Portfolio

James Sullivan | s4529928 | DECO2800

1.0 Individual Work

1.1 Description

Throughout the project, I worked as one of two programmers on the GUI team. My significant achievements include, in order of significance:

- Implementation of the inventory menu, including all associated functionality.
- Implementation of the pause menu, including all associated functionality.
- Significant restructuring of the inventory system, to provide a centralised location for UI observers and two-way communication between the backend and frontend.
- Implementation of the equipped weapon & armour widget, including all associated functionality.
- Implementation of the wand overlay widget, including all associated functionality.
- Implementation of the speech bubble, including the text scroll-out effect and creation of the associated sound effect.
- Implementation of tooltips, including all associated functionality.
- Implementation of a StorylineManager, for central access to and control over dialogue (i.e. speech bubbles) and tooltips, assisting tutorial and cut scene sequences.
- Reimplementation of the potions overlay widget, for better efficiency and aesthetics.

1.2 Methods

1.2.1 UML Class Diagrams

Restructuring the inventory system necessitated the communication of the new structure and usage to the rest of the studio. I achieved this by writing a wiki page which includes the following class diagram, representing the new inventory architecture:

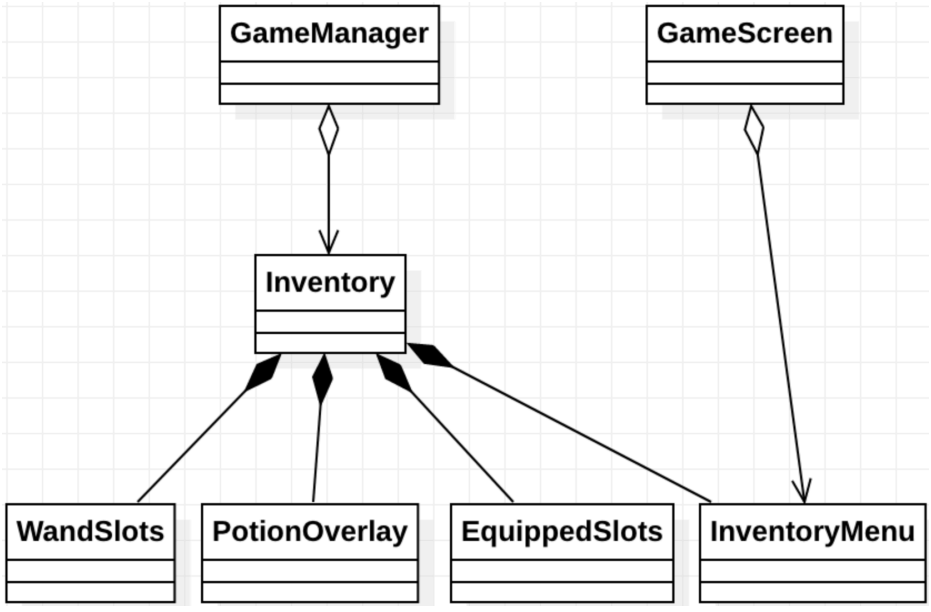

Figure-1

1.2.2 Design Patterns: Mediators

Before the restructuring illustrated in *Figure-1*, the Inventory class was merely a backend class representing the player's inventory, to which the various UI widgets had one-way access, enabling the player's inventory to be modified through their interaction. Furthermore, UI widgets were not implemented cohesively, limiting their ability for their simultaneous intercommunication without implementing complex coupling between them. Therefore, as per *Figure-1*, I converted the backend Inventory class into a mediator for the various UI widgets, which I formed cohesively into single distinct classes, packed into a single package. This granted clean and efficient two-way communication between UI widgets and the backend inventory, as well as the ability for other backend classes to access the frontend widgets through limited methods in the Inventory class, such as for force-refreshing the widgets. This design choice also provides greater flexibility in future development to add and remove inventory-related UI widgets, as there are no dependencies between UI widgets.

1.2.3 Design Patterns: Builders

Several builders were utilised to create dynamic UI for various widgets.

Firstly, the descriptions pane for the inventory menu required the menu background to be dynamically extended and retracted horizontally, without stretching the background image. To achieve this, I created a background builder to replace the static image file originally set as the background. This builder takes dimension and position parameters, and draws a custom background for the menu with these parameters, with identical appearance to the original image file used (see *Figures-2-4*). Having a builder for this purpose, rather than, for instance, multiple images, has the following benefits:

- Higher resolution backgrounds, as they are drawn with a SpriteBatch, rather than a fixed-resolution image file.
- Flexibility for future adjustments to menu background dimensions and style.
- Less images in the codebase, resulting in a lighter texture manager.

