Checkers Game

Connor Walsh
Daniel Cunningham

Abstract

The project covered in this paper is a single screen checkers video game program. Its primary intention is to provide users with a fully functional two-player checkers gaming experience. Users can also play against themselves, if they so choose, to set up and experiment with different checkers scenarios. As a result, this project will be directly targeted at people who enjoy playing checkers either competitively or recreationally. In order to accomplish this, we have used our combined knowledge of both checkers and programming.

1. Introduction

This checkers game project is intended to be an accurate simulation of the board game. We chose checkers because we believed that it would give us an adequate challenge for our coding skills and would be realistically achievable within the restrictions of the time-frame we were given. In this checkers program, the entire board is on one screen. The screen and pieces will be represented by graphics that we create. The checkers game is meant to be played locally (on the same screen) by two players, but a single player could technically control both sides of the board if they would like to set up and test out specific scenarios.

The main goal for this project is to develop a fully functional checkers game that is true to the original board game and our target audience for this project is anyone who wants to be able to play checkers on their computer. This includes competitive players who compete in tournaments and have a high skill level and also casual players who occasionally enjoy checkers in their free time. We hope that we have succeed in creating a well crafted checkers experience and that our target audience will be able to enjoy playing checkers on their home computer or laptop any time they choose. Aside from providing consumers with a high-quality checkers simulation program, another goal for this project was simply to test our skills as programmers and to see if we can successfully create said program within the required time limitations.

1.1. Background

We knew that we wanted to adapt a board game of some sort for this project. We originally considered adapting chess, but chess is a very complex game with many different moves and scenarios. Due to the time restrictions we have, we decided to create a checkers game instead because it is much more straightforward and will still provide us with an adequate challenge. Since we both have familiarity with checkers, we also thought that we could use this to our advantage because we know the rules of the game. The official rules of checkers will be enforced in this program and, in order to understand this project, these rules need some explanation.

In checkers, the game takes place on a board with 64 squares. Half of these squares are a dark color such as black, while the other half are a lighter color such as red. The color pattern alternates in all directions on the board. In checkers pieces can only travel on the dark squares diagonally, moving one space per turn. If a piece is taken on this turn, however, the piece that took it moves another space in the same direction. There are 24 pieces on the board and each player has 12. The pieces are usually red and black. For our game, the red pieces will move first. There does not seem to be a consistent rule for this. A piece can change into a king if it reaches the opposite side of the board from which it has started. In order to win, one player must take all of the other player's pieces or block them from being able to move any of their pieces.

1.2. Impacts

The main impact our project has is simply giving people an enjoyable experience in their free time, preferably with a friend or family member. Stress is bad for people, so it is important to take a break every now and again and have some fun. This checkers game allows people to relax and decompress, preferably with a friend, which is good for both their physical and mental health. Our game will hopefully give users a positive and relaxing experience, similar to how other games and activities do. Aside from this, our project may also help people gain an interest in Checkers as a hobby, potentially leading to them competing competitively in an official checkers tournament.

1.3. Challenges

There were a number of places where we could have ended up stuck during this project. One of them was having to delete pieces off of the board when they are taken. We ended up tying the pieces to the states of the buttons that our board was made up of, which allowed for a quick and easy solution to this problem that also looks good visually without actually deleting anything. Making a piece move two spaces once another piece is taken was also something that we had to figure out how to implement, but we were able to accomplish this by putting in boolean checks to see if there was a piece in the space or not which allowed us to overcome this challenge. Similar logic was used to keep track of which spaces are occupied and which are not to avoid illegal movies from being done. The boolean/button approach allowed us to easily prevent the player from making any illegal moves in general because they can only move pieces to valid squares. The graphics were fairly easy to figure out since WPF has a lot of options in this regard. We ended up drawing the checkers pieces with transparent background and imported them into our project, but aside from this, we were able to generate and color the buttons that made up the board and we were able to put in visually appealing backgrounds using features available in WPF and C#. We made the checker pieces that are normally black in the game blue to prevent them from blending into the board. We also created king pieces that have crowns on them to allow the players to clearly be able to tell which pieces are kings. We kept track of the number of pieces remaining on the board in order to end the game, so once a player has no pieces remaining, the game will end. If a player can no longer move, they must forfeit the game which also ends the game. Both of these cause a message to open that tells the users which player won. Many of these things were challenging to figure out, but through thinking, planning, and research, we were able to overcome these challenges.

2. Scope

The checkers game will be considered done when we have a game of checkers in which you can move pieces, take the other player's pieces, and can king your pieces. It will also include a graphical interface in which the pieces are displayed upon a virtual checkers board. The game must be able to be concluded if the user can no longer move or if their last piece is taken. There will be an option to forfeit the game if the user believes that defeat is inevitable or can no longer move. We also need uniquely colored pieces for both sides in the checkers game and there needs to be a different piece representing a king. Whenever pieces are taken, there must be a way to delete the piece's image from the board while keeping track of the amount of pieces each player has. Finally, there must be buttons for exiting and resetting the game. We originally said that it was a requirement for pieces to jump more than once if the pieces on the board are in the right position for such a move, however, upon doing more research, we discovered that this is not a real checkers rule and is only included as a rule in some versions of checkers.

For stretch goals we would like to include a way to set the board to different themes or colors. We could also potentially allow the user to create different themes for the board themselves and import them in. Along with this, we would like to allow the user to add their own images for certain pieces. Both of these things would allow for the user to personalize the visuals of the game to their liking. The other stretch goal we would like to implement is the ability for the user to save and load a game. This would mean that if the user wants to return to an in progress game they can have the ability to do so. Including functionality like this would allow for the program to be more worthwhile for users, because they wouldn't need to worry about having to lose progress. Also, if the state of the board can be saved, this would allow a player to make their move, save the game, and then send the game or save file to another player who could then make their move and send it back. This would widen the appeal of the game by allowing it to be played without requiring both players to be at the same computer at the same time. Finally, we are also considering putting in the ability for players to play against one another over a network. This would require more research on our part, but giving the game a multiplayer mode like this would also widen its appeal and usefulness.

