Checkers Game
for UTM CSCI 352

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. When it comes to the progress on the project so far, we have thought of various ideas that we will use to make this project a success while also improving our own coding skills in the process.

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. Our target audience for this project is anyone who wants to be able to play checkers on their computer. This includes competitive players who may compete in tournaments and have a high skill level and also casual players who occasionally enjoy checkers in their free time. If we succeed in creating a well crafted checkers experience, 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 is 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, 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. The person with the black pieces moves always first. 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 will have 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 allows people to relax and decompress, which is good for both their physical and mental health. Our game will hopefully give people an enjoyable 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 tournament, which do, in fact, exist.

1.3. Challenges

There are a number of places where we could end up stuck during this project. One of them is having to delete pieces off of the board when they are taken. We will need to implement some kind of animation, likely the piece shrinking into nothingness, so that this does not look lazy or choppy. Making a piece move two spaces once another piece is taken is also something that we will have to figure out how to implement. This could likely be accomplished using a boolean value or some kind if counter that resets to false or zero when the turn ends. We will need to put in logic that lets the program know which spaces are occupied and which are not to avoid illegal movies from being done. We will also need logic to make sure that pieces can't make illegal moves in general, which might be the most difficult challenge of all, because we will need to make sure that all scenarios where a move is not legal are accounted for. The graphics for the game itself may also be a challenge, because we want to make sure that the game looks good and provides users with an immersive experience. This includes making sure that the pieces and board look convincing and that the pieces don't blend into the board making it hard to see. We will also need to make sure the kings are clearly discernible from the regular pieces. Finally, we will need to put in logic that lets the program know when the game is over. All of these things will be challenging, but we will hopefully be able to overcome them with our coding skills.

2. Scope

The checkers game will be considered done when we have a game of checkers in which you can take pieces, can king your pieces, and can jump more than once if the pieces on the board are in the right position for such a move. 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. 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.

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 are listed below because the base requirements are what is needed for the base implementation of this software. These requirements, both stretch and base, is what is expected from a basic game of checkers, and therefore, the requirements for the completion of this software. From graphics to promotion mechanics, the base functional requirements are what is needed to play a very basic game of checkers. Within the nonfunctional requirements, two main things are expected that is performance and reliability. The stretch requirements are also listed below.

2.1.1. Functional. Base Requirements

- Graphical Interface - When starting up the game, you will be greeted with a menu, which will allow you to pick local or multiplayer game, if we achieve the stretch goals. If not, a local game will be available. The game will be played using a graphical interface which will display a board along with the each of the team's twelve pieces. Using the graphical interface, you will interact with the graphical interface to move and take 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. If the player is able to take a piece, they can click on their piece then press on the appropriate square to move the piece there, taking the opponent's piece.

- Move highlighting - When a player presses on one of their pieces, all valid moves using that piece will be displayed upon the board using a lighter-colored image of the piece selected.
- Promotion Mechanics and moving with a king piece - After a player has successfully moved a piece to the last square on the other side of the board, the player's piece will promote to a king. These pieces are able to make valid moves forward, like the rest of the pieces, but they can also make valid moves backwards. This allows them to take pieces forwards and 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 users to add an image file, that is within the constraints of each tile of the board, to change the image of the regular and king pieces. Within this same menu, the user will be able to change the color of the tiles on the board. Both of these combined together will allow the user to create a personally customized game of checkers.
- 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.
- 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 a game or host a game. If the user decides to host a game, he will open a port to allow user connection to his 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 checker.

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 must 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.
- Security - Due to the fact that a connection will be open to another computer, we need to be able to deal with bad requests from one computer to another; therefore, some level of security must be obtained.
- 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

Use Case Number: 1
  Use Case Name: Start Checkers Game
    Description: The player will need to select either the Local or Multiplayer 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.

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

TABLE 1. CHECKERS GAME USE CASE TABLE

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 move the grabbed piece over a highlighted square behind an enemy checker if the piece is in the correct position and press left click again to release the piece. 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.

2.3. Interface Mockups

This figure shows a conceptual design for the main menu of the checkers game. The main menu will feature a button for creating a local game. A stretch goal for a button to search for and connect to or host a multiplayer game is also planned to be included. Clicking the multiplayer button will open another window with these options. 1
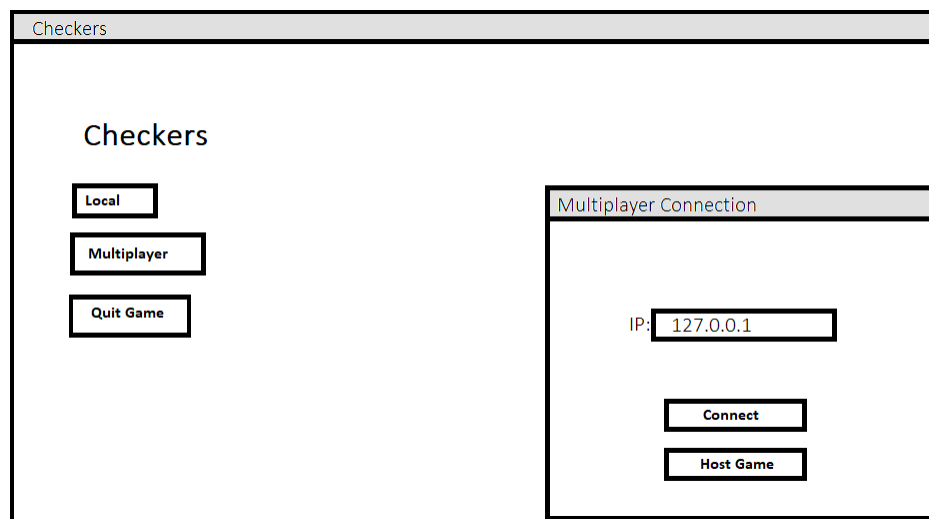

