Catan

Abby Tonkinson and Evan Gray

Overview

- Goal of Catan is to reach 10 victory points
  - Get victory points by having the largest army and/or longest road, building settlements and cities, and through development cards
  - Gain resource cards by building on the board around resource tiles
- 4 agents
  - Random
  - Greedy-settlement/city
  - Greedy
  - Monte Carlo Tree Search

Source: https://myriadsgifts.com/products/catan-the-board-game

Our Experiment and Approach

- Create a complicated board game like Catan
- Train the agent on normal, randomized test games
- Play MCTS agent against more hard coded strategies
  - Simulate human players vs a learning agent
- Related work:
  - jSettlers
  - Uses action-dependent state features to approximate Q-value locally
  - Neural network

RL problem formulation

State Space

- Entire board
- MCTS uses the state history which is essentially the current state of the board

Action Space

- Each action (buy, build, upgrade, trade)
    - What do you want to build, where do you want to do it

Reward

- Victory points

Environment

- Implemented a grid based on hexadecimal coordinates to represent the Catan board
    - Ross Hamish's Catan
    - Includes nodes and edges for building purposes as well as tiles with the respective resource and ports
    - Houses all of the functions/actions needed for game play
- Utilized a dictionary to store and manipulate the actions in the game

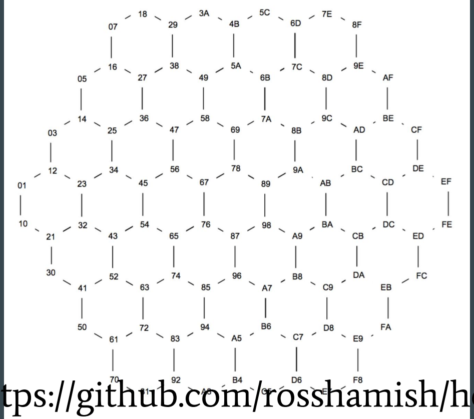


Figure A.2 Node Coordinates

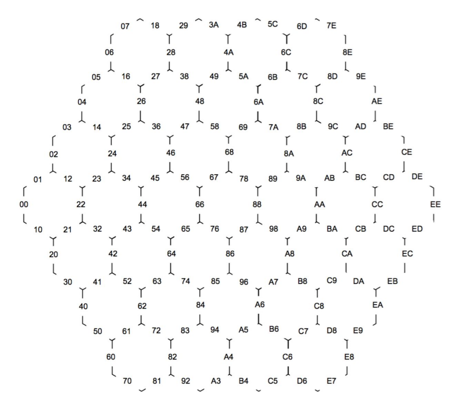


Figure A.3 Edge Coordinates

Source: https://github.com/rosshamish/hexgrid

Agents

- Greedy (Building)
  - Prioritizes building based off reward
    - Cities > settlements > roads
- Greedy (Collecting)
  - Prioritizes buying/playing development cards and utilizing port
- Random
  - Selects a random action from playable actions
- Monte-Carlo Tree Search

Monte Carlo Tree Search

- Treated an overview of the board as the state history
- Tree search approach ideal for a game with such branching actions
- Each action consists of the action a human would take and the options for where the action would take place
  - For example, the agent simulates based on playing the night and each of the available options for the robber to be moved
  - Building a settlement and all available locations for the settlement to exist
- Upper Confidence Bound

$$A_t = \text{argmax}_a \left(\underbrace{Qt(a)}_{\text{Exploit}} + \underbrace{c \sqrt{\frac{\ln(t)}{N_t(a)}}}_{\text{Explore}} \right)$$

Source:
https://www.geeksforgeeks.org/upper-confidence-bound-algorithm-in-reinforcement-learning/

