May 2016

Designing a User Interface for Musical Gameplay

Alexander Ximon Guerra
Worcester Polytechnic Institute

Connor Anderson Thornberg
Worcester Polytechnic Institute

Hongbo Fang
Worcester Polytechnic Institute

Kedong Ma
Worcester Polytechnic Institute

Xiaoren Yang
Worcester Polytechnic Institute

Designing a User Interface for Musical Gameplay

An Interactive Qualifying Project

submitted to the faculty

of

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

Submitted by:

Tech Side:

Hongbo Fang

Alexander Guerra

Xiaoren Yang

Art Side:

Kedong Ma

Connor Thornberg

Advisor

Prof. Vincent J. Manzo

Abstract

A game is made up of many components, each of which require attention to detail in order to produce a game that is enjoyable to use and easy to learn. The graphical user interface, or GUI, is the method a game uses to communicate with the player and has a large impact on the gameplay experience. The goal of this project was to design a GUI for a music oriented game that allows players to construct a custom instrument using instruments they have acquired throughout the game. Based on our research of GUIs, we designed a prototype in Unity that incorporates a grid system that responds to keypress and mouse click events. We then performed a playtest and conducted a survey with students to acquire feedback about the simplicity and effectiveness of our design. We found that our design had some confusing elements, but was overall intuitive and easy to use. We found that facilitation may have impacted the results and should be taken into consideration for future development along with object labeling and testing sample size.

Acknowledgements

We would like to thank Professor Vincent Manzo for selecting us to design an important feature of his game and for is support and encouragement throughout the duration of the project. We would also like to thank Dan Manzo for working with us in designing features that improved the quality of the GUI.

Table of Contents

List of Figures and Tables

Figure 4.3.3: Result collected for question 7

Executive Summary

Introduction and Background

Game User Interface design is different from other applications. A well designed User Interface will not only allow players to communicate with the game effectively, but also enhance the immersive experience. We conducted a large amount of research on the current state of User Interfaces within games, paying particularly close attention to role playing games (RPGs) and games that implemented deep menus. We also gathered research on general theory and classification of interfaces, as well as the underlying psychological principles that act behind them. We believe this research helped to give us a sophisticated understanding of the field we were entering into and the mental processes of players.

Methodology

Our objective was to design a UI system for inventory screen to facilitate the player selecting instruments and use together. We initially focused on designing an extensive UI and control scheme. We spent the majority of our project developing the “Item Grouper” function for inventory system. This function allows the player to create groups of instruments from a grid that stores them. After we developed this function, we conducted a playtesting session using WPI students, to determine how easy it was to use. We collected data about the amount of time that it took for each user to complete tasks, and then had them complete a survey about the tool they had just used.

Results

Our playtesting results showed that the majority of users found the system to be both easy to use and intuitive to learn. The data helps show that our design methods lead to the creation of a UI that embodies good design features. There are some areas of our design that need improvement. Based on the results we should add labels to the final design to make the instruments easier to find. We also found that some of the time data had some numerical outliers, but these may have been caused by error when facilitating the playtest.

Conclusion

Due to a lack of proper facilitation during the playtest, our data may have been skewed, especially in the time data. In the end, we succeeded to create an appropriate interface for the game that the grouper would be installed in with a coherent layout and comfortable controls. More work could be done to improve the visibility of the instruments to help users find items more easily and a larger testing size should be used to acquire more constructive feedback. We also deviated from the desired input method, but in the end it led to a more intuitive and coherent interface.

Chapter 1: Introduction

A User Interface (UI), in general, is a set of control systems to enable a user to communicate with a program. A video game's UI will influence a player's immersion experience, yet remain undervalued by game developers and game critics. Table 1 shows the differences between UIs for productivity software versus gaming software. The design philosophy for game UI is different from other software. Game UI include input methods and gameplay mechanics. Interactivity makes UI designs for games different from productivity software. Productivity software like excel, is not useful if it is difficult to interact with. However, a game would not be fun unless some challenges are involved.

Table 1: UI features in Productivity Software Vs Gaming

	Productivity Software	**Gaming**
Interaction	Mouse, keyboard	Multiple interfaces
Narrative	None	Immersive
Complexity	Minimized	Moderate
Rendering	Menu, pop-up window	Menu, in-game objects

The structure of a UI will differ depending on the goals of the media that it wants to communicate to the player. Even among games and productivity software, the way the UI is designed in not consistent in each product. Shooter games will have most information in the corners of the screen where a strategy game will take up the entire lower fifth of the screen. Is this placement important? What variables need to take into account for each widget on screen? What layout is better and why? Determining an answer to these questions will help us create an interface that is as easy to use as possible. We will research and analyze these topics to determine the best design for our product.

Our goal was to design an appropriate UI for an inventory screen for selecting instruments to use together. This is not a type of game that has a lot of history, so we wanted to

create a system that is as innovative and effective as possible, within this niche. We conducted research on UI guidelines and determined features that should be included in the design and those that are not applicable to the game. We then produced a prototype to test and collect data on our design. Finally, we analyzed and documented our findings and reflected on the process.

Chapter 2: Background

2.1 General Principles of UI Design

Game UI development will design a visual interface and control panel, a series of procedures (sequence of action player has to take to finish the task) and feedback. A tightly crafted game UI must be fun, engaging, aesthetically pleasing, also easy to master, fast to use, responsive and robust. Players will have a more enjoyable and more immersive game play experience if the game design includes these components:

2.1.1 Usability: ease of use, ease of learning

A simple, consistent UI will allow players to jump right into the game play and enjoy the content. If a system is difficult to learn or use, players will blame the bad game design instead of spending more time to get used to the UI.

2.1.2 Playability: A UI should be interactive, immersive and responsive.

All keystroke actions should be intuitively mapped based on players' habits. For example, the most common control for navigation is using "WASD", Common movement keys in First Person Perspective gaming (W = Forwards, A = Left, S = Backwards, D = Right). Sound and visual feedback is also part of user interfaces. Linking unique feedback to specific player action not only makes the game feel responsive, but also trains the player to perform different interactions

2.1.3 Aesthetic: Appealing UI improves game play experience

UI aesthetics is part of the game narrative; it should be consistent with game's setting and related to game play. An attractive UI could also help game sales.

2.2 Types of Game UI

Erik Fagerholt and Magnus Lorentzon from Chalmers University of Technology explored theories of game UI. Figure 2.1 shows four types of interfaces Fagerholt and Lorentzon linked to the narrative and game geometry: non-diegetic, spatial, meta, and diegetic.

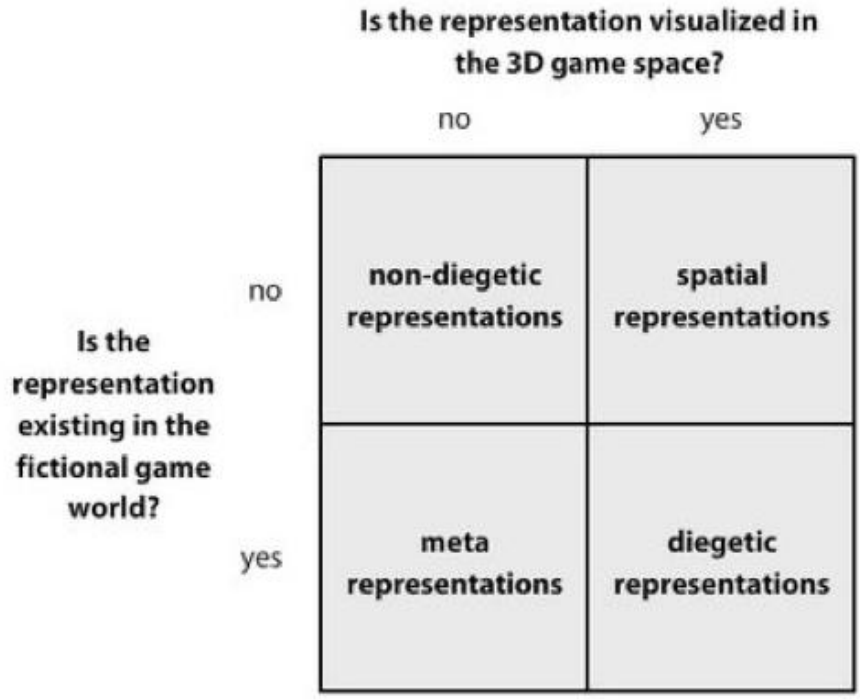


Figure 2.1: 4 types of UI

2.2.1 Diegetic

Diegetic user interface elements that are included in the game world allow the player's avatar to interact with it within the game through visual, audible or haptic means. For example, in the *game The Division*, shown in Figure 2, both the player and the character can see a map that projected on the ground.

Figure 2.2: *The Division* – The hologram map is rendered on the ground around the player

2.2.2 Meta

Meta UI elements do not fit within the geometry of the game world. They sit on the 2D hub plane and are only visible and audible to the players in the real world. For example, as shown in Figure 3, in *Call of Duty* and many first person shooter games, the blood splashing on screen indicates the player is taking damage.

Figure 2.3: *Call of Duty* – The blood spatter effect on screen indicates player is wounded

2.2.3 Spatial

UI elements are not part of the game geometry but exist in the game world providing more information to the player than the character's avatar should be aware. The character highlight in *Assassin's Creed IV: Black Flag* is an example of non-diegetic spatial UI.

Figure 2.4: *Assassin's Creed IV: Black Flag* – In game characters are highlighted in different colors depending on whether they are friendly or hostile.

2.2.4 Non-diegetic

Non-diegetic User interface elements are rendered outside the game world and have the freedom to be completely removed from the game's fiction and geometry. Most of heads-up display and menu screens fit in this category. Figure 5 shows the inventory system in *Skyrim 3*, which pauses the game when it is enabled.

Figure 2.5: *Skyrim 3* – Inventory System takes the player out the game play and pauses the game

2.3 Psychology of UI Design

A UI needs to be simple, yet effective. A well designed UI should “focus on users and their tasks instead of technologies used, consider functions first [then] presentation later, simplicity of [the] UI, [and] promote learning and delivering information” (Chee 1). These four principles combined with cognitive psychology can help enhance the usage of the UI and aid in making the game more comfortable to play. These principles along with general human computer interaction (HCI) guidelines can be related to various parts of a game interface and how it enhances that area.