Figure 1. This diagram shows the main menu of the program.

When the user starts a game, they will be greeted with a screen featuring a checkers board with twelve pieces on each side. The amount of pieces for each player will be displayed and the players will have the ability to input their names. This diagram shows potential implementation for all of these ideas. 2

This diagram shows the player selecting a regular checkers piece to be moved. On both the left and right sides of the piece, the game will highlight 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. 3

This diagram demonstrates a potential hop move to take an enemy piece. In this diagram, the left side of the selected checker has been blocked by a friendly piece, so the checker cannot travel to the left. 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 right diagonal space behind the enemy checker (the piece moves two spaces instead of the usual one). 4

This diagram 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 the diagram this is demonstrated. The king is being 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. 5

This diagram shows a game over screen. Once all of the other player's pieces are taken or they are in a state where they can no longer move any of their pieces, the game will win and the winning player will be displayed in a green message box. The game must now be exited by pressing the quit button. 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. 6

3. Project Timeline

Go back to your notes and look up a typical project development life cycle for the Waterfall approach. How will you follow this life cycle over the remainder of this semester? This will usually involve a chart showing your proposed timeline,
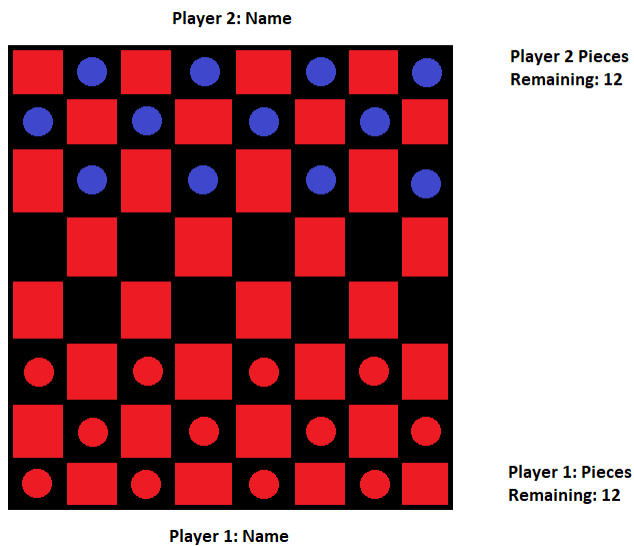
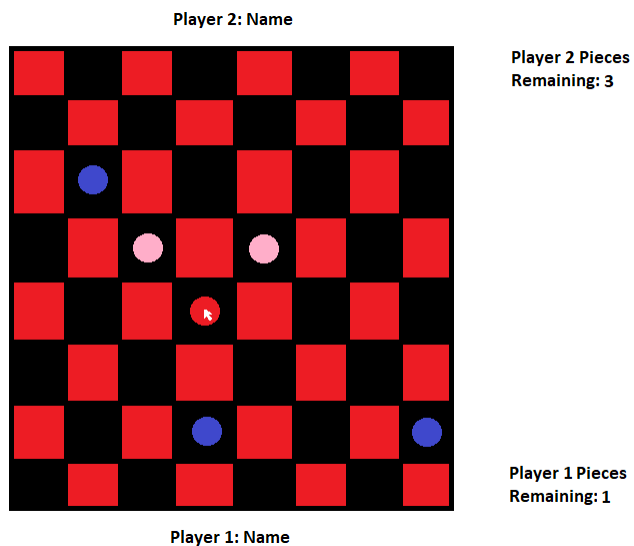Figure 2. This shows the default starting screen for when the player starts a new game.

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

with specific milestones plotted out. Make sure you have deliverable dates from the course schedule listed, with a plan to meet them (NOTE: these are generally optimistic deadlines).

4. Project Structure

At first, this will be a little empty (it will need to be filled in by the time you turn in your final report). This is your chance to discuss all of your design decisions (consider this the README's big brother).

