Project Proposal - Playing Blokus with MaxN

Team Members: Pablo Ordorica Wiener (UNI: po2311)

Course: COMS W4995 - Parallel Functional Programming

- **Semester:** Fall 2025
- **Instructor:**
    - Maxwell Levatich
    - Stephen Edwards
- **TA:** Kyle Edwards

Introduction and Motivation

Blokus (pronounced "block us") is a 2 or 4 player perfect-information turn based game. Players take turns placing their same color "Tetris-like" pieces on the 20x20 board. Each player gets the same 21 pieces of the same color. In the player's first turn they start by placing a piece filling their corner. Then in the subsequent turns, they continue putting pieces on the board following two rules:

1. Each piece must touch at least one other piece of the same color but only on the corners!
2. A piece cannot overlap any existing piece on the board.

The game ends when no one can play another piece. The player with the least amount of squares in their pieces left to play wins that round - like in domino[^1] [^2].

The MaxN[^3] algorithm is an extension of the Minimax algorithm[^4], the classical method for playing perfect-information turn based games with two players. MaxN is for games with two or more players. Instead of alternating min/max nodes for the two participants as in Minimax, each MaxN node represents a move for one of the N players, and the algorithm chooses the move that maximizes *that* player's socre in the resulting N-tuple of evaluation scores. Perfectly suited for Blokus' format.

Objective

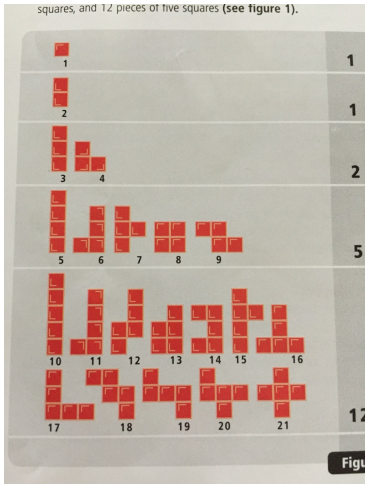
My goal is to parallelize playing Blokus with 4 players using the MaxN algorithm in Haskell with both the Eval and Par Monads.

Game Plan

> Pun intended haha.

Blokus Game Representation

The original board consists of a 20x20 square board and each player receives the following 21 pieces which are polynominoes[^5] from 1 to 5 squares in fixed shapes.



I plan to use the board dimensions and the piece set as knobs to scale the "hardness" for the algorithm implementation.

Sequential MaxN

The MaxN algorithm is similar to the Minimax in that it builds a tree for all possible game states that would result from a player's turn. The difference comes in picking the move: players pick the move that maximizes their chances of winning without regard for how it benefits or hurts the other players.

Pseudo-Code

Here's the pseudo-code for the recursive sequential MaxN algorithm.

```
# For N players
def maxn(node, depth):
    if node is terminal or depth == 0:
        return evaluate(node)  # returns N-tuple (scores for each player)
    children = successors(node)  # all legal moves
    results = [maxn(child, depth-1) for child in children]
    current_player = node.player
    # Choose child maximizing this player's score in result tuple
    best = argmax(results, key=lambda tup: tup[current_player])
    return best
```

> Note
>
> **Sequential Optimizations**
>
> Alph-beta pruning is an optimization technique used with Minimax to reduce the tree size. Unfortunately, alpha-beta pruning is not available for MaxN because, each player maximizes their own scoreand there is no single value for confidently pruning branches as in two-player Minimax.

Parallelization Opportunities

Since MaxN constructs a tree, the primary parallelism is in independently exploring subtrees corresponding to

different moves at each node (i.e., at each branching point, each move can be "sparked" as a parallel computation). Parallel evaluation can occur at the root, or recursively to some fixed depth (breadth vs. depth-based parallelism).

Planned approaches include:

- Parallelizing child node evaluation using Haskell's par, pseq, or the Strategies library (parList rseq, parListChunk, etc.). Each child state is computed independently, making this a natural fit for work-stealing multi-core scheduling.
- Depth-based control: Only parallelize up to a certain tree depth to control work granularity, avoiding too-fine sparks which may incur more overhead than benefit.
- Monadic coordination: Compare Eval vs Par Monad techniques for expressing parallel computation (as explored in the literature), focusing on modularity and composability of the parallel search logic.

References

- [^1]: Blokus (2000) | Board Game Geek
- [^2]: How to Play Blokus in 1 minute (Blokus Game Rules) | Tabletop Duo YouTube Video
- [^3]: C. Luckhardt and K.B. Irani. An algorithmic solution of n-person games. In Proceedings of the 5th National Conference on Artificial Intelligence (AAAI), volume 1, pages 158–162, 1986.
- [^4]: Minimax | Berkley - Introduction to Artificial Intelligence
- [^5]: Polyominoes 101 - The Absolute Basics