2.3.1 Simplicity

Simplicity of the UI is very important as its makes it easier to deliver information and knowledge to the player while not taking away from the main game experience. By considering cognitive psychology more can be done that to not affect the game experience, it can instead be enhanced by a well-designed UI. In Retention Theory, there is a limitation on how much information can be retained in the human short-term memory. This limitation is 7 plus or minus 2 items. When designing a UI the number of important icons should be limited to at most 9 items as to be comfortable on the player’s memory so that they focus on the main game and not having to memorize UI elements. The memorization objects can also extend to actions the user takes to use the UI. For a UI to be simple, the number of actions to use the UI should be low as to not impede the action of the main game and to not stress the memory of the user. Use of the UI should also be fast as the Law of Simplicity states that “users understand faster if the information conveyed is simple” (Chee 1). Games with minimal information on the heads up display (HUD) are easier to learn and the users will tend to understand the game quicker. A user will more likely understand a scrolling shooter game quicker than a first person shooter game. Scrolling shooters usually have few controls and a HUD that displays the score and health of the player. This suggests the player needs to score points while preventing health loss in order to do well in the game. In a first person shooter game, the HUD is usually cluttered with icons, text, animations, and other constantly changing information. Without an addition of a slow paced tutorial, a new user will respond negatively to the user interface and it will take them much longer to understand the game; this process may also be frustrating to the user.

2.3.2 Facilitation

A good UI does not only show information to the user, but should help teach them about the system. There are more laws of cognitive psychology that will help explain design choices if an interface and how they enhance the user's experience. The simplicity of a UI would fall under the law of simplicity, but can also apply to the law of Prägnanz, where simplicity and a "balanced layout [is] considered as good [form]" (Chee 2). This can help deliver information to the user much easier so that they do not feel overwhelmed while playing the game. The law of proximity and figure-ground are two other cognitive tools that can be used to teach the user how the system works. By placing objects closer together than others, you can convey groups to the user. The user can infer that objects in a group relate to each other in some way, thus the UI has taught the user how to use the system more effectively.

2.3.3 Placement and Appearance

There are also three main issues to consider when designing a good UI. These issues are the placement of text and images, limitation of human motor systems and the use of colors in the UI.

2.3.3.1 Placement of Text and Images

"Under [Left-to-Right] theory, it is advised to allocate important information or data on the top left corner of the screen by the mean time allocating less important data on the bottom, or specifically bottom right corner" (Chee 3). The way the human visual field is structured, the left visual field prefers images where the right prefers text. This suggests it is generally better to keep less complex information, like images, on the left side of the screen and have more complex information, like text and charts, on the right side of the screen.

2.3.3.2 Limitation of Human Motor Systems

The limitation of human motor systems is very important to consider when designing a UI. The theory is that cognitive load increases on the user if you provide them with multiple sensory information. For example, if there is a dialogue playing, any background audio would interfere and make it difficult for the user to focus on the dialogue. This is also true if the user

has to read text and listen to sounds simultaneously, it causes the user's cognitive load to increase and makes it harder to focus on each individual sense.

2.3.3.3 Color

Colors are a big consideration in UI design. Colors can be used to grab a user's attention and focus them on important information. The theory of "blue peripheral vision" suggests that blue is the best color to acquire the user's attention (Chee 3). It's also worth noting that the shade of the colors plays a role in comfort. Bright colors are more taxing on the eyes as the muscles must adjust to look at these colors which will tire them out after long periods of time. While soft colors provide more comfort to the eyes and bright colors are better at grabbing attention, they should not be shown at the same time for extended periods. It is again because the eye must adjust from one tint to another that the eyes would tire much quicker and cause the user discomfort.

2.4 Existing UIs

After researching general UI design features, we decided to look at already existing examples of UI from popular games. The goal of this was to analyze the design choices used for that game's UI and areas that are well done or need improvement. Through this method enabled us to build off existing ideas for UI design to then incorporate our own elements to use in our final design.

2.4.1 Minecraft

Minecraft is a sandbox game great for younger and older people to play with friends and adventure or build and be creative. In order to allow this, the UI has to be simple for kids to understand and use with ease, while being complex enough to allow maximum creativity. The inventory and UI of Minecraft are both very simple. All the player needs to do is navigate menus or their inventory and then click and drag the items they desire to the hot bar. In the image you can see the hot bar can hold nine different items. This is the maximum recommended limit of items in a UI as the human attention can hold around seven plus or minus two items at a time. Since Minecraft is at the high end of this scale, players may have to look down at the hot bar once in a while, but can also better organize their hot bar so the items they will use most often can be a short mouse scroll up or down. One area Minecraft could improve on is color.

Figure 2.6: *Minecraft* - the hot bar holds 9 different items

2.4.2 Skyrim

Skyrim is a role playing adventure game for mature audiences where the player explores the world while using weapons and magic to defeat their opponents. The menu system displayed below is the player's inventory of weapons, food, clothing, etc. From a psychological standpoint, this system could use some modifications. For example, the image of the item should be on the opposite side of the screen with the words in the menu on the right side instead of the left. This is because the left eye is better at processing images and the right eye is better at reading text. Even though the text would be on the better side, there is still quite a lot of text to go through. The human mind can recognize symbols better than words. The addition of icons could help make the text better to read, or limit the need of an excess amount of text.

Figure 2.7: *Skyrim* - the menu system displays player's inventory of weapons, food and clothing

2.4.3 Counter Strike

Counter Strike is a first person shooter where the player buys the weapons they want in their inventory before the start of every round. The UI for the weapon selection menu is very well done. There are six options on the wheel, which is one less than the average person's short term memory retention, and only one sub menu to each selection. The load out can be selected easily by new players by moving the mouse and clicking, but professionals can later memorize the key presses needed to complete the selection process in seconds. The colors used are very soft and have a darker tint leading to less strain on the eye muscles. Although it seems a poor design choice to put the images on the right visual field, the use of icons and bars limit the overall need for words, other than to denote the weapons types.

Figure 2.8: *Counter Strike Global Offensive* - Wheel system with sub menus for each section

2.4.4 Assassin’s Creed

Figure 2.9: ***Assassin’s Creed Brotherhood*** **- All of the player’s items are all icons in a wheel**

Assassin’s Creed is a role playing adventure game where the player uses a variety of tools to kill enemies strategically. In the image above the player's inventory can be seen. This UI capitalizes off of its use of icons. All of the player’s tools and weapons are all icons in a wheel. This makes the UI very simple which is good as the player never needs to be in the inventory for too long of a time. To equip a weapon, the player uses the directional pad to set the item the larger wheel is pointing at to that direction on the pad. This makes it that there only 4 equipped items while the player is in the action. This number is well below the average short term memory retention so the player should not have to check their items while they are in combat. There are still many weapons the player can choose from as they progress through the game. However, they can only carry one weapon of a certain weight. This means once players choose the weapon they want in their inventory, they do not have to worry about sorting through all of their heavy weapons to find the giant axe they just bought, they simple have to equip the large weapon icon and they are good to go. The icons in this particular game are a very bright white color which being next to darker colors makes them harsher to the eyes.

2.5 General UI Development Procedure

Figure 2.10: *Battlefield 4* - Spotting in Battlefield 4

2.5.1 Step 1: On-screen information analysis

The first step in UI design is to identify the basic game play information that a player should be constantly aware of. UI designers should be able to describe how it feels being a game avatar. For instance, when designing UI for a first person shooter game, does the player avatar have the ability to spot enemies behind cover? What are the basic stats that associate with the player's avatar? During this step, developers should identify functions that will be present in the UI like crosshair, health, ammo, mini-map, inventory, etc.

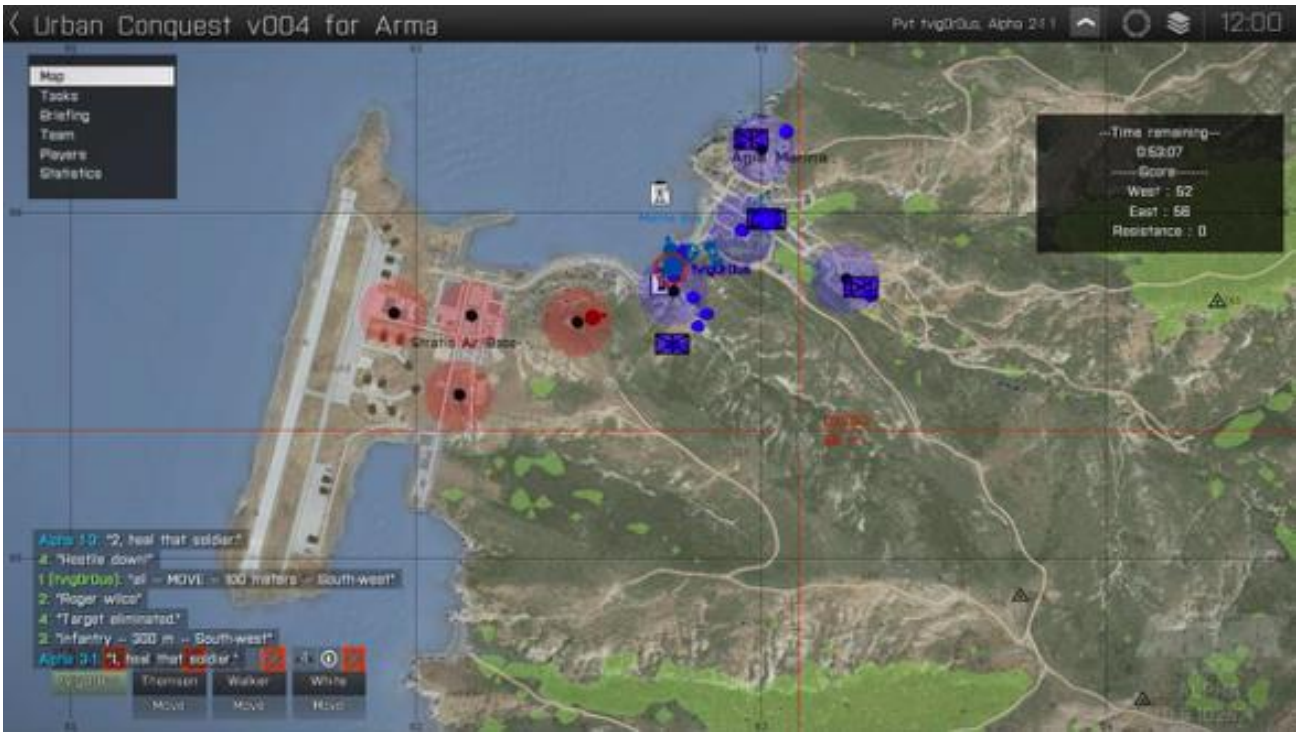


Figure 2.11: *ArmA 3* - Spotting in ArmA 3

2.5.2 Step 2: Design player actions and in-game feedback

At this step, the UI design team will work with game mechanic designers to determine what player can do in this world. For the same task, player will be engaged in different levels of interaction depending on the game genre or theme. For example, Battlefield is arcade shooter in which players shoot for points, while ArmA is a tactical military shooter that focuses on gunplay. Both games have spotting mechanic that allows player has the ability to spot enemies from distance. In Battlefield, this ability might involve a player tapping a button and the system automatically marking the enemy with a red triangle on screen in real time. Whereas in ArmA, the player will be tasked to do a sequence of actions including bringing out the map, zooming in/out, marking the estimated location of the enemy on the map and closing the map. For each action the player takes, there should be a specific feedback to make the game feels responsive, either through sound, visuals, or both.