4.1. UML Outline

Show the full structure of your program. Make sure to keep on updating this section as your project evolves (you often start out with one plan, but end up modifying things as you move along). As a note, while Dia fails miserably at generating pdfs (probably my fault), I have had much success with png files. Make sure to wrap your images in a `figure` environment, and to reference with the `ref` command. For example, see Figure **??**.
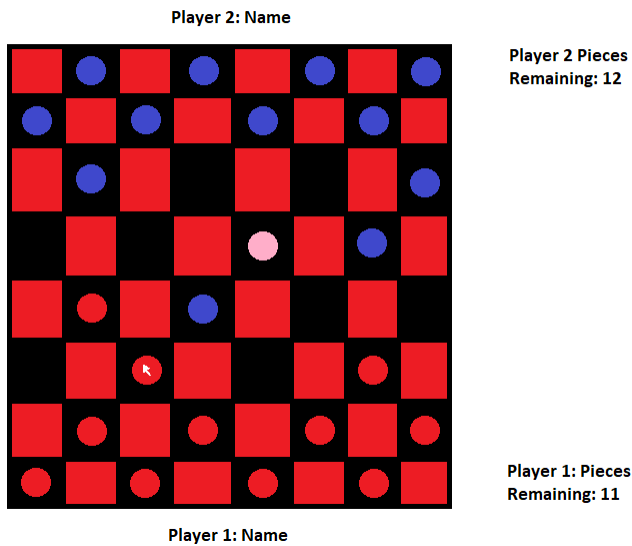
Player 2: Name

Player 2 Pieces
Remaining: 12

Player 1: Pieces
Remaining: 11

Player 1: Name

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

Player 2: Name

Player 2 Pieces
Remaining: 8

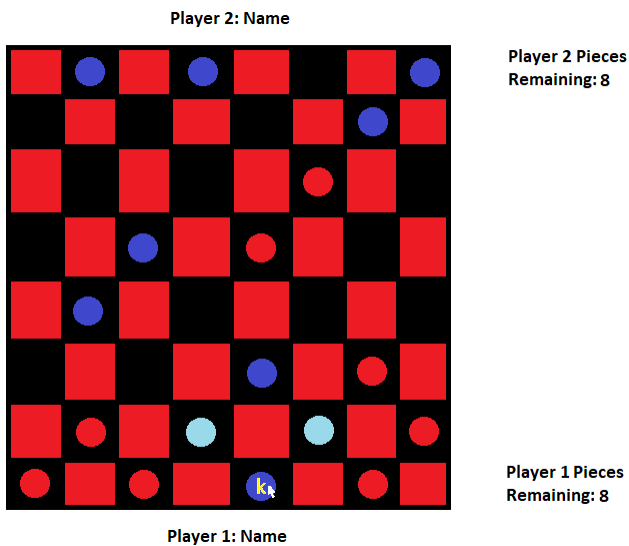Player 1 Pieces
Remaining: 8

Player 1: Name

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

4.2. Design Patterns Used

Make sure to actually use at least 2 design patterns from this class. This is not normally part of such documentation, but largely just specific to this class – I want to see you use the patterns!

5. Results

This section will start out a little vague, but it should grow as your project evolves. With each deliverable you hand in, give me a final summary of where your project stands. By the end, this should be a reflective section discussing how many of your original goals you managed to attain/how many desired use cases you implemented/how many extra features you added.
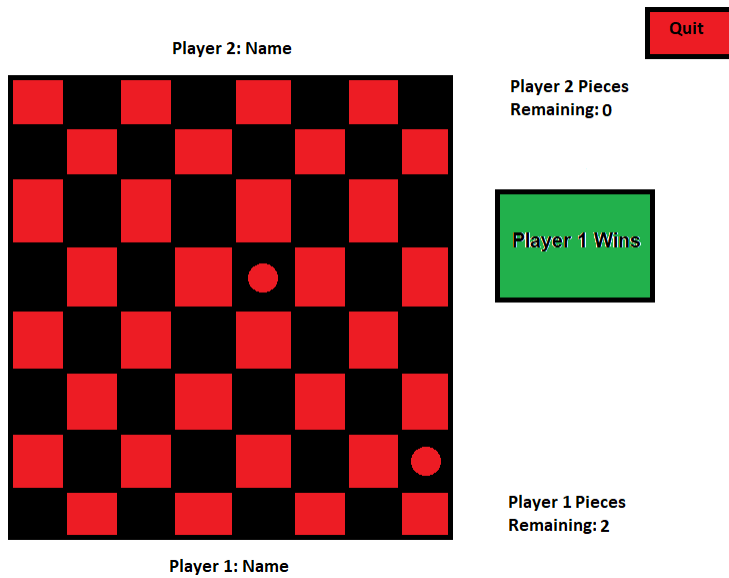
Figure 6. This diagram shows the game over screen.

5.1. Future Work

Where are you going next with your project? For early deliverables, what are your next steps? (HINT: you will typically want to look back at your timeline and evaluate: did you meet your expected goals? Are you ahead of schedule? Did you decide to shift gears and implement a new feature?) By the end, what do you plan on doing with this project? Will you try to sell it? Set it on fire? Link to it on your resume and forget it exists?

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.