Multiplayer Checkers Game

By Jeff Wedding

Computer Science (B.S)

Dr. Sean Hayes

Contents

Statement of Purpose

The project involves the development of a checkers game that combines classic gameplay with modern features, ensuring a fun experience for players of all experience levels. The problem at hand lies in the absence of a modern checkers game that embraces technology and multiplayer elements. This is important because it deprives checkers players of an enriched gaming experience. To address this issue, I developed a feature-rich checkers game that integrates classic gameplay with modern features, including a user-friendly interface and multiplayer functionalities. This solution allows friends and enthusiasts to connect over a shared passion for checkers. Furthermore, this project greatly improved my knowledge of software development, particularly with networking in python. In conclusion, I addressed the issue of the absence of a modern checkers game that embraces technology and multiplayer elements by developing a feature-rich checkers game that integrates classic gameplay with modern features while further developing my skills in networking.

Research and Background

My lack of experience networking required me to do research on the topic. I mainly researched the python library “pickle,” as this seemed to be the consensus best option for python games. Of course, there were several other small things I had to research throughout the course of development, such as displaying objects using pygame.

Project Language, Hardware, and Software

Python

Visual Studio Code and GitHub

Project Requirements

1.
    a. Type – Appearance
    b. Description – Create an 8x8 visual board that the game will be played on.
    c. Rationale – The game needs a board that will be played on. This will make it easier for the players to visualize the game and make moves.
    d. Fit Criterion – The board will be visible once it is implemented.
    e. Priority – High
    f. Dependencies – n/a
2.
    a. Type – Appearance
    b. Description – Create the pieces (red or black dots) that players will use to move.
    c. Rationale – The players need to know where their pieces are.
    d. Fit Criterion – The pieces will be visible once they are implemented.
    e. Priority – High
    f. Dependencies – 1
3.
    a. Type – Functionality
    b. Description – When a player clicks on their piece, they can then move it.
    c. Rationale – Players must be able to move their pieces.
    d. Fit Criterion – The piece will have a Boolean property that will determine if it has been selected or not.
    e. Priority – High

f. Dependencies – 7

4.

a. Type – Functionality

b. Description – When a player clicks on an open space after they selected one of their pieces, that piece moves to the open space that was clicked on. The turn will then end.

c. Rationale – Players need to be able to move their pieces.

d. Fit Criterion – The selected piece will be moved to the desired open space.

e. Priority – High

f. Dependencies – 3, 7

5.

a. Type – Functionality

b. Description – If a player clicks on a piece and then clicks on a part of the screen that is not an open space, the piece will no longer be moveable.

c. Rationale – Players may select a piece to move and then change their mind.

d. Fit Criterion – The Boolean property of the piece will be set to false.

e. Priority – High

f. Dependencies – 3

6.

a. Type – Functionality

b. Description – If a player's piece reaches the opponent's side of the board, that piece will turn into a king.

c. Rationale – King pieces are a critical part of the game of checkers.

d. Fit Criterion – The piece will have a Boolean property that determines whether it is a king piece.
e. Priority – High
f. Dependencies – 7, 8

7.

a. Type – Functionality
b. Description – The board will be implemented as a 2-dimensional array of Booleans. Each entry in the array can either be a 0 (space is empty) or 1 (space is occupied).
c. Rationale – Implementing the board as a 2-dimensional array is the easiest way to visualize the board and where pieces are for me. This will also make it easier when selecting pieces to move.
d. Fit Criterion – Program will run without error if this is implemented correctly.
e. Priority – High
f. Dependencies – n/a

g. Type – Functionality
h. Description – A “piece” class will be created containing information about pieces such as which team they are on, whether they have been selected, whether they are a king, etc.
i. Rationale – This will make it much easier for me to write code for the rest of the project.
j. Fit Criterion – Program will run without error if this is implemented correctly.

k. Priority – Medium

l. Dependencies – n/a

8.

a. Type – Functionality

b. Description – Players will be prompted to double or triple jump if possible.