2.5.3 Step 3: Study target audience

It is important to know your target audience and their ability in terms of learning a procedure. A procedure is a sequence of action the player takes to complete a task, like the spotting in ArmA3. Developers should predict the learning time of each procedure; choose the design with the best combination of learning and execution times. In the end of this step, developers should have chosen a specific input device and came up with a list of keystroke-level actions linked to visual and aural feedback.

2.5.4 Step 4: Start Design a preliminary UI

During this step, developers will create an appealing mock-up of UI elements that are cohesive with the game narrative. Meaningful functions will be mapped to different interface methods - diegetic, meta, etc.

2.5.5 Step 5: Testing

Organizing play testing with the target audiences and professional play testers will provide valuable feedback. Developers should record how long it takes for the average player to learn and execute each task and assess complexity, consistency and ease of learning. A robust UI requires multiple play testing sessions to find system bugs and design flaws.

Chapter 3: Methodology

Our main objective was designing the most appropriate interface for creating custom instruments from the ones the player has collected during the game. We started by inquiring about the control scheme of the game and what controls are already mapped to specific functions. Our initial constraints were that the player can only use number keys and the "-" and "=" keys to navigate the interface. We then decided to look at existing UIs that are mainly constrained to keyboard input and constructed a task analysis to act as a guideline to determine if parts of that interface could be replicated into our design.

3.1 Task Analysis

For our interface, there was a set number of actions the user could do to interact with the system. We call our interface "The Grouper" as it is where the player groups instrument to make their own custom instrument. We determined the actions we wanted to highlight and search for in existing UIs:

- Add and remove instruments from the group.
- Change the volume and mode of each instrument in the group.
- Save and load groups.
- Sort instruments to better find desired instruments.

3.1.1 Counter Strike: Weapon Selection

Our design could use a lot of the elements from Counter Strike's UI (Figure 2.8). The system for selecting weapons for a load out could work for use by having categories of instruments that expand into other menus that have the desired instruments. The way that the inventory is set up is that expert users can quickly fill their load out using a series of quick key strokes. We wanted to incorporate this into our game for experienced users to not feel slowed down by the system.

3.1.2 Minecraft: Grid Layout

After some discussion, we came to the conclusion that a branching menu system like the one described from Counter Strike may lead to some complications. The first issue that arose

was deciding how to group the instruments into categories and subcategories. Some of the instruments that are used in the game are unique and complex and may contain features that belong in more than one category. The second problem was that this system may be hard or frustrating to use, especially for a user who is playing the game for the first time. Minecraft is a game with a younger target audience and shows this with its gameplay and inventory organization. The grid view of the inventory makes it much easier to see all the items the player has and interact with those items.

3.2 Artistic Development

The art team for this project developed a host of assets to be used in the UI. In addition to the assets needed to create the grouper, we also concepted and prototyped several other aspects of the UI.

3.2.1 Early Development

When we first began to deliberate on a design that would work for our game, we broadly concepted several different types of UI. One of the first things we developed was a rudimentary version of the item grouper. This design, shown on the next page, was heavily influenced by the quick radial selection of CS:GO. The instruments are broken up into categories to help the player find the desired item more quickly. One can also use key presses to navigate through the grouper, like in the "buy menu" in CS:GO.

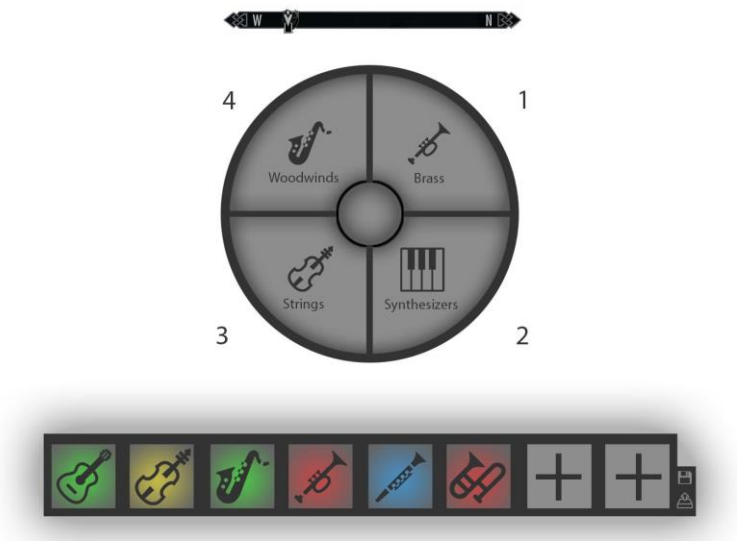


Figure 3.1: UI Initial Concept

The grouper above also shows early versions of two other UI elements that we artistically prototyped. We deemed that it was out of scope to try to implement these elements technically, but that it was still worthwhile to develop the ideas. The aforementioned UI elements are the "Navigation Bar," depicted at the top, and the "Hotbar," depicted at the bottom.

The Navigation Bar was inspired by UI in Skyrim, and consists of a small panel at the top of the screen. It shows the player where they are facing, in reference to cardinal directions, and what important events or activities are around them.

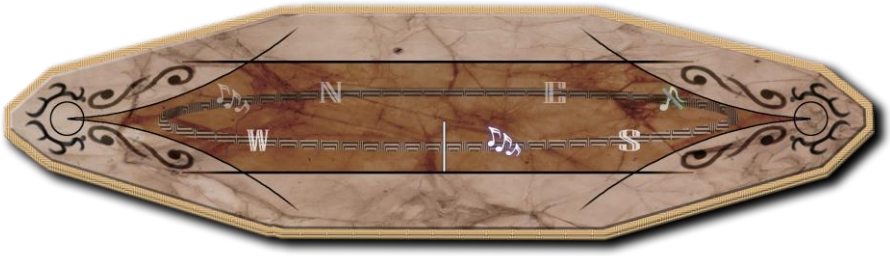


Figure 3.2: Navigation Bar Prototype

The Hotbar is where a player's instrument selections are stored for quick use after they use the item grouper that we developed. An instrument or instrument group can be held in each space. In the latter case, the Hotbar will display the first instrument in the group as the icon. The hotbar also denotes the mode of each instrument or instrument group, by changing the color of the background.

Figure 3.3: Hotbar Prototype

Early in the design process, we planned on having two other functional tabs in the grouper interface. These were the Quest Panel and the Hero Panel. The Quest Panel would keep track of the activities that the player had completed in the larger game and provide information about those tasks to the player. The Hero panel was very similar to the Item Grouper in structure, but had a grid of wearable items and a render of the player character on the right. These designs will be included in the appendix.

3.2.2 Later Development

Once we had a better idea of the scope of our project, we set our sights entirely on the Item Grouper. We refined some of the designs we'd been toying with earlier, and ended up with a grid based system. All of the items that character had access to were displayed in the grid, and

could be added to an active selection at the top of the interface. On the right page of the book, there is a rendered, spinning version of whatever instrument the player hovers over with their mouse.

Artistically, we decided to go with a book theme, because it fit the setting of the larger game and served our purpose as a flat surface the player can interact with. We developed a grungy paper texture and a repeatable frame to be placed around each cell of the grid. Our final efforts are depicted below.

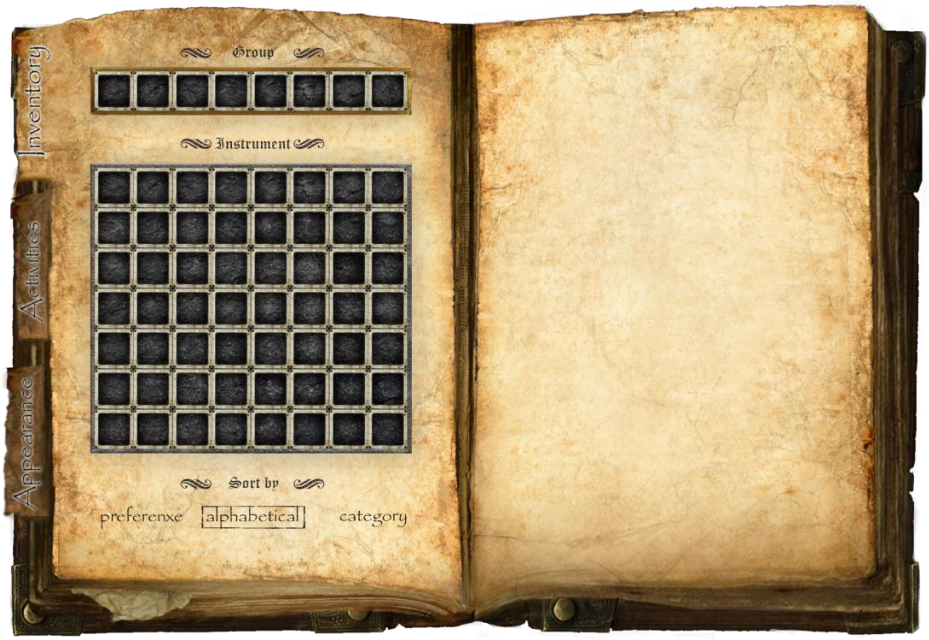


Figure 3.4: Final Grouper Textures

For the Item Grouper to be usable, we also needed to develop instrument icons. This allows the player to have something representational to interact with when using the grouper. 3D models of these instruments had already been created, so we used images of them to create each icon. Each image was posterized, had a cutout filter applied to it, and had image levels adjusted to be more readable to the user. Overall, we created sixteen distinct icons to be used in the Item Grouper. These are included in the appendix.

3.3 Grouper Development

The player can easily switch from gaming interface to inventory system by PRESS “Q”. One major component of the inventory system is grouper system, which enables player to select a group of eight instruments from inventory without much of restrictions and place them into one empty box of hotbar as a group. The chosen instruments can be duplicated to certain pattern required by game or by player. The purpose of designing the grouper was serviced for a major activity which will be triggered in the later part of game. Player can perform multiple instruments at the same time like a solo band. Figure 3.4 is a screenshot of grouper inventory system, which gives a sense of what the inventory system looks like. Limited by the number of current instruments, our grouper prototype only has 16 boxes in the inventory system.

Figure 3.5: Grouper Inventory Interface

Grouper system is generally a selection interface. To show a nice interaction with player, three major features were implemented, flying instrument from one box to another box, rotating the preview instrument over the right side of the page and sorting all instruments in the inventory based on the name in alphabetical order.

