# Rational for this Project

A group of people posing for a photo

Description automatically generatedIn this project we initially decided we wanted to make a game for the female gaming market. We took into account trends that we’d observed in the American and Japanese gaming market aimed at female players, as well as our own tastes in games.

A picture containing indoor, floor

Description automatically generatedThe two trends we zeroed in on were dress up games, a popular example being the mobile game *Love Nikki-Dress UP Queen*, and narrative heavy Choose-Your-Own-Adventure style games, such as the mobile game *Choices*. We also thought the idea of a mobile game would be preferable, as smartphones are rapidly becoming normal for most people to have, which in turn means the potential users do not need additional hardware purchases in order to play.

Figure - Choices. A popular choose your own adventure style game. Image source: https://rhapsodyofsound.wordpress.com/2016/12/12/choices-story-you-play/

Figure - Love Nikki Dress Up Queen. A popular dress up game. Image Source: https://www.pocketgamer.biz/asia/comment-and-opinion/66449/love-nikki-more-than-meets-the-eye/

Although less directly influential, we also looked at the Japanese romance games where there is a female lead with one or more potential male love interests. Anna Carlson had previously been to Japan as an exchange student and considered the genre in light of the potential business opportunities in translating and importing the games. In Japan romances are a popular video-game genre among both sexes. Since we’re interested in appealing to a female market, we considered it wise to examine their approach. A point we thought worth considering about was that these games are often aimed at a young adult market. Within our group we were split as to whether it would be better to aim for an all-ages game, or for a young adult game. We decided to arrange things so we could go in either direction, or we could even potentially make separate games using much of the same code. The art-work in these games also seemed to be very important, and we opted to draw our own art instead of using preexisting assets to insure that the art and code were properly integrated and we weren’t making odd compromises so that one would fit the other.

A close up of a womans face

Description automatically generated

Figure 3-A cutscene from the romance game Reines des Fleurs. Artwork is a central aspect of gameplay experience. Image Source: https://reines-des-fleurs.fandom.com/wiki/CG\_Gallery/Image\_7

With all this decided we settled on a dress up game that takes place at a magical school. The working name we gave it was *Magical School Dress Up*.

# Approach

We broke project down into three parts: the narrative section, the dress-up minigame, and the mobile or UI section. Unfortunately, the mobile section was largely dropped and was replaced with UI on a computer, but as planning for it adjusted many of our choices it may be referred to throughout this report.

We predominantly used Microsoft Visual Studio and programmed in C#. We spent somewhere between half a month and a full month exploring the possibility of switching to Unity, which is a programming environment made for game making, but we decided the learning curve was too high for our schedule, and we were better off continuing in Visual Studio.

# Artwork and Programming Logic Behind Dressing Hero

Figure 4- Hero Drawn in Photoshop using Layers for clothing

The character model for the hero was drawn in Photoshop, clothing models were drawn in Photoshop in layers and then saved as individual files such as: “hair\_wavy\_purple.png”. All artwork was drawn by Anna Carlson.

Hypothetically there was the option to include two Heroes in the game. At the time of the writing of this however, it looks like the female protagonist will be cut from the presentation copy. The program was written, the art was designed, and the story was modified in such a way however that the two possible heroes could share the same wardrobe.

Within the game the classes Hero and Clothing interact quite a bit. The Hero class contained an attribute Wardrobe, and Outfit. Outfit stored only the clothing pieces that the hero was wearing, while wardrobe contained all the pieces he or she had collected for use in later dress-up events.

Enums were used in the clothing section to store and retrieve clothing from an array without wasting time running through an array looking for a string or the programmer memorizing an ID number. There was one Enum which held items like SHIRT, PANTS, HAIR, etc., to represent each type of clothing used for the layering logic in dressing the character, and then separate Enums for each type of clothing. The size of the arrays were calculated by the length of the array. For example, all the shirts in the game are declared in a Clothing array function called initilizeShirts(), which is received by Clothing.Shirts. A simplified version of which is the following:

Clothing[] shirts = new Clothing[(int)Enum.GetNames(typeof(SHIRTSENUM)).Length];

shirts[(int)SHIRTSENUM.WEAVY\_BLUE] = new Clothing("Wavy Blue Shirt", Properties.Resources.shirt\_wavy\_blue);

return shirts;

Within the Hero class there was a method called dressHero(), which would update the hero’s bitmap image by means of using the Graphics class to modify a copy of the base body bitmap, and return a new bitmap after each item in the hero’s outfit was called by using the TypesOfClothing Enum, which make it easier for the programmer to verify they were loading in the proper order. The function dressHero() is called whenever the Hero image needs to be updated.

# Classes

These are ordered alphabetically.

## Choice

Used when selecting what story to switch to at the end of one story.