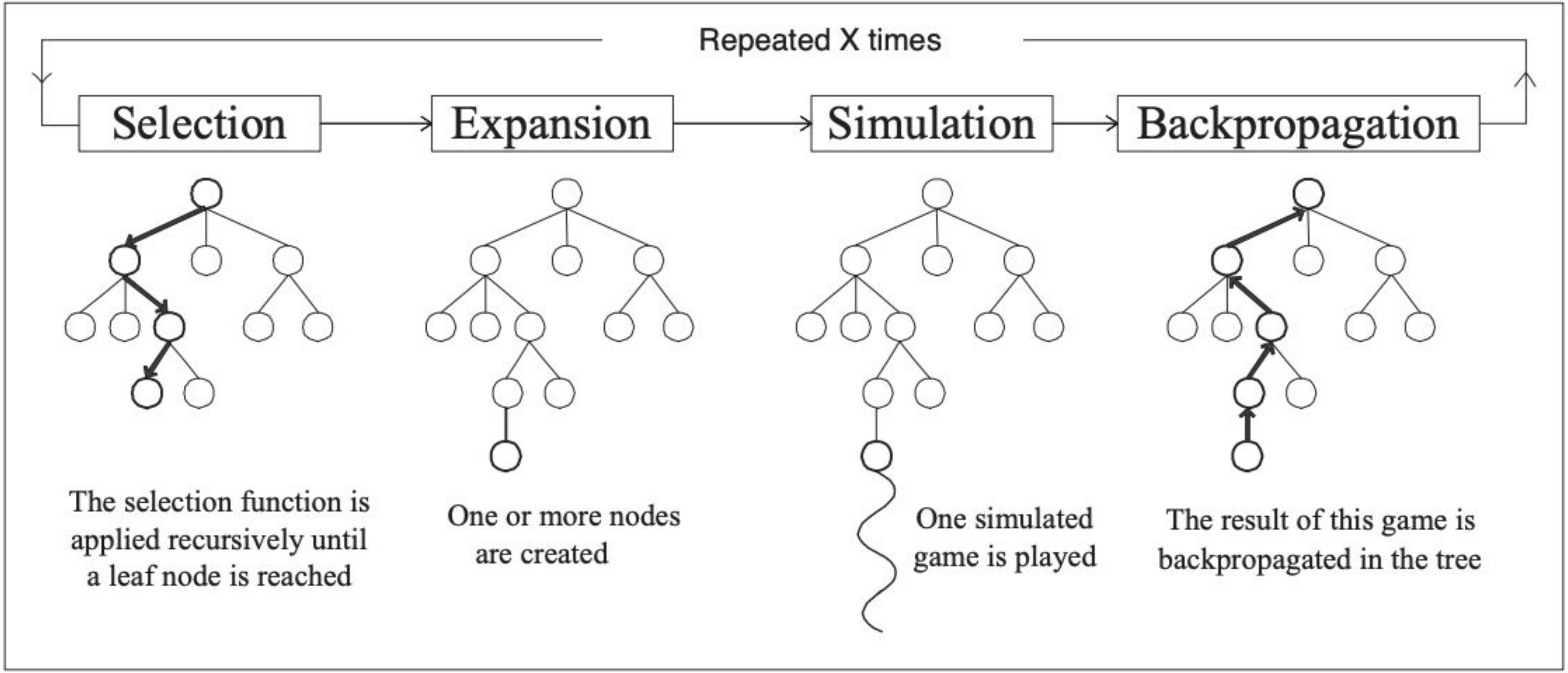


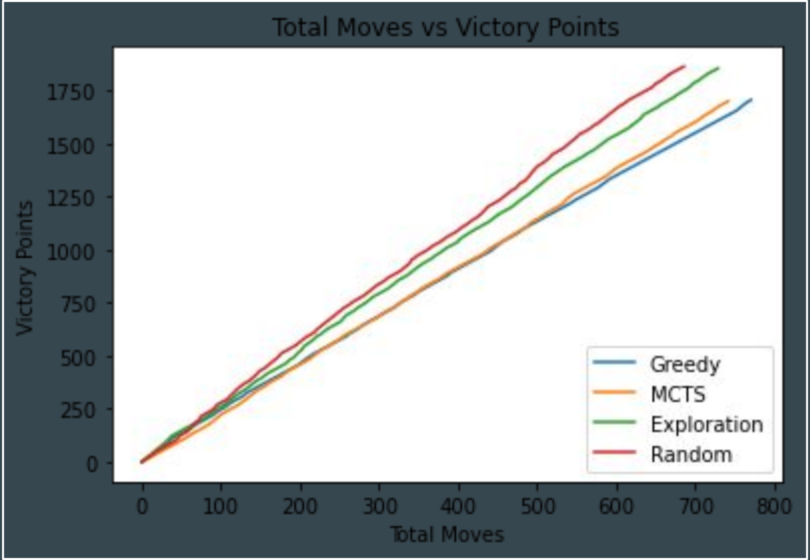
Figure 1: Outline of a Monte-Carlo Tree Search.

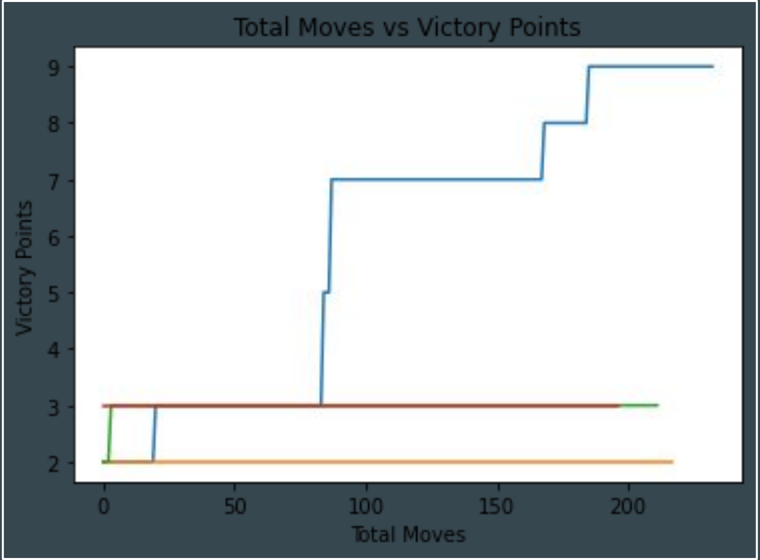
Source: https://paperswithcode.com/method/monte-carlo-tree-search

Challenges

- Using a pre-existing environment or creating our own
    - We found that attempting to use an existing environment was more complicated than making our own
        - Overcomplicated
- Representing the board
    - Tried pygames, GUIs, etc
- Finding adjacent nodes and roads
    - Referencing hexgrid and using hexadecimal locations solved this problem easily
- Trading with ports and other multi decision actions

Results





Cumulative victory points over 100 games

Early game where agents aren’t as trained

Conclusions

MCTS agent needs more work on the state and action space to induce learning

- Broad action space
  - Our implementation based on simpler games, might need more complex implementation
- Improve the algorithm for better planning
- Other approaches have used neural networks to trade between agents

Greedy (Building) directly picks up victory points but performed surprisingly poorly

- Little consideration for placement location in regards to resources and probability of roll

Greedy (Collecting)

- Likely picked up development card victory points granting largest army and longest road achievements

Random

- Best performance overall
- Other strategies were not as effective as we had thought

Future work

- Improve the agent by limiting the state space
- Implement trading between agents
  - Neural network approach
- Reduce adjustments discussed in the report
- Implement a GUI
- Have agents learn from and play against humans