Daydream

Design Document

Sarah Koh, Matt Fredrickson, JC Orolfo

11/17/2014

Advisors
Eelke Folmer

University of Nevada, Reno

Table of Contents

1 Abstract

Daydream is a fully immersive, interactive video game experience like nothing else. Since the advent of commercial virtual reality (VR), the issue of what to use for input has been up in the air for developers. The traditional input device used is a hand-held controller, but this takes away from the primary intention of virtual reality; immersion. Daydream aims to transform interaction with a VR system by introducing a brain-computer interface as the main form of input. Players will be taken on a thought-driven journey to take photographs of fantasy creatures using only their mind as a camera. Deep, focused concentration on the creature will directly translate to beautiful pictures and higher scores. With the power of the brain in our hands, Daydream hopes to stimulate people to delve deeper into their psyche, allow for abundant game accessibility, and open up new horizons for interaction.

2 Introduction

The inner workings of the brain is largely still a mystery to humans. It's only in recent years that computer science began to explore the idea of a brain-computer interface (BCI). Many of these non-invasive BCIs use EEG technology to measure a brain's Alpha and Beta waves to determine more general states of mind such as "relaxed" or "focused". How exactly developers should be utilizing this technology is still not concrete. There are obvious areas of applications, such as for prosthetics, but the majority of people cannot benefit from BCI technology. Daydream aims to find commonplace applicability of BCIs through a virtual reality medium.

Virtual reality itself is a fairly new innovation that still faces issues with effectiveness. One of the biggest concerns with virtual reality is how to take in input from the user. Typically, the user must either hold a controller or another foreign object to control their actions. These external inputs detract from the intention of virtual reality; immersing oneself into a completely different world. A brain-computer interface in combination with a virtual reality environment would allow for a hands-free, completely immersive experience. Daydream's main goal is to demonstrate the practical nature of this form of interaction and the potential for it to be a viable form of input for virtual reality. With the only input being the users own thoughts and emotions the experience is taken to an entirely new horizon of interaction where nothing exists but the person's mind and their fantasy world.

The concepts used in this project come with the challenge of being on the bleeding edge of technology. BCI in particular is a brand new form of interaction, bringing a unique dynamic to the game. Regardless, the innovative notions of BCI and virtual reality hold great promise for the future. These forms of interaction can accommodate a large number of users from as game enthusiasts, to people who practice meditation, to people with impaired fine motor skills, and those with ADHD and anxiety disorders. Because of the breadth of potential users, the options for marketing the product are flexible and adaptable.

Since the specification document, the group has spent time becoming familiar with Unity and working through tutorials. An overall structure for Daydream has also been created. We have also purchased the Mindwave and a few Google Cardboards. Preliminary testing of the Mindwave capabilities show promising results in terms of responsiveness and ease of use. We hope to successfully integrate the hardware components within the week and begin on detailed development shortly afterwards. No significant changes have been made to our project's requirements.

This design document covers the high-level and mid-level design, detailed design, hardware design, user interface design, annotated references, and contributions of the team members. The high-level and mid-level design covers a system-level diagram and a class diagram with class method descriptions. The detailed design contains three dynamic diagrams which detail components of Daydream's behavior. The hardware design section contains a high-level diagram of the hardware components, descriptions of their roles, and snapshots of the components. The user interface design section contains snapshots of the user interface components. A list of annotated references are included that relate to our project. Finally, the contributions section outlines the work done by each team member.

3 High-Level and Medium-Level Design

3.1 System-Level Diagram

Below is a view of the system level architecture. The architecture follows a typical game engine. Each manager contains a pointer to the overlaying engine. This allows each manager to joint functionality whilst maintaining modular design.

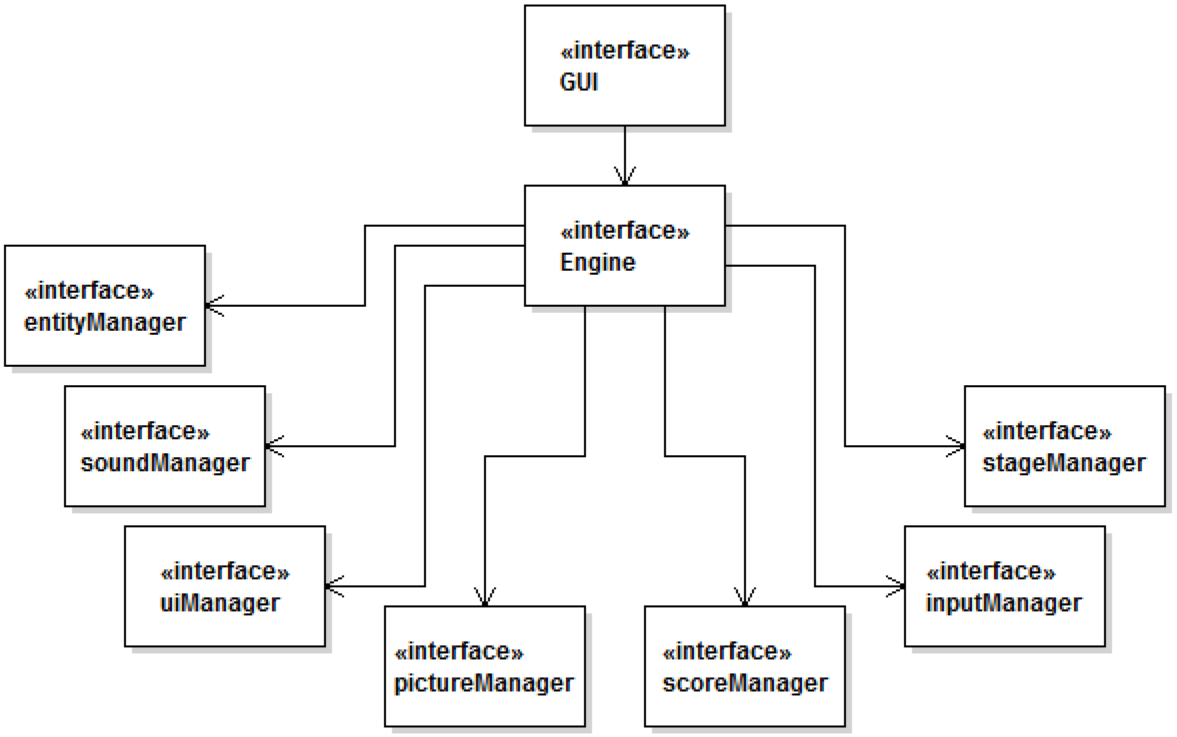


Figure 1: *A view of the system level architecture that makes use of a standard game engine.*

3.2 Class Diagram

The class diagram for Daydream shows the overall structure of the program units. Note that every program manager class refers back to the Engine class. The Engine class is the central point of reference for Daydream. Refer to Figure 2 below.