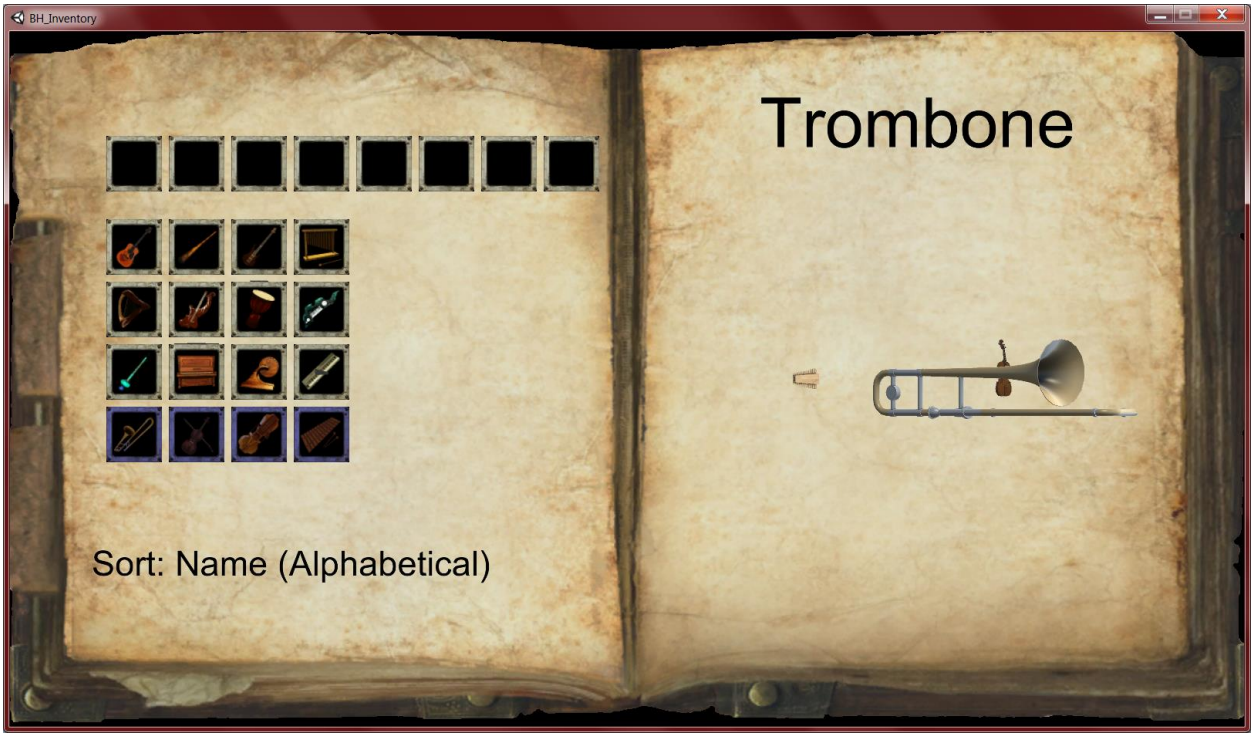


Figure 3.6: Instruments sorted in alphabetical order

3.4 Technical Description Features

3.4.1 Flying

Flying instrument is the case that the instrument can fly from one box to another box with a smooth animation. When a box is selected by the mouse or the keyboard input, the instrument in that box will fly to one empty box on the hotbar. The algorithm we implemented for the flying feature is not straightforward. Since the instrument in the box would not be removed after the flying, we cloned the instrument with the same position and rotation first. Then the instruments were able to fly to the designated box with a slow rate. “FlyBehaviourScript.js” contains the core code of flying behavior.

```
function Update() {
    if (Input.GetMouseButtonDown(0)) {
        hasClicked = true;

        //var prefab :GameObject = AssetDatabase.LoadAssetAtPath("Assets/Instrument.prefab", GameObject) as GameObject;
        //var prefab : Object = AssetDatabase.LoadAssetAtPath("Assets/Instrument.prefab", typeof(GameObject));
        //Object prefeb = Resources.Load("Instrument", GameObject);
        //clone = Instantiate(prefab, transform.position, transform.rotation) as GameObject;
        //cloneObject = GameObject.Find("Cube(Clone)");

        clone = Instantiate(transform, transform.position, transform.rotation);

        Debug.Log("instantiate one here");

        Debug.Log("locate at "+index);

        target = hotbars[index].transform;

        Debug.Log(target);

        hotbars[index].GetComponent(isFilledScript).isFilled = true;
    }
   if(hasClicked && !hasReached){

        Debug.Log("hasClicked");
        //transform.Translate(Vector3.up * moveSpeed * Time.deltaTime);

        // The step size is equal to speed times frame time.
        var step = moveSpeed * Time.deltaTime;


        // Move our position a step closer to the target.
        transform.position = Vector3.MoveTowards(transform.position, target.position, step);
        if(Vector3.Distance(transform.position, target.position) == 0){
            hasReached = true;
        }
    }
}
```

Figure 3.7: Script of flying update

3.4.2 Rotating

Rotating an instrument is happening in the preview of the selected instrument rotating on the right half side of the inventory system. The selected instrument will show a preview over the

right half side of the inventory system. Rotating instrument gives a detailed view of the instrument over 180 degrees, so player can have a clear idea of what each instrument looks like. Besides the rotation around the center point, the instrument would also rotate itself. Certain degrees difference between the direction of two rotation enable the player to have a clearer sense of what the instrument looks like up and down.

To implement rotating function, we first cloned the selected instrument with the same size, the same position and the same rotation. Secondly, we translated the selected instrument to the right half side. Thirdly, we scaled it with a reasonable degree, so the preview of instrument has enough detail shown to audience and also fits the system. Two different rotations applied to the preview instrument were implemented by calling Unity function twice with different parameter. As the result, one always rotated around a constant point in the center while the other one rotated around the position of itself. The rotation direction and frequency were customized separately.

```
function Update(){
    var attachedText = GameObject.Find("Name_Text");

    if(isSpin){
        transform.RotateAround(Vector3(4,0,-2), Vector3.up, 30 * Time.deltaTime);
        distScale = Vector3.Distance(transform.position, Vector3(Camera.main.transform.position.x + 9,
            Camera.main.transform.position.y, Camera.main.transform.position.z));
        if(transform.position.z < -3 && transform.position.x > 5){
            attachedText.GetComponent(TextMesh).text = InstrumentName;
        }
        transform.localScale = Vector3(2/(distScale-7.0f), 2/(distScale-7.0f), 2/(distScale-7.0f));
        transform.RotateAround(transform.position, Vector3.up, 60 * Time.deltaTime);

    }
    if(isSpinAlone){
        transform.position = Vector3(4,0,-2);
        attachedText.GetComponent(TextMesh).text = InstrumentName;
        transform.localScale = Vector3(4.5,4.5,4.5);
        transform.RotateAround(transform.position, Vector3.up, 60 * Time.deltaTime);

    }
}
```

Figure 3.8: Script of rotating update

3.4.3 Sorting

Toggle with “S”, inventory system can switch between sorted state and unsorted state quickly. In sorted state, all instruments are placed based on name in alphabetical order. The process of switching between two states were designed to look like all instruments flying from

original box to the destination box. The purpose of developing sorting functionality was to make the selection easier for player when inventory system contains massive number of instruments possibly even multiple pages of instruments.

Place the name of all instruments in an array and sort this array by calling Unity sorting method. Call the flying method implemented before, fly the instruments from the original box to the target box.

```
//toggle sort mode
if(Input.GetKeyDown(sort_key) && InstrumentProperties.wait_cnt == 0){
    sort_mode++;
    if(sort_mode > 2)
        sort_mode = 1;

    var ptr : int;
    var ptr2 : int;

    var cases : GameObject[];
    cases = GameObject.FindGameObjectsWithTag("Case");

    if(sort_mode == 1){
        sort_text.GetComponent(TextMesh).text = "Sort: Default";
        //Debug.Log("Default sort");
        var Items : int[];
        Items = GameObject.FindWithTag("IM").GetComponent(ItemList).owned_ids;
        for(ptr = 0; ptr < cases.Length; ptr++){
            for(ptr2 = 0; ptr2 < Items.Length; ptr2++){
                if(cases[ptr].GetComponent(SelectCase).heldObj.GetComponent(InstrumentProperties).InstrumentID == Items[ptr2]){
                    clone = Instantiate(cases[ptr].GetComponent(SelectCase).Instrument, cases[ptr].transform.position, cases[ptr].transform.rotation);
                    clone.transform.position.z = -5;
                    clone.GetComponent(InstrumentProperties).UpdateInfo(cases[ptr].GetComponent(SelectCase).heldObj.GetComponent(InstrumentProperties).InstrumentID);
                    clone.GetComponent(InstrumentProperties).fly(cases[ptr2], 10f);
                }
            }
        }
    }
    else if (sort_mode == 2){
        sort_text.GetComponent(TextMesh).text = "Sort: Name (Alphabetical)";
        //var AllNames : String[];
        var AllNames = new Array(cases.Length);
        for(ptr = 0; ptr < cases.Length; ptr++){
            AllNames[ptr] = cases[ptr].GetComponent(SelectCase).heldObj.GetComponent(InstrumentProperties).InstrumentName;
        }

        AllNames.Sort();
        //Debug.Log("Length:" + AllNames.length + "[" + AllNames + "]");
        for(ptr = 0; ptr < cases.Length; ptr++){
            for(ptr2 = 0; ptr2 < AllNames.length; ptr2++){
                if(cases[ptr].GetComponent(SelectCase).heldObj.GetComponent(InstrumentProperties).InstrumentName == AllNames[ptr2]){
                    clone = Instantiate(cases[ptr].GetComponent(SelectCase).Instrument, cases[ptr].transform.position, cases[ptr].transform.rotation);
                    clone.transform.position.z = -5;
                    clone.GetComponent(InstrumentProperties).UpdateInfo(cases[ptr].GetComponent(SelectCase).heldObj.GetComponent(InstrumentProperties).InstrumentID);
                    clone.GetComponent(InstrumentProperties).fly(cases[ptr2], 10f);
                    //cases[ptr].GetComponent(SelectCase).heldObj.GetComponent(InstrumentProperties).fly(cases[ptr2], 10f);
                }
            }
        }
    }
}
```

Figure 3.9: Script of sorting update

3.4.4 Saving and Loading

Our inventory system included saving and loading functionality for the instrument grouper, so that after user edited the instrument grouper, they can retrieve the grouper after closing the inventory window and even after closing the game. The inventory system used ID numbers to track each instruments. In order to preserve the instruments that user edited in the

instrument grouper, our system had to save the ID numbers to the hard disk in a format that's easy for our operations.

In order to decide the format for storing the instruments, the operations that we apply on these instruments must be identified. Currently, for demonstration purpose, our system had to be able to save and load only one group of instruments. Also, as we expected, in the future, our system will need to store multiple groups of instruments. Besides adding and removing instruments in an instrument group, user may also want to order the instrument groups that they created as their preference, so that they can easily select one of the instrument group at an easier manner. Therefore, the format for storing instrument groups had to be able to represent ordered lists for instruments IDs. Furthermore, in the future, developers of the game may decide to add status to each instruments. For example, in case developer decide they want to add trading system to the game, an instrument can have different duration and therefore different price. Each instruments may have different status. For example, a common instrument has a price so that user can sell the instrument to exchange other desired instruments, while the instruments that are required for main story should not be sold, so that those instruments should not have a price.