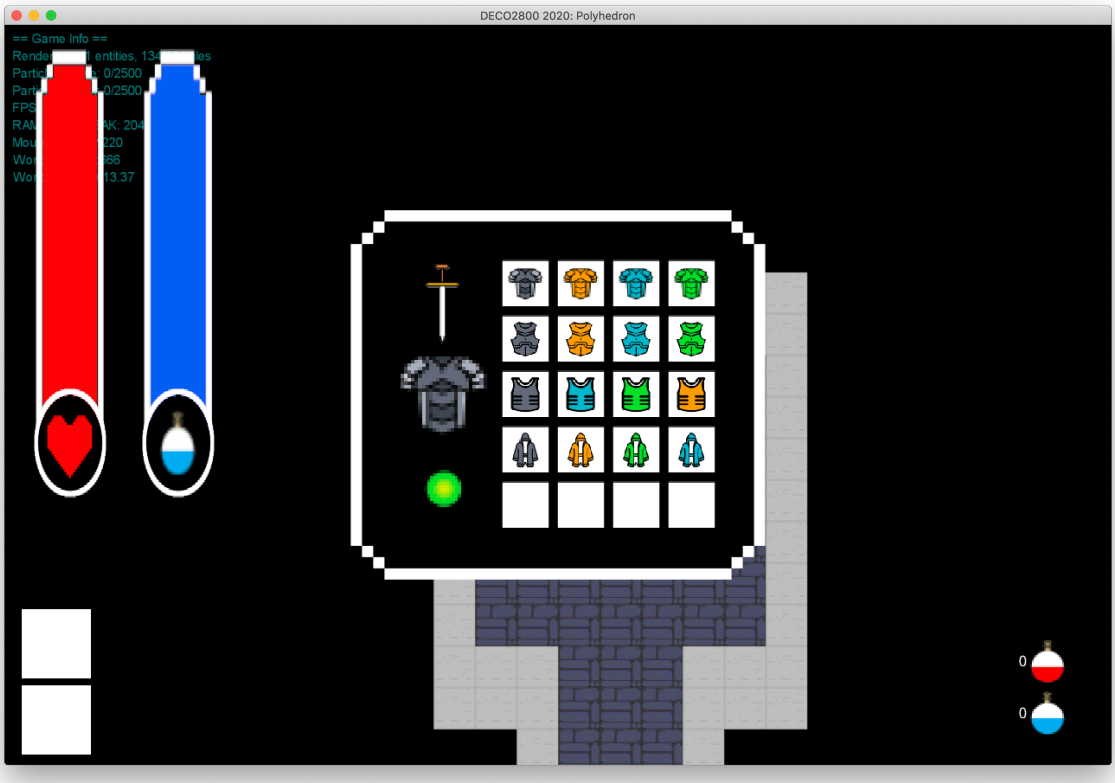

Figure-2: Builder-generated inventory menu background, retracted

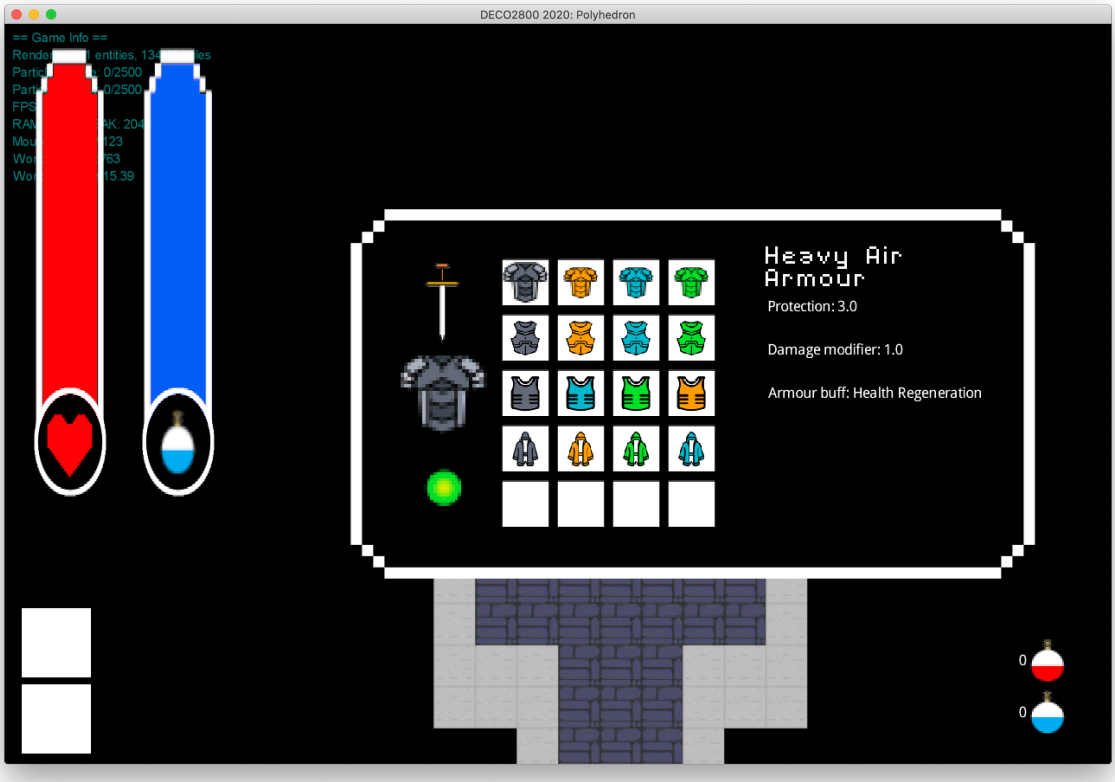

Figure-3: Builder-generated inventory menu background, extended

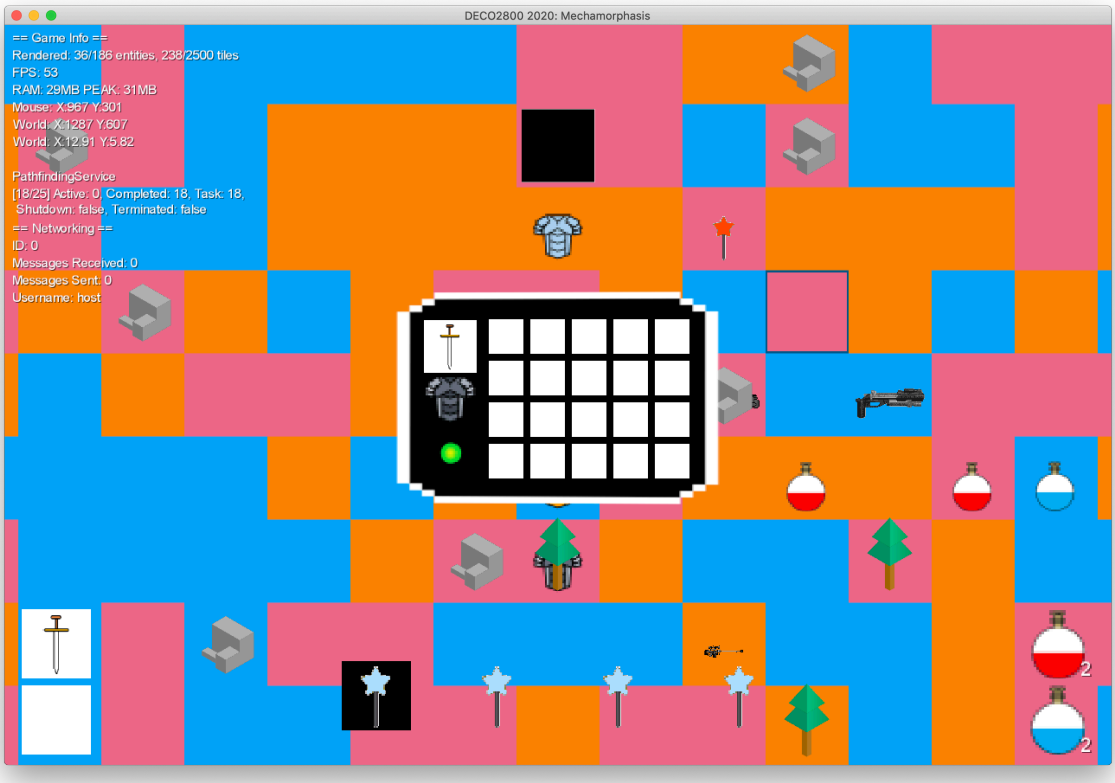

Figure-4: Original static-image background

Another builder I created was for the wands overlay, which consists of four ImageButtons representing the four wands. The images for these buttons are complex, including the wand image and rune images stacked on top in particular locations. To create these images, I created a builder which combines wand and rune drawables into a single drawable (see *Figures-5-6*).

```
    /** Helper class for creating ImageButton wand slots. This is a Drawable class which draws the Drawables ...*/
    private class WandSlotImage extends BaseDrawable {
        Drawable wand;
        Drawable slot1;
        Drawable slot2;
        Drawable slot3;

        /** Constructor for this WandSlotImage. Draws slot1, slot2 and slot3 over the top of wand. This way, the wand ..
        private WandSlotImage(String wand, String slot1, String slot2, String slot3) {
