Adversarial Search Methods for Two-Player Board Games

Artificial Intelligence

Turma 10 - Grupo A1 91

Ntsay Zacarias up202008863

1 de Abril de 2024

Contents

1 Introduction

The objective of this project was to explore and highlight the differences in the implementation of various adversarial search algorithms in board games. Among these, I selected the somewhat lesser-known game, **Focus**, for analysis. Focus represents a zero-sum, inherently adversarial game with a complex state space, making it an ideal candidate for the application of techniques such as MiniMax, Monte Carlo Tree Search (MCTS), and their variants. The game's deterministic nature further complements these strategies, enabling precise planning and execution of moves. My primary goal was to deepen my understanding of Focus to develop and implement effective heuristics that capitalize on the unique aspects and rule nuances of the game.

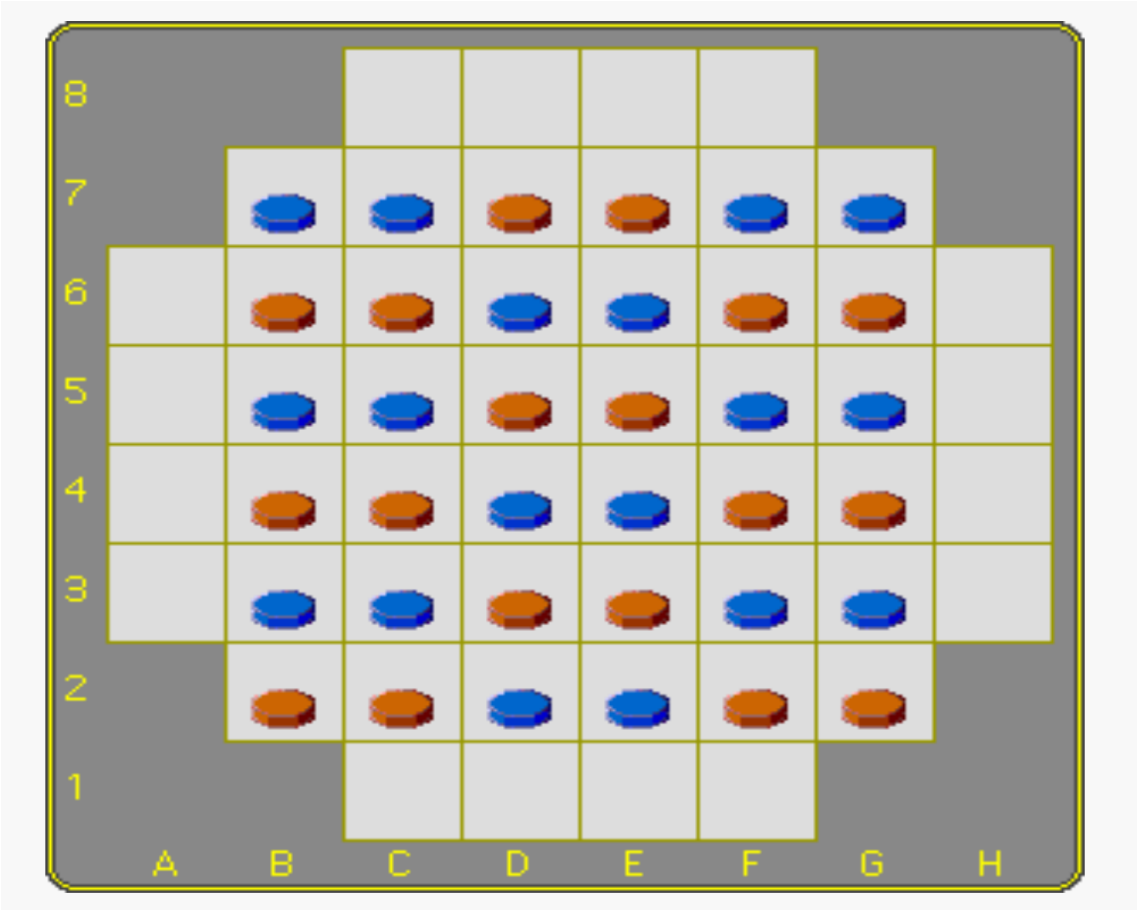

Figure 1

Focus is a game played on a square board with some of the corners removed. It is an abstract strategy game in which players attempt to make moves and capture pieces in such a manner that their opponent(s) have no moves remaining

1.1 Rules

Each player has 18 pieces, let's call each of them a **man**, and they can stack on each other, creating stacks of **men**.