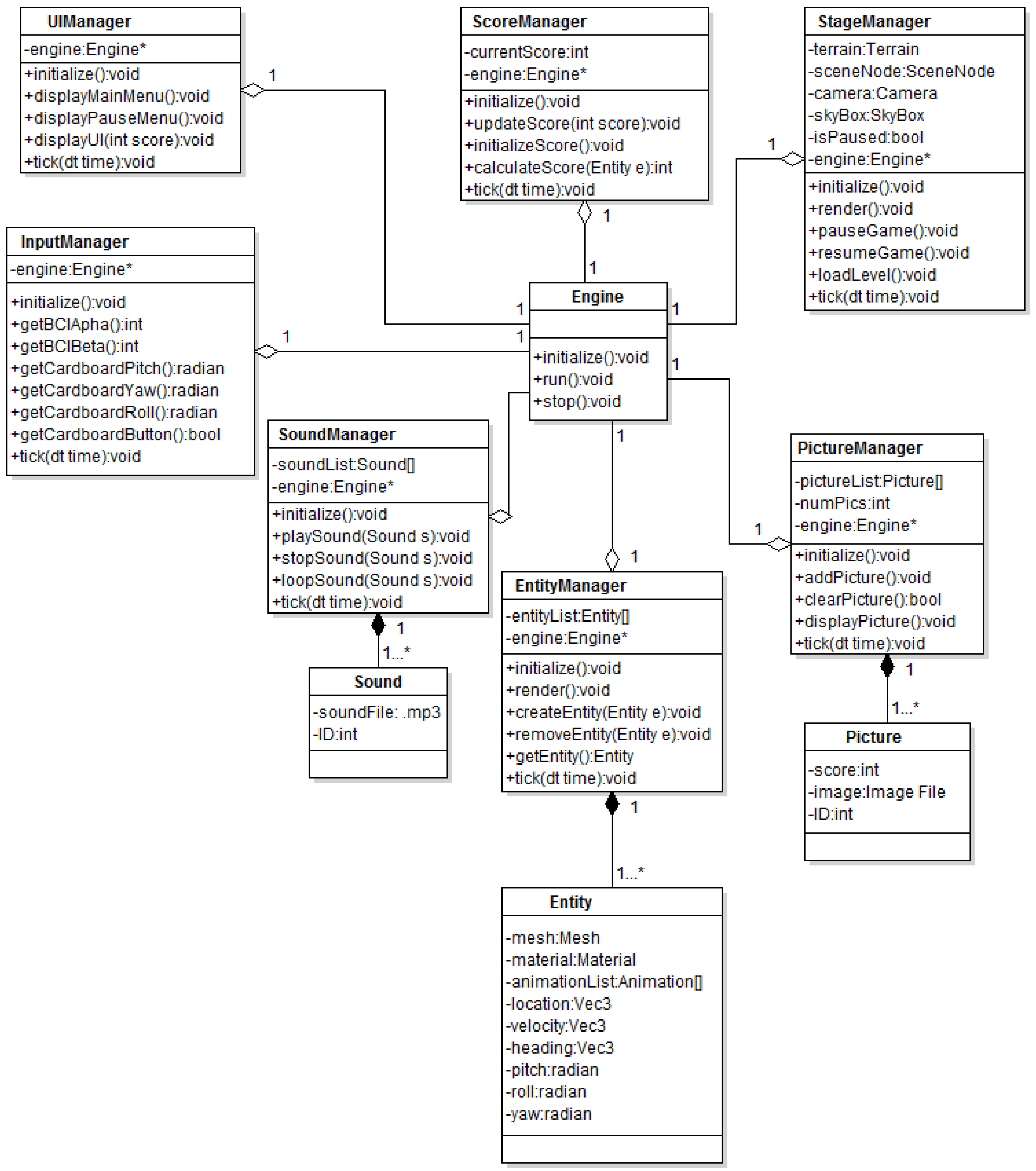


Figure 2: *Daydream Class Diagram*

3.3 Class Method Descriptions

3.3.1 Engine Class

The Engine class is the central point of reference for all of the managers. All manager classes described below contain a reference to the Engine. It acts as the mediator to inform manager classes of events, start the game, and stop the game.

Class	Engine
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method initializes the engine and establishes pointers to the manager classes.

Class	Engine
Method	run
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method runs the game loop, incrementing each manager’s tick function.

Class	Engine
Method	stop
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method halts the game engine and begins shutting the system down.

3.3.2 UIManager Class

The UIManager class controls which menu is displayed, the user interface in-game, displaying the score, and initializing the menu system.

Class	UIManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method initializes the user interface manager and the engine pointer.

Class	UIManager
Method	displayMainMenu
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method displays the main game menu to the user.

Class	UIManager
Method	displayPauseMenu
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method displays the pause menu to the user.

Class	UIManager
Method	displayUI
Visibility	public
Return Type	void
Parameters, Types	score:int
Description	This method displays the in-game user interface, including current score, to the user.

Class	UIManager
Method	tick
Visibility	public
Return Type	void
Parameters, Types	time:dt
Description	This method maintains consistent time with the rest of the game with the provided delta time (dt) value.

3.3.3 InputManager Class

The InputManager class interacts with the hardware components to get information on brainwave readings from the BCI (Mindwave), heading information from the Cardboard, and button press events from the Cardboard.

Class	InputManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	Initializes the engine pointer and input manager so the user may use input.

Class	InputManager
Method	getBCIAlpha
Visibility	public
Return Type	int
Parameters, Types	none
Description	Reads the level of Alpha waves from the BCI device.

Class	InputManager
Method	getBCIBeta
Visibility	public
Return Type	int
Parameters, Types	none
Description	Reads the level of Beta waves from the BCI device.

Class	InputManager
Method	getCardboardPitch
Visibility	public
Return Type	radian
Parameters, Types	none
Description	Gets the pitch aspect of the Google Cardboard device.

Class	InputManager
Method	getCardboardYaw
Visibility	public
Return Type	radian
Parameters, Types	none
Description	Gets the yaw aspect of the Google Cardboard device.

Class	InputManager
Method	getCardboardRoll
Visibility	public
Return Type	radian
Parameters, Types	none
Description	Gets the roll aspect of the Google Cardboard device.

Class	InputManager
Method	getCardboardButton
Visibility	public
Return Type	radian
Parameters, Types	none
Description	Detects the Google Cardboard slider button being actuated.

Class	InputManager
Method	tick
Visibility	public
Return Type	void
Parameters, Types	time:dt
Description	This method maintains consistent time with the rest of the game with the provided delta time (dt) value.

3.3.4 ScoreManager Class

The ScoreManager class maintains the status of the current score as well as updating the score, initializing the score, and calculating the score using the picture scoring algorithm.

Class	ScoreManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method initializes the current score and the engine before the game runs.

Class	ScoreManager
Method	updateScore
Visibility	public
Return Type	void
Parameters, Types	score:int
Description	This method updates the score for the player after a picture event occurs. The score parameter used is calculated in the calculateScore method.

Class	ScoreManager
Method	initializeScore
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method sets the score to an initial value after the game begins.

Class	ScoreManager
Method	calculateScore
Visibility	public
Return Type	int
Parameters, Types	e:Entity
Description	This method calculates a score for a picture based on an algorithm that considers entity boundaries.