c. Rationale – Double and triple jumping is an essential part of the game of checkers.

d. Fit Criterion – The spaces that can be used to double and triple jump will be highlighted.

e. Priority – High

f. Dependencies – 1, 4, 7

9.

a. Type – Functionality

b. Description – Pieces will be able to move forward and diagonally forward.

c. Rationale – Pieces need to be able to move.

d. Fit Criterion – Pieces will visibly move.

e. Priority – High

f. Dependencies – 9

10.

a. Type – Functionality

b. Description – The king piece will be able to move diagonally, forwards, and backwards.

c. Rationale – King pieces are an essential part of the game of checkers.

d. Fit Criterion – Pieces will visibly move.

e. Priority – High

f. Dependencies – 9

11.

a. Type – Functionality

b. Description – Two players will be connected to each other to play.

c. Rationale – Players need to be connected to play.

d. Fit Criterion – Players will not join the game until they are connected.

e. Priority – High

f. Dependencies – n/a

12.

a. Type – Functionality

b. Description – Players will capture the opponent's pieces by hopping over it.

c. Rationale – The most important part of checkers is capturing pieces.

d. Fit Criterion – A "piece captured" message will appear on the screen when a player captures a piece.

e. Priority – High

f. Dependencies – 10

13.

a. Type – Usability

b. Description – If a player clicks on one of their pieces and that piece can capture an opponent piece, the space where they should jump will be highlighted.

c. Rationale – This will make it a lot easier for players to see where they should jump.

d. Fit Criterion – The spaces will be highlighted yellow.

e. Priority – Medium

f. Dependencies – 3

14.

a. Type – Functionality

b. Description – The game will end once a player has captured all the opponent's pieces.

c. Rationale – The game needs to end.

d. Fit Criterion – A message saying the game is over will come up on the screen.

e. Priority – High

f. Dependencies – None

Project Implementation & Description

The checkers match begins when both players have successfully connected to the server. The board starts as your typical checkers board that everyone knows and loves (Fig. 1). The second player who connects will be player 1 and will start the game with their move. When this player has decided which piece they would like to move, they simply click on that piece to view the open spaces available. These spaces are highlighted (Fig. 2). Once the player clicks on one of these open spaces, their piece is shown in its new place on both their board (Fig. 3) and their opponent's board (Fig. 4). It is important to note that when a player can jump the opponent's piece, they **must** do so. They will not have the option to move any other piece. The interface for jumping a piece is the same as moving any other piece; the same highlighted space appears (Fig. 5), and once the jump is made, the player's piece is removed from the board (Fig. 6). If during the game, a player manages to move a pawn to the opponent's end of the board, it becomes a king piece (Fig. 7 and 8). This king piece has the ability to move in both directions (Fig. 9). A sample king jump can be seen in figures 10 and 11. The game ends when one player has collected all of the opponent's pieces. When this happens, the game window will close, and the command line will display either a winning or losing message (Fig. 12 and 13).

