OTHELLO

CSC 242 final project

By Anis Kallel, John Donner, Nathan Contino

aka Team Daybreak

Our project implements the minimax best-move chooser alongside a heuristic of our own design, shittyHeuristic, to create an AI for the popular tile-flipping game Othello.

As Dr. Frankenstein once said, "I've created a monster."

This zip folder contains two folders:
	Human Friendly, which contains a version of the game that prints out a board with each move as well as a friendly suggestion of valid moves for a human player
	Tournament Friendly, which only prompts/prints moves in an “X Y\n” format.

MINIMAX

Our implementation of minimax works primarily through the Game3 class, which is assigned a depth by its caller (in our case, the caller is always an instance of class Othello). This class continuously calls itself on every possible move at decreasing depths until it reaches depth 0, at which point each instance of Game3 (there will be quite a few) calls a new instance of Game using every available move. Game contains our actual move-quality heuristic, titled "shittyHeuristic," which assesses the quality of a particular move. Each instance of Game3 then "harvests" the validity of each branching game state, recording the worst and best moves at a particular time. As the Game3 calls return, their parent Game3 calls check the best of their worst possible moves and the worst of their best possible moves. This continues up the "tree" of Game3 calls until we have a final "chosenValue" (a veritable "minimax") for our given gamestate. Our AI then chooses the corresponding move aligned with "chosenValue" that indicates the most promising possible value of our given move choices.

GAME

Every good minimax algorithm needs a good heuristic, and we've got the best one we could whip up in two weeks: shittyHeuristic. ShittyHeuristic is rooted in a (relatively) reliable heuristic designed to emulate a very basic AI- it's merely a 2D array that

assigns each tile in the game a particular value regardless of the current board state. Corners are always weighted most heavily, followed by edge tiles, with tile values decreasing as you approach the center of the board. The one exception to this rule is corner-adjacent tiles (of which there are 12), which are actually weighted *negatively*, since it's in a player's worst interest to take a corner-adjacent tile, leaving a corner open to the opposing player. ShittyHeuristic augments this tile value system with a "frontier weight" system that either adds or subtracts value to a given board state based on the number of frontier tiles belonging to the current (AI) player, which directly correspond to movement choices for the opposing player. ShittyHeuristic doesn't stop there, though: when an instance of Game is created, *a new game begins at that point, played out entirely between two players using the two-pronged tile value heuristic described above. The final state of the board is then evaluated, returning a negative number of tiles if the AI player lost and a positive number of tiles if the AI player won. Using this heuristic, we can roughly estimate the number of tiles the AI would receive if he were to play optimally in that given branch of the game tree. This heuristic is very different from the traditional "tile counting" heuristic - and, we believe, far superior- because it actually estimates board end states for a particular game. Because of this approach, the heuristic isn't particularly potent in early moves of the game, but becomes more and more clever as the game goes on because the "projected" board end-states become more and more accurate.

As a result of time and space restraints, our game can only travel to a depth of 4 or so in reasonable time. However, this is not unusual- many of the best minimax algorithms only travel 3 to 5 moves in the future due to the high branch factor. It’s also very important to note that our “DEPTH” constant actually represents the real depth minus two: one level can be attributed to the “zero” level (where depth = 0) and another may be attributed to the “leaf” level of “Game” objects, each of which report their winning state (computer tiles - player tiles). Our heuristic calculates the percentage of wins projected with our shittyHeuristic v. shittyHeuristic Game object for every single move up to DEPTH + 2 moves in the future. Reasonable results have been seen at a depth of 4 (where DEPTH = 2) in reasonable time, so the game defaults to that in all but the most extreme of circumstances.

Currently, alpha beta pruning is in alpha stages. While pruning is a very attractive concept, our current percentage implementation makes it difficult (as you’d have to calculate percentages on the fly, adding a lot of unnecessary (and expensive) division). Instead, we’ve focused on our unique heuristic that we believe will give us an edge over traditional tile-counting implementations of minimax for Othello AIs. Instead, our AI plays out projected leaf nodes at depth + 2, offering insightful information beyond mere tile counts- for instance,

a player with ONLY the four corner tiles near the end of the game is much better off than a player with 50% of the board, but no edges or corners. Our algorithm focuses on many often-neglected aspects of the Othello game that go beyond simple tile counting, such as strategic tile values and player/computer frontier counts, giving us what we believe is an advantage in AI tournaments. In an environment where many students write the same algorithm with the same efficiency and the same heuristic, a slightly different heuristic can make a world of difference, and that’s why we chose shittyHeuristic.