2.1. Requirements

The requirements listed below have been separated into base and stretch requirements. The base requirements are what is expected for a basic game of checkers, while the stretch requirements are are features that we would like to try and implement to give the users of this software a better experience. Within the nonfunctional requirements, the two main things are expected are performance and reliability. All of these requirements are based on our own ideas, things that we have seen in well designed software, things that we expect and like to see from well designed software, and also the rules of checkers itself.

2.1.1. Functional. Base Requirements

- Graphical Interface - When starting up the game, users will be greeted with a menu, which will allow them to start a local game. The game will be played using a graphical interface which will display a

board along with the each of the player's twelve pieces in their starting positions. Using the graphical interface, users will be able interact with the graphical interface to move and take the other player's pieces.

- Ability to move and take pieces - After the graphical interface has loaded, the player will be able to click on a piece and make a valid move. Then, the graphical interface will display the move that was made, shifting the position of the piece that was moved. If the player is able to take a piece, they can click on their piece then click the appropriate square to move the piece there, taking the opponent's piece in the process.
- Move highlighting - When a player presses on one of their pieces, all valid moves that the piece can make will be displayed on the board using a lighter-colored image of the selected piece.
- Promotion mechanics and moving with a king - After a player has successfully moved a piece to the last square on the other side of the board, the player's piece will be promoted to a king. These pieces are able to make valid moves in the forward direction from where they started, like regular pieces, but they can also make valid moves backwards. This makes kings more powerful than a regular piece and can allow them to take pieces by moving backwards.

Stretch Requirements

- Theme Creation - If this stretch goal is implemented, there will be an options menu within the main menu, allowing the user to change their theme. This will allow the users to customize the checker board by changing its color, which will allow the user to better customize their experience. There could also be a feature where custom images could be imported to color the tiles on the board or change the pieces.
- Save/load games - When playing the game, there will be an option to save the game to a folder. It will be saved to a special file type that the game will be able to be load. After loading, the user will be able to continue the game from where they last left off.
- Playing against someone using the Save/load game method - After saving the game, the user will be able to send the file to another person. When the person receives the file, they will be able to open it using the game. After double-clicking the file, the game will open, allowing the user to make their move, save and send it back to the other player.
- Networking - If multiplayer is implemented, the user will have the option to join or host a game. If the user decides to host a game, they will open a port to allow user connection to their computer so that the game may process the requests from the non-host player. When playing, the game would allow the host player to make the first move. The game would then package the data and send it to the other user, updating their game and allowing them to play their move. The game would then do the same thing with the non-host player. It would allow them to move, package the data, and send it back to the host to update.

2.1.2. Non-Functional. Base Requirements

- Performance - When playing locally, the application must update within real time whenever a piece is moved, allowing for the next person to move as well. This will allow the player or players to play the game without latency or delay.
- Reliability - When playing, whether the stretch goals are reached or not, the application must be reliable. It should not crash, bring the performance promised, and maintain the game according to the rules of checkers.

Stretch Requirements

- Performance - When playing across a network, we would expect the application to update whenever it receives a valid request. The application will await the move from the user and send it as a response. Whenever playing using the save game method, we would like to allow the user to load the game in a matter of seconds by double-clicking on the file, skipping the menus and loading the game instantly.
- Scalability - This application will be scalable. It will be able to be used by any number of people wanting to play at the same time due to the fact that it opens a port and allows connection to another computer running the game.
- Data Integrity - When saving the game and sending it to another player, we wish for our games to be incorruptible and not editable.

2.2. Use Cases

These use cases are the different scenarios that the average user will experience when using our software. They describe how the user is going to interact with its different features. The complexity level in the table is how difficult we believe the

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Start Checkers Game	Player	Low	1
2	Move Checker	Player	High	1
3	Hop (take piece)	Player	High	1
4	Make King	Player	Med	2
5	End Game	Player	Low	3
6	Forfeit Game	Player	Low	4

TABLE 1. CHECKERS GAME USE CASE TABLE

different cases are to implement and the priority ranking represents how important it is for each feature to be implemented. The number one is the highest priority ranking while the number five would be the lowest (see Table 1).

Use Case Number: 1

Use Case Name: Start Checkers Game

Description: The player will need to select either the Local or Multiplayer (stretch) button on the main menu of the game. For a local (offline) game they will immediately be greeted with the checkers board and the correct amount of pieces. For Multiplayer they will have to connect to a valid game at the main menu (see Figure 1).

1) User hovers the mouse over the Local button on the main menu.
2) User left clicks on the Local button.
3) The checkers game is loaded and the user is greeted with a checkers board with 24 pieces.

Termination Outcome: The game has been loaded (see Figure 2).

Alternative: Multiplayer Game (stretch goal) (see Figure 1)

1) User hovers the mouse over the Multiplayer button on the main menu.
2) User left clicks on the Multiplayer button.
3) User will be met with another window that opens that allows them to type in an IP address and scan for a game. The user must hover over the IP box and left click. The user can then type in the IP address or they can hover over and left click on the host button to host their own game. Once a valid game is found the checkers board screen will be loaded.

Termination Outcome: The game has been loaded.

Use Case Number: 2

Use Case Name: Move Piece

Description: When it is a player's turn, they can select a piece to move on the board. Once a piece is selected the game will highlight valid moves that can be made by the piece. They can then drag and drop said piece on a valid move location and move the piece. Regular pieces can only move to the other side of the board, never backwards, but kings can move backwards (see Figure 3).

1) User hovers the mouse over the piece they would like to move while it is their turn.
2) The user will left click on the piece and this will grab the checker.
3) The game will highlight the positions of valid moves for the piece. The player can move the grabbed piece over a highlighted square and press left click again to release the piece. The piece will then be moved to that position.

Termination Outcome: The piece has been moved.

Use Case Number: 3

Use Case Name: Hop (take piece)

Description: When it is a player's turn, they can select a piece to move on the board. If there is an enemy piece one space to the front left or right of the piece (or back if the piece is a king) a user can select their piece and hop over the piece (move two spaces to the left or right). This takes the enemy piece and removes it from the board (see Figure 4).

1) User hovers the mouse over the piece they would like to move while it is their turn.
2) The user will left click on the piece and this will grab the checker.
3) The game will highlight the positions of valid moves for the piece. The player can click on a highlighted square behind an enemy checker which will be highlighted if the move is valid. The piece will then be moved to that position and the enemy piece will be deleted.

