Final Project

Milestone 1 due 11/15, 6pm
Milestone 2 due 11/22, 6pm
**Milestone 3 due 11/29, 6pm**
Milestone 4 due 12/6, 6pm
Complete project due 12/13, 6pm

Just like other homework assignments, you'll be submitting by posting the URL to your repo in Canvas every week and pushing your code regularly to your github repo. Please make a new folder in your `cs5001` directory called `final_project`. Submit each milestone and the finished project to the appropriate assignment in Canva. See the deliverables for each milestone at the end of this document.

For this project, you will make a graphical game of Checkers (AKA Draughts). [Play online here](). This is by far the most significant homework you will do in CS 5001. It's a lot to consider all at once but very doable in small chunks. Start by reading the game rules below and playing the online game linked above to get familiar with how it's played. Then, read all project expectations at least once, very carefully. There are weekly milestone submissions to break up the workload and keep you on track to finish strong.

Checkers Rules

The game is played with black and red pieces on an 8x8 board with light and dark squares in a checkerboard pattern. The goal of the game is to capture all of your opponent's pieces. Play begins with both players' pieces arranged like this:
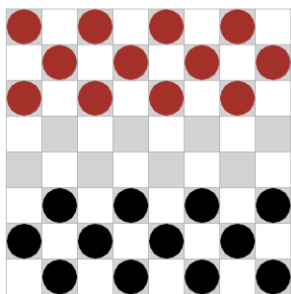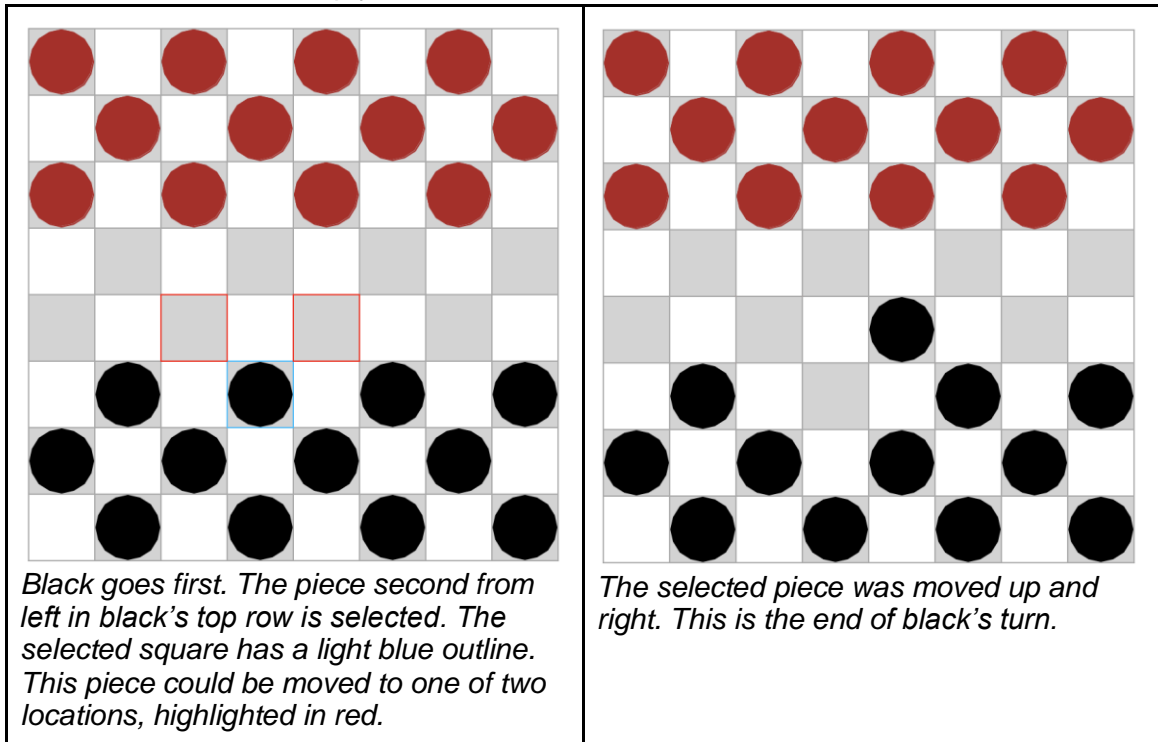