To be compatible with our need for demonstration and also the potential future requirements, we decided to use JSON format. First of all, our current goal is to implement a system that for demonstrating the feature of saving and loading, convenience is one of the criteria for choosing the format. As we are using JavaScript for developing in Unity, JSON format is naturally easy to both be generated and parsed by JavaScript programming language. JavaScript in Unity doesn't come with JSON parser, so in our code, we imported SimpleJSON library for parsing the stored JSON file when loading the instrument group. In addition to the convenience, JSON format provided a structural syntax that allows storing the information in an objective-oriented manner, so that it is compatible with potential requirements of adding status to instruments.

An alternative way of storing format is the XML format. The game Running With Rifle used XML format to store all the weapons available in the game and also a list of weapon that player unlocks.

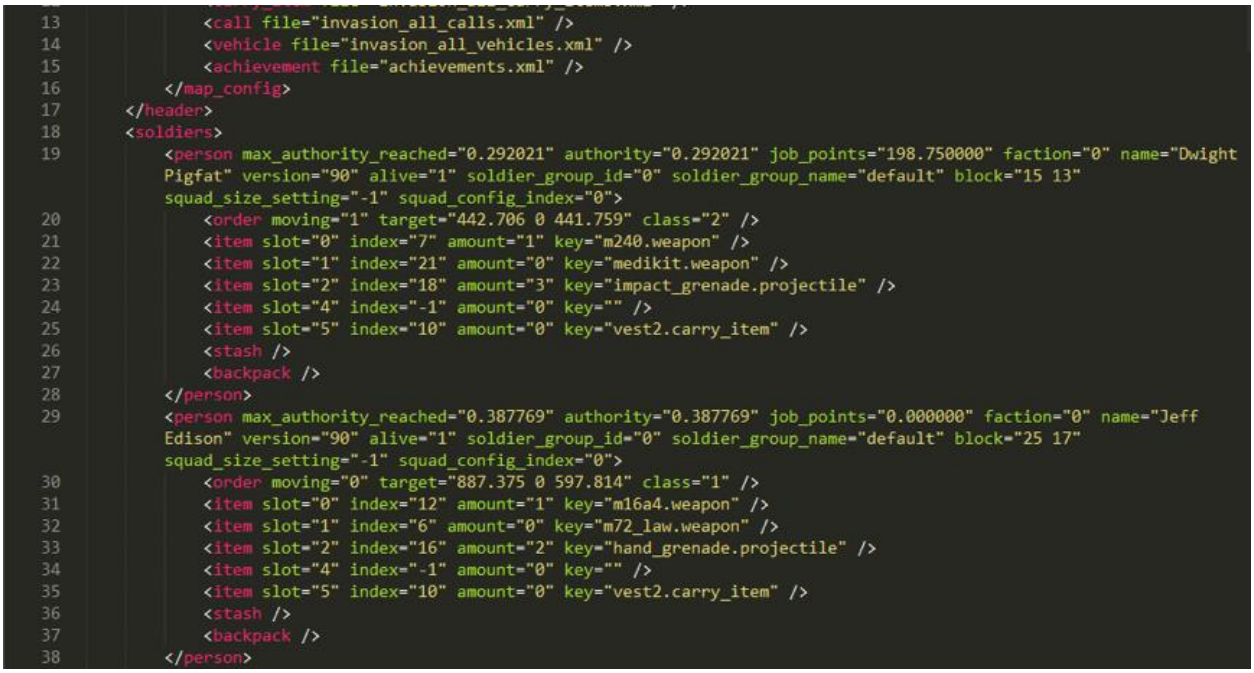


Figure 3.10: Game Running With Rifle used XML format to save characters and their weapons

Compared to JSON format, XML format had benefit of easier data extraction. Programmer can use XPath path expression to specify the selector expression of desired data. For example, XPath can help programmer to easily find the first two weapons of every soldiers that stored in XML format. For JSON format, in order to achieve the same goal, the developer has to manually program the data extractor methods in an objective-oriented manner. However, such feature is not our first priority. The convenience of JSON format outweigh the easy data extraction feature of XML format.

Another decision that we made is to store the JSON file on the game folder as a file in JSON format. An alternative way to store the data is to use PlayerPrefs feature of Unity. In Unity, PlayerPrefs is a library that let developer to store data in a form of key-value pair. As a built-in function, PlayerPrefs is idea to save the game progress. To the players, the difference between saving game data by PlayerPrefs and saving data to a JSON file in game folder is that, JSON file can be open by common text editors and is more accessible to the players. PlayerPrefs stores the data in list format in Mac OS system, and in registry in Windows, therefore not easy to be accessed by players. On one hand, letting players to access the store game data could be a good thing, when players can quickly try out different instruments in game without being

constrained by the progress of the game. On the other hand, this action may also be considered as cheating the game, since many of the sophisticated mechanism may fail due to the unexpected player behavior, thus compromising the musical educational purpose.

3.4.5 Volume control

In an instrument group, the volume of each instrument should be customizable by the player. We implemented the grouper so that when user hover the mouse on one of the instrument, and scroll the mouse wheel up or down, the color value of that instrument will be changed, and therefore representing that the associated volume of that instruments are being turned up or down. We provided this programming interface that developer in charge of audio system can use to control the volume of the instruments if necessary. In the later development, we added wooden frame to the instrument icons, so that the color change when scrolling mouse wheel is not visible. However, the internal value is still responsive to mouse wheel scrolling event.

3.5 Survey and Data Collection

In order to evaluate our inventory interface, we conducted a survey to 19 people. The survey included play test and 8 questions. In the play test, the participants were not given any instructions on how to use the inventory interface and asked to pick three sets different instruments in our inventory interface. The difficulty of each set of instrument was increasing, with the first set of instrument having only two types of instruments and the last set of instrument having 6 different types of instruments. After the play test, the participants were asked for 7 questions about simplicity and efficiency of our inventory design and also provided space for respondent to freely comment on what could be added to our interface design.

Chapter 4: Results

After the playtest session, we collected all the text files and the survey documents and put all the data into a spreadsheet to review and analyze the results (Appendix C). The information we received from the participants is divided into three separate topics to pinpoint areas of our project that may need improvement. The majority of the survey responses are numerical, where five represents a very positive reaction and a one represents a very negative reaction.

4.1 Simplicity

The first two questions asked the participant how easy the system was to use and how intuitive it was to learn.

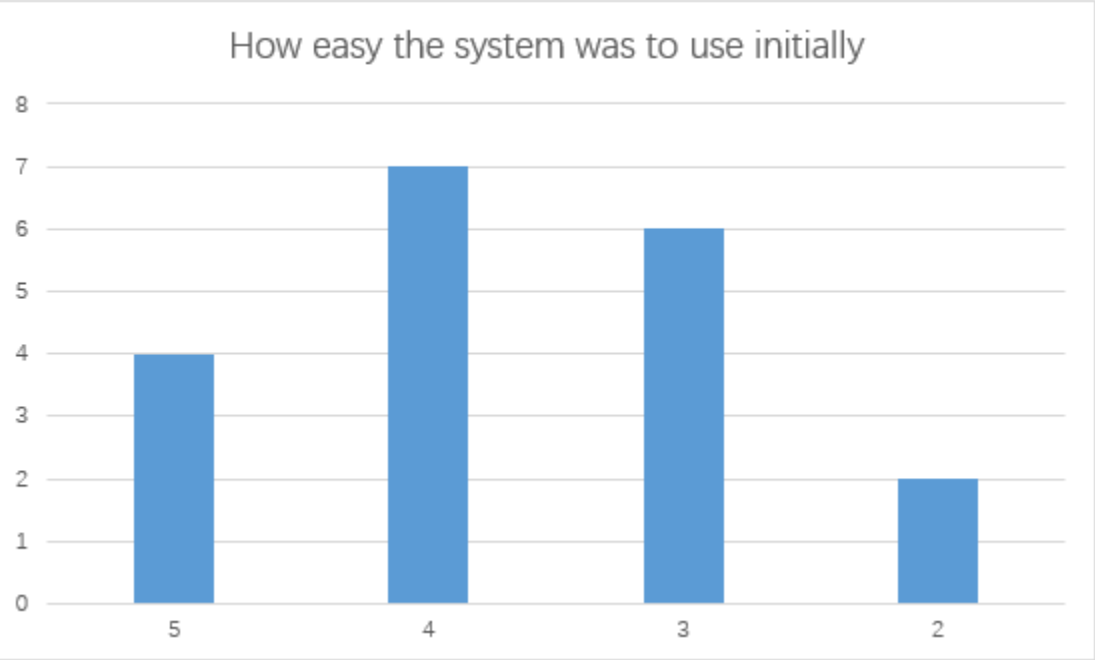


Figure 4.1.1: Result collected for question 1

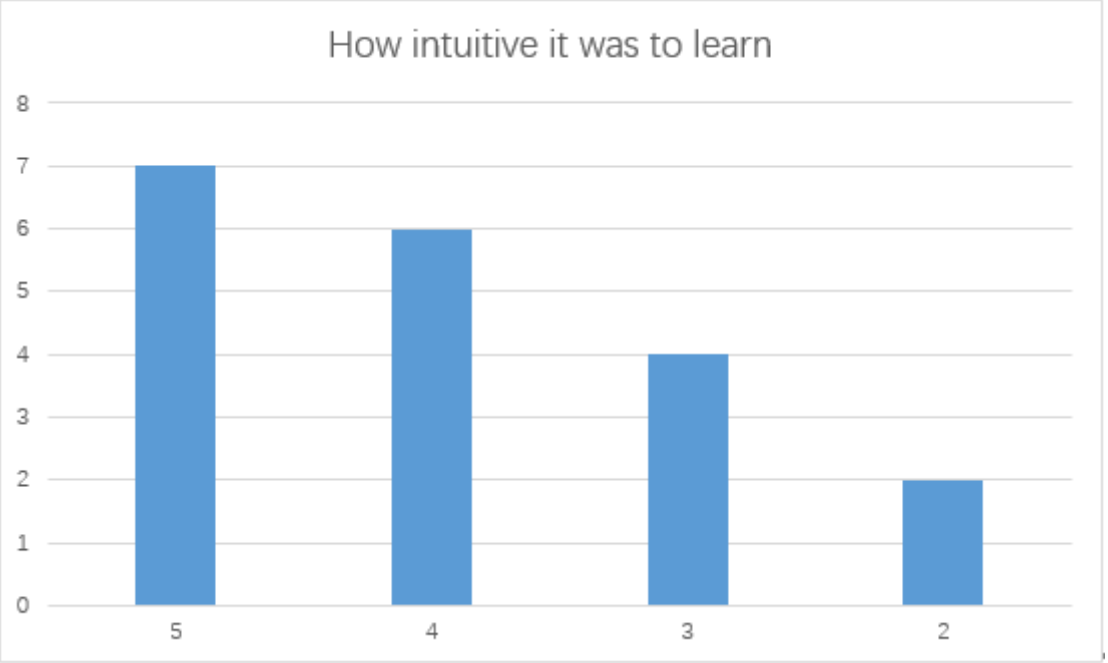


Figure 4.1.2: Result collected for question 2

We received averagely positive responses for how easy to use of our interface, with 89.5% responses to be neutral and positive and 57.9% responses to be positively easy to use. Also, 36.8% people rated our system very intuitive to learn. The result showed that our system provided enough visual clue for users to adapt our interface and to understand how to use our interface.