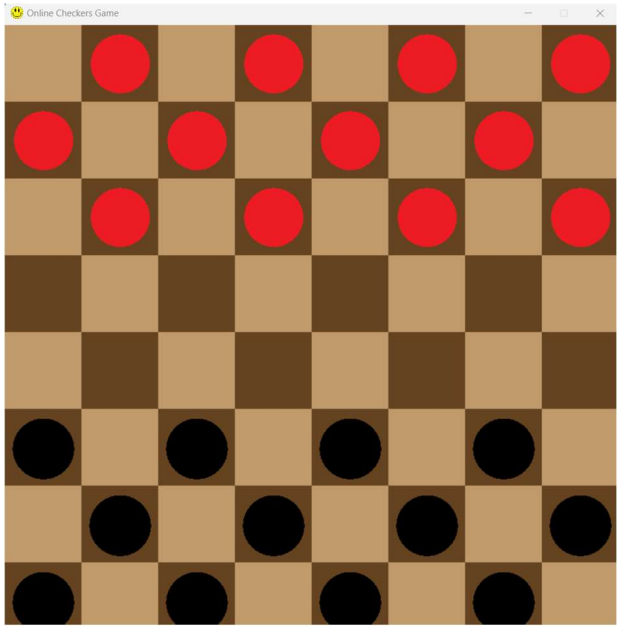

Fig 1. The starting board

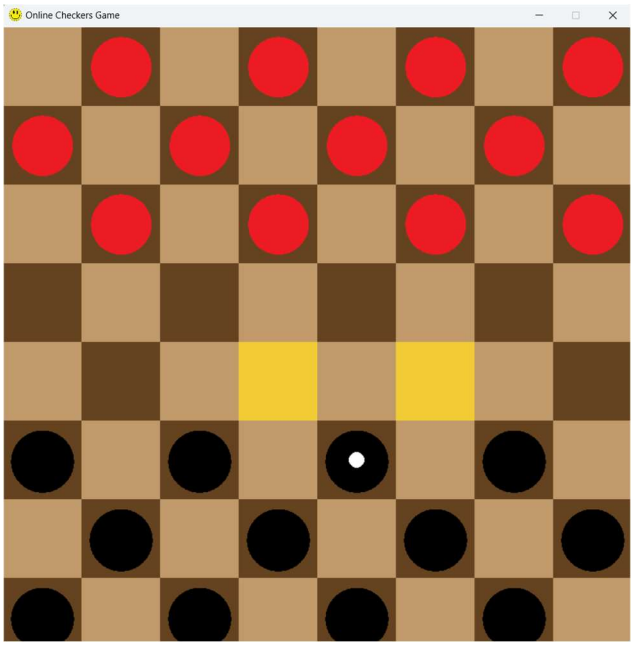

Fig 2. Available spaces are highlighted

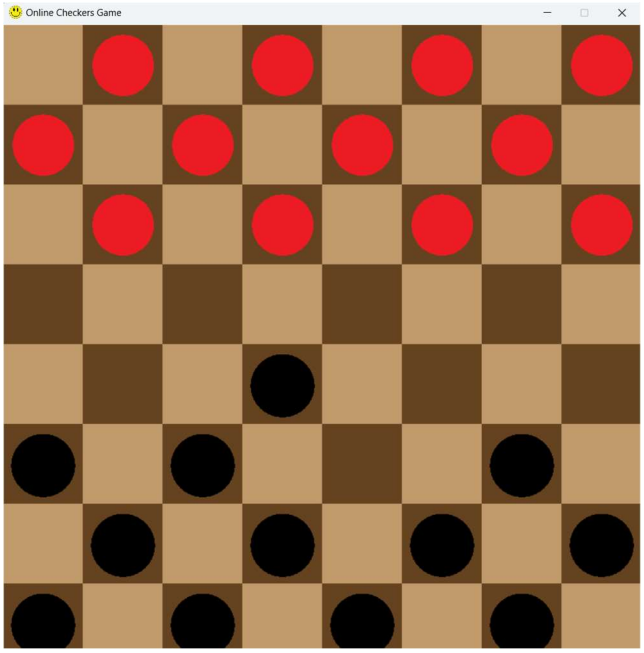

Fig 3. Updated board

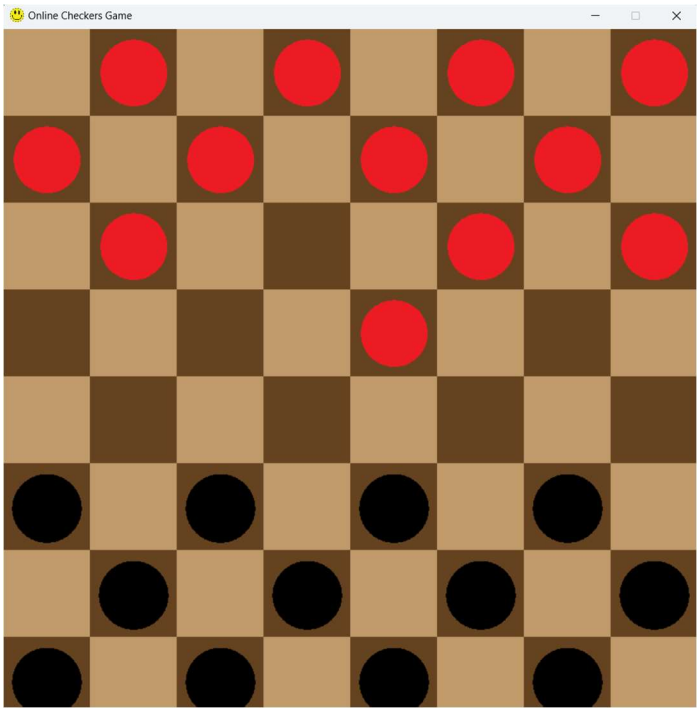

Fig 4. Updated board on the opponent's end

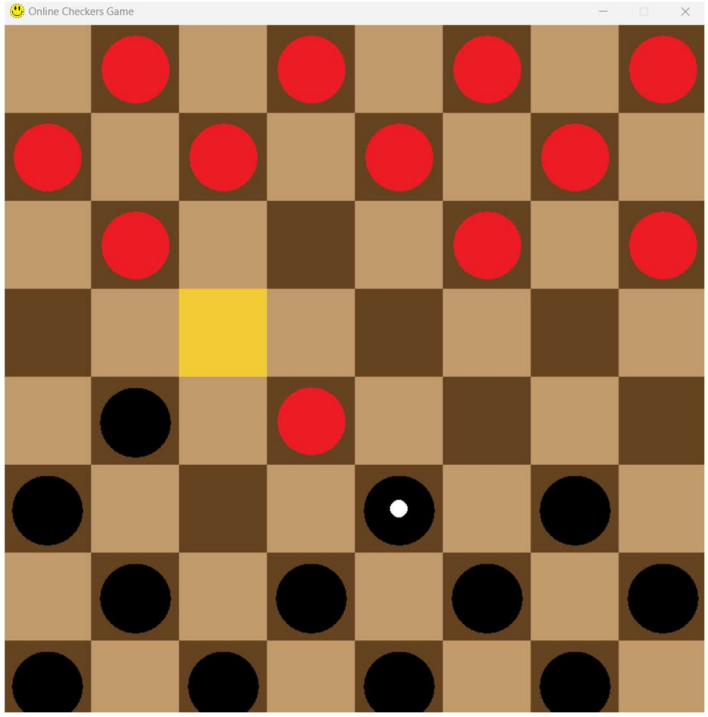

Fig 5. Jumping a piece is just like moving one

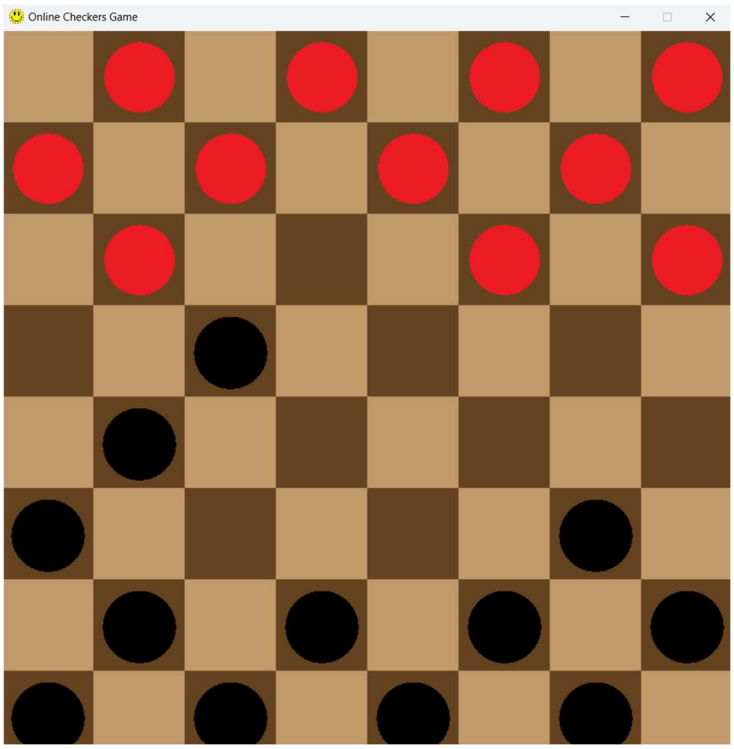

Fig 6. Piece is removed after it is jumped