The board at the start of the game
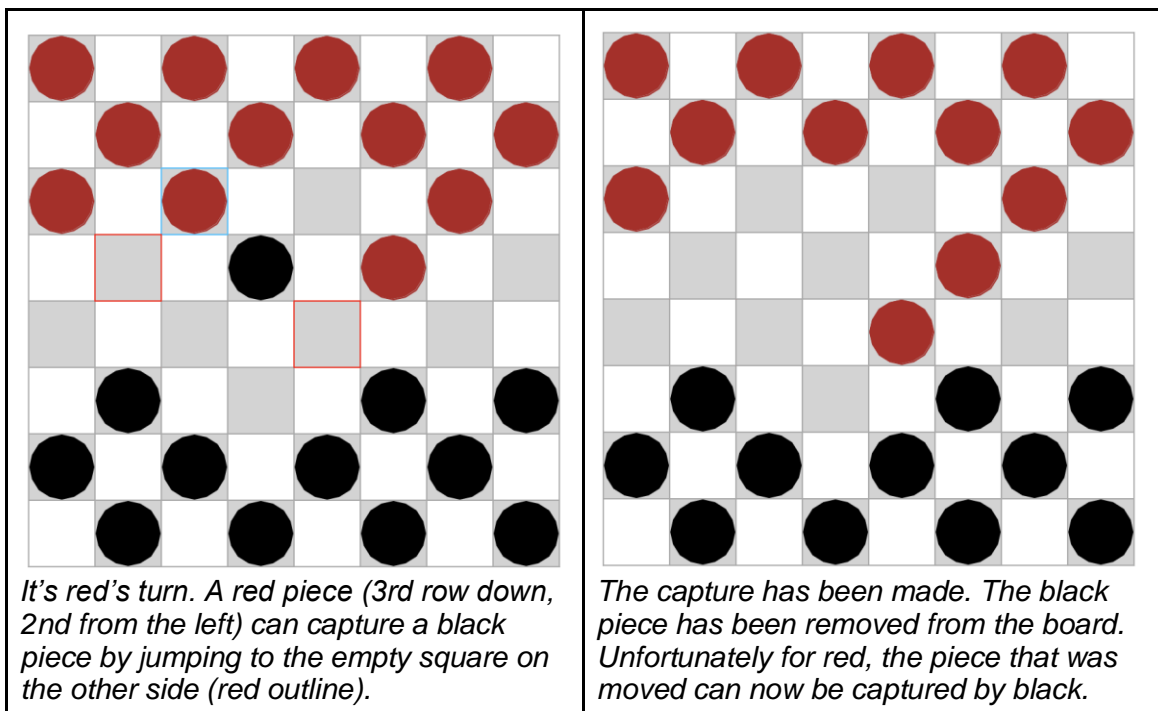
Making moves
A turn consists of moving one of the player's own pieces. **Pieces may only be moved diagonally and in a forward direction**, with one exception (see "King pieces" below). "Forward" means from the player's start side to the opposite side—red pieces move from the top of the board to the bottom, and black pieces move from the bottom to the top. There are two types of move a piece can make:

1. **Non-capturing move**. A piece is moved diagonally forward one square. The square that the piece is moved to must be empty.

Black goes first. The piece second from left in black's top row is selected. The selected square has a light blue outline. This piece could be moved to one of two locations, highlighted in red.