4.2 Efficiency

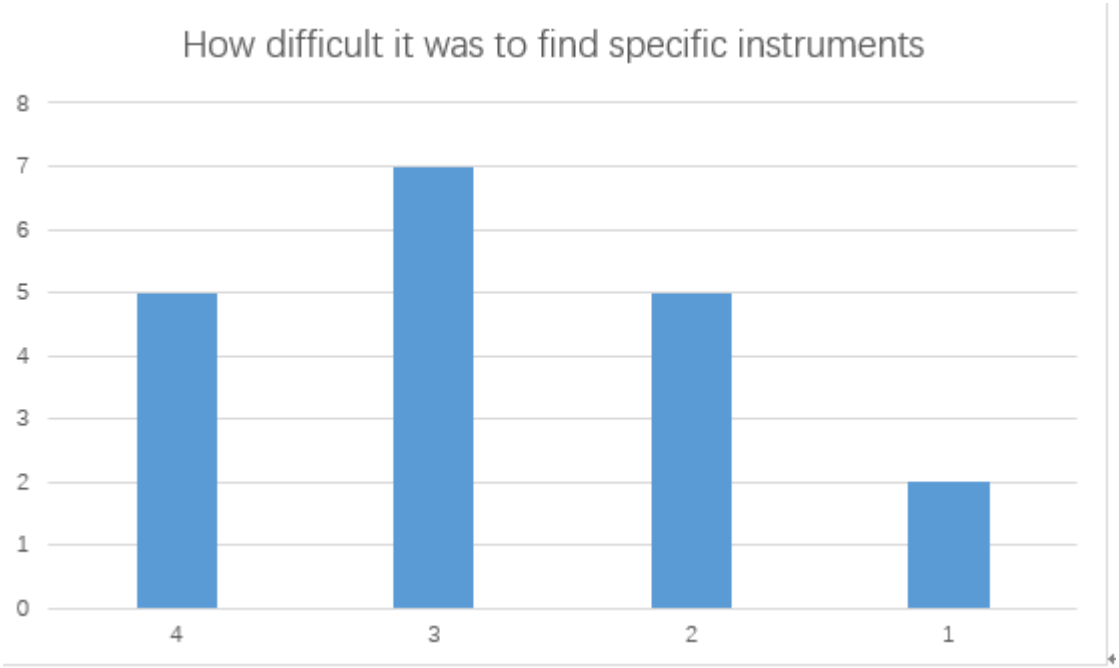


Figure 4.2.1: Result collected for question 3

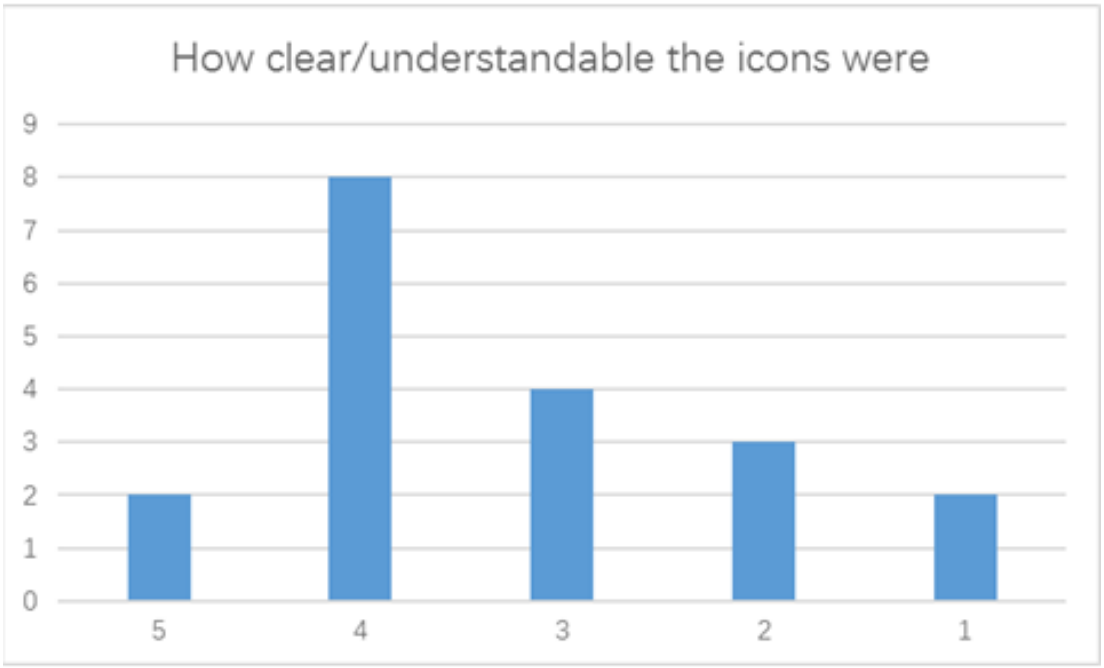


Figure 4.2.2: Result collected for question 4

With 63% people rated that the specific instruments were hard to be found and 57% people rated that icons were clear to be recognized, the efficiency of our system did not achieve as much as the simplicity and intuitive. Two of the respondent suggested that we could add

labels of the instrument's' name attached to the icons. This indicated that the icon along is not efficient enough for users to quickly identify different instruments. On one hand, the icons are relatively small, and some of the instruments were fictional so that they cannot be associated with the names easily. On the other hand, the participants were given only the names of the specific instruments, therefore they have to translate the names into unknown icons. The only way they could achieve that is to hover the mouse over all of the icons to see the larger display of both 3D instrument models and names, and further learn the icon.

4.3 Design

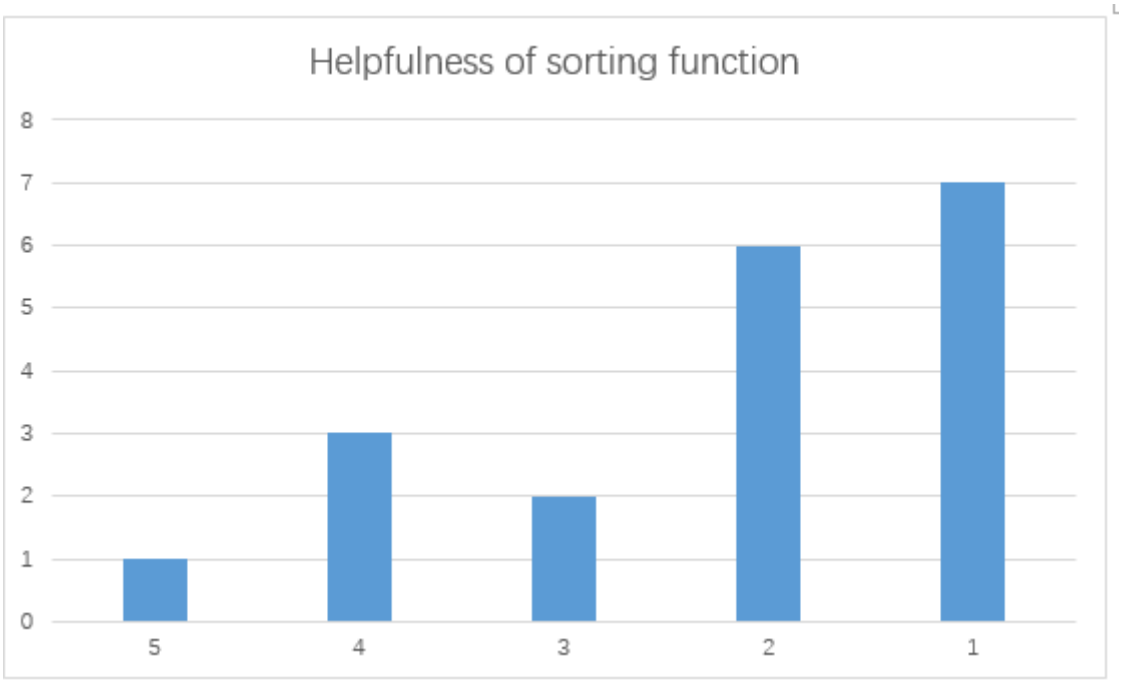


Figure 4.3.1: Result collected for question 5

The result showed that 68.4% people rated that the sorting function was not helpful during the survey. As the sorting function sorted instruments in alphabetical order, without displaying the name label for each icon in the inventory interface, it was difficult to utilized the advantage of alphabetical order.

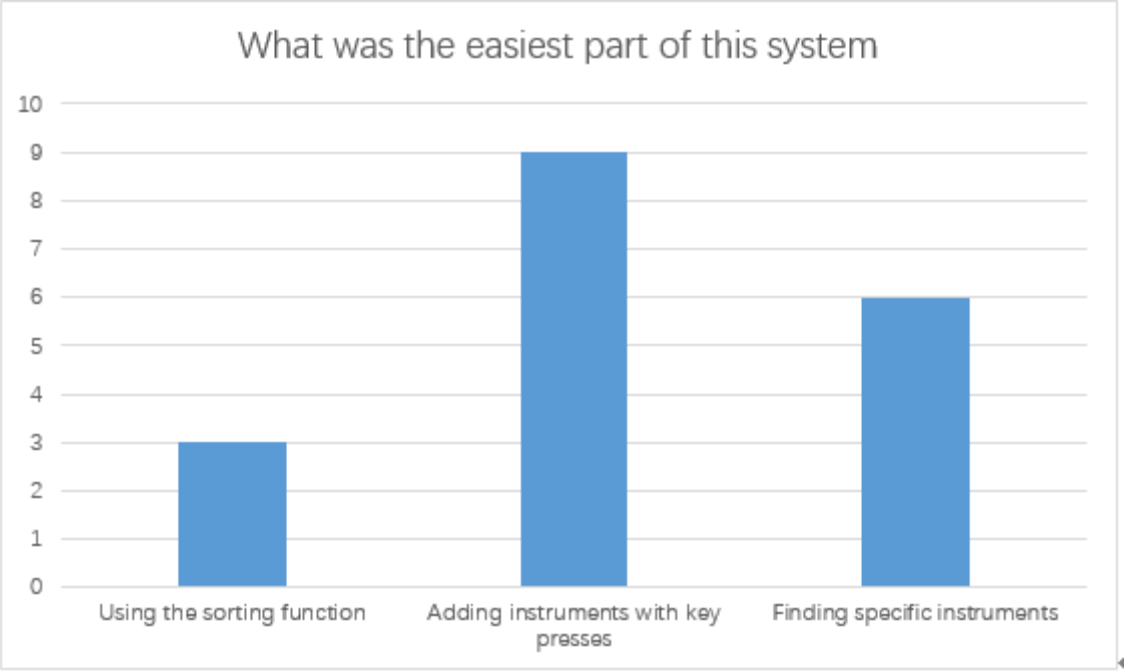


Figure 4.3.2: Result collected for question 6

Our design of selecting instruments with key presses ranked first place, demonstrating that pressing two number key to select instruments is easier to remember than moving mouse to specific location on the screen to select instruments.

