AI Project Milestone Report

Jessica Fedor, Marc Bacvanski
Dec. 3rd, 2021

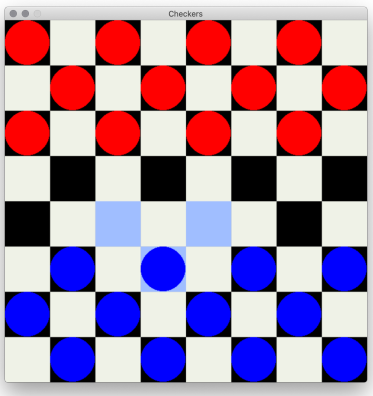
1. Report: What have you done so far? What is the current state of the project? To ensure best use of time, we recommend treating this question as a partial draft of the final report – ideally, if your project does not change, you can write 1–2 pages for this section in each milestone report, and then copy and paste them into the final report.

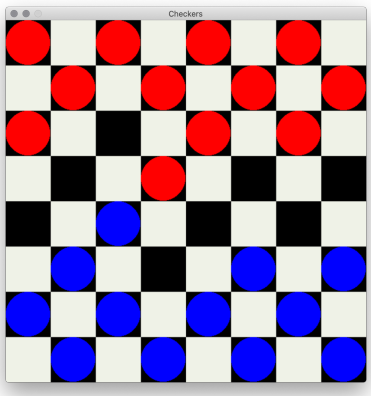
So far, we have completed a working checkers UI that allows a human to play against an AI. Our current algorithm runs minimax with up to depth 3, with a simple evaluation function used when the depth limit is reached. Any higher depth causes the AI to run for longer than 10 seconds for each turn (likely due to the high branching factor), so we only test it with depths 1, 2, and 3.

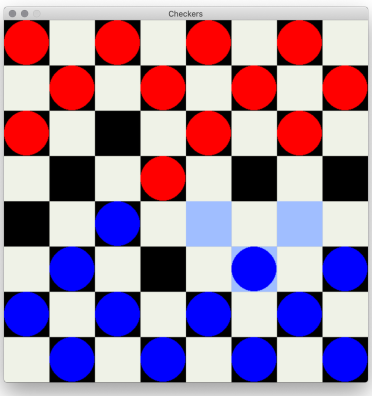
We wrote a new GameState class to hold information pertinent to a checkers game state. This class does not deal with the interface between the game action and the visual board (we didn't write that code - see below for specifics), but instead manages how the game state changes after moves are made.

Currently, our evaluation function simply returns how many more pieces the agent has than the opponent has left on the board. This means higher value is given to states where the AI's color (red) has more pieces on the board and the human player has fewer blue pieces on the board. This evaluation function causes the AI to choose actions that "capture" the human player's pieces, since that decreases the number of blue pieces on the board by 1 for each jump/capture. However, sometimes the AI will choose to not jump. This occurs when the minimax algorithm has detected that, should the AI jump, then the "min" action for the human would be to jump the AI immediately after. Since this would result in both players losing pieces, and therefore the ratio remaining unchanged, the AI does not necessarily detect this is a high-value move.

We found a checkers UI from a public GitHub repo that seemed to work well for our purposes. We adapted the code relevant to running and rendering the game itself. The code for the AI agent's algorithm was written entirely by us. The UI displays the current game state and makes it pretty easy to play, considering all you have to do to make a move is click on the piece to move, observe your move options (highlighted in light blue on the board), and then click whichever highlighted square you want to move the piece to (not written by us).







We do not have any tests for our game as of right now, but plan to develop them in the next steps.

2. Reflect: Did you achieve your milestone(s)? If not, why not (e.g., some part was challenging because of [insert challenge here] and hence took longer, some other task became irrelevant)? Did you do something extra that you did not mention in your proposed milestone(s)?

We did indeed achieve our milestone. We have produced a working checkers environment using Python, and have one simple algorithm working that plays checkers against a human. The algorithm isn't very smart yet, but we expected that for our milestone. It still managed to beat Marc once.

The most challenging part was translating our formulation of the search problem into code. We wanted to build our search algorithms to be general enough to work on any problem type, and so wrapping the state of the board in a GameState class was key. We modeled nodes in our state space as configurations of the board with a given player's turn, and with methods to get the list of possible actions and the next state given a particular action.

We based our implementation of minimax search on the algorithm pseudocode presented in the lecture notes, and our own versions of minimax in past programming assignments. This was very helpful in our understanding of how minimax would translate into a checkers game and increased our efficiency, which helped us reach our milestone.

We hope that given the framework we've developed en route to our milestone, it will not be challenging to develop future agents.

3. Replan: Based on your progress so far and the knowledge you have acquired, what is the up- coming plan (week-by-week)? What is your immediate next step? If your project milestones / goals have changed from the initial proposal (e.g., you underestimated / overestimated the difficulty of the project), propose new milestones / goals.

For these next few weeks, we plan to develop our minimax algorithm to use more sophisticated evaluations to determine what truly is the best move. Since there are a lot of nuances in the value of potential moves (and consequent opponent moves) in checkers, it will require some deeper thought to plan this out to help our AI pick the best action. Our immediate next step is to ensure our minimax function works as intended (since we haven't written any formal tests yet). Once we have verified this, we'll probably end up making our evaluation function much more complex.

To compare our agents, we plan to develop a system to have two agents running a different version of our algorithm play each other and record the number of wins and losses of each. Through running a set of these games with different agents, we can create a tier list of the best agents, i.e. those who win the most. The idea is to have each agent running a different version of our algorithm (for example, one running minimax and the other running Monte-Carlo) and see which one fares better after a series of games. This will allow us to test

the general performance of our algorithms relative to each other, and ensure we're making progress with our algorithms.

If we have time, we will try to implement a reinforcement learning approach. For example, if the AI detects that the human player is aggressive and often takes risks (ex. Advancing their pieces when they know it leads to potentially being captured), then it should adjust its decision-making. Likewise, if the AI detects that the human player is a defensive player and avoids risks at all costs (ex. Advancing their pieces into corners to avoid potentially being captured), then it should adjust its decision-making for that as well. How exactly it would "adjust its playing" is still undecided.

Areas for improvement moving into the next stage of the project could include:

- Looking at more advanced evaluation functions
- Expectimax
- Monte-Carlo Tree Search (MCTS)
- Reinforcement Learning approaches
- Self-play, in conjunction with RL