Termination Outcome: The piece has hopped over an enemy checker and the enemy checker has been removed from the board.

Use Case Number: 4

Use Case Name: Make King

Description: If a piece is moved by a user all the way to the opposite side of the board from which it started, that piece turns into a king, which is a piece that can move both forwards and backwards (see Figure 5).

1) User hovers the mouse over the piece they would like to move while it is their turn.
2) The user will left click on the piece and this will grab the checker.
3) The game will highlight the positions of valid moves for the piece. If a player is one square away from the opposite end of the board that they started on, the piece can be placed on an unoccupied square to the left or right of the piece. This will turn the piece into a king.

Termination Outcome: The piece has been moved and has transformed into a king.

Use Case Number: 5

Use Case Name: End Game

Description: Once all of a player's pieces are taken the game will end and a green message box will appear declaring the winner. The user can then click the quit button and return to the main menu of the game (see Figure 6).

1) User selects a piece and uses it to take the other player's final piece. The victory message will appear.
2) User hovers the mouse over the quit button.
3) User presses the left click button and the program returns to the main menu.

Termination Outcome: The game has been ended and the user is returned to the main menu.

Use Case Number: 6

Use Case Name: Forfeit Game

Description: During a players turn, they can chose to press the forfeit game button at the top right corner of the screen to give the win to the other player and return to the main menu. This button is mainly for if the player can no longer move, meaning that the game is over, but it can also be pressed if they do not think that they can win anymore or no longer wish to play.

1) Once a user's turn begins, they can hover the mouse over the forfeit button.
2) User presses the left click button.
3) A window proclaiming the winner will pop up.
4) The user left clicks "ok" or exit on the window and they will be returned to the main menu.

Termination Outcome: The game has been ended, the winner has been proclaimed, and the user is returned to the main menu.

2.3. Interface

Figure 1 shows a screenshot of the main menu of our checkers game. The main menu features a button for starting a local game. If the user presses the "Settings" button the window will close and a new window will open where they can set the color of the board. The "Local Game" button will close the menu window and open a window with the checkers game board. Pressing "Quit Program" will exit the program (see Figure 1).

Figure 2 shows the defult starting screen for a checkers game. When the user starts a game, they are greeted with a screen featuring a checkers board with twelve pieces on each side. The amount of pieces for each player is displayed on the right hand side of the board and the game can be exited by pressing the "Quit Game" button in the top right of the screen. A player also has the option of ending the game by forfeiting during their turn. This is accomplished by pressing the "Forfeit" button (see Figure 2).

Figure 3 shows the player selecting a regular checkers piece to be moved. On both the left and right sides of the piece, the game highlights the two valid moves that the piece can make. If either the left or right space is occupied by another friendly checker, the player will not be able to move to that position. If both spaces are occupied by a friendly piece, the checker can not move. If the spaces are occupied by enemy checkers, then a hop can be performed (see Figure 3).

Figure 4 demonstrates a potential hop move to take an enemy piece. Since the space to the right diagonal of the selected checker is occupied by an enemy checker and the space behind that enemy checker is not occupied, the enemy piece can be hopped. This removes the enemy piece from the board and moves the attacking checker to the left diagonal space behind the enemy checker (the piece moves two spaces instead of one) (see Figure 4).

Figure 5 shows a king piece being moved. In order for a piece to become a king it must reach the opposite side of the board from which it started. Once a piece becomes a king, it can move both forwards and backwards in the usual diagonal manor. In this screenshot, the king has been selected and the game is telling the player that the king can either move to the left or right black square in the backwards direction for a blue piece (see Figure 5).

Figure 6 shows a game over screen. Once all of the other player's pieces are taken the game will end and the winning player will be displayed in a message box. The game must now be exited by closing the message box. This will return the program to the main menu and the user can either start a new game or exit the program by pressing quit (see Figure 6).

Figure 7 shows the forfeit button being used to end the game and give the win to the other player. As was already mentioned, this button's primary function is to allow the user to end the game when no more moves can be made, but it can also be used to end the game and give the win to the other player at any point in the game. When it is a players turn, they always have the option of pressing the forfeit button if they would like to (see Figure 7).