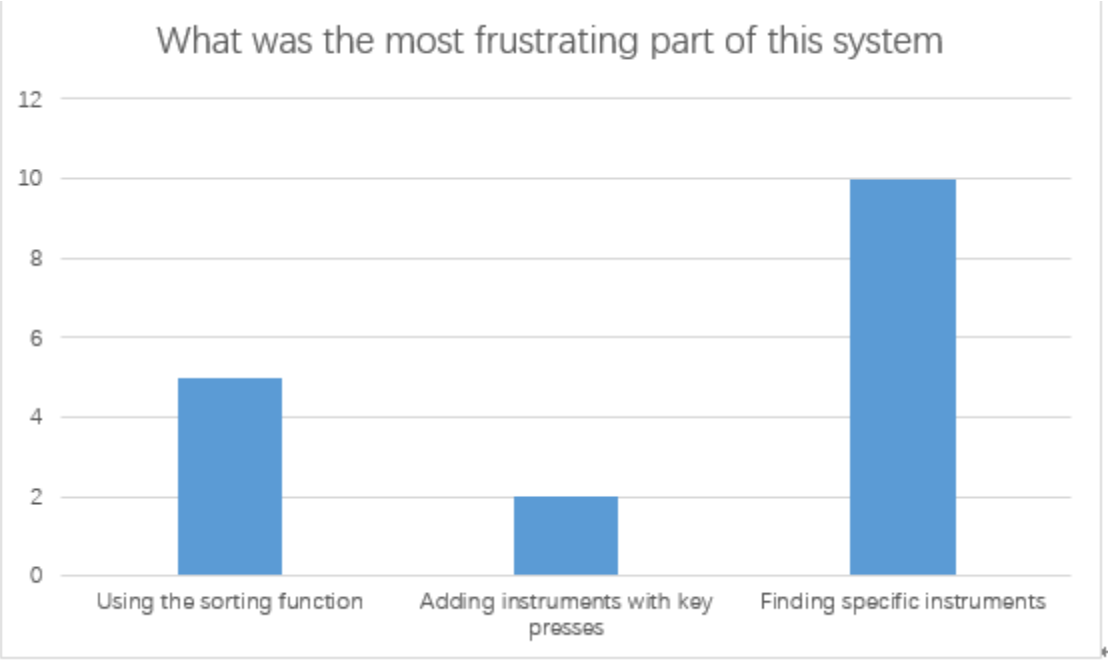


Figure 4.3.3: Result collected for question 7

Finally, most people reported that finding specific instrument difficult, confirming the previous results.

Chapter 5: Conclusion

The Grouper interface that we implemented is a great tool for a player to design their own custom instrument. It provides a coherent layout and control scheme such that all the instruments are visible and in one location, preventing the player from getting lost in a menu hierarchy. We found that users found our design relatively easy to use and found the interface to be very intuitive. We believe we have succeeded in making a UI that blends into the game well. First time users found searching for the instruments to be tough at first, but found the use of the keys to be the easiest part of our design.

5.1 Errors

There are some errors that occurred during our playtesting session that may have affected our data. When we started the playtesting session, we failed to facilitate any information about the game that the player might need in order to complete the tasks assigned to them. An example of this is not informing the testers that they needed to close the grouper after making an instrument in order to move on to the next task. The system prompts the user to open the screen using the "Q" key, but never tells them to close the screen, or even the key to do so. This lack of instruction affects our time data and may have made the interface appear confusing.

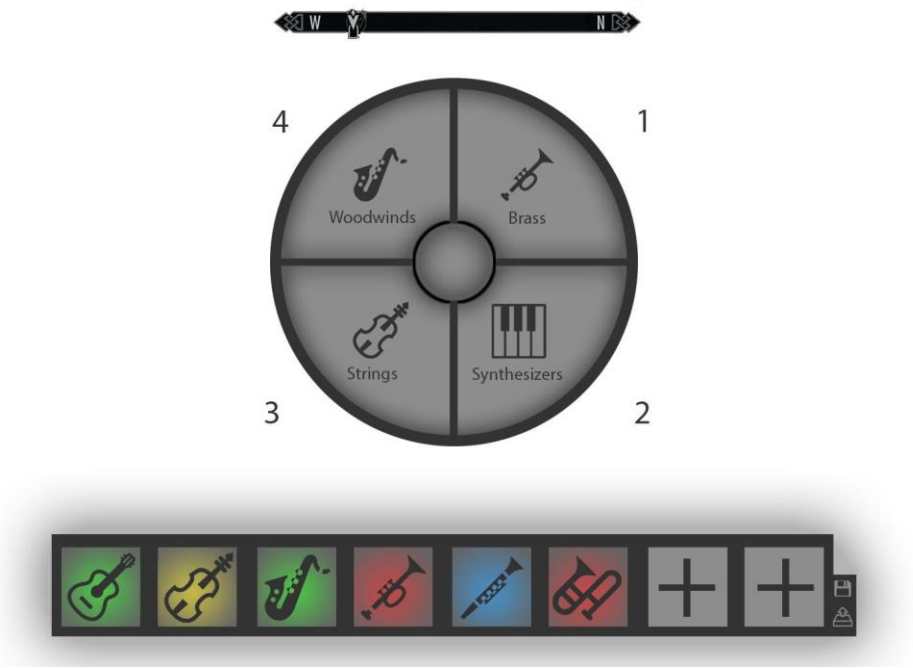
Another error occurred when some of the computers did not have the text file with the results of a playtest. We believe this is due to the player not completing the tasks all the way through as the last task generates the text file after being completed. This made the size of our test much smaller and provided us with less data to analyze.

5.2 Future Work

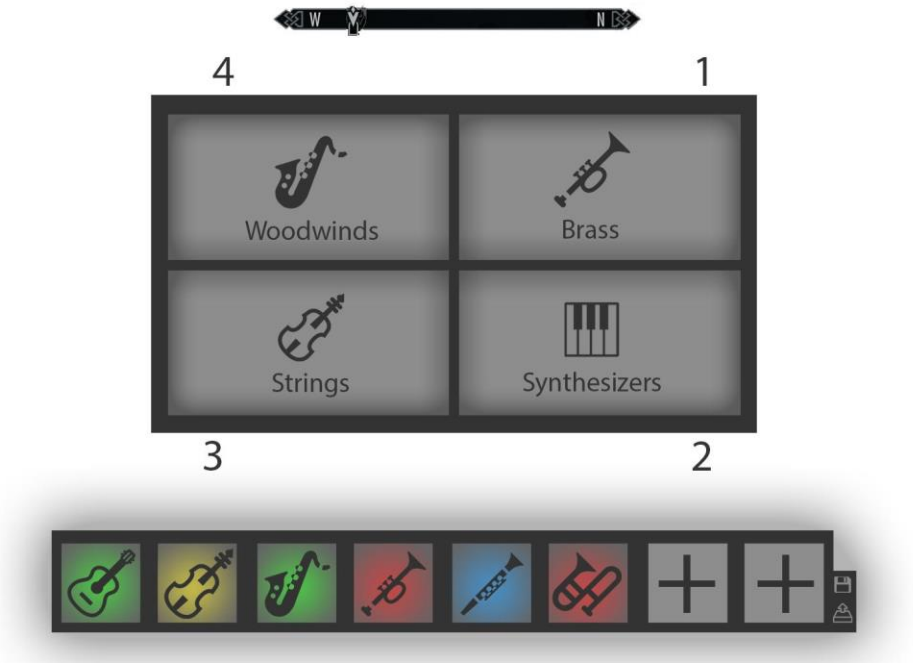
For future implementations and design, we suggest that playtest sessions should be conducted on a larger sample size in order to get more constructive feedback on the design. Clear instructions should be given to testers before using the interface so that their lack of expertise on the system does not affect the data. We would also note that in our final implementation, we have setup the interface to respond to mouse input for ease of use. In later versions of this design,

this functionality should be removed to adhere to the initial goal of only having keyboard input as the game may implement its own control surface that does not map to mouse actions.

