The driver class for this package is checkers.c; this program creates an instance of a game of checkers, as well as a small set of checker-playing AI's that a player may compete against or pit against one-another.

This project was much more challenging than we anticipated it would be. The first several days after its assignment were spent wading through the C-code to figure out what everything did and how the game was represented in memory.

Our actual implementation began by extended the material-advantage heuristic. Pieces were and always have been weighted at 1.0, and Kings at 1.7. Immediately, we modified the scoring to be a ratio instead of a difference, which is more conducive to late-game trading. At that point, we had essentially exhausted our checkers intuition and moved to some basic research.

We learned that certain positions and piece orientations are more valuable than others when evaluating a board. In particular, there are two "rings" in the center of the board, which should be valued more highly when occupied than other nearby squares. For this, we simply added a weight, such that an additional bonus was awarded to a player with a piece occupying the center-most ring, and a slightly smaller bonus if the outer ring was occupied.

With respect to orientations, while the IDS would naturally calculate the value of some piece configurations over time, we also found that some setups could be immediately evaluated and incur bonus scores. For instance, a piece occupying a space adjacent to an allied piece would be awarded a small bonus (for its defensive strength). A piece positioned such that the the turn player might move into a capturable position in the following turn would be awarded a

marginally larger offensive bonus. This slight inequality was to encourage forcing the action and avoiding loops.

Unfortunately, this didn't play out quite as planned. While this implementation could consistently beat the random player, it would always lose to the depth-5 player, and it took some adjustments of how we weighted different positions and orientations to get to a point where, instead of losing, we looped against it instead. A minor improvement, but still problematic nonetheless.

Fixing this proved difficult; changing weights, swapping indexes while iterating, increasing depth, swapping heuristics based on game state, all were ultimately ineffective. In the end, we reverted to a simpler state andre-introduced some of the above changes, which got us to the point where we could beat d5 consistently, but only if we went second. If we went first, we would loop with d5.

The solution to this was a stupid one: in the bit of our evaluation function where we give bonuses for occupying the first and second center rings of the board, we were only awarding points for pieces, not kings.

Our final bit of implementation was an early-game move-book, which would skip the evaluation function and simply use a dictionary of moves which we knew were strong openers. With more time, we would extend the dictionary to mid-game and end-game scenarios, too, so that some particularly slow evaluations could be skipped.

Testing the base requirements involved running 5 to 10 3-second games per heuristic change against d5 and random to assess the nature of losses or loops. Naturally, d5 was the most educational. Being checkers novices, we essentially just looked for moves that appeared to turn

the game against us and took our best guesses as to which of our evaluation heuristics was failing, and how we could improve it.

Building the move-book was a process of actually playing against our own implementation instead of pitting it against another AI.