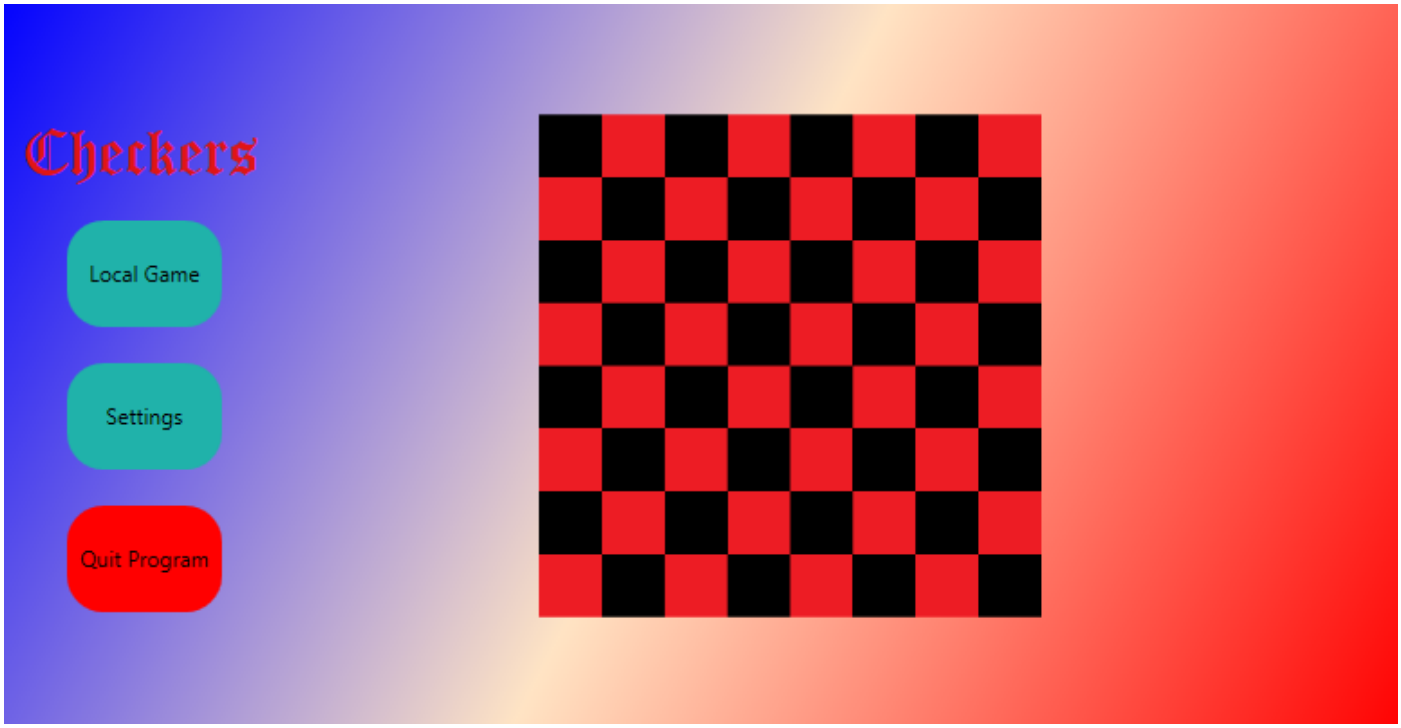

Figure 1. This diagram shows the main menu of the program.

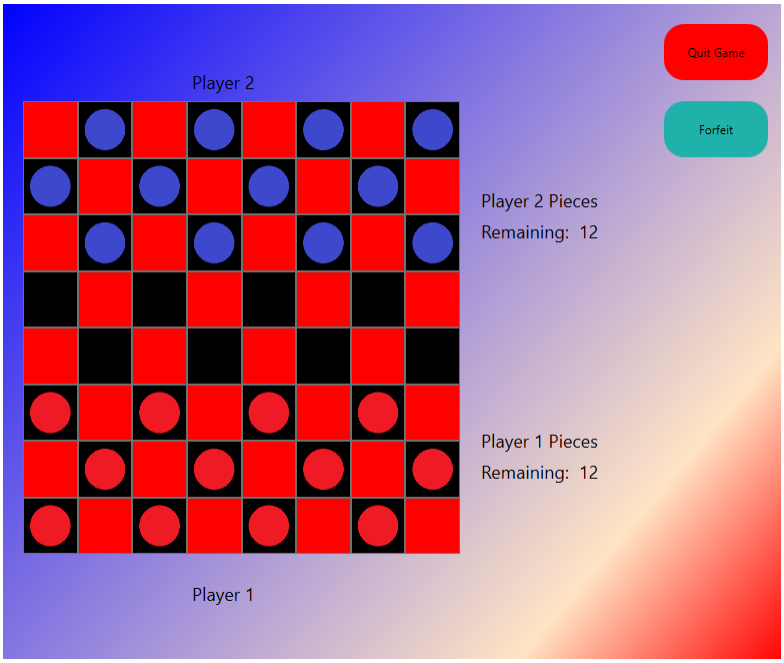

Figure 2. This shows the default starting screen for when the player starts a new game.

3. Project Timeline

Figure 8 shows the development process of our checkers game based on the Waterfall model of project development. As a result, the timeline showing our expected milestones and the development life cycle of our project is also based on the Waterfall model. The deadlines in the timeline that we have created are the same deadlines that have been set for this project in class. These are highlighted in red because these tasks have to be completed by the specified date. We have been

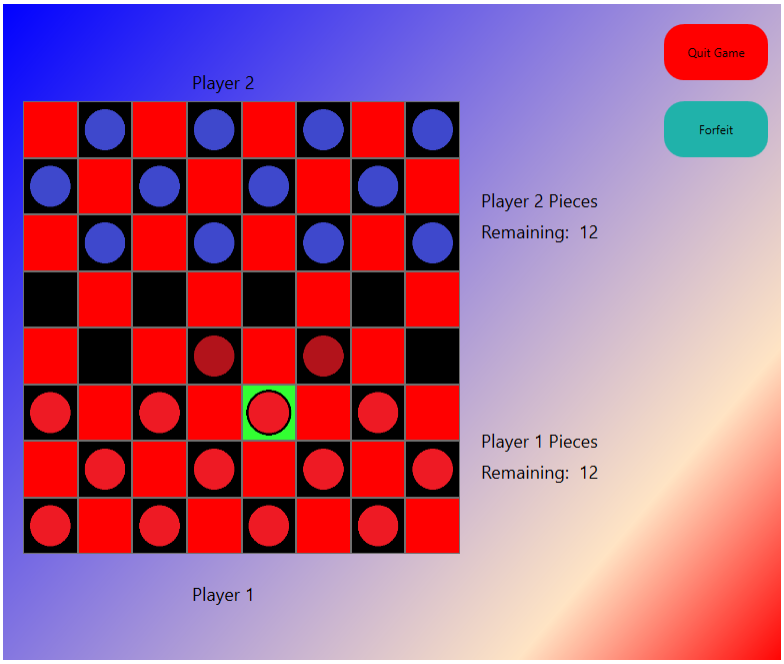

Figure 3. This screenshot shows the player selecting a regular piece and it's potential moves .

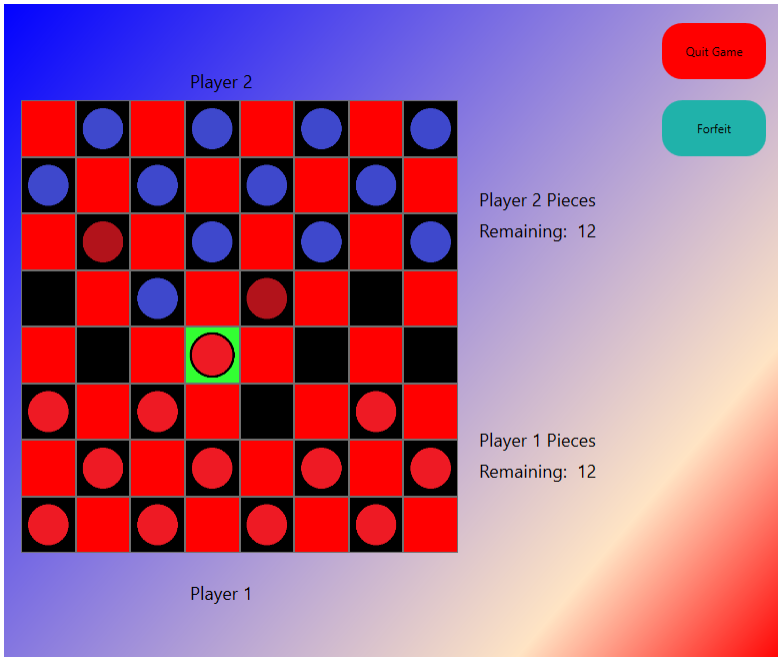

Figure 4. This screenshot shows hopping (taking an enemy piece).

