A7 Sprint2: Progress Report: Checkers & AI

Joo Hyun Lee (jl3272) , Guofeng (Timothy) Shi (gs522) ,

Athena Cheung (afc65) , Henry Bender (jhb332)

Vision

We are building checkers platform that includes multiplayer and singleplayer modes. We implemented an AI bot for the single player mode for this sprint. The bot always takes a valid move and always takes (jump over) the opponent's piece when possible. Also, we aimed for the AI bot to move a piece so that it does not get captured after the move unless that is the only move possible.

Summary of progress

In this sprint, our group started off by implementing the remaining rules of chess, such as mandatory captures of pieces if available, multiple captures within the same turn, promotion of King pieces and backward jumps, and the win condition now including the case in which a player cannot make any moves. This was a collaborative effort in all writing functions that would come together in our game loop to force captures and automatically commit multiple captures (i.e. possible_moves, multiple_jump, etc.) Our second goal was to create an AI to play against. We achieved this in this sprint through first checking if the AI could make any captures, and if not, picking a move for it that would not result in being captured (if possible). Lastly, we implemented the AI game loop. This involved having to parse for the Player vs AI option in std in and then calling a different loop. This AI loop only accepted user input moves for the Red side, and for the White side it would Unix delay for 1.5 seconds before executing the move chosen for it.

Activity breakdown

- Joo Hyun
    - Board.ml and Board.mli: Worked on adding the win condition, which if the opponent has no more moves the player wins. Also, implemented get_captures, which is a function that returns only the captures (jumps) out of the possible moves. Also, fixed the parser and board printer to use alphabets for rows instead of integers.
- Athena:

    - Board.ml and Test.ml: Worked on the valid_moves function, which returns a list of legal moves given the current game state. The function uses a few helper functions, which evaluate the board row by row and calls valid_move on pieces that are owned by the current player. The moves that are Valid are then accumulated in a List.
- Henry:
    - Board.ml and main.ml: Worked on implementing automatic/required multiple jumps for players/AI. Created the AI function to choose moves that are captures or don't result in immediately being captured. Worked on the new game loop for the Player vs Ai game mode.
- Timothy:
    - Main.ml and Parser.ml: To distinguish the AI and Person vs. Person mode, I created another function called game_input_ai which will be called in the game_loop if the game mode is AI mode. On the other hand, if the game mode is Person vs. Person, the game_loop function will call the normal game_input instead. To implement the [game_input_ai], I check the available moves for the Black (the default AI) when it is AI's turn. To make the AI smarter, we also check the consequence of this move (will it result the capture, multiple moves, or nothing?) to make the game tree have the depth 1.
- All:
    - Worked together modifying the main.ml to use the newly implemented functions in board.ml.
    - We wrote test cases for the functions each of us wrote

Productivity analysis

In this sprint, our team was productive. We accomplished all of our goals as well as a few of our reach goals such as making a smart AI. We first met to decide on what new features we want to implement, and then we added functions for those features to the relevant mli files. We decided together what each function would do and what arguments and return values they would have. For example, we decided that a "move" should be a position*position*bool and that valid_moves should return a list of these moves so that get_captures can figure out if the current player has a mandatory capture. Then we split up and assigned the features that we considered modular such as AI interface and enabling multiple jumps. We set a due date and agreed to have our implementations done by Monday. We also agreed to document as we go and write tests for the functions that we implement. On Monday, we met after Discussion and worked together on bringing all of the individual features together. There were a few bugs that took some time to debug, but we all debugged together and eventually figured out what was wrong.

Coding standards grades

- Documentation: Meets Expectations. We documented all the functions, helper functions inside board.ml, main.ml, parser.ml. We made sure to not include the documentation for helper functions in the mli files.
- Testing: Meets Expectations. We wrote tests for most of the functions inside board.ml except for the helper functions. We added the test cases for the newly implemented functions and also tested edge cases, such as empty boards, deadlock board, etc.
- Comprehensibility: Exceeds Expectations. We used empty lines to separate definitions and used value, type names that are meaningful. We used type synonyms such as piece (color*rank), position (int*int) for comprehensibility. We also used begin, end to make the code more easily readable.
- Formatting: Meets Expectations.We tried not to get over the 80 column limit. We used minimum parentheses and indented the nested if statements, pattern matching clearly.

Scope grade

Here is what we consider a satisfactory, good, and excellent solution:

- Satisfactory: Fix the errors caused by the previous sprint, implement the functions [get_capture], [possible_moves].
- Good: Implement the function [enforced_captures], which returns the list of each move that is a capture in [movelist]. Implement the functions [multiple_jumps] to check if there are possible multiple captures after each. If there are, the piece will jump multiple times. Implement the AI mode in main.ml, which enables the auto-playing.

- Excellent: A smarter AI. Instead of choosing randomly from all the possible moves, the AI should move smartly by checking the possible captures and multiple jumps after the moves, then making the right decision to move.

We give our team “Excellent” for this sprint, because besides all of the functions are successfully implemented, we even make the AI smarter by developing a game tree which has depth 1. What’s more, we make sure all of the test cases are passed.

Reference

http://www.cs.cornell.edu/courses/cs3110/2017fa/handouts/style.html

http://www.cs.cornell.edu/courses/cs3110/2019sp/standards_rubric.pdf