# Connect-M Implementation

Nash Henry, Eric DiGiocchino

nwh@students.uwf.edu

etd7@students.uwf.edu

*Abstract –* **The popular kids game Connect 4 poses interesting challenges for students. In this document we encounter a version of the Connect 4 problem - Connect M. Rather than the traditional 6 x 7 board, students had to generate a board of size N, and build an artificial intelligence using the Minimax algorithm that could connect M pieces contiguously.**

**Keywords – Minimax, Tree, Alpha-Beta, Pruning, Game**

1. Introduction

In the beginning of the game the user determines several factors: board size, number of contiguous pieces required to win, and indicate the first player. From there the program must be able to track players and pieces, verifying a win condition has or has not been satisfied. Should either player win, the program should put a prompt to the screen marking which player won. If there is no winner when the board is full, the game should show the game was a draw and no player won.

## Game Begin

When the user starts the program they should enter the three command-line arguments. These arguments are the axis size of the board **N**, the number of contiguous pieces required to satisfy a win condition **M** and the first player **H**.

**N** {3 ≤ **N** ≤ 10}

**M** {2 ≤ **M** ≤ **N**-1}

**H** {H = [0,1]} 1 = human moves first

## The Board

After receiving the command-line arguments the game generates a board. The board is a NumPy matrix of the shape: (**N**, **N**, 0) and is initialized with zeros. NumPy is an obvious platform for this application for its ability to handle simple and semi-complex matrix operations with low runtimes.

## Heuristic Evaluation

For strategic evaluation, the game ignores the relationship between a game piece and a player. This allows the program to consider blocking and advancing moves simultaneously without requiring extra compute time.

Within the assumption that piece association does not matter, directions are selected for evaluation within the game parameters. Due to the fact a piece cannot be placed under an existing piece, we can ignore the up direction when evaluating, so the remaining linear directions are selected: Left, Right, Down, Diagonal Up and Left, Diagonal Up and Right, Diagonal Down and Left, and Diagonal Down and Right.

In any of the listed directions the program calculates the heuristic value of a direction as:

**For some direction d:**

**:**

For extensibility the piece association is maintained in a tuple (h, p) where h is the heuristic value and p is the piece association. But, since association is ignored in this case, the heuristic values are summed and returned to the calling function.

A point of note: since there is no up direction to calculate for, the heuristic for the downward direction has a bias placed on it to give it focus. This voids most bias in the strategic evaluation.

## Game Tree Formulation and Evaluation

Given the game has an adversarial nature, it makes sense to use an adversarial solution to solve the problem. The description of the project required a Minimax search algorithm with alpha-beta pruning.

Implementing the minimax algorithm is simple enough. But can accumulate time and space complexity given the wrong implementation.

The most effective way to evaluate the tree is through recursive means.

At the bottom of our recursion tree the strategic score is calculated for some position passed into the function. The tree is then evaluated from the leaf nodes to the stem. Each odd-valued layer is a maximization layer, and each even numbered layer is a minimization layer.

Minimization swaps the replaces the beta value for a layer with the least valued heuristic. While maximization swaps the alpha value for the highest valued heuristic.

Rather than iterate across all branches of a tree to find the best solution, which could result in NN branches; the evaluation of max and min happens as the program iterates. To avoid overflow, we implement the Alpha-Beta pruning condition to avoid parsing branches that are unnecessary.

Our algorithm evaluates leaf nodes at the first parent layer, which will always maximize the solution. Each successive parent layer does the operation associated with it as determined by depth in the tree, also passing coordinates as each layer is updated.