able to meet all of the required deadlines over the course of this project. The goals, which are highlighted in orange, are tasks that we would like to have completed by the specified date in order to keep ourselves on track to meet the actual required deadlines. Because of this, these goals are looser than the actual deadlines, but would be good things to get done by or around the specified date to avoid any close calls with the actual deadlines. Over the course of the project we have been able to meet the goals in this timeline around the dates that were specified. The blue squares in this diagram are the different phases of the Waterfall model and we thought that including them in our timeline would make it easier to know what phase in the Waterfall model we are currently in. Generally speaking, we planned to meet all upcoming deliverable

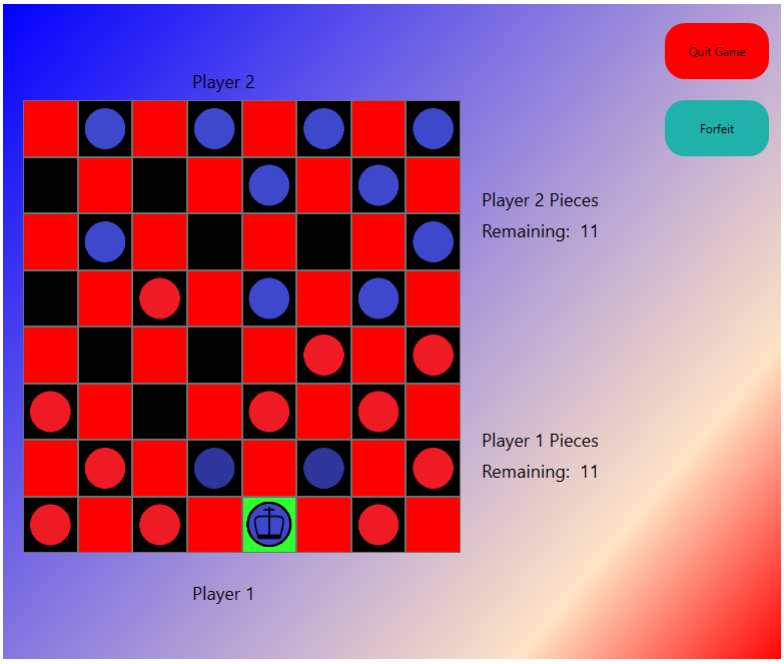

Figure 5. This screenshot shows the player selecting a king and it's potential moves.

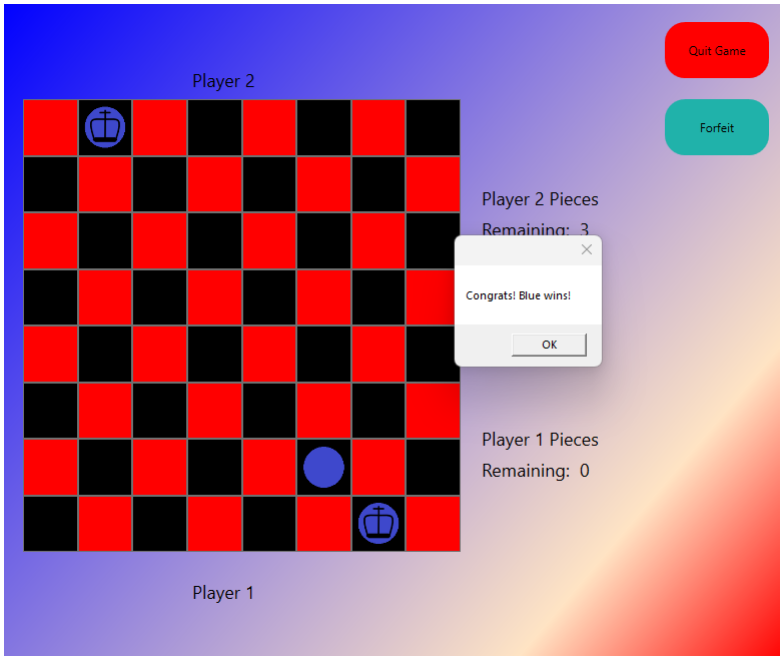

Figure 6. This screenshot shows the game over screen.

deadlines for this project and our projected goal deadlines by working hard on coding as a team and testing our project daily to make sure that it is meeting our established requirements. Over the course of the project, we managed to stay on task and we were able to meet all of our base requirements along with some of our stretch goals.

4. Project Structure

The following section explains the structure of our project and how we created it. Our checkers game begins with a main menu that greets the user with the ability to start a new local game, multiplayer game, or open the settings window.

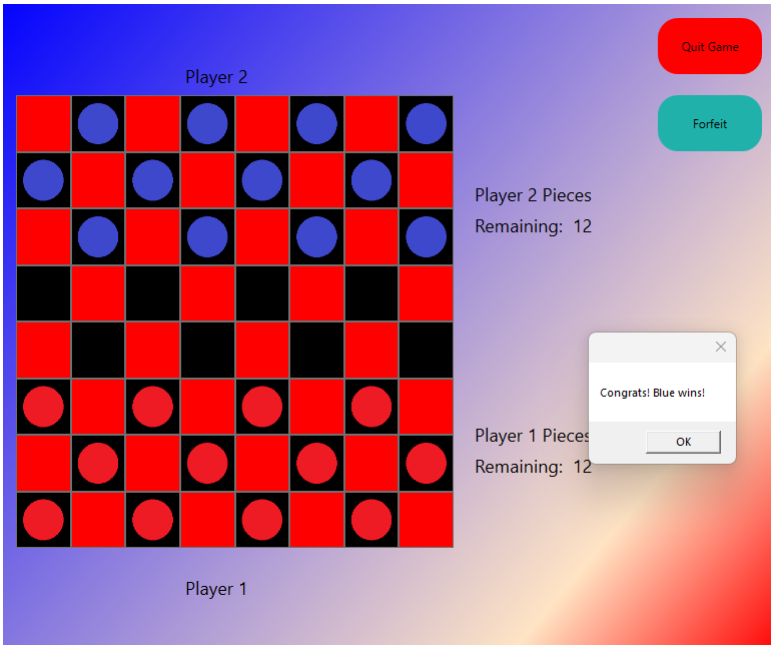

Figure 7. This screenshot what happens when a player forfeits the game.