### Functions

* Choice(string buttonText, Story thisStory) [Choice.cs, line 11]:
  + Creates a Choice object.
  + Establishing the text that will appear on a button created from this
  + Sets the story it will switch the user to.

## Clothing

### Enums

We’ve been using reference to the enums in the clothing class in order to implicitly back each clothing item with a two dimensional array: Clothing [][] Wardrobe.

A screenshot of a cell phone

Description automatically generatedThe first location is backed by the enum TYPESOFCLOTHING, which holds things like SHIRT, PANTS, HAIR, etc. The second array varies. There is a separate area for each individual clothing item. So for example, there is an array called SHIRTENUM that holds things such as WEAVY\_BLUE, MAGIC\_BLACK, and TSHIRT\_RED.

Figure - Clothing Enums

### Important Variables

* string Name
* Bitmap Image
  + In order to change or alter the Picturebox used in the form, it is most useful to store the potentially new information in a Bitmap type
* int Score
  + Used for dress-up minigame calcuation
* int Saved\_ClothingTypeEnumNumber, and int Saved\_ClothingUniqueItemEnumNumer
  + Used to store wardrobe reference numbers inside each clothing item itself.
  + Useful for situations where the program wants to display or reference the outfit the Hero is wearing

### Functions

* Clothing(string name, Bitmap image, int score = 1)
  + Sets the name, an image, and a base score used for minigames that use clothing to calculate the score
* A screenshot of a computer screen

  Description automatically generatedSets up several static Clothing arrays based on different clothing parts that return the second part of the 2D wardrobe array.

Figure - Calling Clothing Arrays backed by Enums

* The Clothing[][]Wardrobe variable calls a varaity of clothing arrays based on the enum used in it.

## A screenshot of a computer screen Description automatically generated

Figure -Wardrobe variable. Constantly cross referenced using the clothing arrays established earlier

## A screen shot of a monitor Description automatically generated

Figure - Example of a clothing initialization function. initializeFaces()

## Game Data

### Functions

* InportStoryfile() [GameData.cs, line 31]: Takes a text file and puts it in the narrative section of the game interface
* initilizeGlamorHobosAdventure() [GameData.cs, line 52]:
* initilizeStory() [GameData.cs, line 79]: Initializes the story with the correct files

## Hero

### Functions

* Hero() [Hero.cs, line 21]: Creates characters in story with all of their attributes
* starterOutfit() [Hero.cs, line 82]: gives the user a default outfit
* addToWardrobe() [Hero.cs, line 151]: Adds new clothes into the user’s usable wardrobe
* dressHero() [Hero.cs, line 172]: Puts clothes on hero
* updateDrawImage() [Hero.cs, line 216]: Draws hero with current outfit
* changeShirt() [Hero.cs, line 228]: Updates shirt image on hero
* changeClothing() [Hero.cs, line 237]: Updates clothing image on hero

## Item

### Functions

* Item() [Item.cs, line 13]: Makes a new item

## Minigame

### Functions

* Minigame() [Minigame.cs, line 19]: Initializes a new minigame instance
* start() [Minigame.cs, line 27] [Story.cs, line 209]: Begins minigame; returns true if won, false if lost; begins story

## Occurrence

### Functions

* Occurance() [Occurance.cs, line 10]: Initializes an occurance with a name, description, and image

## Program

### Functions

* Main() [Program.cs, line 15]: Begins game

## Story

### Function

* Story() [Story.cs, line 17]: Initializes the story with story text, choices on or off, and images
* updateImageArraySizes() [Story.cs, line 30]: Makes arrays to hold different aspects of the game
* addMiddleCharacterImage() [Story.cs, line 52]: Makes character image visible
* addBackgroundImage() [Story.cs, line 67, 75]: Inserts background image
* addForegroundImage() [Story.cs, line 81, 90]: Inserts foreground image
* imageArrayVisibleLoop() [Story.cs, line 95]: Adjusts position of image on background
* addOccurance() [Story.cs, 126]: Creates occurrence in story
* next() [Story.cs, line 223, Form1.cs, line 81]: Progresses through story text, checks for minigames or occurrences, checks for image change
* last() [Story.cs, line 227]: Ends story
* findItem() [Story.cs, line 303]: Makes occurrence where item is found
* branchStory() [Story.cs, line 318]: Branches story
* addMinigame() [Story.cs, line 329, StartPage.cs, line 101]: Starts new minigame

# Forms

Since there are only a few of these, they are ordered in encounter order. Each image was checked against a grey scale filter to double check for readability for the color-blind.

## StartPageA screen shot of a computer Description automatically generated

Figure 9- StartPage form in greyscale and normal color

A list of stories and a hero are declared on this page.