            this.wand = new TextureRegionDrawable(textureManager.getTexture(wand));
            this.slot1 = new TextureRegionDrawable(textureManager.getTexture(slot1));
            this.slot2 = new TextureRegionDrawable(textureManager.getTexture(slot2));
            this.slot3 = new TextureRegionDrawable(textureManager.getTexture(slot3));
        }

        @Override
        public void draw(Batch batch, float x, float y, float width, float height) {
            int wandSize = 80;
            int runeSize = 16;
            // draw the wand
            wand.draw(batch,  x: x-(wandSize/2f),  y: y-(wandSize/2f), wandSize, wandSize);
            // draw the runes in a vertical fashion over the top of the wand
            slot1.draw(batch,  x: x-(runeSize/2f),  y: y+5, runeSize, runeSize);
            slot2.draw(batch,  x: x-(runeSize/2f),  y: y-15, runeSize, runeSize);
            slot3.draw(batch,  x: x-(runeSize/2f),  y: y-35, runeSize, runeSize);
        }
    }
}
```

Figure-5: Code for the wand image builder: takes one wand drawable and three rune drawables and merges them into a single drawable.

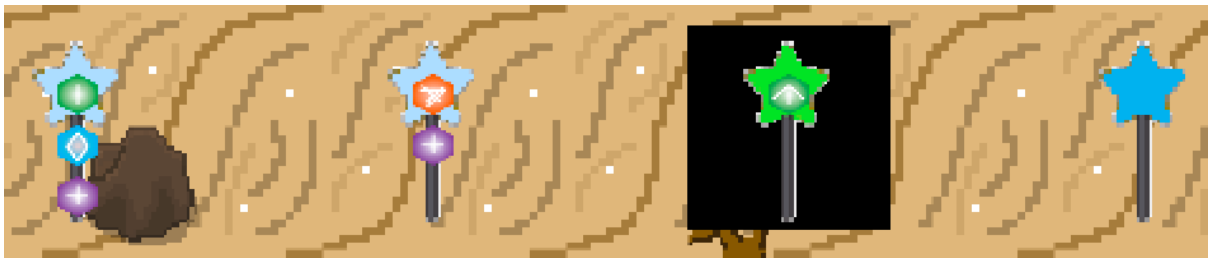

Figure-6: The wand overlay, with various quantities of runes attached to the wands, and the third wand selected.

1.2.4 Design Patterns: Composite

The pause menu is a large and complex widget, with the ability to navigate to multiple panes. To achieve this functionality, I created the PauseMenu class as a composite of all of its various panes, with functionality to hide and show particular panes depending on the state of the menu.

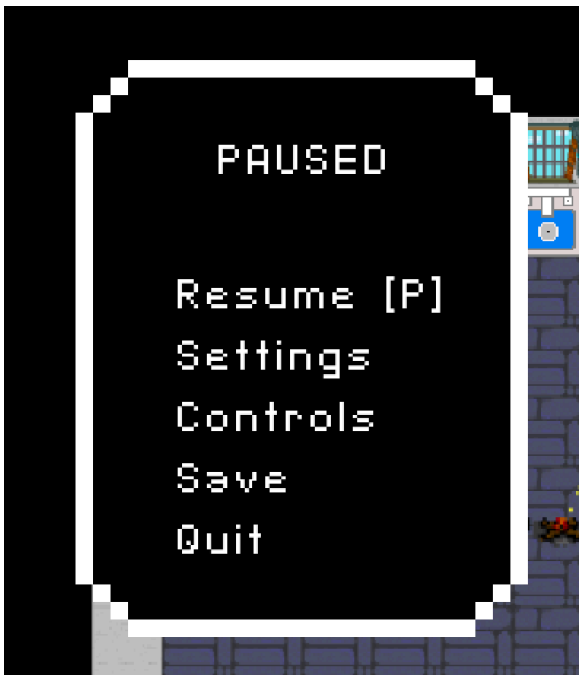

Figure-7: The home pane of the pause menu

Figure-8: The controls pane of the pause menu

1.2.5 Documentation and Collaboration

Many of the widgets I created, such as the inventory menu, tooltips and speech bubble, were designed to be taken advantage of by other programmers. For instance, the speech bubble was used by one team to enhance their cut scenes by providing character dialogue, and tooltips were used by another to indicate to the user when checkpoints are triggered. Thus, I understood that proper documentation and communication was important in my work. I did three things to attempt to achieve this:

1. Made generous use of Javadocs and code comments. I included Javadocs for all classes and methods I implemented, and always ensured to leave comments in all code blocks which were at least mildly complex.

```
 * Handles the instant-display, text-scroll-appearing effect, and associated sound of the bubble.
 */
public void onTick() {
    String textDisplaying = toBeDisplayed.peek();
    if (textDisplaying == null) {
        // nothing more to display
        isDone = true;
        return;
    }
    rootTable.setVisible(true);
    GameManager.get().getWorld().getPlayer().setInCutscene(true);
    if (scrollIndex == textDisplaying.length()) {
        // no more characters to scroll out
        return;
    }
    if (scrollIndex == 0) {
        // we just started displaying this bubble: begin playing the scroll sound
        soundId = soundManager.playMusic( id: "speech_bubble");
    }
    scrollIndex++;
    textLabel.setText(textDisplaying.substring(0, scrollIndex));
    if (scrollIndex == textDisplaying.length()) {
        // we just finished displaying all text: stop playing the scroll sound
        soundManager.stopMusic(soundId);
    }
}
```

Figure-9: A small example of my Javadocs and commenting

2. Created numerous pages on the wiki explaining the architecture and recommended use of widgets, such as for tooltips, inventory and pause menu.
3. Always implemented at least one example of a widget when first publishing it. For example, I implemented the "Press 'E' to pick up item" tooltip when first creating tooltips, and a "This speech bubble is for demonstration purposes" speech bubble when first creating speech bubbles.

1.3 Reflection

Overall, I have found this course to be extremely valuable for my individual development. I have greatly excelled in my ability to investigate and understand large and complex frameworks like LibGDX, read and understand the code of many others, utilise many functions of git, make use of design patterns, comprehend large and highly conceptual codebases, efficiently write significant portions of Java code in ways which integrate into such codebases successfully and much more.

One of the most significant of these developments was my ability to investigate and understand large and complex frameworks. Having no prior experience with LibGDX, and being (practically speaking) the sole programmer on the GUI team, I faced the unique challenge early-on of learning how to use LibGDX UI widgets with almost no pre-existing material in the codebase to learn from. I learned these widgets gradually: first learning tables, buttons and skins to implement a pause menu in sprint 1, then learning how to change cursor images and build custom drawables to style inventory-related widgets in sprint 2, then learning actor actions to fade in tooltips in sprint 3. By the time I had begun work on sprint 4, I was able to make a sophisticated speech-bubble widget, complete with animated text, in just 1.5 hours. This was something I was very proud of, as it clearly demonstrated a significant advancement in competence with LibGDX since sprint 1. I currently feel that I can create any GUI widget I can think of. This experience of diving deep into LibGDX has greatly increased my confidence in investigating large and complex frameworks.

A lesson learned the hard way, however, was proper testing philosophy. Throughout my sprints, I routinely failed to leave enough time for adequate testing, which resulted in lost marks for the team's sprints. My initial approach to testing, as I had adopted through other courses, was to write first, comment second and test last. Until now, I have been successful using this approach, but I have since learned that writing tests for large codebases can be far more difficult than for small ones. During my sprints, I had routinely left myself just half a day to write tests, and, whenever I would attempt them, I found myself running into many issues, panicking, and giving up. Taking the time to properly learn testing tools such as Mockito, and writing tests first, is a big lesson I will take away from the course.

Another of these lessons is leaving adequate time for refinement. For most sprints, I had undertaken significant responsibilities in creating widgets. I also, during the creation of these widgets, occasionally decided to spend some more time implementing additional functionality which I believed could enhance the user experience. The combination of these strategies resulted in little time left for refinement of these widgets at the end of the sprint, and I was disappointed to submit work which appeared unfinished relative to the designs on which they were based (see *Figures-10-13*). In future, I plan to adopt the

approach of "quality over quantity", working within the initial design and avoiding the adoption of any extra responsibility which may hinder the quality of my work.

Figure-10: The intended design style for the pause menu controls pane.