All of these tasks can be accomplished by pressing their associated buttons which are clearly defined on the menu. From the main menu screen, the user can also exit the program by pressing the quit button. This main menu is backed by a xaml file that uses classes that are built into WPF to give the background visually pleasing color effects. The menu also contains a picture of a checkers board to the right of the buttons as a decoration and the buttons have been rounded off by creating a button style in the xaml. In order to open other windows, the buttons are programmed to create a new instance of the window that the user wants to open and close the current window when clicked.

In order for the checker board theme to be changed, the user must click the settings button which will close the main menu and open a settings window. The settings window contains buttons that are labeled with options for different colors and when the user clicks one of these buttons, the settings window will close and the user will be brought back to the main menu. Once a new game has been started, the color selection made at the settings window is used to generate the squares of the board. This feature is accomplished by way of the factory design pattern which will be elaborated upon later. The settings window also uses the same WPF classes to give it a visually pleasing aesthetic, but the colors are different on this menu than they were in the main menu for visual variety. The buttons on this screen are also rounded using the same type of style that was set in the main menu window.

If the user presses the local game button at the main menu, an offline two-player checkers game will start in a new window. This window can be exited at any time by pressing the quit button in the upper right corner of the screen. Pressing this will return the player to the main menu. The player 1 and player 2 sides of the board are labeled according to the rules of checkers where the person with the dark colored pieces would be considered the second player. The number of pieces each player has remaining on the board is also displayed on the right hand side of the screen so that the players know who is currently winning. This window contains a grid featuring a checkers game which is made out of buttons. The colored buttons are not usable by the player but the black buttons are the ones that the pieces can move on. When a piece is selected, valid moves are highlighted by a ghost piece of the same color showing the potential move. Currently, the moves are all valid and pieces can be taken and kinged. When a piece is kinged it gains the ability to move backwards as well as forwards and this occurs when a piece reaches the other side of the board from which it started. Once all of a players pieces are taken, the game ends with a pop-up window that lets the players know who won. Once this window is closed, the program returns the user to the main menu.

4.1. UML Outline

Figure 9 shows the UML for the factory design pattern we used for theming in our checkers program. As can be seen in the image, there is a Factory class that inherits from a SquareFactory interface that contains a GetSquare method that accepts a ColorType enumerator value. The method contained inside of the Factory class has a switch statement that calls classes that correspond to the different color options for squares on the board. These color classes inherit from the ISquare

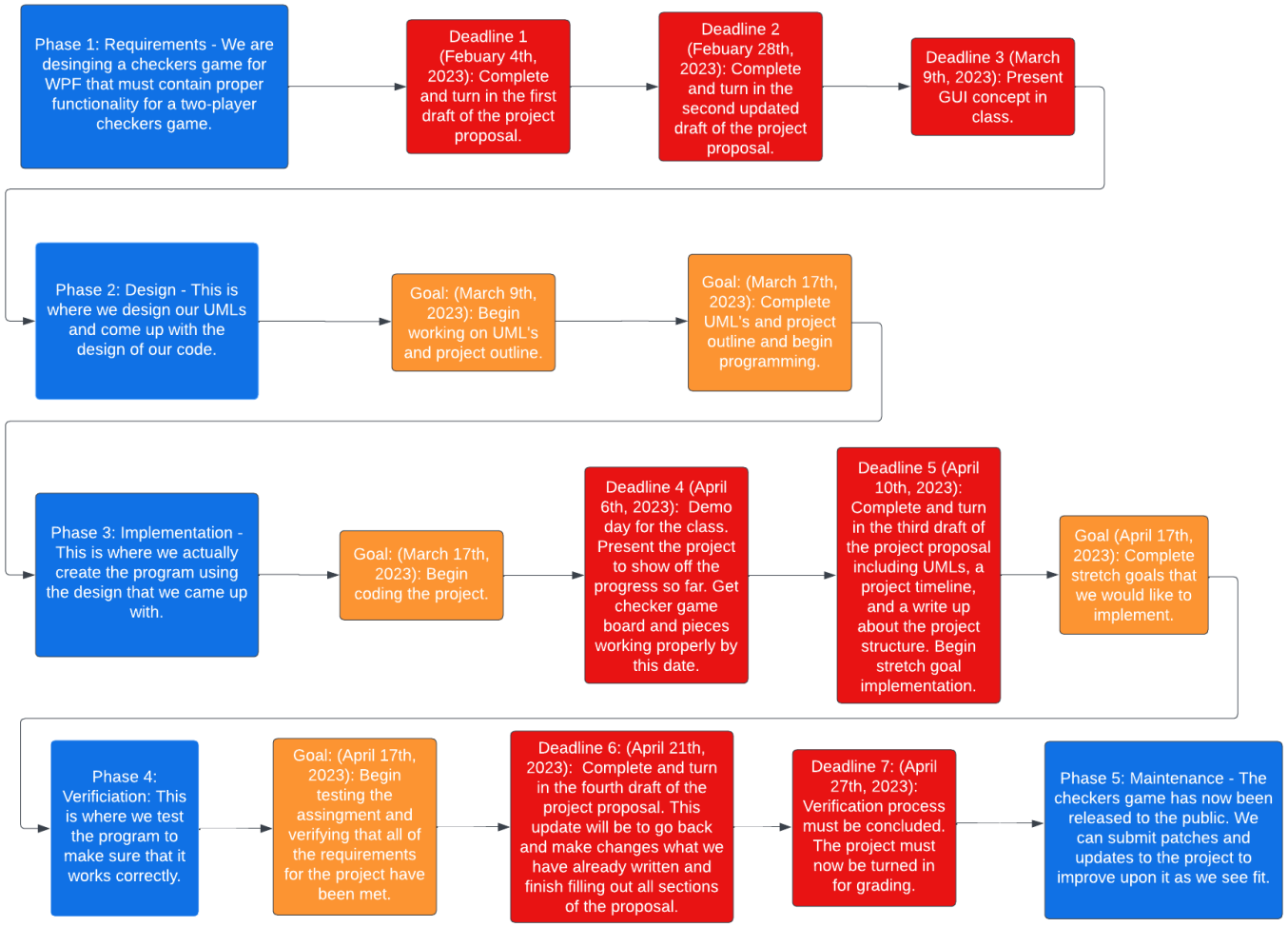

Figure 8. This diagram shows our projected timeline for the completion of our checkers project.