- A stack is owned by the player whose color is on top
- On his turn a player moves a man or a stack of men horizontally or vertically, based on the number of pieces to be moved (Stacks may be split in this process)
- When a stack grows over five pieces tall, the remaining men are removed from the bottom of the stack
  - Men of a player's own color become 'reserves' to be re-entered into the game at a later time
  - Pieces of the opponent's color are captured
- Instead of moving a piece, a player may choose to enter one of his reserves on any square of the board, whether vacant, or occupied by a piece of either color

1.2 Objectives (End States)

- A player wins when his opponent cannot move a piece, nor enter a reserve on the board

2 Related Work

3 Problem Formulation

3.1 State Representation

A state is represented as a 2D matrix, where each cell (i, j) contains a list of integers representing the stack of pieces at that position. Player 1's pieces are denoted by 1, and Player 2's pieces by 2. Non-playable cells are represented by `None`.

3.1.1 Initial State

The initial state is the board setup with each player's pieces in their starting positions, and corners of the board removed (set to `None`).

3.1.2 Operators

Move

- **Preconditions**: The source cell contains a stack, the move is within board limits, and it adheres to the game's movement rules.
- **Effect**: The stack is moved or split, capturing or merging with stacks at the destination.
- **Cost**: 1

Enter Reserve

- **Preconditions**: The player has at least one piece in reserve.
- **Effect**: A piece from the reserve is placed on any cell on the board, starting or adding to a stack.
- **Cost**: 1

Winner	Moves		Captures		Avg Move Time (s)		Time to Finish (s)
1	46	46	5	5	0.01	0.02	5
1	40	40	5	5	0.02	0.02	4

Table 1: AB vs AB (Random) depth=1

Capture

- **Preconditions**: A stack grows over five pieces tall after a move.
- **Effect**: Pieces exceeding the stack limit are removed. Own pieces become reserves; opponent's pieces are captured.
- **Cost**: Implicit in the move cost.

3.1.3 Objective Test

The objective is to reach a state where the opponent cannot make a valid move or enter a reserve, thereby winning the game.

4 Implementation

This game was made using Python3 and Pygame for the GUI

5 Results Analysis

In my simulations, I noticed a couple of critical behaviors when running two MiniMax algorithms against each other, often leading to infinite loops due to a lack of randomness in decision-making. To address this, I made a slight adjustment in the heuristic to break out of repetitive states.

The main heuristic I used focuses on dominating the board by prioritizing control over stacks of pieces and encouraging moves that lead to captures. This strategy is what fundamentally drives the gameplay, aiming for a more aggressive and piece-capturing approach.

However, when I tried implementing Monte Carlo Tree Search (MCTS), it didn't perform as well as I hoped. The Alpha-Beta (AB) pruning with its robust heuristic consistently outperformed MCTS in both efficacy and speed, winning all games. I think that the MCTS implementation might not have fully aligned with the game's rules.

I also observed that AB pruning significantly improved performance. Yet, as I increased the search depth, the returns started diminishing, particularly beyond depth 2. This made me realize that depth 2 is the sweet spot for balancing efficiency and strategic depth, providing the best compromise between fast decision-making and competitive gameplay.

Interestingly, the MCTS didn't fall into the same traps of circular solutions as MiniMax did, probably because its simulation-based approach encourages exploring different moves from previous ones. This difference highlighted a unique advantage of MCTS, despite its overall lesser performance compared to AB pruning in my simulations.

Winner	Moves		Captures		Avg Move Time (s)		Time to Finish (s)
2	80	81	0	5	0.04	0.05	13

Table 2: AB vs AB (Random) depth=2

Winner	Moves		Captures		Avg Move Time (s)		Time to Finish (s)
2	94	95	5	5	0.15	0.23	44
2	53	52	5	3	0.21	0.21	28

Table 3: AB vs AB (Random) depth=3

Winner	Moves		Captures		Avg Move Time (s)		Time to Finish (s)
MiniMax	22	22	2	0	0.2	0.02	10

Table 4: AB (Random) vs MiniMax depth=1

Winner	Moves		Captures		Avg Move Time (s)		Time to Finish (s)
MiniMax	21	22	2	1	0.30	1.59	30

Table 5: MCTS vs MiniMax depth=1