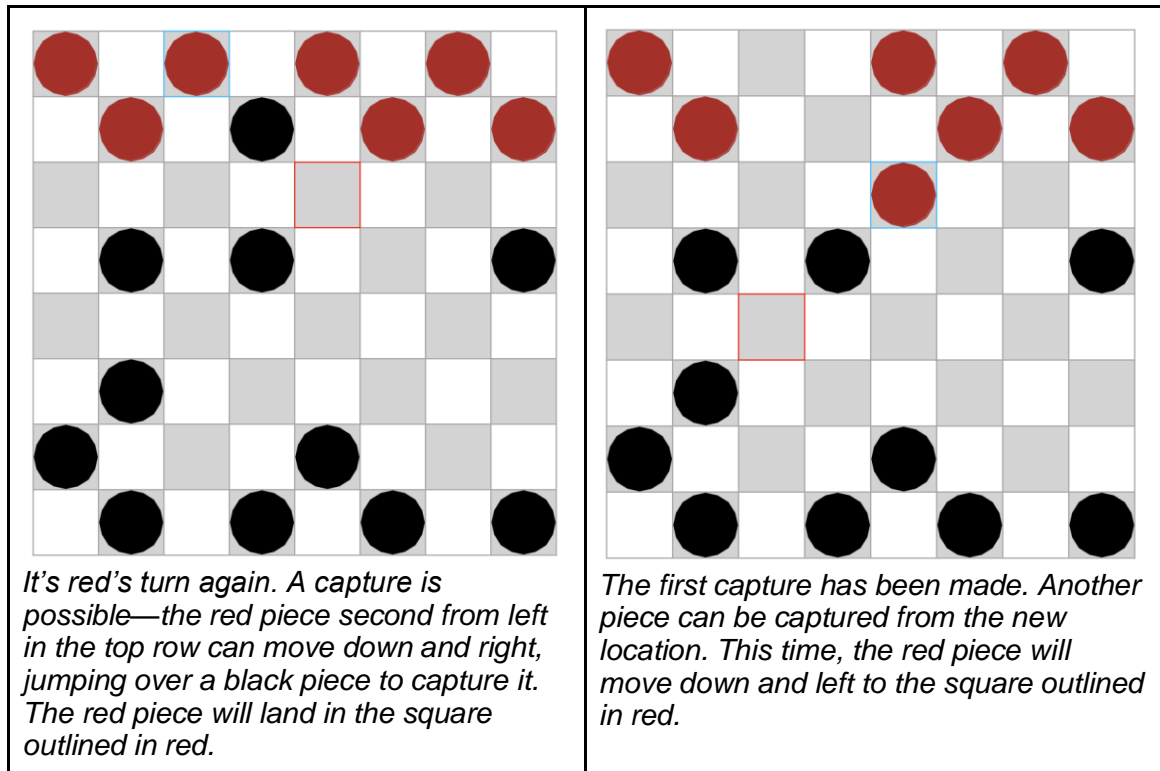*The selected piece was moved up and right. This is the end of black's turn.*

2. **Capturing move**. If an enemy piece is next to the player's piece, and the next square in the same direction is empty, the enemy piece can be captured as long as the move is in a forward direction. The captured piece is removed from the board.

It's red's turn. A red piece (3rd row down, 2nd from the left) can capture a black piece by jumping to the empty square on the other side (red outline).

The capture has been made. The black piece has been removed from the board. Unfortunately for red, the piece that was moved can now be captured by black.

If a capture is possible, it must be made. In the example above, the only legal move for red was to capture the black piece, even though this ultimately sacrificed red's own piece.

Sometimes it's possible to make multiple captures in a single move. Jumps can change diagonal direction, as long as the capturing piece continues forward.

It's red's turn again. A capture is possible—the red piece second from left in the top row can move down and right, jumping over a black piece to capture it. The red piece will land in the square outlined in red.

The first capture has been made. Another piece can be captured from the new location. This time, the red piece will move down and left to the square outlined in red.