interface that contains one method for changing the color of the squares on the checker board. The classes that inherit from this interface generate and return buttons of different colors that are used to change the theme of the board. The board itself is a grid that is made out of buttons, so this can be used to change its color. After the buttons are generated they are returned to the Factory which then returns the ISquare object to the part of the program that called the factory.

Figure 10 shows the UML for the decorator design pattern that we used for the pieces and also contains the design for the checkers game part of the program overall. In the UML, the code backing the main menu can be seen along with the abstract piece class and the concrete decorator classes that inherit from it. Here the logic for the concrete position and player classes can also be seen along with the GameState enumerator. When the game starts, a player object is created that stores a 2D array of Pieces. This uses the abstract class to provide a basis for any type of Piece decorator to be held within the Piece array. Whenever someone wishes to move, the player uses the CheckValidMove method it contains to call the CheckValidMove method of a certain piece. Whenever this is called, the CheckValidMove method uses its specific implementation that it has for its different decorators to find the valid moves. If the decorator being called contains a component (KingDecorator), it will search all four directions that need to be checked instead of the two. After checking for valid moves, the player object uses the move method to either take pieces or swap the piece that is the argument and the BlankPiece on the board.

4.2. Design Patterns Used

For this project we used the decorator and factory design patterns. The factory design pattern has been used to implement theming for the checkers board. As was previously seen in the UML, there is a SquareFactory interface which provides

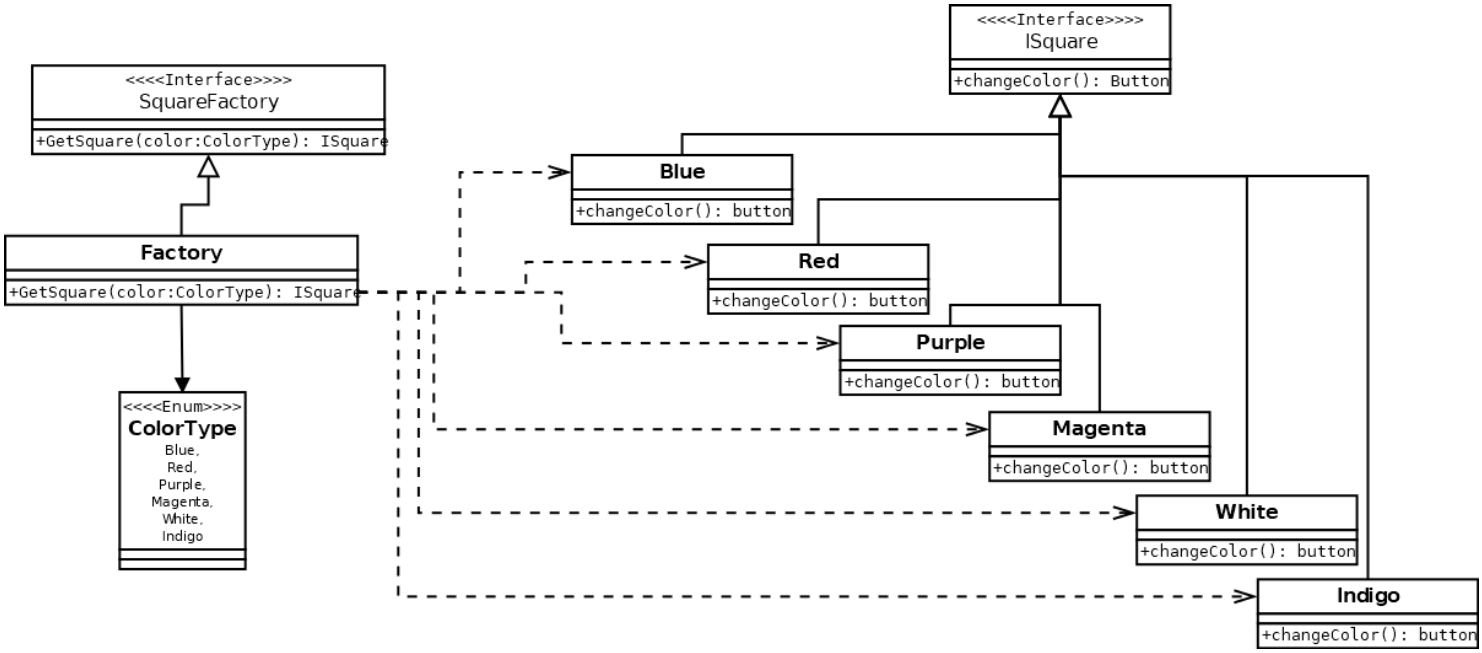

Figure 9. This is an image of the UML that was created to show the logic of the board square factory.