### Functions

* StartPage() [StartPage.cs, line 22]: Initializes game components
  + When I run this for the first time I declare a large number of variables that are used elsewhere in the program.
  + I make use of the story class to declare the timing of background changes and character appearances that can occur throughout the presentation later on in form1. I use a static method initilizeStory(string storyFile,Hero hero, Boolean hasBranchingChoices = false) from the GameData class to make sure some dependent data down the line is up to date when I fist declare the story.
  + I then declare where in the narrative timeline items are found, minigames occur, and cloths are found. For the sake of this demo however, the timing of unlocking the clothing was cut and all clothing items are available at the beginning.
  + I then repeat that process with any story branches.
  + After that I put in the Choices for the branching story.
  + At the end I add each of the branching story paths to the varible List<Story>magicSchoolStories I declared at the beginning. This makes sure I have access to these details later in the game.

#### Button Functions

* newGameBtn\_Click()
  + Creates a new Form1 based on the variable List<Story> magicSchoolStories information and opens it in a new window.
* exitBtn\_Click()
  + Closes Application

## Form1

A screenshot of a video game

Description automatically generated

Figure 10- Form1 in greyscale and normal

### Functions

* Form1(List<Story> inportStory, int position=0) [Form1.cs, line 23, 30]: initializes Form
  + (Alternatively) Form1(Story inportStory, int position=0)
  + Used to import stories that were set up in the start up page
* Form1\_Load() [Form1.cs, line 41]: Loads images into the form
  + Make sure that the foreground image is turned off
  + Set up the images to display the character on top of the background.
  + Run checkForImageChange() function to make sure all images and related updates have been applied.
  + Output start fist line from text file into the output label.
* checkForImageChange() [Form1.cs, line 113]: Positions or changes image along with the story
  + Checks for image changes
    - Foreground, background, and characterImage
  + Checks for occurrences
  + Checks if this Multiple choices menu needs to appear
  + Checks if button color needs to be changed and does so if conditions are met
* occuranceSwitch() [Form1.cs, line 176]:
  + Makes popup widow when occurances occur
  + Turns window invisible when not in use
* showMultipleChoiceRoutes() [Form1.cs, line 186]:
  + Makes popup window for story branching
  + Creates up to three Buttons that move to another section of story
* colorButtonGrey(Button button)
  + Used when text color needs to be greyed out.
* colorButtonBlack(Button button)
  + Used when text color needs to be turned to black
* checkButtonColor()
  + Discovers if there is more store before or behind the current position and changes button color between black and grey accordingly.
* selectNextStory(int arrayPosition) [Form1.cs, line 238]:
  + Selects next Story to run
  + Resets position to 0
  + Sets outputLbl text to first line of new story
  + Turns button selection box to invisable
* playLoad()
  + There is a loading screen included when the minigame is running, this starts it.
* next()
  + used to advance the text
  + check for image changes
  + check for events
* prevButton\_Click()
  + Works as the reverse of the next function
  + Unlike the next() function, which can be called multiple ways, this one can only be called by clicking the previous button
* loadingTimer\_Tick()
  + Used to assist the loading screen

## DressUpContest

A screenshot of a social media post

Description automatically generated

Figure 11-DressUpContest in greyscale and normal color

### Important Variables

* Minigame minigame;
  + As the Minigame class was written to enable a variety of potential minigames to occur, this variable provides this form with winning conditions and relevant information that needs to be passed into this form from elsewhere.
* Hero hero;
  + Most player information is stored in the Hero class. So things such as working wardrobe the player possesses needs to be accessed to populate the selection lists.
* Int Score
  + The score achieved in this game will need to be compared with the winning conditions.

### Functions

* DressUpContest(Minigame minigame) [DressUpContest.cs, line 18]: Initializes dress up games
  + Initializes the minigame and hero variables
  + Fill out each clothing selection box using the fillClothingSelectionBox() function.
  + Select the starting index for each clothing item.
    - I was in the process of developing a function to set this based on the current outfit of the hero. Due to the project deadline however, it was decided to hardcode most of it.
* getHerosOutfitItem(TYPESOFCLOTHING typeOfClothing)
  + Set to work with the Wardrobe part of the Hero class and is used to return an integer used to set the starting index position for each clothing item selection box based on what the hero is wearing.
* fillClothingSectionBox(ListBox box,Clothing[] arrayToFill) [DressUpContest.cs, line 42]: Shows clothing user has in wardrobe in listboxes
  + Adds items to clothing selection box by checking to see if the hero has them in his wardrobe
  + Adds a “nothing” option to all selection boxes except for “face”, which is called make-up from the user’s perspective. Because missing a face, rather then making the player look undressed, makes them look inhuman.
* changeIndividualClothingItems(ListBox listBox, TYPESOFCLOTHING typeOfClothing) [DressUpContest.cs, line 90]: changes an individual clothing item on the character
  + Used in a number of selection events to update what the hero is wearing in the image, and update the score appropriately.
* button1\_Click()
  + Calculates the total score and tests to see if hero has met the requirements to win the minigame.
  + Closes out the form