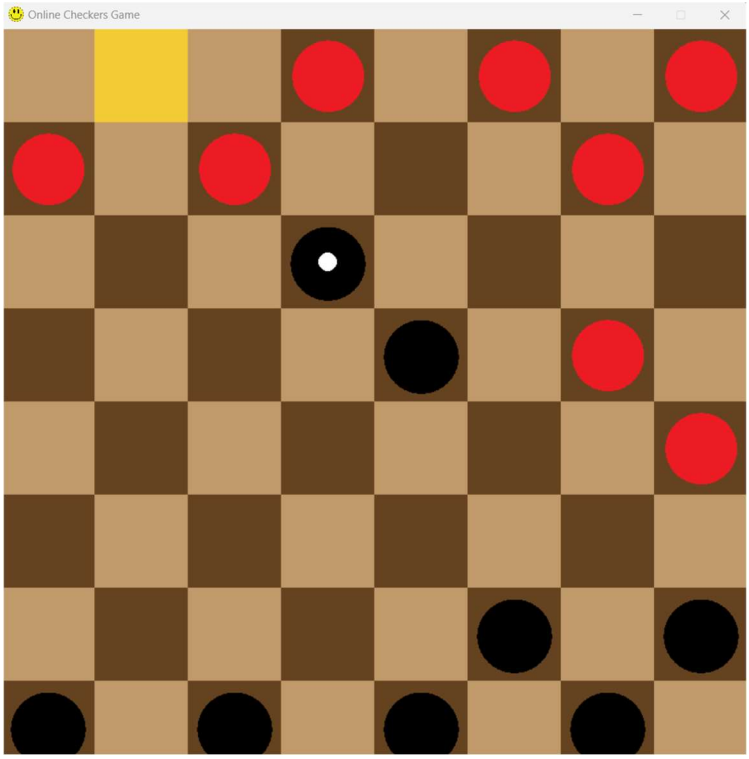

Fig 7. This piece can become a king with this jump

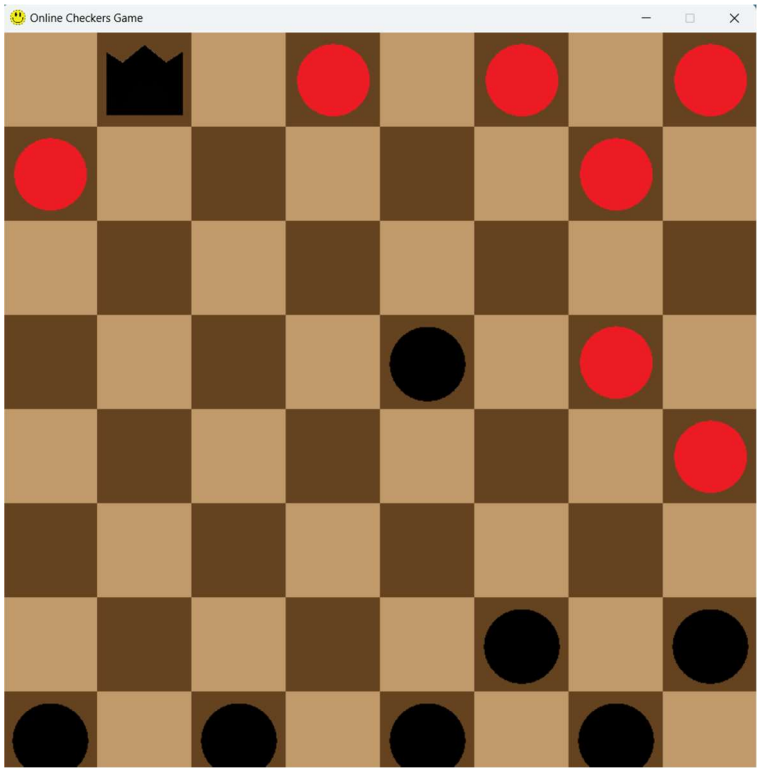

Fig 8. Updated king piece

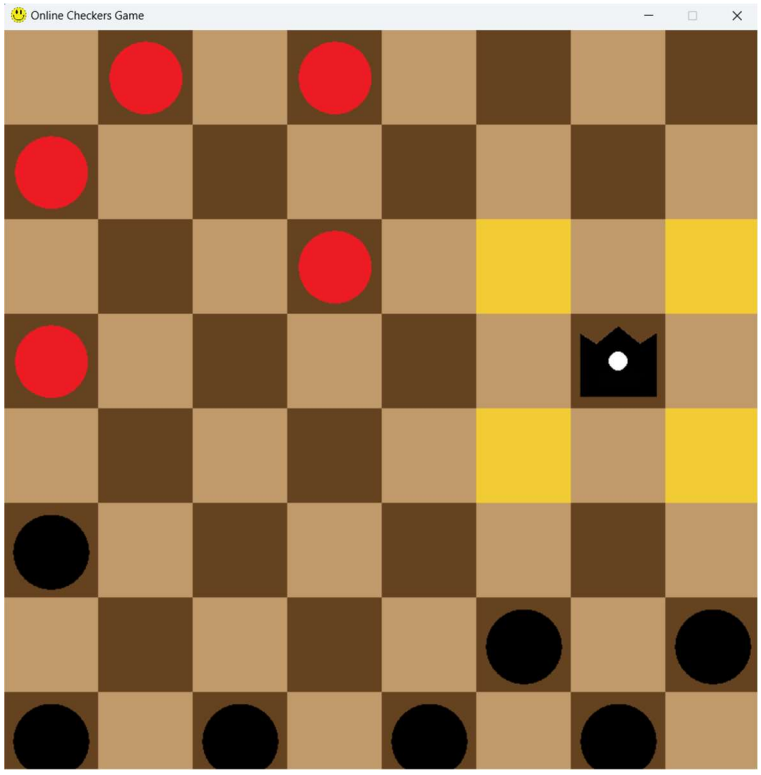

Fig 9. King pieces can move in both directions

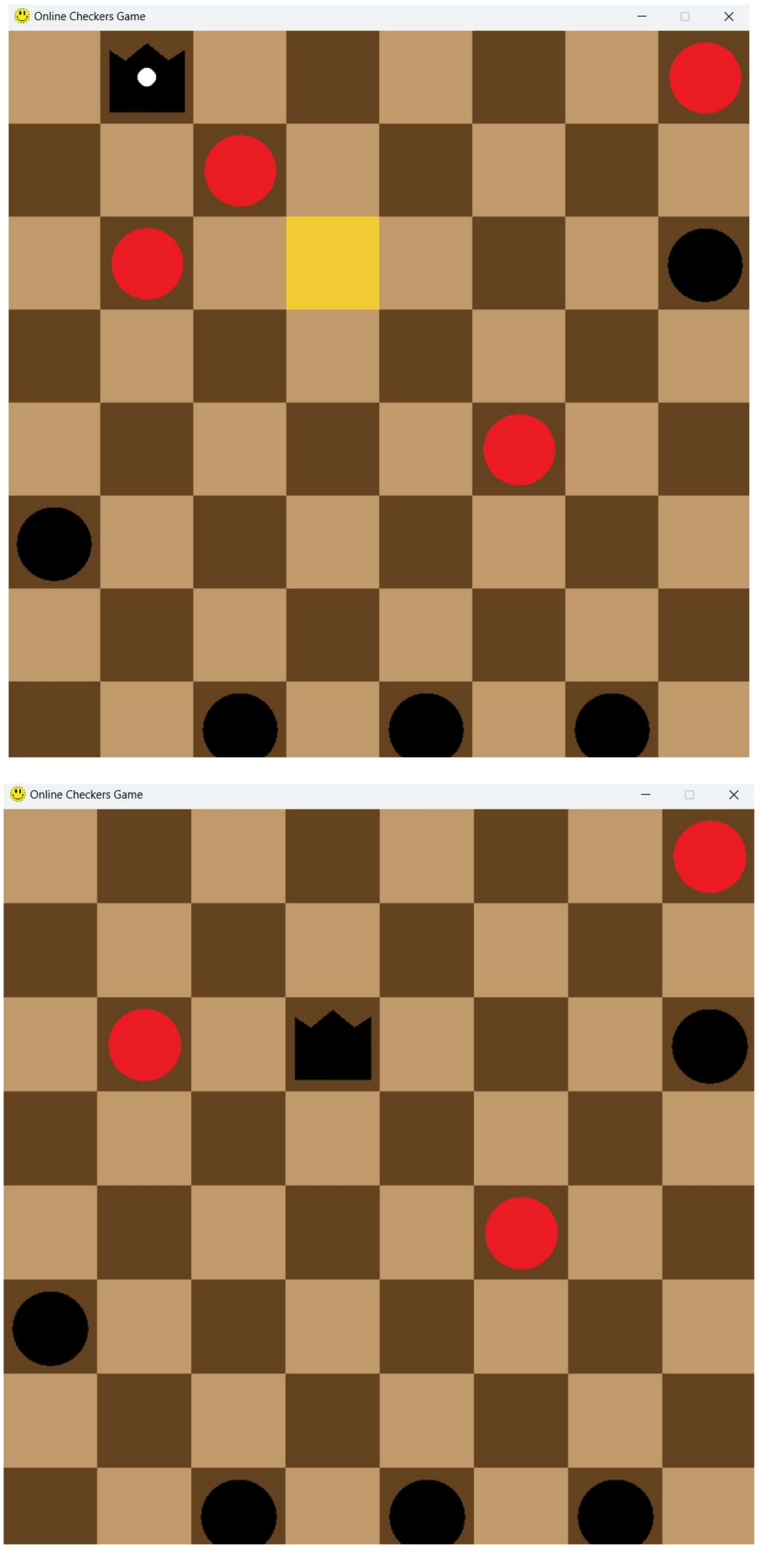

Fig 10 and Fig 11. A king piece jumping a pawn

```
PS C:\Users\jwedd\Downloads\Senior Project\Testing> python3 client.py
pygame 2.5.2 (SDL 2.28.3, Python 3.11.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
