Memory Game Project
for UTM CSCI 352

Tony Anderson and Vrushank Mali

Abstract

This Project is going to be a Memory Game that keeps score of how many matches the player got, Reset button, Start Screen, Game over Screen, basic things that computer games have. The target is really anyone that likes memory games or wants to test their memory. We have come up with the general idea of what its going to look like.

1. Introduction

With this project, we are going to try to make a fun but simple memory game and by the end have something that could maybe be turned into an app later on. If people play this game, we hope they just have fun and not compare it to one of those bad ad infested games on the app store. This is a first step into maybe creating games for a living or just to get a better understanding on how 2D video games are made.

1.1. Background

The concepts you need to know are good C++ knowledge as well as some c-sharp or object oriented programming language (the more you know the better you will understand). If you don't know any of what was just said, I highly advise you to go read up on them before you read this paper. We decided to do this type of project because we play video games all the time and want to make our own and learn how complex or easy it is.

1.2. Impacts

The only impact that this project would really have is on health. Computer Eye Strain is a real and sometimes bad thing when looking at a computer screen for long periods of time. We highly recommend if you feel your eyes getting tired or worse start getting blurry even when you look away from the screen, take a break from your work, game, or whatever you are doing. Don't ruin your eye sight just for a silly game.

1.3. Challenges

We think the challenges of this project will be the: Start Screen, Game Over Screen, Animating the rotation of the pictures, Randomly shuffling the pictures around, Fading in the pictures for the user to memorize (just an idea, might not implement), Having the pictures fade away when the player gets a match (just an idea, might not implement),

2. Scope

The goals for this project are simple: 1. Keep up a good pace so we don't fall behind on coding it, 2. Have a working game that plays good and doesn't feel hacked together, 3. Have everything working 2 weeks before due date to go over bugs or problems, 4. Be happy with what we made and how it all came together

Some stretch goals we have are to try to have the pictures fade in for the user to memorize, and having the pictures fade away when the player gets a match.

We feel the project will be done when we are happy with how it all looks and plays, as well as being able to play an entire game and have no major hiccups (in a perfect world we would want it to have no hiccups at all, but this is a project for a semester of class and limited time to work every bug out). As long as we feel like it plays well and seems fun to play, it will be done. As far as time goes, we would like to have at least two weeks before the project is due to go over bugs and problems with code.

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.

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

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 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.

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!

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

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.