Class	ScoreManager
Method	tick
Visibility	public
Return Type	void
Parameters, Types	time:dt
Description	This method maintains consistent time with the rest of the game with the provided delta time (dt) value.

3.3.5 StageManager Class

The StageManager class manages all aspects of the scenery including the terrain, the camera view, and the sky box. It is responsible for rendering and updating the scene, loading levels, pausing the game, and resuming the game.

Class	StageManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method initializes the scene and environment which consists of native Unity types for the terrain, the scene node, the camera, and the sky box.

Class	StageManager
Method	render
Visibility	protected
Return Type	void
Parameters, Types	none
Description	This method renders the stage and all stage components and updates as the game progresses.

Class	StageManager
Method	pauseGame
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method sets the isPaused attribute to true when the user looks far down or far up. This event will halt the game and trigger a pause.

Class	StageManager
Method	resumeGame
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method sets the isPaused attribute to false which will allow the game to resume rendering.

Class	StageManager
Method	loadLevel
Visibility	public
Return Type	void
Parameters, Types	none
Description	This method loads in the selected level and the scenery components attached to that level.

Class	StageManager
Method	tick
Visibility	public
Return Type	void
Parameters, Types	time:dt
Description	This method maintains consistent time with the rest of the game with the provided delta time (dt) value.

3.3.6 SoundManager Class

The SoundManager class maintains a list of sounds to be used. It contains methods that control which sounds are played, which sounds are stopped, and functionality to loop sounds.

Class	SoundManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	Initializes the sound manager.

Class	SoundManager
Method	playSound
Visibility	public
Return Type	void
Parameters, Types	s:Sound
Description	Plays a sound file, will automatically stop playing sound when file is done.

Class	SoundManager
Method	stopSound
Visibility	public
Return Type	void
Parameters, Types	s:Sound
Description	Stops sound file playback.

Class	SoundManager
Method	loopSound
Visibility	public
Return Type	void
Parameters, Types	s:Sond
Description	Plays a sound file, will restart sound file when the file reaches the end of its duration.

Class	SoundManager
Method	Tick
Visibility	Public
Return Type	Void
Parameters, Types	time:dt, soundList:Sound[]
Description	Runs through the list of sounds and plays and stops sounds as needed

3.3.7 Sound Class

The Sound class packages information about the sound file and assigns an ID for each sound created. A list of Sounds is maintained and managed in the SoundManager.

No methods

3.3.8 EntityManager Class

The EntityManager class maintains a list of entities to be rendered and placed. The functionality of the class extends to rendering entities, creating entities to be placed in the list, removing irrelevant entities, and retrieving entities.

Class	EntityManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	none
Description	Initializes the entity manager.

Class	EntityManager
Method	render
Visibility	public
Return Type	void
Parameters, Types	none
Description	Renders a specific entity.

Class	EntityManager
Method	createEntity
Visibility	public
Return Type	void
Parameters, Types	e:Entity
Description	Creates an entity and appends it to the list of entities.

Class	EntityManager
Method	removeEntity
Visibility	public
Return Type	void
Parameters, Types	e:EntityList, id:int
Description	Searches list for entity whose id matches the given id, then removes that entity from the list.

Class	EntityManager
Method	getEntity
Visibility	public
Return Type	Entity
Parameters, Types	e:EntityList, id:int
Description	Looks through list of entity and returns the entity whose id matches the given id.

Class	EntityManager
Method	tick
Visibility	Public
Return Type	Entity
Parameters, Types	time:dt
Description	Looks through the list of entities and updates each entity based on the change in time.

3.3.9 Entity Class

The Entity class encapsulates concrete information about an entity. This includes the mesh, the material, any animations it may perform, placement, velocity, and heading.

No methods

3.3.10 PictureManager Class

The PictureManager class stores a list of pictures that the player takes and the number of pictures. It is responsible for adding pictures to the picture list, clearing pictures, and displaying pictures when selected by the player.

Class	PictureManager
Method	initialize
Visibility	public
Return Type	void
Parameters, Types	-
Description	Initializes the picture manager.

Class	PictureManager
Method	AddPicture
Visibility	public
Return Type	void
Parameters, Types	-
Description	Takes the current view and adds it as a picture to the current list. Uses ScoreManager functions to generate score to associate with the picture.

Class	PictureManager
Method	clearPicture
Visibility	public
Return Type	void
Parameters, Types	-
Description	Clears out all pictures in the picture manager.

Class	PictureManager
Method	displayPicture
Visibility	public
Return Type	void
Parameters, Types	id:int
Description	Displays the picture whose value id matches the given id.

3.3.11 Picture Class

The Picture class contains all information about a picture. This includes the score it received, the actual image file, and an ID. This is used by the PictureManager to store information about a picture.

No methods

4 Detailed Design

4.1 *Activity Diagrams*

4.1.1 Main Menu

Figure 3 depicts the activity diagram for the main menu. This includes selecting between Start Game, Start Tutorial, View Photos, and Exit. The diagram details what is displayed through the choices made with the end state being that the game is exited.

