Mary Catherine Scott
Trevor Rice
CS 457
Checkers

Initial Player Development

The initial development of the checkers player was based solely on the pseudo code for min-max that was given in the book. It was improved upon when we discussed an initial implementation in class and were given code that would work, but may not be the most efficient or effective. This was very helpful as a start to make sure we could at least say our min-max was working as a min-max, regardless of the heuristics we would begin to implement.

Improving the Player

After the discussion in class about min-max we felt pretty confident that it was working as intended and thus the heuristics would need to be the focus of our time as well as setting up the timer to make sure we did not run out of time. Trevor set forth to improve the timer function while I set off to try to improve the heuristics so that we would hopefully, start winning. Improvements also came as I realized I had typed in the given code incorrectly, once fixed we started winning a lot more games. It's amazing what happens when the board is being looked at correctly rather than not at all.

Testing the Player

Most of our testing was letting the games run. We developed a way to stop the games after so many turns to determine if it was a stalemate or draw. With this tool in place we were able to use scripts to run many games at a time while only printing the end result rather than the entire game. There were times that we wanted to watch an individual game, and others that we just wanted to see if we were winning most, if not all, of the matches. After much thought, I was not sure there was a way to determine if we really could beat random 100% of the time. We could test and test and test and maybe just miss the one scenario where we would lose. But the goal began as beating random for 100% of the games we played against it.

Things began to improve and there were runs that would beat random every time there was not a stalemate. Then we would run it again and it would lose 1 out of the 30 games it played. We would get very close and then begin to feel very far all over again.

There was a rather major set back when we remembered that there would be times that we would be player 2, then it stopped working quite so nicely. This lead to more code reviews to find where we missed something from the other direction of play.

Heuristics Development

- Mary Catherine

I have to admit most of my time was spent on this aspect of the development of our player. I started this process by trying to mimic what I would do to evaluate the board. I soon found that this was harder than I thought and I was missing edge cases, or not thinking deeply into the game enough. It did not take me long to find that I needed to slow down and find a more simplistic way to look at this heuristic. After discussing with a number of people about the project and about checkers itself I realized I had missed a very important aspect of the game. A winning scenario. If there was ever a point in time that I had pieces and the other person did not then I wanted to go in that direction. In the same way, if there was ever a point where I did not have any pieces left, I did not want to make those moves. These heuristics went in and prospects started to look better. It was such a small thing, and yet it made a large difference in how the game played and in how I felt I should be thinking about the development of the heuristics.

I started to look at the different scenarios in regards to numbers of players on the board. The simple heuristic we started with had many flaws. The biggest one I remember running across was there is a huge difference between the board with 6 red and 4 white players and the board with 3 red and 1 white player. Although their difference is the same, the strategies involved here differ, the need to stay alive is much more significant with only 1 player left. I started to look at ratios and actual counts of players not just the difference and where they were located on the board, although that was important as well.

Towards the end we kept getting stalemates along with the occasional loss. I started thinking end-game strategies. What should the player do if there are only two pieces left. It would often put itself in a corner and wait there. This did not seem like the greatest of ideas, so we tried to give it other ideas on how to win. Such as if the other player gets close it could put itself in position to lose. End-game I feel is what is really needed. If I have time to figure it out I will implement it, but I don't think there will be time. I feel that it is needed that once the player has fewer than 4 or 5 pieces on the board the min-max needs to stop and some newer, better endgame strategies need to start, especially once the player is down to one piece. I witnessed a game, where with one piece left my player moved into a position to be jumped. I am still not certain how this happened, but it did and it was very concerning.

- Trevor Rice

For my part in the heuristics development I thought about the ideal board location for my pieces, Mary Catherine had thought about the same idea as well but when I talked to others they mentioned a slightly different strategy. In the end we combined the two. Stay to the edges of the board during the early part of the game and move to control the middle of the board at about the middle of that game. This is solely based on number of turns but maybe we should be considering number of pieces. Additionally, as we move to controlling the middle of the board we also focus on getting pawns to the other end of the board. As soon as they become kings they are no longer influenced by this heuristic.

