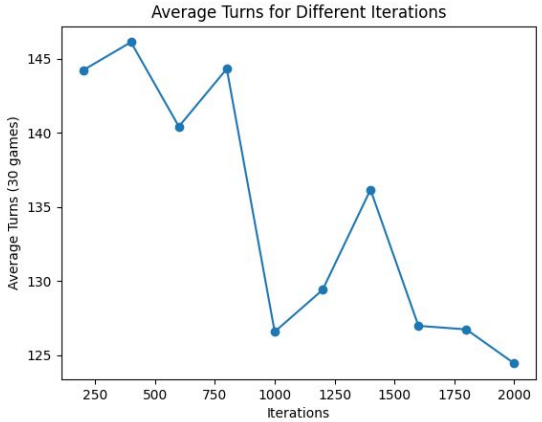


Iterations

- The number of iterations of the MCTS Algorithm for each turn

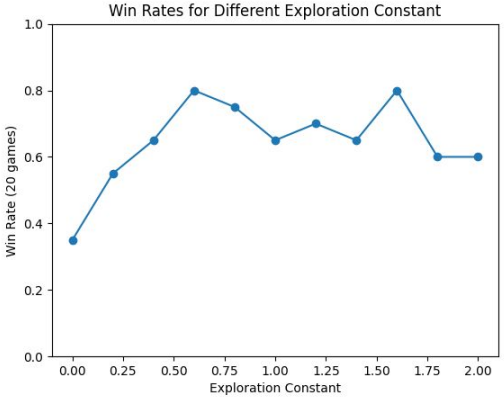


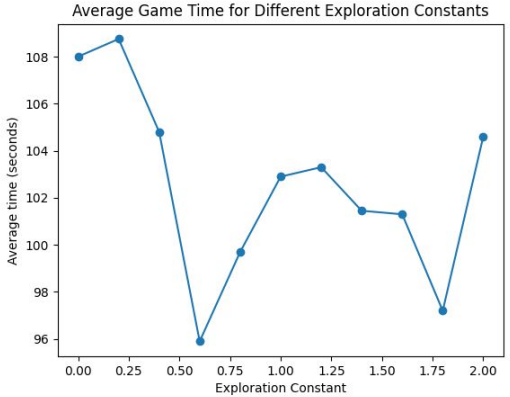


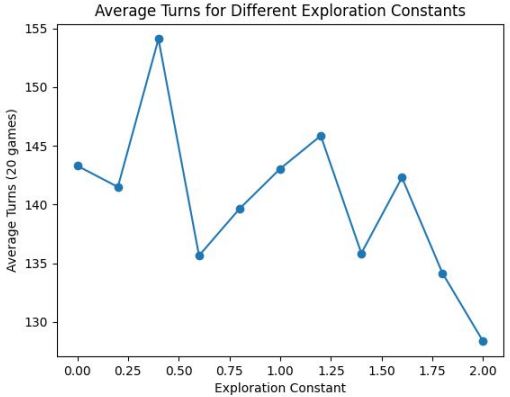


Exploration Constant

- The choice between exploring more nodes or visiting previously seen good nodes



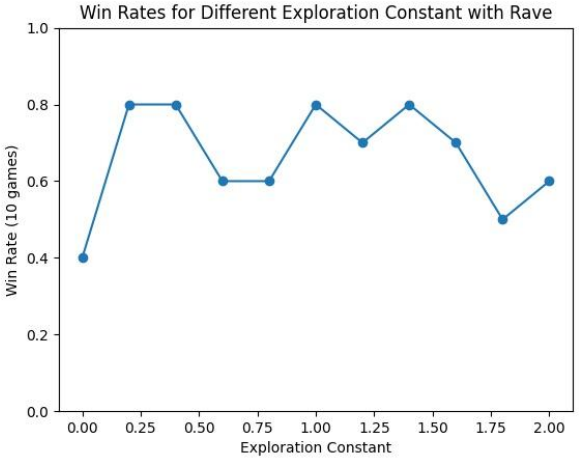


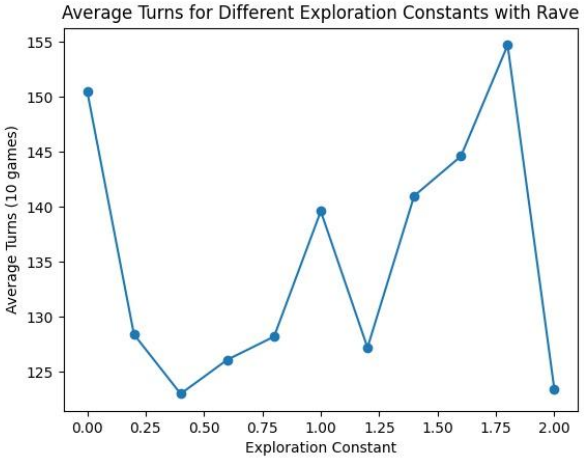


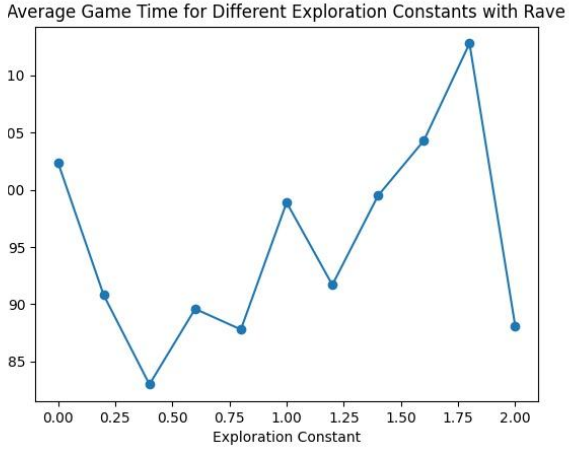