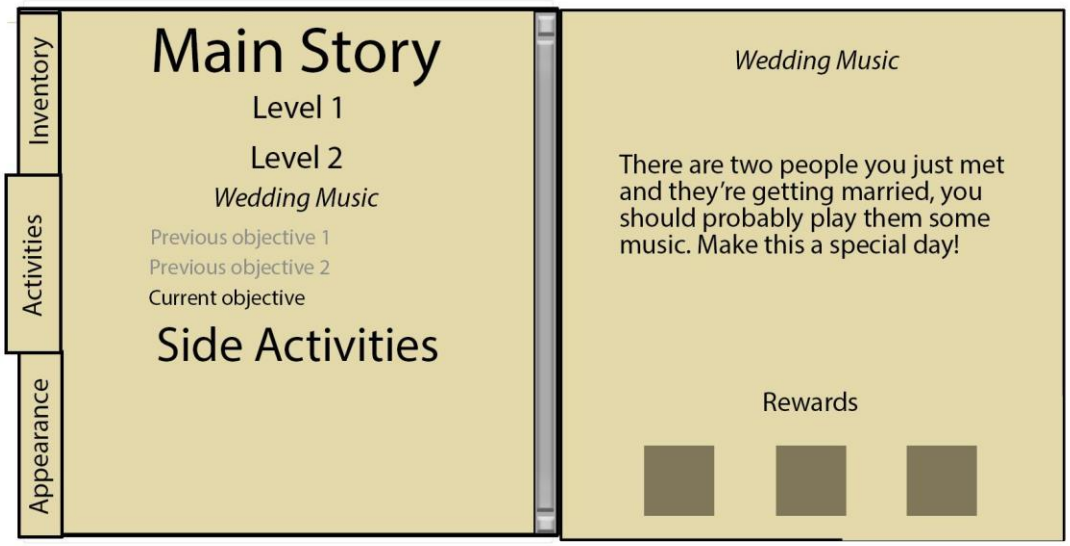
Appendix A: Initial Design



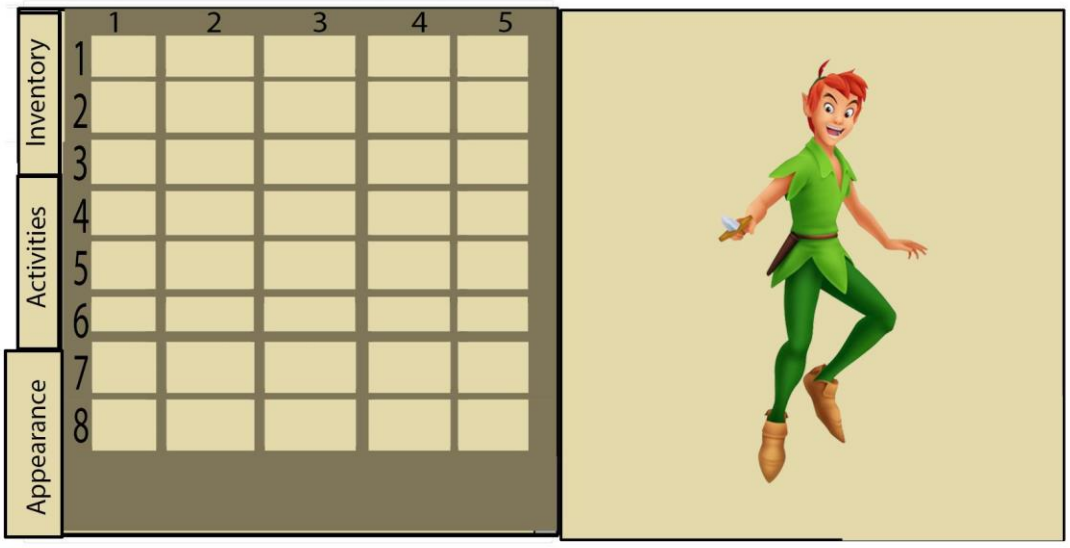
UI Mockup Radial



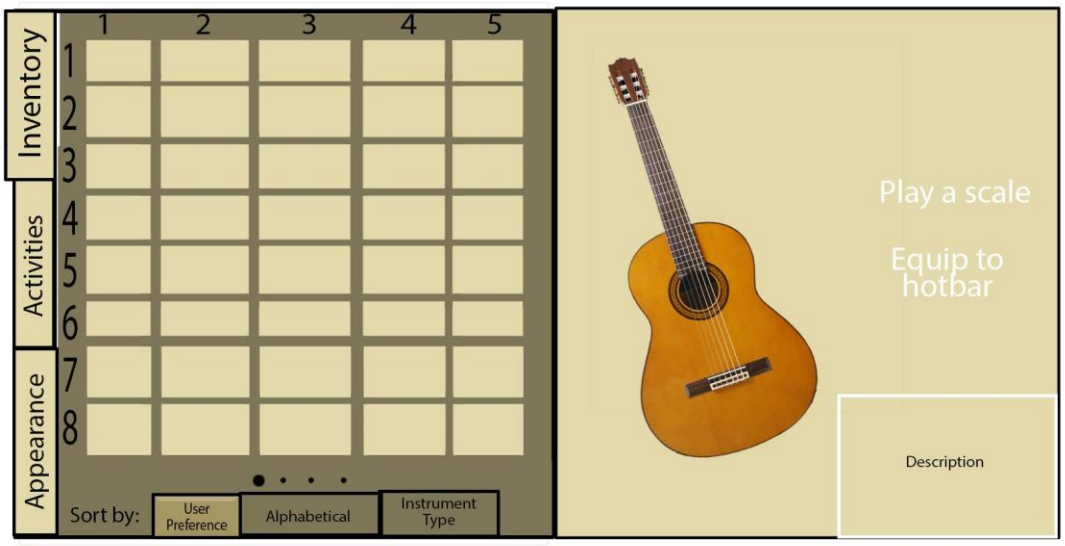
UI Mockup Rectangular



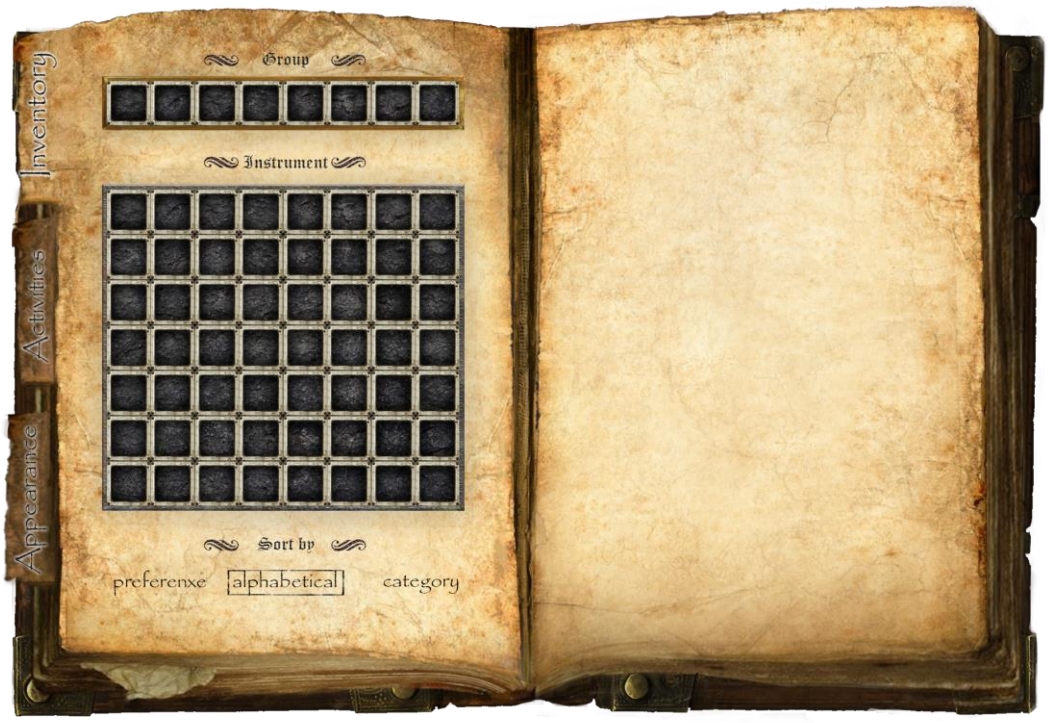
UI Activities Tab



UI Appearance Tab



UI Inventory Tab



UI Mockup Inventory Tab

Instrument Icons

Appendix B: Survey Document

Process of playtest:

- Open grouper
- Make instrument group
- Start timer, let them go
- Measures how long it takes (under the hood)
- Repeat with varying difficulty

Questions:

On a scale from 1-5:

- Rate how easy the system was to use initially
- Rate how intuitive it was to learn
- Rate how difficult it was to find specific instruments
- Rate how clear/understandable the icons were
- Rate the helpfulness of the sorting function

Multiple Choice with optional field

- What was the easiest part of this system?
    - Using the sorting function
    - Adding instruments with key presses
    - Finding specific instruments
        - Optional: Why? _____
- What was the most frustrating part of this system?
    - Using the sorting function
    - Adding instruments with key presses
    - Finding specific instruments
        - Optional: Why? _____
- Would you add anything to this system?

Appendix C: Raw Playtest Data

Result of Survey Question 1 -8

index	easy to use	intuitive to learn	how difficult to find	icon clearness	sorting function	easiest part	frustrating part
1	3	5	2	4	2	3	1
2	3	5	2	3	3	1	3
3	4	4	3	4	1	3	2
4	4	3	3	2	3	2	3
5	4	5	4	4	1	3	3
6	3	4	3	2	1	3	2
7	5	5	4	3	3	2	3
8	5	5	4	4	4	2	3
9	4	4	3	4	2	2	3
10	3	4	4	4	3	1	3
11	3	2	2	4	3		Prompt not clear
12							
13	2	2	2	1	1	1	interface design
14							
15							
16	5	5	1	5	1	3	1
17	4	4	1	3	5	2	1
18							
19							
20	2	3	3	1	1	2	3
21	3	3	3	3	1	2	1
22	4	4	3	5	4	3	1

23	5	5	4	2	3	2	3
24	4	3	2	4	4	2	3

Time spent on each task

time1	time2	time3	time4
21.60848	29.65563	31.46756	27.73489
59.09338	33.77276	19.05819	24.92207
40.54671	23.60381	43.32398	42.06399
14.21831	11.53448	42.60394	28.74563
25.69205	21.77962	20.25332	48.008
25.63716	28.48975	21.3222	21.09608
24.07728	25.294	24.79499	50.42499
119	56	14	47
107	15	18	69
55	61	40	64
18.08753	38.35416	22.17294	66.22972
72.53802	42.70922	38.57963	44.58821
16.97058	12.07032	26.37395	47.82747
91.19829	28.67665	50.65824	53.83138
11.12377	54.17061	23.35899	25.69392
14.51161	14.74939	18.7991	33.31545

Suggestions:

I didn't know to press Q after I picked the instruments. No indication. What are the key presses? I didn't know you could use keys. There was no indication.
Maybe add instructions on to just click the instruments to add because I thought it was drag and drop for the first couple of tries.
the icons were very buggy when you would hover over different ones
Add labels under the instruments

Names on the instruments icons
the instruments didn't stack and weren't remembered when I left the menu screen; add stacking, and sometimes the wrong name would display above the items

Bibliography

Fagerholt, E. Lorentzon, M. (2009). *Beyond the HUD - User Interfaces for Increased Player Immersion in FPS Games*

Kieras, D. (2009). *User Interface Design for Games,* University of Michigan

Russel, D. (2011). *Video game user interface design: Diegesis theory.* Retrieved May 2, 2016, from http://devmag.org.za/2011/02/02/video-game-user-interface-design-diegesis-theory/

Andrews, M. (2010). *Game UI Discoveries: What Players Want.* Gamasutra. Retrieved May 2, 2016, from http://www.gamasutra.com/view/feature/4286/game_ui_discoveries_what_players_.php

DellaFave, R. (2014). *Designing an RPG Inventory System That Fits: Preliminary Steps* Retrieved May 2, 2016, from http://gamedevelopment.tutsplus.com/articles/designing-an-rpg-inventory-system-that-fits-preliminary-steps--gamedev-14725

Yee, C. Ling, C. Yee, W. Nazmee, W. *GUI Design Based on Cognitive Psychology: Theoretical, Empirical and Practical Approaches.* Retrieved May 2, 2016, from http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6268617

Nijholt, A. (2015) *More Playful User Interfaces: Interfaces that Invite Social and Physical Interaction.* School of Computer Sciences, Universiti Sains Malaysia

Krieger, D. (2001) *Designing A Good Interface.* Retrieved May 2, 2016, from http://www.gamespy.com/articles/491/491801p1.html

Saunders, K. Novak, J (2012) *Game Development Essentials: Game Interface Design 2nd Edition.*