The two types of moves cannot be combined. For example, it is not possible to make a non-capturing move and continue on to make a capturing move in the same turn.
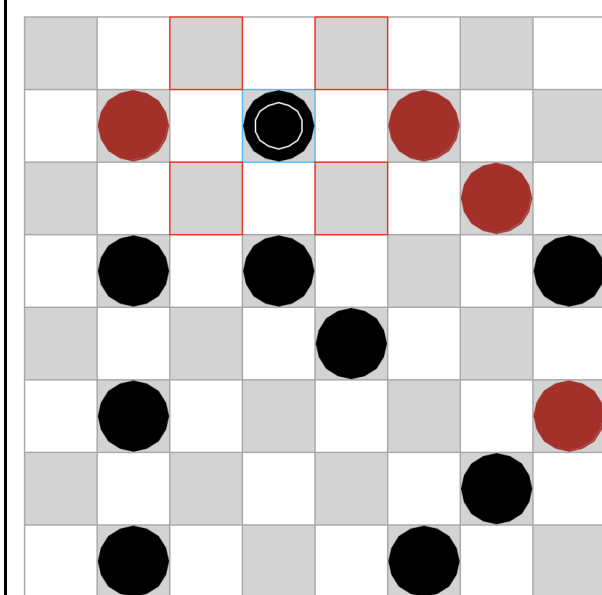
King pieces

When a piece gets all the way to the opposite side of the board, it is crowned a King. King pieces are very useful because they can move backwards as well as forwards. Moves are still diagonal. In "real" checkers, a King is made by stacking two pieces together. In my Python version of checkers, I identify King pieces with a white ring on the piece.
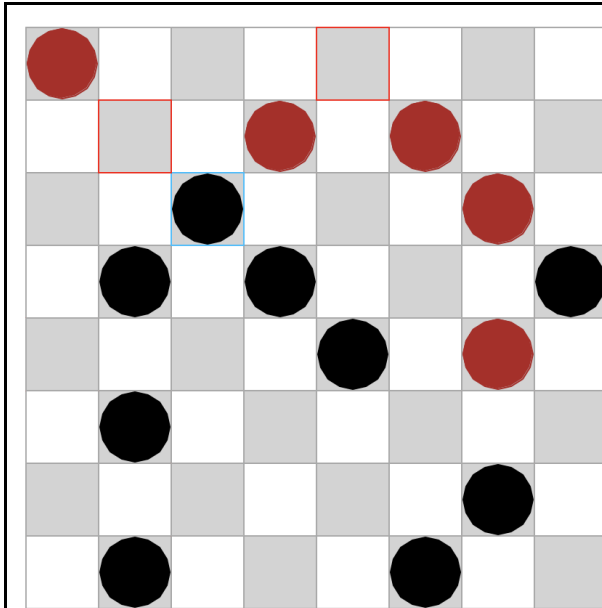
It's black's turn. A capture can be made by the black piece closest to the top of the board.
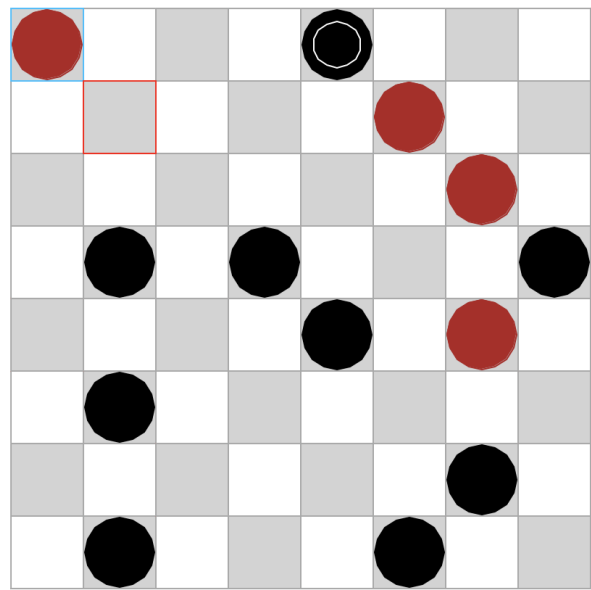

After the capture is made, the black piece reaches the top and is crowned a King. Notice the white ring on the King piece.