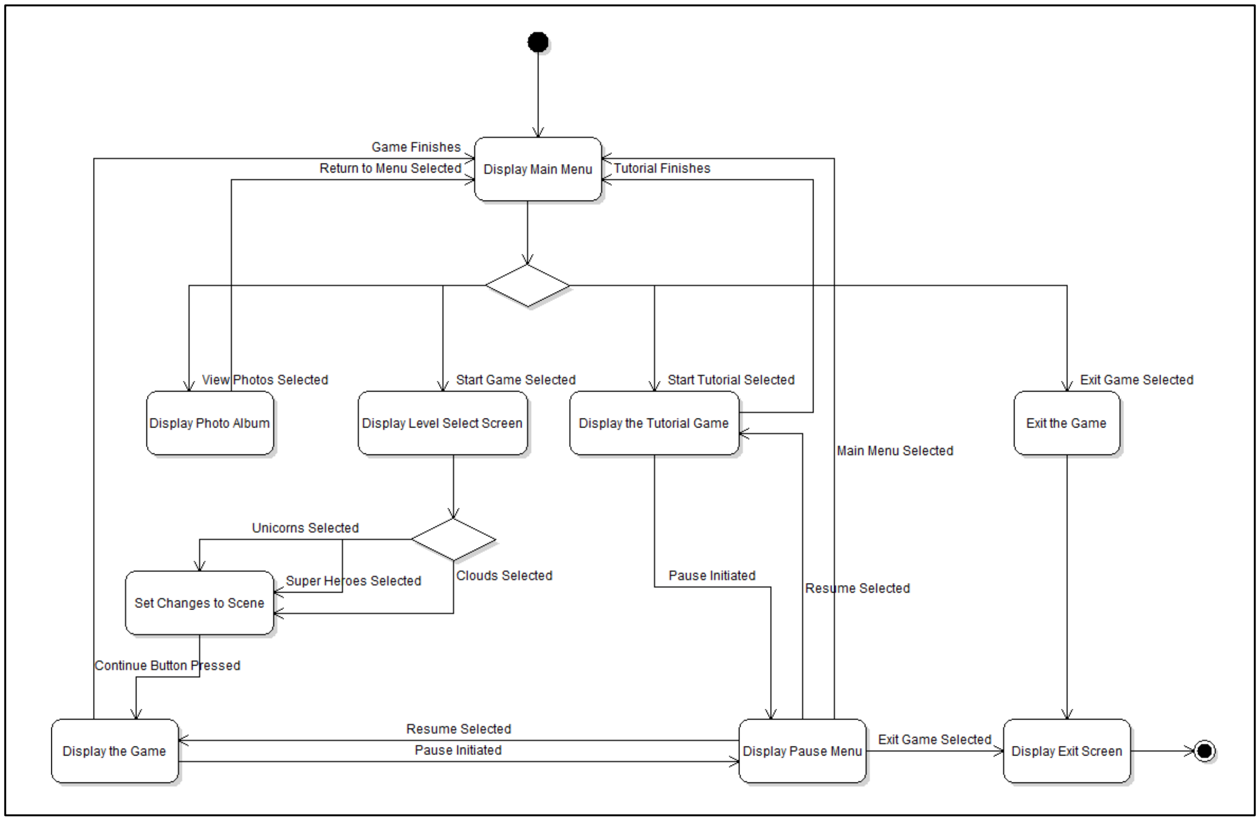


Figure 3: *Activity Diagram for the Main Menu*

4.1.2 Game Loop

Figure 4 below contains the activity diagram for the main game loop. This is a simple function that is the very core of our game: it initializes, then runs the game until the user pauses or quits the game.

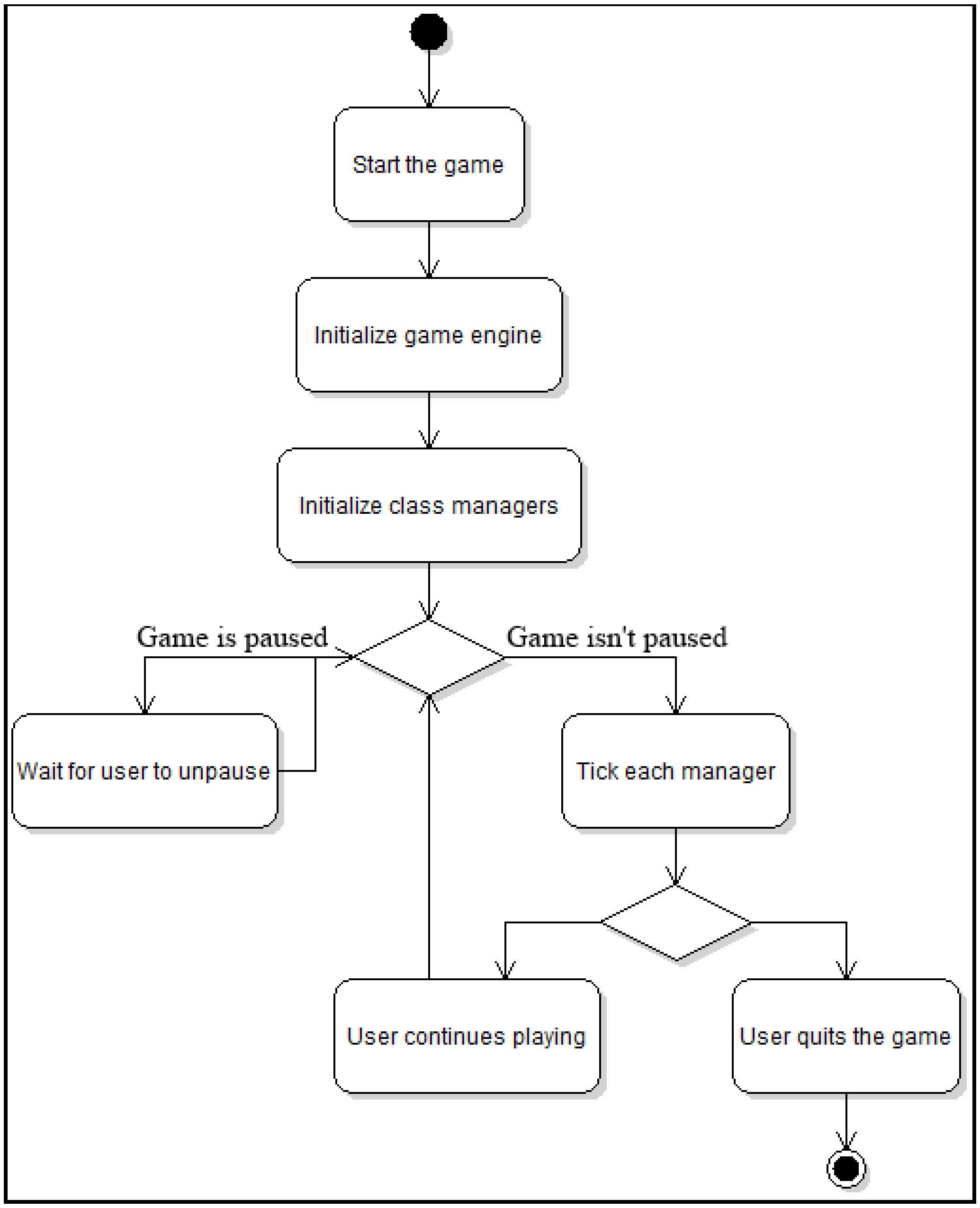


Figure 4: *Activity Diagram for the main game loop*

4.1.3 Taking a Picture

Figure 5, below, shows the activity diagram for taking a picture. When the picture button is pressed, and the timer is zero, the game will check to make sure there is an entity in view. If there is, the game will proceed to save the image and calculate a score through the score manager.

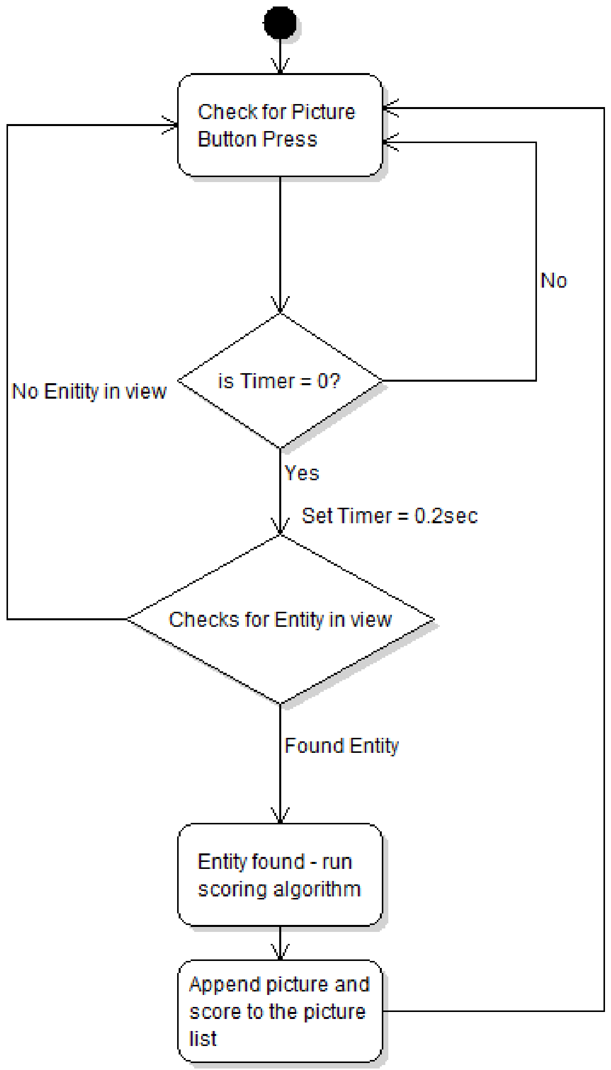

Check for Picture Button Press
No
No Enitity in view
is Timer = 0?
Yes
Set Timer = 0.2sec
Checks for Entity in view
Found Entity
Entity found - run scoring algorithm
Append picture and score to the picture list

