Report for EWN Game Implementation Homework 2 & Final Project

Name: Nguyen Minh Trang
Student ID: 411021365
Course Name: Theory of Computer Games

1. How to compile the programs

In the terminal, navigate to the directory and compile the `hw2.cpp`, `final.cpp`, and `sample.cpp` (given sample code) files using these commands:

```
g++ -o hw2 main_hw2.cpp hw2.cpp -std=c++11
g++ -o final main_final.cpp final.cpp -std=c++11
g++ -o sample main_sample.cpp sample.cpp -std=c++11
```

The results of the compilation are the `.exe` files, which will be used for testing with the game platform. Then, to test two programs against each other, navigate to the directory that has the package provided by the platform, open the `open` and `enter` folders in two separate terminals, and type in the following command:

```
java -jar Launcher.jar -cli
```

Follow the instructions to fill in the information and the paths to the `.exe` files before starting the game.

2. Implementation

a. HW2: UCT Algorithm (MCTS with UCB)

For Homework 2, the Monte Carlo Tree Search (MCTS) algorithm was implemented with Upper Confidence Bound (UCB) to select the optimal moves. The algorithm uses four key helper functions to explore the game state and determine the best possible move.

Key Functions:

- **Select:**
  - **Purpose:** Navigate the tree to find the most promising node using the UCB formula.
  - **Formula:**

$$\text{UCB} = \frac{\text{wins}}{\text{visits}} + C \cdot \sqrt{\frac{\ln(\text{parent\_visits})}{\text{visits}}}$$

    where C balances exploration and exploitation.
  - **Implementation:**
    - Starts at the root node and recursively selects child nodes with the highest UCB score.
    - Ensures that unvisited nodes have the highest priority.
  - **Outcome:** The algorithm progressively learns by selecting paths with high potential.
- **Expand:**
  - **Purpose:** Add child nodes to the current node by generating all possible legal moves.
  - **Implementation:**
    - Calls a helper function to generate moves based on the current board state and player's pieces.
    - Each child node represents a new game state resulting from a legal move.
  - **Outcome:** Increases the breadth of the search tree by expanding new states.
- **Simulation:**
  - **Purpose:** Randomly play out a game from the expanded node to the terminal state to estimate the value of the node.
  - **Implementation:**
    - Performs random legal moves for both players until the game reaches a win/loss/draw state.
    - Assumes uniform randomness during simulations, which introduces stochasticity to explore diverse paths.

- **Outcome:** Provides a win/loss estimate for the current node, which is then propagated upwards.

- **Backpropagate:**
  - **Purpose:** Update the win/loss statistics of all parent nodes along the path from the simulated node back to the root.
  - **Implementation:**
    * Iteratively updates the visits and wins for each parent node based on the result of the simulation.
    * Adjusts the statistical data of each node to refine UCB calculations for future selections.
  - **Outcome:** Helps the algorithm learn from previous explorations by improving the UCB scores for promising nodes.

b. Min-max with Alpha-beta Pruning (Star0 Algorithm)

For the final project, a Min-max algorithm with Alpha-beta pruning was implemented, incorporating the Star0 algorithm to handle chance nodes. This approach factors in the probability of dice rolls to make more informed decisions.

Key Functions:

- **F_MAX (Max Node):**
  - **Purpose:** Evaluate the maximum utility for the player at the current state.
  - **Implementation:**
    * Iterates through all legal moves and recursively calculates the utility values of successor states using Alpha-beta pruning.
    * Prunes branches where the utility cannot exceed the current best value.
  - **Outcome:** Identifies the optimal move for the player while minimizing computational overhead.

- **G_MIN (Min Node):**
  - **Purpose:** Evaluate the minimum utility for the opponent at the current state.

- **Implementation:**
  - Similar to F_MAX but selects the minimum utility among the successor states.
  - Uses Alpha-beta pruning to skip branches that do not affect the outcome.
- **Outcome:** Models the opponent's optimal response, which is used to calculate the player's best move.

- **Star0_F (Max Chance Node):**
  - **Purpose:** Calculate the expected utility for the player at a chance node.
  - **Implementation:**
    - Considers the probability of each dice roll to compute the weighted average utility for all possible outcomes.
    - Rolls back to the Max Node (F_MAX) for evaluation of subsequent moves.
  - **Formula:**
$$\text{Star0\_F} = \sum_i p_i \cdot \text{Utility}_i$$
where p_i is the probability of dice roll i.
  - **Outcome:** Incorporates randomness into the evaluation to simulate realistic game conditions.

- **Star0_G (Min Chance Node):**
  - **Purpose:** Calculate the expected utility for the opponent at a chance node.
  - **Implementation:**
    - Similar to Star0_F but models the opponent's perspective by evaluating their expected utilities.
    - Rolls back to the Min Node (G_MIN) for further computations.
  - **Outcome:** Ensures that both players' perspectives are accounted for when dice rolls introduce randomness.

