AI & GAMES

Group Proforma:

Two Algorithms to be used:

- Monte Carlo Tree Search (MCTS) with AlphaGo:
    - Strength:
        - Suitable for games with large state and action spaces.
        - Doesn't require an explicit evaluation function.
        - Can handle complex decision spaces and uncertainties.
        - AlphaGo integrates deep neural networks for evaluating board positions and making strategic decisions.
- Minimax Algorithm with Alpha-Beta Pruning:
    - Strength:
        - Efficient in practice, especially with a well-designed heuristic.
        - Effective for games with a smaller state space.
        - Provides a deterministic evaluation of game states.

Team Division:

- Chittesh and Vedant - Monte Carlo Tree Search (MCTS) with AlphaGo
- Suphal and Amaan - Minimax Algorithm with Alpha-Beta Pruning

Tuning for Hex game for the above two algorithms:

- requires a good heuristic function.
- refine rollout strategy
- adapt backpropagation strategy
- hex-specific heuristics
- optimize computational resources
- test on different board sizes
- optimize tree policy

Research:
Monte Carlo Tree Search (MCTS) with AlphaGo

1. Monte Carlo Tree Search: (https://www.chessprogramming.org/UCT)
    a. Select
        - Upper Confidence Bound for trees

In UCT, upper confidence bounds (UCB1) guide the selection of a node [3], treating selection as a multi-armed bandit problem, where the crucial tradeoff the gambler faces at each trial is between exploration and exploitation - exploitation of the slot machine that has the highest expected payoff and exploration to get more information about the expected payoffs of the other machines. Child node j is selected which maximizes the UCT Evaluation:

$$UCT_j = X_j + C * \sqrt{\frac{ln(n)}{n_j}}$$

where:

- X_j is the win ratio of the child
- n is the number of times the parent has been visited
- n_j is the number of times the child has been visited
- C is a constant to adjust the amount of exploration and incorporates the sqrt(2) from the UCB1 formula

The first component of the UCB1 formula above corresponds to exploitation, as it is high for moves with high average win ratio. The second component corresponds to exploration, since it is high for moves with few simulations.

https://www.chessprogramming.org/Match_Statistics#ratio

The optimal range for the exploration constant *C* in Monte Carlo Tree Search (MCTS) can vary widely depending on the nature of the problem, the characteristics of the search space, and other factors. There is no one-size-fits-all answer, and finding the right value often involves experimentation and tuning specific to your application.

1. **Typical Starting Value:**
    a. A common starting point for the exploration constant C is often in the range of 1.0 to 2.0. This is a reasonable range to begin with, and you can adjust from there based on empirical testing.
2. **Problem Size and Complexity:**
    a. For larger and more complex problem spaces, you might need higher values o *C* to encourage more exploration. Conversely, for smaller and more constrained spaces, lower values of *C* may be more appropriate to focus on exploitation.
3. **Empirical Testing:**
    a. Conduct experiments with different values of *C* and observe the performance of your MCTS algorithm. Run simulations or trials on representative problem instances and assess how well the algorithm explores the search space and converges to good solutions.
4. **Domain-Specific Knowledge:**
    a. If you have domain-specific knowledge about your problem, use that knowledge to guide your choice of C. Understanding the characteristics of the optimal solutions and the structure of the search space can inform the selection of C.
5. **Grid Search or Optimization Techniques:**

a. If feasible, perform a grid search over a range of C values and evaluate the algorithm's performance for each. Alternatively, you could use optimization techniques, such as Bayesian optimization, to automatically search for the optimal C value based on observed performance.

Bayesian Optimisation:

- probabilistic model-based optimization technique that can be used to find the optimal set of hyperparameters for a given objective function

Chapter -3:https://www.studocu.com/en-ca/document/dalhousie-university/computer-science/hex-game-heuristic-search/23592313

https://www.cs.utexas.edu/~pstone/Courses/394Rspring11/resources/mcrave.pdf

https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=CF874E2439C88B2D68278ADEE5D708B9?doi=10.1.1.726.1164&rep=rep1&type=pdf#:~:text=Monte%2DCarlo%20tree%20search%20and,the%20improvement%20of%20the%20simulations.

https://github.com/masouduut94/MCTS-agent-python/blob/masthttps

Self.state - in MCTSnode
backpropagate () - node.state.get_player()

b. Expand
c. Simulate
d. Backpropagate

With the inconsistent defeat of Bob from the implementation of the minimax algorithm, the whole team was able to figure out that minimax will no longer be used for the final proposal since it was not efficient enough to beat all the test agents.