Figure 5: *An activity diagram for taking a picture.*

5 Initial Hardware Design

5.1 Hardware Component Diagram

Our initial hardware design is extremely simple: it consists of an Android device as our main system, with the MindWave and Google Cardboard integrated into it for input and output purposes. Eventually we hope to merge all three components into a single elegant system.

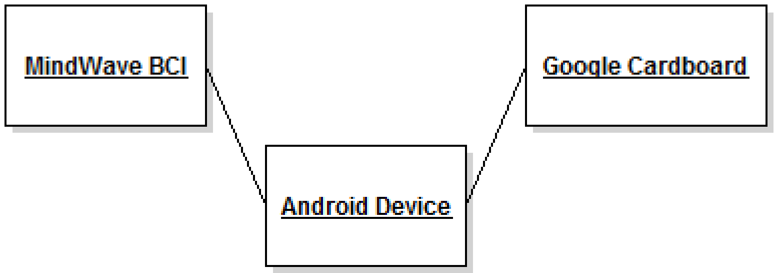


Figure 6: *Initial hardware design*

5.2 Hardware Component Listing

Android Device	Our central system, where the game will be run and all computations take place.
Google Cardboard (Figure 7)	3D-enabling device built for Android smartphones. Used for visual output and single-button input.
MindWave Mobile (Figure 8)	Brain-Computer Interface. Allows the user to interact with the software with their brain waves.

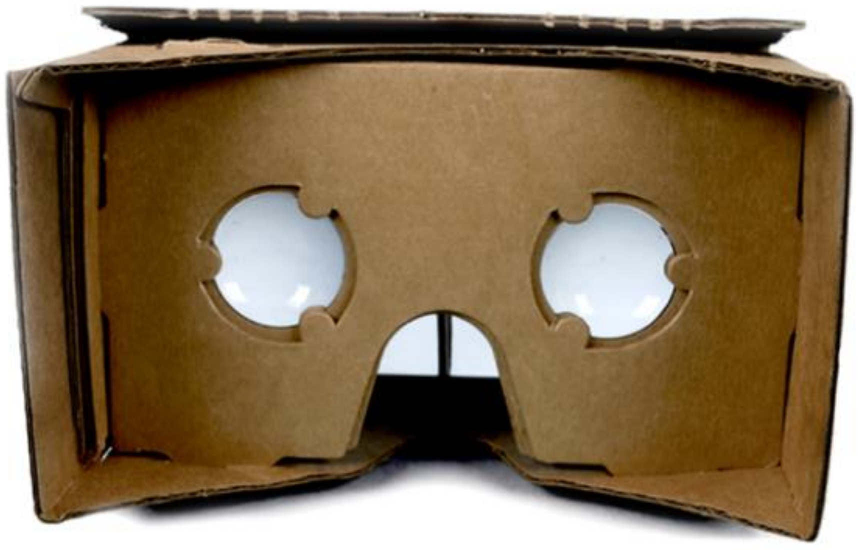


Figure 7: *Google Cardboard (image from http://cardboard.withgoogle.com)*

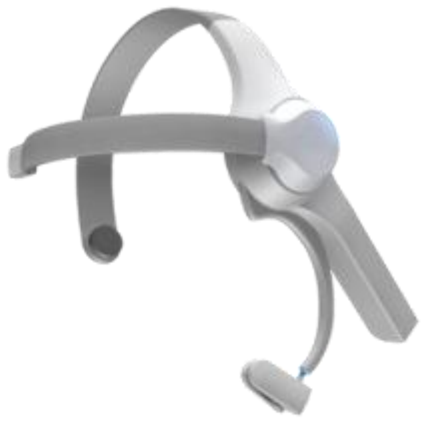


Figure 8: *MindWave Mobile (image from http://www.neurosky.com)*

6 User Interface Design

Title Screen

The figure below shows a mockup of the title screen. The aesthetic design of the title screen is a work in progress; however, the layout will be very similar to the final version.