You Win!
PS C:\Users\jwedd\Downloads\Senior Project\Testing>
```

Fig 12. The winning message

```
PS C:\Users\jwedd\Downloads\Senior Project\Testing> python3 client.py
pygame 2.5.2 (SDL 2.28.3, Python 3.11.9)
Hello from the pygame community. https://www.pygame.org/contribute.html
You Win!
PS C:\Users\jwedd\Downloads\Senior Project\Testing>
```

Fig 13. The losing message

Test Plan

Introduction: This test plan involves a multi-faceted approach to ensure the game's functionality, performance, and reliability across various scenarios. The primary goal is to verify seamless gameplay between two players on separate computers while adhering to the rules of checkers. Constraints include compatibility testing across different network environments to guarantee smooth communication and synchronization between players. Performance testing will be conducted to assess the game's functionality. Overall, the test plan aims to deliver a robust and enjoyable gaming experience.

References: Project Proposal, Requirements Specification, Test Plan Specification

Test Items: PowerShell 5.1, Python 3.11, Pygame 2.6

Features to be Tested: Functionality of the game including but not limited to: players are able to move and jump over pieces, players are able to "double jump", pawns turn into kings if they reach the other side of the board, game will end once a player has collected all of the opponent's pieces. The connection between the two players will also be tested.

