Project: Bitwise Dice Duel AI

Overview

In this project, you will design an AI agent to play a custom two-player stochastic board game. The emphasis is on **game tree search (Minimax and Expectiminimax)** and applying **evaluation functions** to guide the search.

Your grade will be based mainly on your **Expectiminimax implementation, evaluation function, and analysis of search behavior**, not on trivial mechanics such as drawing the board or coding bitwise operations.

Game Rules

- **Board:** ~40-square linear track. Each player has **1 token**, starting at square 0. The first player to reach the end wins.
- **Dice:** On each turn, roll 3 eight-sided dice (values 1–8).
- **Moves:** Choose two dice and apply one of:
  - Bitwise AND ($\wedge$)
  - Bitwise OR ($\vee$)
  - Bitwise XOR ($\oplus$)

    The result is the number of spaces to move your token.
- *Example: roll (3, 6, 7)*
  - $3 \wedge 6 = 2$ → move 2 spaces
  - $3 \vee 6 = 7$ → move 7 spaces
  - $3 \oplus 6 = 5$ → move 5 spaces
- **Collision:** If you land exactly on your opponent's token, their token is sent back to start.

Minimax and Expectiminimax

You have already studied **decision trees**, where results at the leaves are propagated upward. Game trees are similar, with two new ideas:

- **Minimax:**
  - Alternate turns between a **maximizing player** (you) and a **minimizing player** (your opponent).
  - The leaf value is a score (e.g., win = +1, loss = –1, ongoing = heuristic).
  - Each level of the tree chooses the max or min of its children, depending on whose turn it is.
- **Expectiminimax:**
  - Adds **chance nodes** for dice rolls.
  - At chance nodes, compute the **average expected value** over all possible outcomes.
  - This allows you to handle randomness in games like Bitwise Dice Duel.

Requirements

- Implement game state, move generation, and transitions.
- Implement **Minimax** search for deterministic play.
- Extend to **Expectiminimax** with chance nodes for dice rolls.
- Design an **evaluation function** that considers:
  - Distance to the goal (farther = worse, closer = better).
  - Collision opportunities (reward bumping the opponent, penalize being bumped).

    - Win/loss states (assign very high/low values).
- Provide **instrumentation** (logs, counters) to show node counts at different depths.
- Book a **10-minute final check-in** with me:
    - Demonstrate your program.
    - Explain any line of code I ask about and justify its purpose.

Tips

- Start small:
    - Build a basic interface to play manually.
    - Implement the GameState struct and move generation.
    - Write Minimax with a simple evaluation function.
    - Extend to Expectiminimax for dice rolls.
- Use **depth limits** to control branching.
- Instrument your code: count how many nodes are expanded.
- Displaying the board:
- Use Unicode tokens:
    - `cout << “🁣”; // Empty Space`
    - `cout << “🁤”; // Computer Occupies`
    - `cout << “🁪”; // Player Occupies`
    - `cout << “🁫”; // Both Occupy`
- To clear the screen each turn:

```
#ifdef _WIN32
    system("cls");
#else
    system("clear");
#endif
```

Stretch Goals (+5% each)

1. **Alpha–Beta Pruning:** Add pruning to reduce the number of nodes explored. Show logs comparing node counts.
2. **Graphical Interface:** Replace text display with a simple graphics library (e.g., SFML or Raylib).

Rubric (100 points + up to 10% extra credit)

- **Core mechanics (10 pts)**
    - Board, move generation, and transitions (5 pts)
    - Playable manual interface (5 pts)
- **Minimax implementation (30 pts)**
    - Recursive minimax search (10 pts)
    - Working evaluation function (20 pts)
- **Expectiminimax implementation (35 pts)**
    - Correct chance node handling (20 pts)
    - Reasonable stochastic play (15 pts)
- **Instrumentation and explanation (25 pts)**
    - Node count logs at different depths (10 pts)
    - Clear explanation of code at final check-in (15 pts)

- Note: I reserve the right to award 0 points for the entire project if you cannot explain and or justify any two parts of your code.

Stretch Goals:

- Alpha–Beta Pruning (+5%)
- Graphical Interface (+5%)