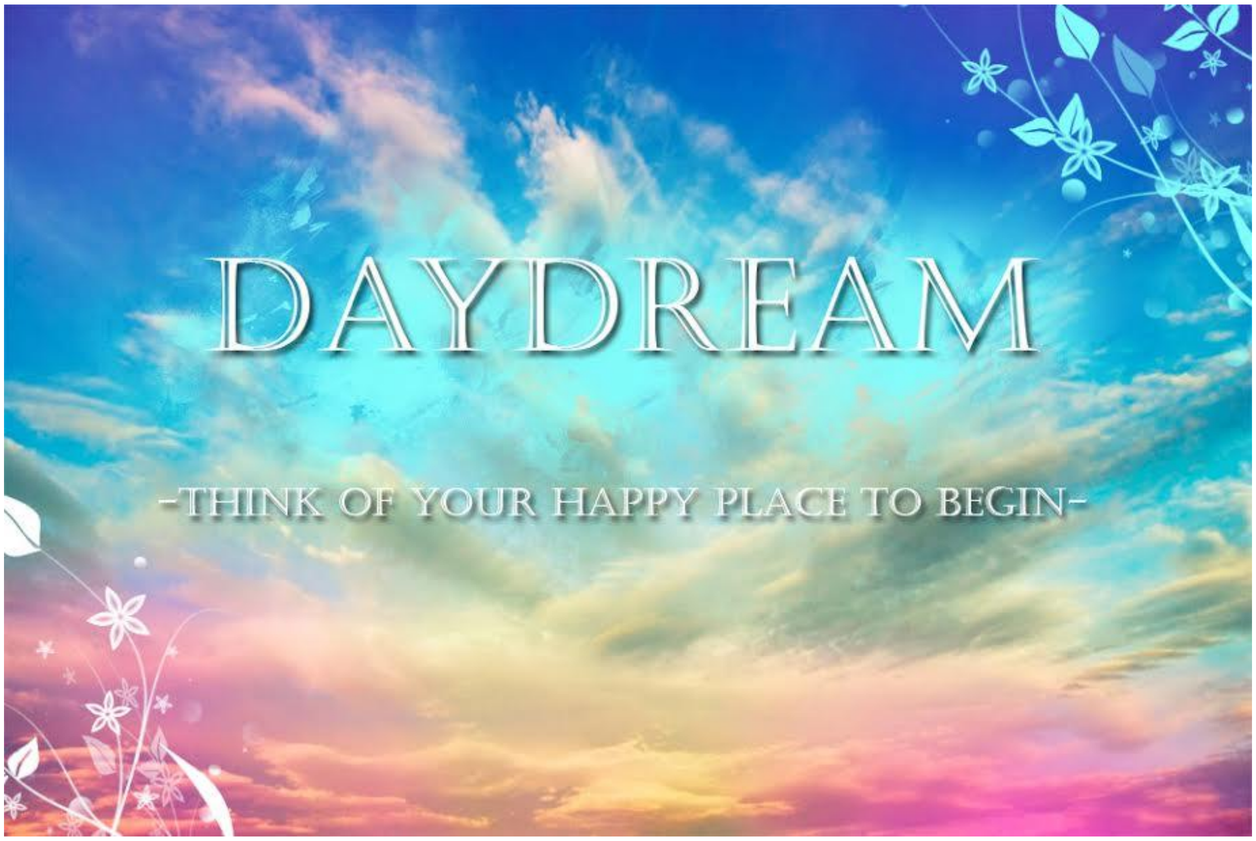


Figure x: *Display of mockup title screen. Title screen will hold for a couple seconds, then will start to take in input. Game starts when a certain level of brain activity is read. Original image used in title screen taken from:*
<http://wallalay.com/clouds-30-382768-desktop-background.html>

Main Menu

The figure below shows a mockup of the main menu screen. The aesthetic design of the main menu screen is a work in progress; however, the layout will be very similar to the final version.

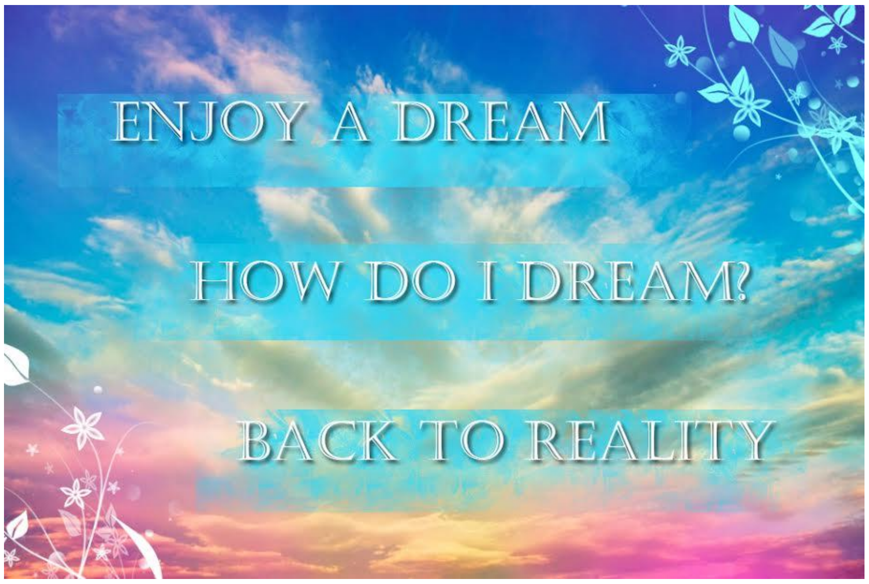


Figure x: *Display of main menu. User can choose to select a stage, run the tutorial, or to exit the application. Original image used in title screen taken from: <http://wallalay.com/clouds-30-382768-desktop-background.html>*

Level Select

If implemented, the figure below shows a mockup of the level select screen. The aesthetic design of the level select screen is a work in progress; however, the layout will be very similar to the final version.

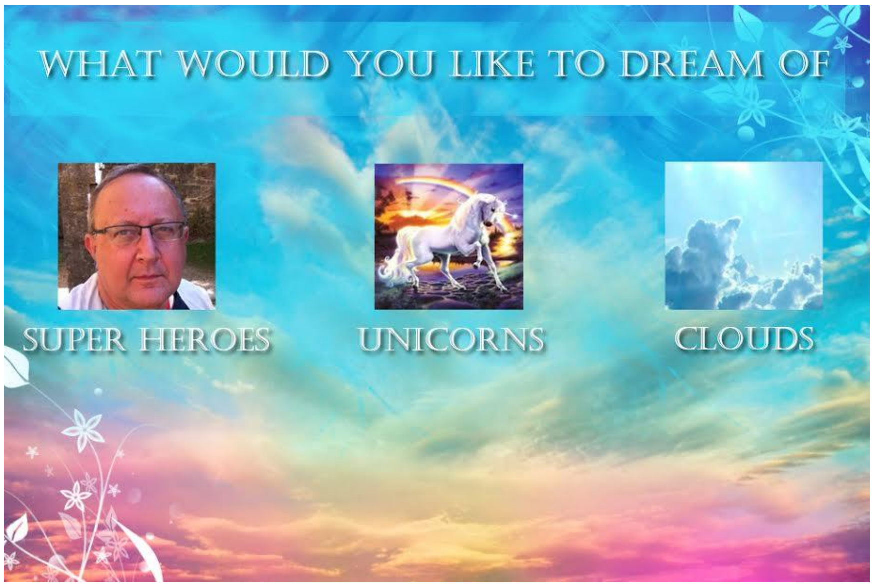


Figure x: *Display of level select if implemented. User selects a stage to proceed to. Original image used in title screen taken from: <http://wallalay.com/clouds-30-382768-desktop-background.html>, <http://simonemccallum.files.wordpress.com/2012/06/unicorn_and_rainbow.jpg>, <http://www.wired.com/wp-content/uploads/blogs/insights/wp-content/uploads/2012/11/clouds_660.jpg>, <http://www.cse.unr.edu/~dascalus/>*

Example of a good picture

The figure below shows a well zoomed picture. A circular reticle in the middle of the screen gives the user the feel of looking through the lens of a camera.

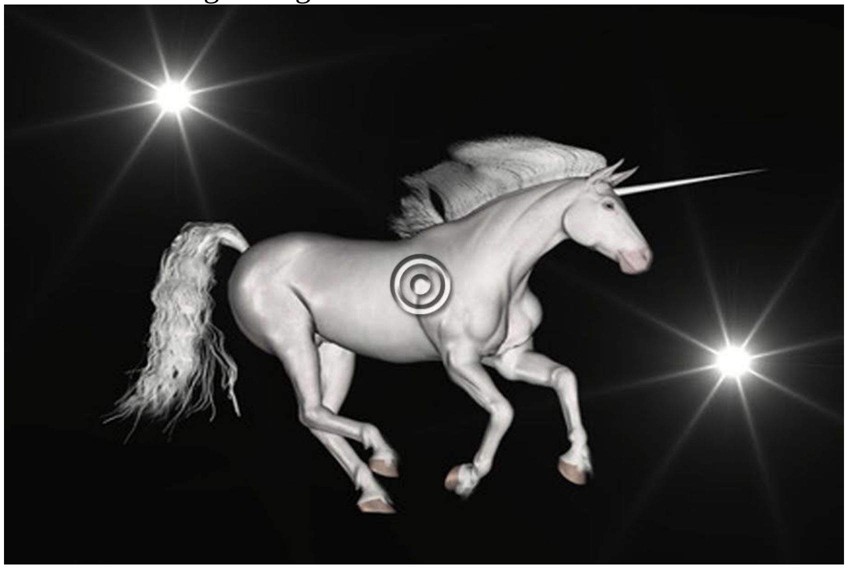


Figure x: *Display of a well zoomed entity to take a picture. Entity is centered well is in good view in for the user to take a picture. Such a picture would net the most points. Original image taken from: <http://abstract.desktopnexus.com/wallpaper/358514/>*

Example of a picture zoomed in too far

The figure below shows picture zoomed in too far. A circular reticle in the middle of the screen gives the user the feel of looking through the lens of a camera.