Exploration Constant with Rave

$$(1-\beta(n_i,\tilde{n}_i))\frac{w_i}{n_i}+\beta(n_i,\tilde{n}_i)\frac{\tilde{w}_i}{\tilde{n}_i}+c\sqrt{\frac{\ln t}{n_i}}$$

- Similar results, although average game time seems to increase whereas game time in normal MCTS decrease

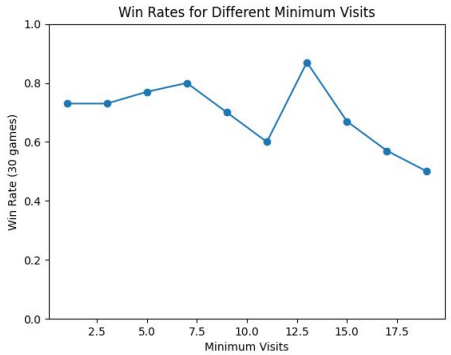




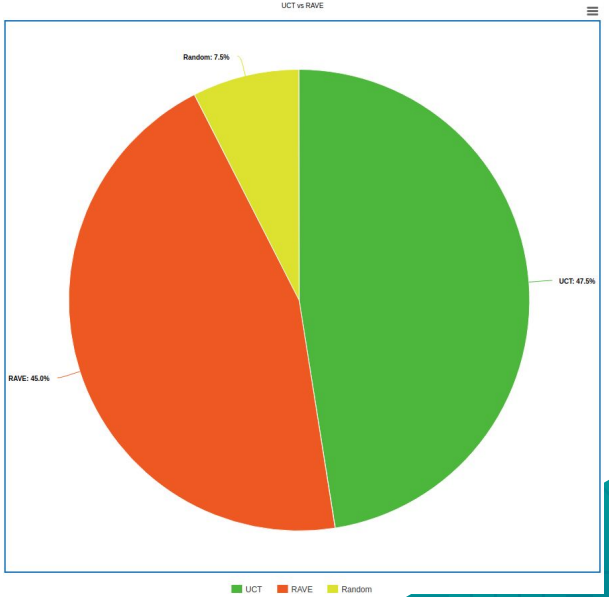


Other Parameters

Minimum Visits: The number of times child nodes need to be visited before using the calculation