- **Evaluation Function:**
  - Considered positional advantages based on proximity to the target and control over key areas.
  - Incorporated piece values and mobility.

3. Self-play results of MCTS vs Star0 Algorithm

- Self-play results: 44W / 56L

4. Self-play results vs sample code (randomness)

- **MCTS vs. randomness:**
  - Results: 58 W / 42 L
- **Star0 vs. randomness:**
  - Results: 70W / 30L

5. Evaluation Function Design for Min-max Alpha-beta Algorithm

The evaluation function was designed in accordance with a study on EWN evaluation function (2013) [1] with the following criteria:

- **Distance to Goal:**
  - **Purpose:** Encourages pieces to move closer to their respective goals.
  - **Implementation:**
    - For Red pieces (maximizer), the distance is measured as the maximum of rows or columns to the bottom-right corner.
    - For Blue pieces (minimizer), the distance is measured as the maximum of rows or columns to the top-left corner.
    - Weighted by a constant `DISTANCE_WEIGHT` to prioritize goal proximity.
  - **Impact:** Pieces closer to their goals contribute positively, while those farther away reduce the score.
- **Adjacent Opponents (Probability Factor):**

- **Purpose:** Penalizes pieces based on the number of adjacent opposing pieces, simulating a higher risk of being captured.
- **Implementation:**
  * Iterates through all adjacent squares to count opponents near each piece.
  * Blue pieces are penalized for adjacent Red pieces, and vice versa.
  * Weighted by a constant `PROBABILITY_WEIGHT` to emphasize the strategic value of safe positioning.
- **Impact:** Positions with higher opponent proximity reduce the score, reflecting increased risk.

- **Attack and Defense Weights:**
  - **Purpose:** Balances offensive and defensive strategies by prioritizing the importance of Red and Blue pieces differently.
  - **Implementation:**
    * For Red pieces (maximizer), scores are scaled using `ATTACK_WEIGHT`, emphasizing their offensive role.
    * For Blue pieces (minimizer), scores are scaled using `DEFENSE_WEIGHT`, emphasizing their defensive role.
  - **Impact:** Encourages aggressive play for Red and cautious positioning for Blue.

- **Remaining Pieces:**
  - **Purpose:** Considers the material advantage by accounting for the number of remaining pieces for each player.
  - **Implementation:**
    * Adds a bonus to the score for the number of captured opponent pieces.
    * Scaled by `ATTACK_WEIGHT` for Red and `DEFENSE_WEIGHT` for Blue.
  - **Impact:** A player with more remaining pieces gains an advantage, aligning with material dominance in the game.

Final Calculation: The evaluation function aggregates these factors into a single score:

- Positive scores favor Red (maximizer).
- Negative scores favor Blue (minimizer).

To ensure the evaluation reflects the current player's perspective:

- If the AI is playing as Red, the score is returned as is.
- If the AI is playing as Blue, the score is negated to reflect the minimizer's strategy.

6. Possible Enhancements

Resource and Time Constraints: Due to the limited computational power of the machine and strict time constraints for development, the implementation opted for smaller parameters to ensure the feasibility of the solution. These constraints influenced the following decisions:

- **MCTS Parameters:**
  - The number of iterations in the Monte Carlo Tree Search (MCTS) was kept relatively low.
  - This decision traded off some potential accuracy in decision-making for faster computation, allowing the AI to make moves within reasonable time limits.
  - The current MCTS implementation employs random move selection during the simulation phase. This approach, while straightforward, introduces variability and often fails to explore the most promising paths effectively.
- **Star0 Algorithm:**
  - The Star0 implementation uses a shallower depth for search layers, reducing the complexity of evaluating chance nodes.
  - While this reduces the depth of strategic insight, it ensures the algorithm remains computationally efficient.

Future Work:

- **Heuristic-Driven Simulations:** Researching and implementing heuristic functions to guide the simulation phase of MCTS could significantly improve performance.

- **Parallelization:** Leveraging parallel computing to evaluate multiple nodes simultaneously in MCTS or Star0 could allow for deeper searches without significantly increasing time complexity.
- **Enhanced Evaluation Functions:** Expanding the evaluation function to account for additional factors, such as positional dominance or strategic threats, could further refine the AI's decision-making capabilities.

References

1. Wang, Z., & Zhang, Y. (2013). Research of Einstein würfelt nicht! Monte Carlo on board game AI. *Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG)*, pp. 1–7. Retrieved from `https://www.semanticscholar.org/paper/Research-of-Einstein-w%C3%BCrfelt-nicht!-Monte-Carlo-on-Wang-Zhang/68b228d112094bd48932932bd85b2b2`