A couple more turns have been played and it's black's turn again. The King piece is selected. Notice that it can now move forward or backward (see the squares outlined red).

Ending the game

The game ends when a player has captured all enemy pieces (most common), or when one of the players is not able to move their remaining pieces.

Black has won the game. Red still has a piece on the board but it is blocked from moving.
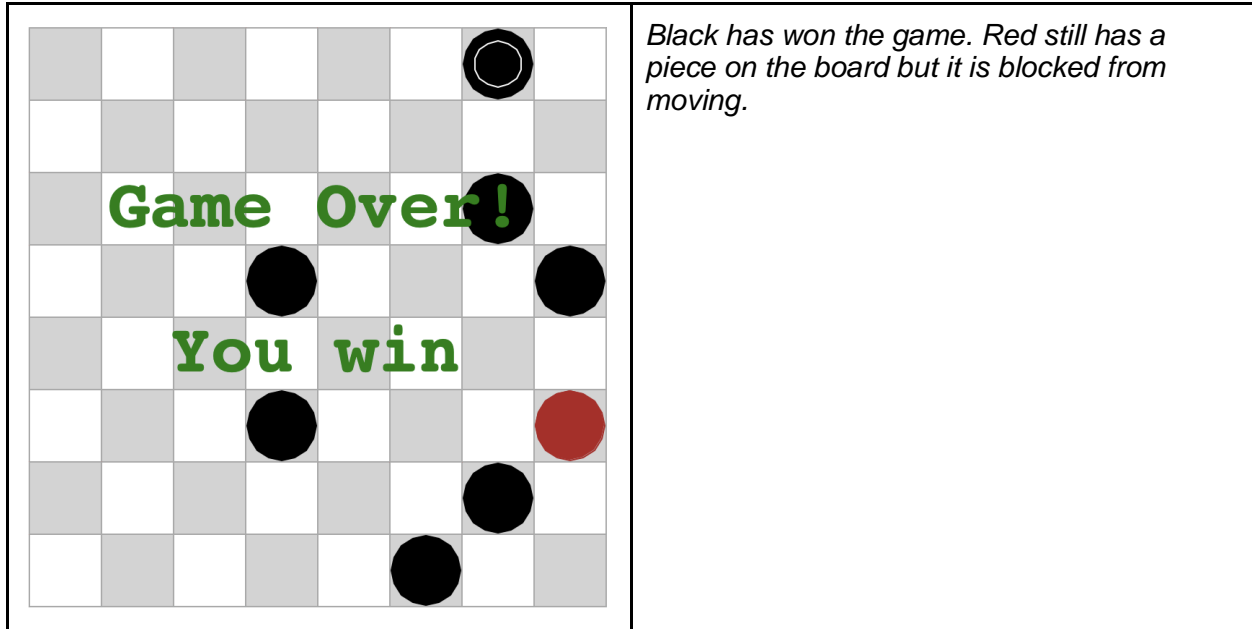
Project Expectations: Functionality

Your implementation of Checkers does not have to look and function exactly like mine. Here are the expectations:

- **You must display a board with an 8 x 8 checkerboard pattern made up of light and dark squares.** You can change the overall dimensions of the board and the specific colors used, as long as you have the right number of squares and the top left square is your dark color.
- **Each player starts with 12 pieces, arranged as shown in the first image in this document.** Black and red are the traditional colors. You may change the colors but, if you do, be sure to document which color makes the first move.
- **Moves must follow the rules above.** If a capturing move is available, the player MUST make that move. If multiple capturing moves are available, it doesn't matter which one is chosen, as long as a capture is made. If, after a player has captured an enemy piece, another capture is possible for the same piece, that capture must also be made.
- **When a capture is made, the captured piece must be removed from the board.**
- **When a piece reaches the opposite side, it should be made into a King piece.** King pieces should be visually distinguishable from "normal" pieces. You don't have to use the same styling that I did. King pieces should be able to move as described above.
- **One player is a (human) user, the other player is the computer.** The human user should make the first move. If you're sticking to the traditional colors, black will be human. When it's the computer's turn, your program will need to identify all legal moves and pick one to make. If a capturing move is available, it must be made. Beyond that, it's up to you to decide how "smart" the computer should be when it picks a move.
- **The user should make moves by clicking on the square containing the piece they would like to move, then clicking on the square the piece should be moved to.** Only allow a move to be made if it is valid according to the rules above. If the user