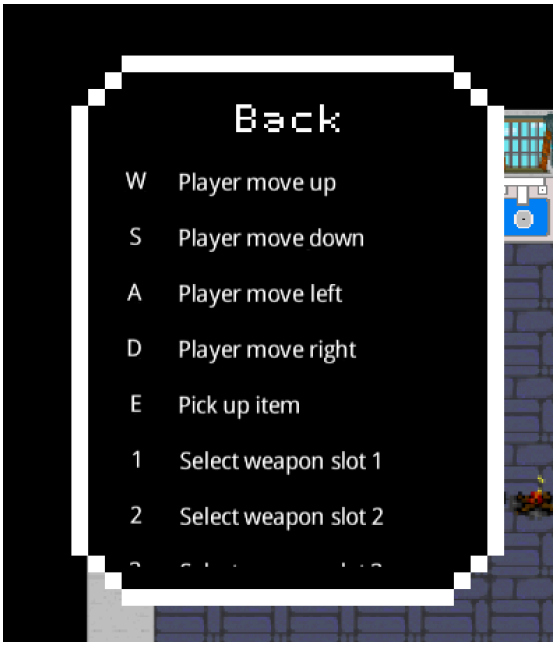

Figure-11: The implemented design style for the pause menu controls pane.

Figure-12: The intended design style for the equipped weapons & armour slot.

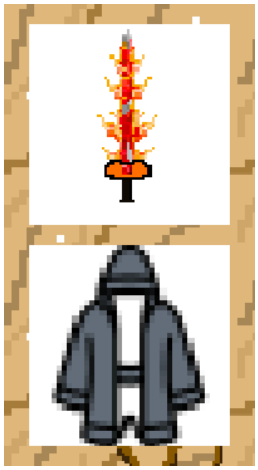

Figure-13: The implemented design style for the equipped weapons & armour slot.

2.0 Team Work

2.1 Description

My team consisted of two designers and two programmers, and were responsible for the GUI of the game, including splash screens, menus, overlays and pop-ups. The group constructed feature tickets for each sprint during studio sessions, where we verbally determined our task allocations. These were usually confirmed shortly afterwards via online messages on our Discord server, which the team used for online communication, making use of individual text channels for *coding*, *design*, *menu content* and *general* communications. When deemed necessary, we also held in-person meetings on campus after studio sessions, to further clarify team aims and individual responsibilities. We found task allocations to be mostly effortless, as our responsibilities could be divided such that they could be fulfilled independently of each other.

2.2 Reflection

It was quite apparent by the end of the semester, to the studio tutor as well as members from other teams, that I was not content with some of my team members, particularly, the other programmer, who I will refer to as W. The little code he did write was of extremely poor quality, and, due to a severe lack of communication and collaboration on his part, resulted in our work for sprint 3 being overall much worse than if he had not done anything at all.

Ultimately, I produced by far the most work for the team of all our team members, and wrote the vast majority of the code we pushed. However, since the PAF system is fundamentally based on here-say, and other team members insisted on providing equal distributions in their PAFs, I did not receive individual grade compensation in proportion to my efforts, which made me consistently frustrated with W and the course.

This experience has taught me the great importance of team member accountability. As a relatively reserved individual by temperament, I was initially resistant to confront W about his lack of work, and although I attempted to passively communicate this issue to him through Discord, his lack of online availability in the chat made this strategy ineffective. Therefore, I believe a clear mistake we made as a team was failing to consistently meet in person at our scheduled time after the studio, to discuss team progress. Had we accomplished this, I believe we would have been able to communicate this issue more effectively to W.

Another such opportunity we missed was proper and consistent use of milestones and individual tasks on the feature ticket. Our team's feature tickets for most of the sprints did not have milestones, and merely included a checklist of features and sub-features the team intended to implement. In hindsight, I believe that if we listed milestones and the

team members responsible for each of these items, the productivity of all team members would have increased, and we could have had stronger justification to more firmly hold W accountable for his work.

Another area for improvement for our team was active communication amongst all team members. Our team's designers were not proficient with programming, and our programmers did not want to partake in any part of the design process. As a result, communications between designers and programmers gradually broke down over the semester, leading to designers constructing prototypes which were difficult to implement in code. One example of this is the health and mana bars, which had rounded tops, increasing the difficulty of implementing the filling for the bars, as a simple stretchable rectangle could not be used. As a programmer in future teams, I will make a better attempt to communicate more effectively with non-programmers, such as by partaking in the design process.