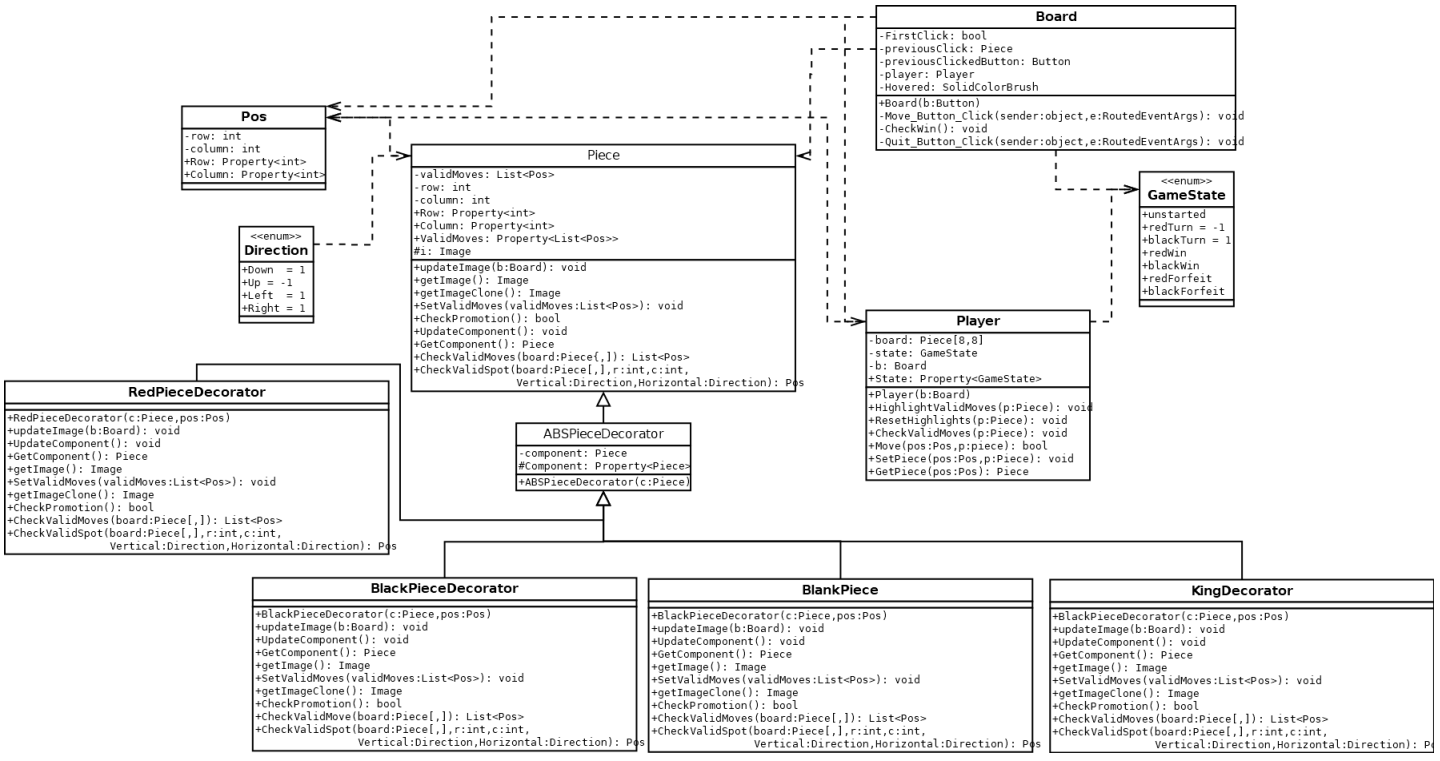

Figure 10. This diagram shows the design of the decorator pattern for the pieces.

the required method for a factory that can generate squares for our checkers board. In this particular case, we are only using one Factory class which can be seen inheriting from the SquareFactory interface. There are six different color options including, but not limited to, a blue square, a red square, and a purple square that are called using a switch statement inside of the Factory class. In our checkers game, these squares are buttons. The different color choices are the different board theme options that are currently available to the user in the checkers game. Depending on a selection that is made by the

user in the settings window, a new checkers game board will be opened and the method corresponding to the color choice that they picked will be called to generate a button object of that color, which is then used to fill in the colored squares on the board. The black squares cannot be changed because this could make the pieces hard to see. Finally, it is important to note that all of the different button classes that generate the different colored buttons inherit from the ISquare interface which contains the changeColor() method that they all must contain in order for this pattern to work correctly.

We also used the decorator design pattern in order to create a 2D array of pieces. This allows us to manipulate the pieces in the backend. We created four different decorators to accomplish this:

- BluePieceDecorator
- RedPieceDecorator
- BlankPiece
- KingDecorator

The abstract class for all of these decorators is ABSPieceDecorator. It contains a private Piece component so that each decorator can contain another decorator. Within the backend, the top-most(visible) decorator will always be a BluePieceDecorator, RedPieceDecorator, or BlankPiece. The component for a BlankPiece is always null. No component methods are used here. However, for the BluePieceDecorator and RedPieceDecorator, if the component is KingDecorator, the methods for the KingDecorator are called using the methods for the KingDecorator. This allows the pieces to move forwards and backwards. Each time the player object's method is called to manipulate pieces, it lets the pieces perform their checks, and then the player object moves the pieces (see Figure 10).

5. Results

In our project we have managed to meet all of our base requirements and one of our stretch goals. The base checkers game is fully operational and enforces the rules of checkers automatically. We originally intended for multiple piece hopping to be a feature, but later found out that this is not an official checkers rule and is only included in some versions of the game. Therefore, this idea was cut from the game. In this checkers game, both players can win the checkers game, both the king and regular pieces work exactly as intended, and we added the ability for the player to forfeit the game during their turn, which prevents the game from locking up if the player no longer has the ability to move any of their pieces. We also managed to implement different color options to give the user the ability to customize the visuals of the checker board to better suit their preferences which was one of our stretch goals. We also put in a lot of effort to make the game look visually appealing by using WPF to create colorful background and rounded buttons to make the game more visually pleasing. We worked on the networking/multiplayer component for the game, which was another one of our stretch goals, however this has not been included in the game at the moment because it does not fully work. The stretch goal that would give the player the ability to save the game's progress and load it at a later date was never implemented due to time constraints. When we began this project, we were not sure how far we would get and we had almost no idea of how we were going to create this checkers game in WPF, but due to hard work and a lot of research, we have managed to meet all of the required goals that we set for this project in the beginning. Overall, we believe that this project has been a success.

5.1. Future Work

Overall, we have put a decent amount of work into our stretch goals for this project. Because of this, our next goal with this project is going to be to finish the networking component and then add it to the game. We would not want the work that we have put in on this feature to go to waste and it would be good if we could get this part of the game working, but it is currently not operational and we did not want to have features that do not work correctly in our game until we get them figured out. We might also put in the save game feature at a later date to give the game more selling points over other checkers simulation programs. Finally, we might also go back and put in the ability to import custom images for pieces and board tiles, which was another stretch goal. For now though, this checkers game is an offline, two-player experience, which is what our original goal was to begin with.