attempts an invalid move (e.g. clicking on an empty square or an enemy's piece etc.), provide some feedback. Printing a message to the terminal counts as feedback.
- **Update the UI appropriately after each move.**
- **When the game is over, declare the winner and stop the game play.** In my implementation, the winner is printed in the UI but this is not required—printing to the terminal will meet this requirement.

You are free to add extra features or enhancements beyond these requirements. For example, my sample solution provides visual feedback on where a selected piece can be moved and delays the computer move to make it easier to follow.

Project Expectations: Design and Testing

You have a lot of freedom in how you design your solution but there are a few requirements:
- **You must have a file called `main.py`, which includes a `main()` function**. This will be the "driver"—the file that is run to draw the board and start the game.
- **You must use classes,** but it is up to you to decide what classes you define. There is no single right approach to this.
- Whatever classes you come up with, **put each class in its own file**.
- Depending on your design, it might not make sense to put everything in a class, so you may want to write some functions to handle some aspects of the game. **Put standalone functions in `main.py`**.
- **Write unit tests where you can**. You will not be able to unit test anything that interacts directly with Turtle.
- **All your files must have a file comment. Classes, methods, and functions should all have docstrings.** When documenting classes, the Methods section only needs to include methods that are intended to be "public".
- **Continue to use all good practice principles we've covered so far** e.g. no magic numbers; functions/methods should be single purpose; give classes, variables, methods, and functions meaningful names.

Project Milestones

Make sure you submit to each milestone assignment in Canvas by the due date. You are free to go beyond specific milestone requirements e.g. if you've made progress toward milestone 2 before the milestone 1 date, submit all your work so far to the milestone 1 assignment—don't try to split up your code to match the milestone. Milestone requirements carry over. For example, when you're working on milestone 3, it's expected that everything covered in milestone 2 is still functioning.
There are no style / design / testing / documentation requirements for the milestones beyond having a main.py file that contains a `main()` *function.*

11/15 Milestone 1: Implement the basic UI
Submit a `main.py` that contains a `main()` function and does the following:
- Draws the 8 x 8 board with a checkerboard pattern.
- Draws all the pieces in their correct starting positions.
- Clicking on the board should print out the coordinates of the click and whether or not the click was in bounds i.e. in a valid square on the board. (If you're already moving pieces by the time you submit milestone 1, you don't need to print the coordinates/in bounds information)

11/22 Milestone 2: Represent the game state and non-capturing moves
Submit code that does the following:
- There should be a notion of the game "state" i.e. where each player's pieces are on the board and whose turn it is. Using a class for this is highly recommended!
- Non-capturing moves are implemented e.g. clicking on a cell containing a piece, then clicking on an adjacent empty cell (in a forward direction) moves the piece.
- There is some notion of turn-taking (black moves, then red moves, then black moves, and so on), but the computer player may not be implemented—the human user may play both black and red at this stage.

11/29 Milestone 3: Implement capturing moves and the King upgrade
Submit code that does the following:
- Capturing moves are implemented.
- If a piece can be captured, only a capturing move should be allowed. This includes multiple-capture moves.
- When a piece reaches the opposite side, it becomes a King and can move backwards as well as forwards.

12/6 Milestone 4: Implement the computer player and game ending
Submit code that does the following:
- Red (or whatever color you're using for the computer) is moved by the computer.
- The game ends and a winner is declared.

12/13 Complete project
Spend the last week documenting your code, writing tests, passing pycodestyle, and improving your code design as needed. It is common for Align students to include their CS 5001 project in their portfolios so you may also want to add some extra polish once the assignment requirements are met.