Checkers Game: Final Project for UTM CSCI 352

Connor Walsh
Daniel Cunningham

Abstract

The project covered in this paper is a single screen checkers simulation 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 comparatively 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 a simulation program. 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 can be played by one or two players, but the intention is for two players to play against one another. The one-player experience will solely be for players who wish to experiment with the game and set up hypothetical 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 comparative 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

When we were first given this assignment, we knew that we wanted to adapt a board game of some sort. We originally considered adapting chess, but chess is a 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. 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, they move two spaces. There are 24 pieces on the board. Half of these pieces go to one player and the other half go to the other. The pieces are usually red and black. The person with the black pieces moves 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.

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. Making a piece move two spaces once another piece is taken is also something that we will have to figure out how to implement. We will need to put in logic that lets the program know which spaces are occupied and which are not. We will also need logic to make sure that pieces can't make illegal moves, which might be the most difficult challenge of all. The graphics for the game itself may also be something we could get stuck on. We will need to make 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 put in logic that lets the game 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.

2.1. Requirements

As part of fleshing out the scope of your requirements, you'll also need to keep in mind both your functional and non-functional requirements. These should be listed, and explained in detail as necessary. Use this area to explain how you gathered these requirements.

2.1.1. Functional.

- User needs to have a private shopping cart – this cannot be shared between users, and needs to maintain state across subsequent visits to the site
- Users need to have website accounts – this will help track recent purchases, keep shopping cart records, etc.
- You'll need more than 2 of these...

2.1.2. Non-Functional.

- Security – user credentials must be encrypted on disk, users should be able to reset their passwords if forgotten
- you'll typically have fewer non-functional than functional requirements

2.2. Use Cases

This subsection is arguably part of how you define your project scope (why it is in the Scope section...). In a traditional Waterfall approach, as part of your requirements gathering phase (what does the product actually *need* to do?), you will typically sit down with a user to develop use cases.

You should have a table listing all use cases discussed in the document, the ID is just the order it is listed in, the name should be indicative of what should happen, the primary actor is typically most important in an application where you may have different levels of users (think admin vs normal user), complexity is a best-guess on your part as to how hard it should be. A lower number in priority indicates that it needs to happen sooner rather than later. A sample table, or Use Case Index can be seen in Table 1.

Use Case ID	Use Case Name	Primary Actor	Complexity	Priority
1	Add item to cart	Shopper	Med	1
2	Checkout	Shopper	Med	1

TABLE 1. SAMPLE USE CASE TABLE

Use Case Number: 1
Use Case Name: Add item to cart
Description: A shopper on our site has identified an item they wish to buy. They will click on a "Add to Cart" button. This will kick off a process to add one instance of the item to their cart.

You will then go on to (minimally) discuss a basic flow for the process:

1) User navigates to page listing desired item
2) User left-clicks on "Add to Cart" button.
3) User cart is updated to reflect the new item, this also updates the current total.

Termination Outcome: The user now has a single instance of the item in their cart.

You may need to also add in any alternative flows:

Alternative: Item already exists in the cart

1) User navigates to page listing desired item
2) User left-clicks on "Add to Cart" button.
3) User cart is updated to reflect the new item, showing that one more instance of the existing item has been added. This also updates the current total.

Termination Outcome: The user now has multiple instances of the item in their cart.

You will often also need to include pictures or diagrams. It is quite common to see use-case diagrams in such write-ups. To properly reference an image, you will need to use the `figure` environment and will need to reference it in your text (via the `ref` command) (see Figure 1). NOTE: this is not a use case diagram, but a kitten.

After fully describing a use case, it is time to move on to the next use case:

Use Case Number: 2
Use Case Name: Checkout
Description: A shopper on our site has finished shopping. They will click on a "Checkout" button. This will kick off a process to calculate cart total, any taxes, shipping rates, and collect payment from the shopper.

You will then need to continue to flesh out all use cases you have identified for your project.

Figure 1. First picture, this is a kitten, not a use case diagram

2.3. Interface Mockups

At first, this will largely be completely made up, as you get further along in your project, and closer to a final product, this will typically become simple screenshots of your running application.

In this subsection, you will be showing what the screen should look like as the user moves through various use cases (make sure to tie the interface mockups back to the specific use cases they illustrate).

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, 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 2.

Figure 2. Your figures should be in the *figure* environment, and have captions. Should also be of diagrams pertaining to your project, not random internet kittens

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.

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.