In the case of kings, we award bonus points to the opposition (making this a bad thing) for having more than 2 kings while awarding us bonus points for having more than 3. One other scenario we've implemented was driving our players close together. This works in many cases but even at the end, for some reason if there is only one player on each side, our player moves to the corner and stays there just moving between positions 1 and 5, causing a stalemate.

How Well It Works

As of right now, the player does not work well. Sadly, we cannot beat random every time. I have run out of time to really figure it out. I fear that I have overthought the heuristics and need to take step back and minimize them a little bit, that or I need to rebuild more things and keep track of more in the state struct. Either way, our player does not beat random every time and it does not beat depth 5 at all. I know this will mean a much lower if not a failing grade over all, but we did spend a lot of time on it and I felt it at least deserves to be turned in.

We do have a functioning program and the heuristics we've implemented seem sound yet they do not do the job. We are not sure at this point if the problem is with the values set or if we are just really poor checkers players. Since we only tend to play against our kids, we don't face too fierce of competition. The rub certainly does come from the hours put into the project. Our min and max functions are working perfectly as well as the timer but the heuristics we've developed are where we are failing. Our goals were to:

1. Maintain the most pieces on the board
2. Control the back row and sides during the start of the game
3. Control the middle of the board after our first seven moves
4. Drive pawns to the be made kings

5. Have more than three kings on the board
6. Avoid letting the opponent have more than two kings on the board
7. Aggressively set the opponent up for being jumped once there we only a few pieces left.

Splitting Up The Work

Trevor did the work on the timer, he had an idea to do it with threading and he ran with it. I did code reviews on it and tried to help with bugs, but in the end Trevor did the timer work pretty much on his own. I was then left to do the heuristics work, although Trevor definitely helped with it a lot. I will admit I think Trevor spend more time on this project than I did, although I know I spend many hours on it as well. The heuristics had me writing and deleting code rather quickly and frequently.

Timer

To ensure we didn't go over the project we decided to implement a similar sleep timer as was used by the checkers.c program but in it's own thread. I (Trevor Rice) tackled this part of the program first. Initially I implemented a global pthread_mutex_t and a global pthread_t but it never seemed to update the global 'times up' variable I'd created. I then tried an array of pthread_t's with no luck. I implemented pthread_cond_t's and still couldn't get anything working. I finally realized I'd my initial idea was correct and that I'd just been casting too big of a net. I went back to using the single mutex and a single pthread_t to call on our timer function. The difference being that I am only initializing the global mutex in the main method and only declare and create the pthread_t locally inside the FindBestMove function. This allows the timer thread to update the global 'times up' variable to be updated for the parent thread to see and react to. When this variable has been updated from zero to one, the depth is then changed to zero so that all of our min and max functions can return the evaluated board state. Initially the timer was inconsistent and I went through several different sleep methods and their man pages before I came across the clock_nanosleep() method which turns out to be consistently accurate and many of the other man pages referred to it as the most ideal solution. We are taking in the SecPerMove value, reducing it by one and create a timespec struct with SecPerMove – 1 seconds and 500000000 nanoseconds. This allows us to have .5 seconds to return the best evaluated board state and make our move. Once this last piece of the puzzle was put into place, our timer worked flawlessly. Even when setting the depth to as much as 20 levels we have yet to run out of time.

Last Minute Details

After our initial submit we set to work to fix what was not working properly. We soon found that we had completely forgotten to implement an iterative deepening function. Ours was just a deep search. This initial switch was not as helpful as we had hoped. I (Mary Catherine) decided it was time to drop the heuristic back to it's most basic position. Once I did this everything started to fall into place. We had a stop feature for when a winning move was found, after our discussion with you this got taken out as it was causing a lot more stalemates than we realized. After that a few more heuristics were added and taken away. At this point our player is doing much better. It beats random every time, with the very rare time out. We also beat d5 more than we lose and sometimes more than we stalemate.