COSC 322 Progress Report

2025-02-26

Group Members:

Kyle McLeod (22892269)

Dhevan Singh (62395199)

Jordan Truong (40559650)

Chase Winslow (45880549)

1. Progress Summary

Over the past couple weeks, our team has been very focused on the organization of our project. We started off by creating a Git repository so all members of our team could contribute and be aware of the changes made by other group members. This Git repository has a detailed README file that goes over project setup, instructions and development guidelines to ensure we keep a structured workflow.

One of our milestones was completing the warm up tasks. These tasks helped us set up our project and get the GUI visible. Players are able to enter any room , and spectators can also join the room and see the gameplay.

Since none of our group members were familiar with the Game of Amazons, we spent some time learning the rules of the game. We used this guide to learn the basics and tested the gameplay using this game link.

To maintain a clean codebase, we made sure to plan our file structure before implementing our main logic. This ensures that different components of our project are modular and easy to maintain. The planned folder structure can be seen in Section 3.2, where we document every folder's use case and specify some classes that we will use.

Finally, we conducted research on potential algorithms to use for our game-playing agent. We initially considered the Minimax algorithm, which is very popular for turn based games. Minimax with alpha beta pruning should always find optimal moves with a deep enough search. However, since the tournament has a 30 second move timer, we were worried that Minimax would be too computationally expensive and wouldn't be able to explore enough states in the 30 second timeframe. After some more research, we decided that a Monte Carlo Tree Search (MCTS) algorithm would be best for the Game of Amazons. MCTS is great for games with large search spaces because it doesn't require exhaustive search depth. MCTS instead runs randomized simulations and refines the search tree based on moves that are more likely to lead to a better outcome.

2. Technical Plans

2.1 Algorithm Decision & Breakdown

2.1.1 Our Decision

The algorithm that will be used for our AI agent is Monte Carlo Tree Search (MCTS). It is an algorithm used in turn-based games with very large search spaces like the Game of Amazons. Here's how we plan to implement it in our program:

2.1.2 Overview of Monte Carlo Tree Search

MCTS is a heuristic search algorithm that also uses random sampling. It builds a tree and uses results of random simulations to find the best move. It can be broken down into four steps:

1. Selection - Starting from the root node, the algorithm traverses the tree based on a selection policy.
2. Expansion - If the leaf node is a non-terminal node, one or more child nodes are added and these child nodes represent the possible moves from that specific state.
3. Simulation - Random moves are played from that node until the game ends.
4. Backpropagation - The result of the simulation is propagated back up the tree. Update the number of visits and win/loss values.

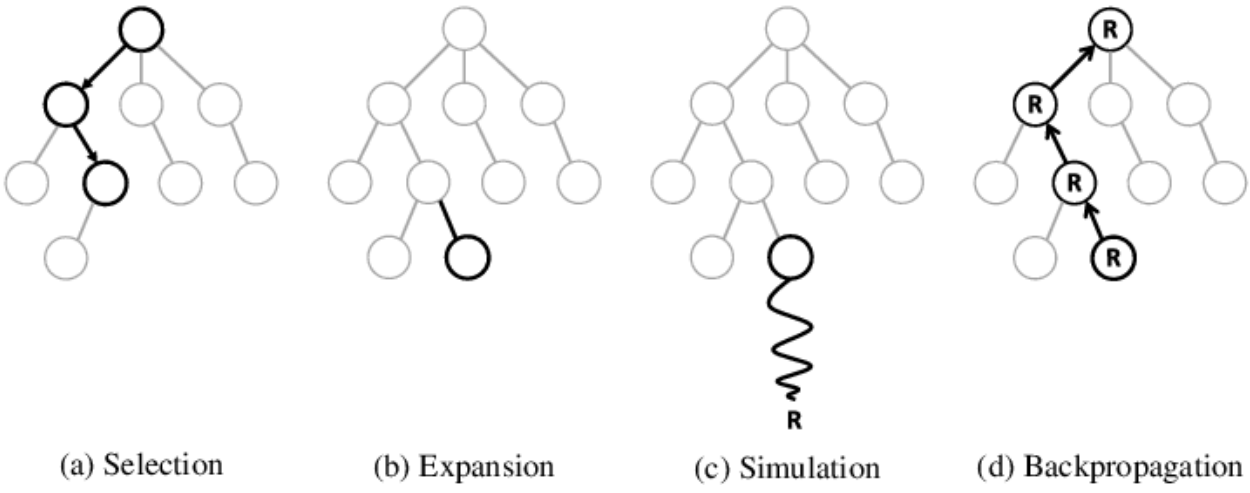

Figure 1) Monte Carlo Tree Search Visualization

2.1.3 Implementation of MCTS in the Game of Amazons

We will use the GameClient class that represents the game state. This class will be used to get:

- Piece positions
- Current player
- Arrow locations
- Any other information that MCTS will need

Each node in MCTS will represent a game state. We will need a class for the nodes that includes:

- Current game state
- List of child nodes
- Statistics such as number of visits and win/loss ratio

We will implement the four steps of MCTS within a dedicated class that manages the search process. This class will handle:

- Initializing the search tree
- Selection, expansion, simulation, backpropagation
- Returning the best move

2.1.4 Challenges

The 30 second move timer will be a challenge for the MCTS implementation. The time might restrict the depth and breath of the search, which would lead to an un-optimal move. We should take this into consideration, and make sure that the AI does not spend too much time exploring branches with low expected outcomes. Instead, we should focus the AI more towards branches with high expected outcomes and if there is extra time on the move clock, then the AI could explore the lower expected outcome branch.

Since MCTS involves random moves, the quality of the simulations will vary. This means we should focus a lot on providing MCTS with quality heuristics. This might be at the cost of computational complexity, but we believe the benefits of improved decision making will be worth it.

2.2 Proposed Project File Structure

This is our proposed file structure for our implementation of MCTS in the Game of Amazons. All of our classes will be kept in the ubc/cosc322/ folder and we will have unit tests for each class we create.

```
├── src/                                    # Source code directory
│   ├── docs/                               # Documentation
│   │   ├── progress_report.md              # Progress report notes.
│   │   └── user_guide.md                   # User guide for the game.
│   │
│   ├── main/                               # Main source code
│   │   ├── java/                           # Java source files
│   │   │   └── ubc/cosc322/                # Main package
│   │   │       ├── COSC322Test.java        # Main game player class
│   │   │       ├── MCTSNode.java           # Class representing a node in the
tree
│   │   │       ├── MCTS.java               # Class implementing the MCTS
algorithm
│   │               ├── HumanPlayer.java    # Class for human player
│   │   │           └── LocalTest.java      # Class to test locally
│   │   │
│   │   └── resources/                      # Resource files (configs, properties)
│   │       ├── config.properties           # Configuration file for game settings
│   │       └── images/                     # Images for GitHub repo
│   │
│   └── test/                               # Test source code
│       └── java/                           # Test Java files
│           └── ubc/cosc322/                # Test package
│               ├── MCTSNodeTest.java       # Unit tests for MCTSNode
│               ├── MCTest.java             # Unit tests for MCTS algorithm
│               ├── HumanPlayer.java        # Unit tests for HumanPlayer
│               └── LocalTest.java          # Unit tests for local testing
│
├── target/                                 # Compiled output (GITIGNORED)
│
├── .gitignore                              # Git ignore file
├── pom.xml                                 # Maven project configuration
└── README.md                               # Project overview
```

3. Goals

Short-Term Goals

1. Implement the MCTS algorithm and integrate it with the GameClient class to get game state information.
2. Write unit tests for the MCTSNode and MCTS classes and test edge cases.

Long-Term Goals

1. Prepare for the tournament by optimizing the algorithm even more for speed and accuracy.
2. Final testing of the game, including AI vs. AI and human vs. AI matches.
3. Refine Heuristic function to account for other aspects of the game such as blocking opponents.

4. Conclusion

Over the past couple weeks, our team has made good progress with getting started on the project. We learned the rules of the Game of Amazons, set up a Git repo, and researched potential algorithms for our agent. Our final decision was to use the Monte Carlo Tree Search (MCTS) because of its suitability for games with large search spaces. We also made an organized file structure and development workflow to make the development process smooth and easy. Our next steps will be implementing the MCTS algorithm and creating tests for the classes, then we will focus on preparing for the tournament.

5. References & Resources

1. Minimax Algorithm:
    - Russell, S., & Norvig, P. (2020). Artificial Intelligence: A Modern Approach. Pearson.

2. Game of Amazons Rules: [Link]
3. Monte Carlo Tree Search (MCTS): [Link]