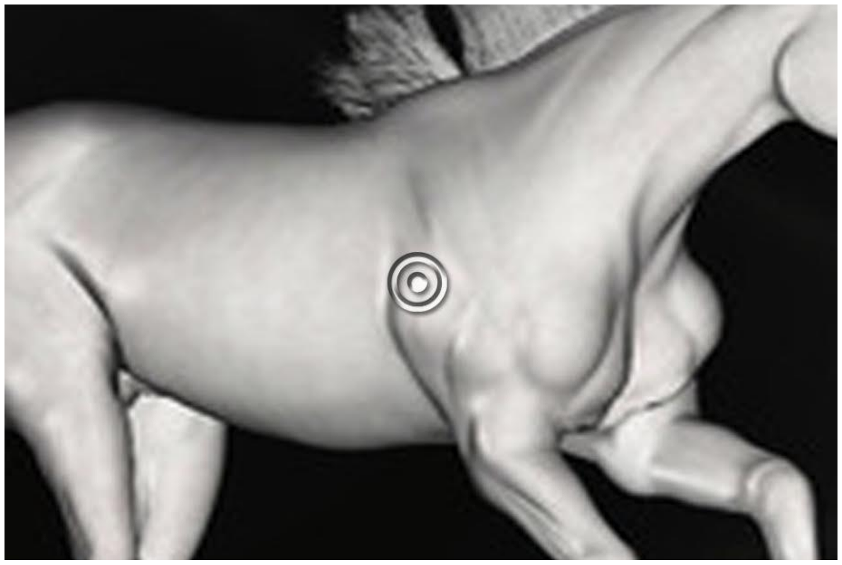


Figure x: *Example of an entity zoomed in too far. This is what happens when the user focuses too much on an entity that is relatively closer. Such a picture would not score many points. Original image taken from: <http://abstract.desktopnexus.com/wallpaper/358514/>*

Example of a picture zoomed out too far

The figure below shows picture zoomed out too far. A circular reticle in the middle of the screen gives the user the feel of looking through the lens of a camera.

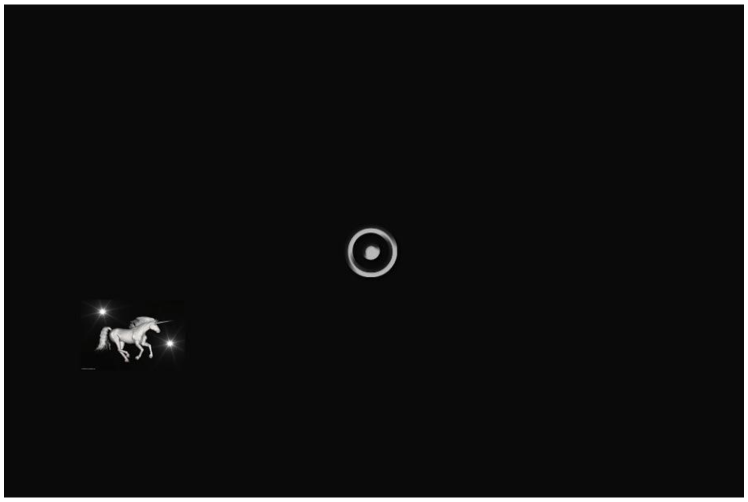


Figure x: *Example of an entity zoomed out too far. This is what happens when the user focuses too little on an entity that is relatively far away. Such a picture would not score many points. Original image taken from: <http://abstract.desktopnexus.com/wallpaper/358514/>*

Example Tutorial Screen

The figure below shows a tutorial screen. This is one of two different concepts we can implement with tutorials. If more time is available, our team would implement an interactive tutorial over a simple how-to-play list.

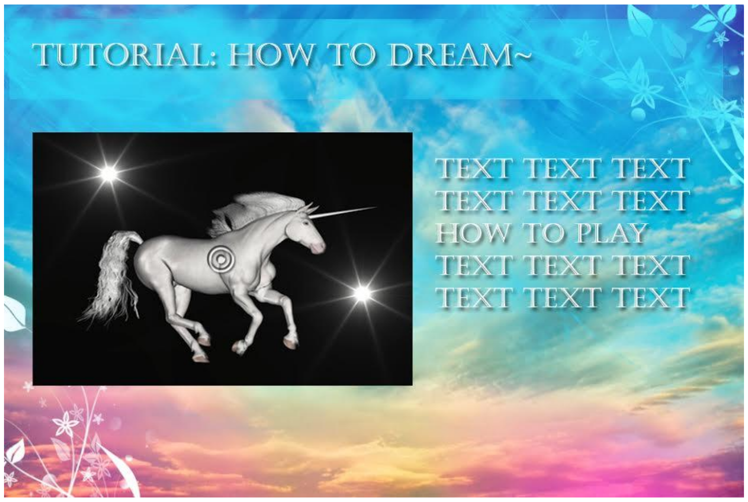
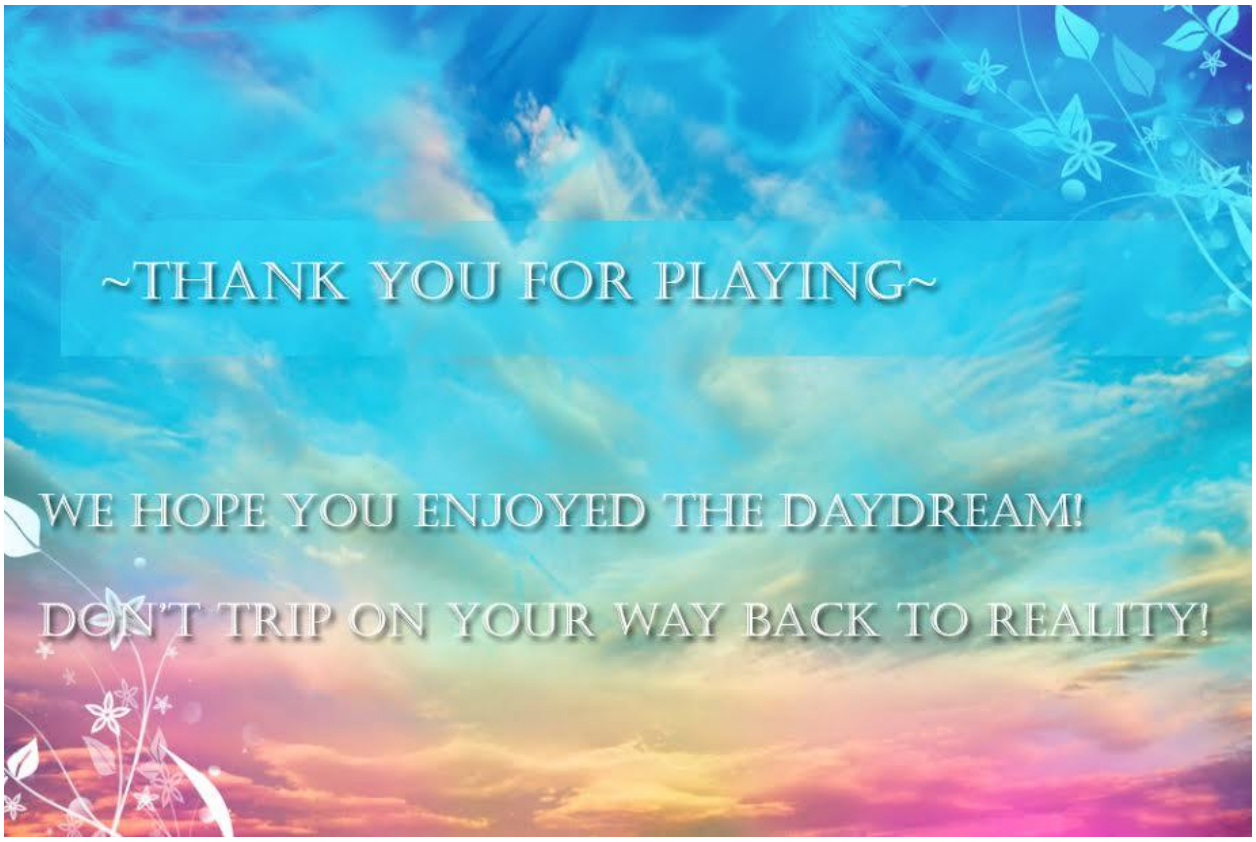