UCT VS RAVE selection strategies

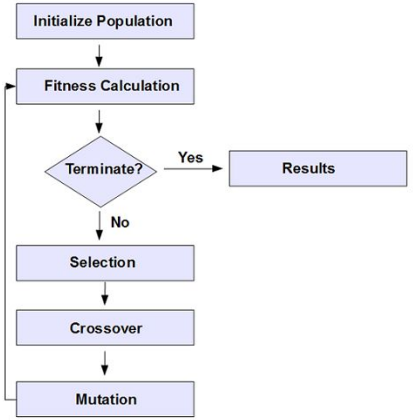


Genetic Algorithm

How can we optimize the MCTS player’s parameters?

Genetic Algorithms

- We've seen these before! (HW9)
- Summary:
  - Create random population
  - Calculate fitness of each individual
  - Preserve the fittest
  - Select several to crossover, mutate the crossovers
  - Repeat with our new population until we're done



Why a genetic algorithm?

- Several variables at play with MCTS players:
  - Max Iterations
  - Exploration constant
  - Selection Policy (UCT? RAVE? PUCT?)
  - Tree Size
  - Min Visits
- Challenging to predict how any individual will perform with none of the variables held constant
- Genetic algorithm allows us to achieve a strong MCTS player with all these variables at play

Implementation

How we made it

Implementation: High-Level Overview

- Goals:
  - Run Stac Settlers simulation for x games
    - 1 MCTS (with specific traits), 3 random
  - Extract and plot MCTS win rates
  - Encapsulate this into a genetic algorithm
- Chose **Python**
  - Easy to handle/plot our data
  - Familiar to all of us, easy to stand up

Implementation: Classes

- MCTSConfig
  - Generates config file for Stac Settlers with 1 MCTS player (and defined genome), 3 random players
  - Runs simulations and pulls MCTS player's win-rate
- MCTSPlayer
  - MCTSConfig, genome
  - crossover(), mutate(), get_fitness() and calculate_fitness()
- Environment
  - Population, population_size
  - create_population() initializes random, starting population

Implementation: Challenges

- Ideally:
  - Pop = 64 individuals
  - Fitness = win% over 200 games
  - Different set of opponents each 40-game segment
- In reality:
  - Each MCTS game takes ~1-2 mins! Need more cores for that
- So we simplified things to prove viability
  - Pop = 8 individuals
  - Fitness = win% over 10 games, same opponents each game
  - Comprehensive? No. Can we still see a result? Yes.

Results

- Top player: 87.5% win-rate over 40 total games, ~32 second runtime / game
  - (simulations stacked each round for the fittest, kept individuals)
- Stats:
  - Iterations: 2033
  - Exploration constant: 1.8747
  - Min visits: 3
  - Max tree size: 5822
  - Selection policy: PUCT

Further Work

Optimization

- Run multiple games in parallel
- Run longer experiments to help reduce random error

Source Code Modification

- Create and implement new selection algorithms
- Add early game heuristics for earlier exploitation

Sources

- Catan MCTS: https://github.com/sorinMD/MCTS
- Catan Simulator: https://github.com/sorinMD/StacSettlers
- MCTS Optimization Paper: http://www.incompleteideas.net/609%20dropbox/other%20readings%20and%20resources/MCTS-survey.pdf
- Our Repo: https://github.com/MWRose/CatanAI

Appendix: Defining our genome

- Goal: represent our genome uniformly
- Result:
  - Held each trait as a float **0.0 ≤ n ≤ 1.0**
    - Looks like [0.08122069, 0.48083098, 0.18265367]
    - Handling in crossover(), mutate(), etc. is easy now!
  - When generating MCTS player's config file, multiply each trait by its max constant to get our values

- MCTS Traits:
  - Iterations
  - Exploration constant
  - Selection Policy
  - Tree Size
  - Min Visits

Appendix: Calculating Fitness

- Determine by matching individual against 3 random-type players
- Fitness = [win% + sqrt(30 / runtime)] / 2
  - Win rate and runtime are prioritized