Features Not to be Tested: N/A

Approach: The most logical approach to testing this program involves both manual and white box testing methods. Manual testing entails simulating various game scenarios, such as different moves, captures, and game-ending conditions, to ensure smooth gameplay and accurate rule enforcement. White-box testing involves examining the game's source code to identify potential bugs or vulnerabilities, such as incorrect game state handling or network communication errors, ensuring robustness and reliability. By combining these approaches, the testing process aims to deliver a polished and error-free experience for players.

Item Pass/Fail Criteria: There will be an expected outcome for each test. If the expected outcome matches the actual outcome, the test is considered passing. If not, the test is considered a failure.

Suspension Criteria and Resumption Requirements: N/A

Test Deliverables: Test plan, Test cases

Test Environment: Powershell, Python, Pygame, Microsoft Word

Risks: There may be some security vulnerabilities within the program, as I have not learned how to test for them. Furthermore, I have very little experience in network communication protocols, so I do not yet know how to identify and test vulnerabilities within the network. I will reduce this risk by researching proper techniques for creating communication over a network without risk.

Test Results

Test No.	Action	Input	Expected Output	Actual Output	P/F
1	Launch game	“Python3 client.py”	Game opens	Game opens	P
2	Players can move once their pawn forward and diagonally	Click on pawn then click on open space	Pawn moves	Pawn moves	P
3	Available spots to move are highlighted	Click on pawn	Pawn and available spots to jump are highlighted	Pawn and available spots to jump are highlighted	P
4	Players’ turn ends once it is over	Click on open space	Other player is able to move	Other player is able to move	P
5	A piece is clicked on and then another piece is clicked on	Click on a pawn and then click on another pawn	New pawn will be movable	New pawn is movable	P
6	Pawn turns into king when it reaches the other side of the board	Reach other side of board with pawn	Pawn will become king	Pawn becomes a king	P
7	Game states the winner	Win game with black	“Black wins”	You win!	P
8	Players will ‘take’ pawns when they jump over them	Jump over a pawn	Pawn will disappear	Pawn disappears	P
9	Invalid moves are rejected	Click on a pawn to move	Pawn will not move	Pawn does not move	P

10	Game will state the game has ended with no winner if a player exits the game	Exit out of game	“No winner. Game exited.”	“Player disconnected”	P
11	Players must jump pawns if they are able to	Set up a pawn that can be jumped	Player is forced to jump the available pawn	Player is forced to jump the available pawn	P
12	Game closes when there is a winner	Player 1 wins the game	“Player 1 wins!” (on command line)	Game closes	P
13	Players are connected over a network	“python3 server.py” “python3 client.py”	Two visible boards that communicate	Two visible boards that communicate	P

Challenges Overcome

I did not face many challenges creating the game side of the program. Rather, the challenges presented themselves far more on the networking side of the program. I had very little experience networking coming into this, so it was a challenge. Even after doing some research on pickling in python, I still had trouble getting data to properly send. It turned out that I was greatly overcomplicating the process. Towards the beginning of coding, I was getting some errors that led me to believe there was a problem with the size of the data being sent and received. This led me down a rabbit hole. I then started to troubleshoot by trying to send the data in chunks to be packed and unpacked, among several other things. It turned out that the problem was a result of not properly initializing the board at the start of the game.

Future Enhancements

While creating the game, there were several enhancements that I came up with. I would like to be able to allow multiple games to be played at the same time. As of right now, my program only supports one game at a time. Also, it would be beneficial for some sort of indicator to be on the screen to let players know whose turn it is. With the current program, a player must be watching the screen and see the other player move to know that it is their turn. Another thing I would like to add is a timer. As the program is currently written, players can take however long they would like on each turn. This could cause integrity problems; if a player knows they are going to lose then they don't have to play their turn. As for quality-of-life enhancements, I would like to add a chat room for players to use in-game, and a waiting room for players that have connected but do not have someone to play against. I'm not sure what this waiting room would consist of, I still need to brainstorm some ideas.