Figure x: *Example of a tutorial in action. The aesthetic design of the tutorial is a work in progress; however, the overall layout is the same. The actual text would be much more helpful.*

Example Thank You Screen

The figure below shows a Thank You Screen. The screen would be displayed after the user has chooses to leave the game. Like the tutorial screen, this is one of two different concepts. If more time is available, the Thank You Screen would not be static, and the user would be placed in one interactive scene. This scene would have credits and additional information available if the user looks around.



~THANK YOU FOR PLAYING~

WE HOPE YOU ENJOYED THE DAYDREAM!

DON'T TRIP ON YOUR WAY BACK TO REALITY!

Figure x: *Example of a thank you screen in action. The aesthetic design of the tutorial is a work in progress; however, the overall layout is the same. The actual text would be much more helpful.*

7 Annotated References

Below is a listing of some references our group has taken into consideration. All of these references deal with software that implements BCI.

Folgieri, R., & Zichella, M. (2012). A BCI-based application in music: Conscious playing of single notes by brainwaves. *Comput. Entertain., 10*(3), 1-10. doi: 10.1145/2381876.2381877
This paper details the possible application of BCI instruments to play musical notes. The detail this paper takes into the effects of a combined paradigm of audio, gesture, and visual stimuli on the user is especially important for the considerations our team will take on this project.

J, M., & rvinen. (2013). *Beyond Five Senses: Non-Sensory Output in Brain-Computer Interface (BCI) Research*. Paper presented at the Proceedings of International Conference on Making Sense of Converging Media, Tampere, Finland.
This paper proposes a research review on the topic of non-sensory output in BCI. This paper proposes alternative outputs rather than visual or audio stimulus and notes the advantages of implementing additional output. The paper also does well in illustrating the limitations imposed by both HCI and BCI input and output. Our team will consider these limitations and design input to accommodate.

Kapeller, C., Hinterm, C., ller, & Guger, C. (2012). *Augmented control of an avatar using an SSVEP based BCI*. Paper presented at the Proceedings of the 3rd Augmented Human International Conference, Megève, France.
This paper focuses on the use of BCI instruments to control an avatar in the popular video game, World of Warcraft. This paper details the process for appropriating working BCI control onto a video game and will greatly influence the considerations our team will take on this project. The complexity of World of Warcraft and the methods Kapeller takes to accommodate it will greatly influence the methods that we deal with user input.

Yoh, M.-S., Kwon, J., & Kim, S. (2010). *NeuroWander: a BCI game in the form of interactive fairy tale*. Paper presented at the Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing - Adjunct, Copenhagen, Denmark.
This paper focuses on the use of a specific BCI device, the NeuroSky Mindset, in the implementation of an interactive version of a fairy tale. The NeuroSky functions very much like the Mind Wave device we plan to implement. The NeuroSky's use of a "attention" and "meditation" state will be especially valuable as our team uses a "focused" and "unfocused" state to accomplish the same task and similar roles.

8 Contributions

Sarah Koh contributed the following:

- Activity diagram: Main Menu
- Class diagram
- Class diagram descriptions
- Method descriptions for ScoreManager
- Method descriptions for StageManager
- Abstract
- Introduction

Matt Fredrickson contributed the following:

- Activity diagram: GameLoop
- Method descriptions for Engine
- Method descriptions for UIManager
- Method descriptions for InputManager
- Initial hardware design section
- Cover page
- Table of contents
- Glossary
- Organizing individual contributions and compiling the final document

JC Orolfo contributed to the following:

- Activity diagram: Take a Picture
- Method descriptions for Sound Manager
- Method descriptions for Entity Manager
- Method descriptions for Picture Manager
- System-level diagram
- UI design snapshots
- Revised annotated references

9 Glossary

Alpha Waves	A type of brainwave that registers the state of physical and mental relaxation. It is associated with deep relaxation and ranges between 7.5-14Hz.
Augmented Reality	A live view of real-world locations and objects, with digital overlays generated via computer.
Beta Waves	A type of brainwave that is emitted during and alert or agitated state. It is associated with waking life and ranges between 14.40Hz.
Brain Wave Activity	A measure of the amount of synchronized electrical pulses generated from neurons communicating with each other.
Brain-Computer Interface (BCI)	A communication between a computer system and the brain.
Calibration	Optimizing of device by adjusting and standardizing input readings in respect to user’s latent input range.
Electroencephalography (EEG)	A test to detect electrical brain activity using a contact point called an electrode. EEG data is expressed using waves.
Focused State	A state in which brain wave activity is above a certain threshold.
FPS	Frames Per Second.
Google Cardboard	A low-cost, do-it-yourself eyepiece that integrates with Android smartphones to provide a mobile virtual reality experience.
NeuroSky Mindwave	A simple BCI device that measures EEG signals for the purpose of development, entertainment, and education.
Non-invasive BCIs	A brain-computer interface that does not break the skin but rather sits on top of the head.
Unfocused State	A state in which brain wave activity is below a certain threshold.
Unity	A cross-platform video game creation system that includes a game engine and an IDE.
Virtual Reality	A world that a user may experience which is entirely virtual, though means of electronic devices such as a computer or smartphone.