Basic Implementation:

- We divided the whole implementation in the two classes Node and MCTS where the Node class consists of adding children to calculating UCT value to decide which node to explore next.
- UCT or Upper confidence bound for trees is calculated to maintain balance between the nodes explored and which are yet to be explored so that maximum nodes are explored.
- selectNode(),
- expandTree(),
- rollOut(),

- back propagation()

Enhancement:

1.tune the UCT formula by adding AMAF

- The first enhacnemnt we try doing to increase the node value based on UCT by including AMAF (all moves as first) -
- With this optimization we were able to explore all new nodes which were not explored before.

2. RAVE is a modification to the standard UCT (Upper Confidence Bounds for Trees) algorithm in Monte Carlo Tree Search. It focuses on updating statistics based on the results of simulations involving the move being considered and similar moves from previous simulations.

Implementation and Results

Role and Responsibility:

My roles and responsibilities:

- I worked in a sub-team with Vedant where we planned to implement the Monte Carlo Tree Search Algorithm.
- I started researching different techniques used by others for developing the algorithm such as understanding about AlphaZero.
- I learned about the basic mechanism of MCTS along with UCT (Upper Confidence Bound for Tree).
- I used python to implement the basic functioning of the algorithm and used UCT for node selection. Also, We implemented those strategies (RAVE, UCT tuning) into our MCTS code by doing pair programming.
- Experiment - I collaborated with Suphal to convert the python file into java for better efficiency.

Entry 1: Project Kickoff
Date: 15th November 2023

Objectives:
Discuss project goals and deliverables.
Assign roles and responsibilities.
Set project timeline and milestones.
Decisions:
Weekly team meetings will be held twice every week, possibly on Tuesdays and Thursdays.

Entry 2: Research and Planning

Date: 22nd November 2023

Objectives:

Understand the rules of Hex Game
Do research on AI algorithms
Choose the algorithm for the project
Choose the programming language

Challenges:

Confusion between efficiency of Java and Python
Confusion between efficiency of MCTS and Minimax Alpha Beta.

Decisions:

Subdivided the team into equal team members.
Team 4A consisted of Amaan and Suphal
Team 4B consisted of Chittesh and Vedant
Team 4A worked on Minimax algorithm
Team 4B worked on MCTS
We decided on using Python for this coursework.

Entry 3: Design and Development

Date: 29th November 2023

Objectives:

Track the progress of each subteam
Understand the heuristics for the algorithms
Implement strategies like RAVE for efficient functioning of the algorithm

Challenges:

Lack of experience in MCTS.
Gathering the correct resource from the internet to understand various strategies.

Decisions:

To complete both the algorithms by next week and compare which one is most efficient.

Entry 4: Mid-Project Review
Date: 5th December 2023

Objectives:
Identify potential errors and solutions.
Merge the teams by making a decision of the final algorithm to proceed with.
Optimise the functionality of the code.
Understand the strategy of the test agents

Challenges:
Time out error.
Python was executing at a slow pace.
Minimax Alpha Beta was not providing desired results.

Decisions:
We decided to merge and do pair programming to work on the MCTS algorithm.
Use of certain python libraries like Numpy to optimise the code's implementation.
Do some research on Java to see if it can be beneficial to change our programming language at this time of the project.

Entry 5: Final Stages and Testing
Date: 12th December 2023

Objectives:
Complete the presentation slides.
Test the code on all the Test Agents
Further optimise the code to reduce execution time.

Challenges:
Had trouble in maintaining consistency in defeating the tests agents.
High execution time

Decisions:
To stick on Python after performing a few experiments with Java.
Optimisation of the code.
To add comments in the code.
To Make sure the AI agent is defeating all the test agents.

Result Analysis before submission:

We run the code at least 20 times for each test agent. The latest result is displayed below.

- Bob - Defeated in 26 turns. Execution time was 105.129s . Time per move was 8.001s .
- Rita - Defeated in 22 turns. Execution time was 88s . Time per move was 8.001 .
- Jimmie - Defeated in 28 turns. Execution time was 175.844s . Time per move was 8.001s .
- Joni - Defeated in 37 turns. Execution time was 163.177s . Time per move was 8.001 .

Time per move before submission was set to 8 seconds.