Daniel Chavez, Samuel Apker

Tim Anderson

Artificial Intelligence

10/13/2017

Crafting an Artificial Checkers Player

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. We found that a lot of people were taking a completely opposite approach and highly valuing all of the edges of the board. At face value it seems like a decent strategy but ultimately we concluded that this would be a misstep. They value the opposite edge of the board because that's how a piece is kinged but any material advantage heuristic will take care of that in a more optimal way. The back rank is valued because then the opponent cannot get a king, but this simply leads to voluntarily splitting your forces and at a certain point those back rank pieces will have to leave anyway, and finally they valued the sides of the board because in those positions there pieces can’t be jumped. With this I believe it does more harm than good because it’s a completely defensive position allowing for less opportunity of attack and further splitting your forces. For that exact reason of giving yourself more opportunity of attack we found that the center of the board was highly advantageous.

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. After fixing our original center evaluation it became clear that we could add back in the adjacent ally and capturable enemy heuristic.

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 end-game scenarios, too, so that some particularly slow evaluations could be skipped. Building the move-book was a process of actually playing against our own implementation instead of pitting it against another AI. Online there were a lot of resources that detailed the best strategies in the opening. We only ever found specific move lists for the first move on either side of the board. Having our player choose these specific moves involved us changing the move return process, based on a turn counter, to return a specific int in the move list. Since this data structure wasn't visible it involved hard coding the computer to return a random move in the legal move list and manually iterating through that until we found the move we wanted. This was simpler in the instance where the computer is red since it will always return the one we wanted, which happens to be the move 11 - 15. For the instance where the computer is white we had an optimal follow up for the first 7 moves that the Red player could make.

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.

The final implementation is essentially as follows: the board is evaluated position by position for the presence of a piece or a king. The color which possesses the piece is awarded points to reflect the power of the piece. This was the material advantage aspect of our heuristic. Then, at each position, if there is a piece or king there, in addition to the material score incurred already, that player scores further points in a few additional cases. First, if the there are more than 8 pieces on the board between the two sides, pieces that are in the first center and second center rings of the board are awarded a fairly significant positional advantage (marginally more for the first ring than for the second). Second, if the game is just over halfway done (meaning that there are 11 or fewer pieces on the board between the two sides), positions adjacent to the current position are evaluated, and the player is awarded further points for adjacent allies ($i \pm 1$, $j \pm 1$), or potentially capturable enemies ($i \pm 2$, $j \pm 2$). The former awards less than the latter to encourage forcing the action instead of looping about in a safe position. This captures the notion that it's not enough to have more pieces, they also have to be positioned properly. Finally, and in addition to material and positional advantage, we implemented a move-book for the early-game, which gave us a stronger start.

Our heuristic actually performs reasonably well. In our tests, it always beats random and d5, maintaining a consistent lead throughout. However, other players like rat, diff, and year2, give us issue. We can take the lead against them at certain points in-game, but ultimately end up either looping or outright losing. We have not implemented a solution for this, but a good first step would likely be multi-threading the search process to allow for greater depths to be investigated. We also tested a few games against other students and found we were consistently winning.