CS 429 - Really Cool Game

Nathan Handler

Kyle Nusbaum

Jonathan Albrecht

Andrew Roth

Ian Canaday

Matt Verzak

Table of Contents

1. Project Description 3

2. Process Followed by Team 4

3. Requirements & Specifications 6

4. Architecture & Design 8

5. Future Plans 11

1. Project Description

Our project was to create an action RPG video game. The goal of the game is to explore the world searching for objects, fighting various enemies during the process. Once all of the objects are collected, a final boss will appear, which must be defeated to complete the game. We pulled a lot of inspiration for the game from some of the original Legend of Zelda and Super Mario video games.

The game was created using Python 2.7.6 and the `Pygame` library. This allowed us to make the game cross-platform; however, our target platform was Debian GNU/Linux. Our main goal behind this project was to learn the basics of developing a game since the university does not offer a course dedicated to this topic.

2. Process Followed by Team

2.1 Issues of Iterative Development

The biggest issue we experienced with iterative development was that we assigned multiple tasks that depended on each other to the same milestone. This meant that if one of those tasks took longer than expected to complete, then all of the other tasks would also get behind schedule. This typically resulted in all of the dependent tasks getting pushed to later milestones.

A second issue we faced was that we initially had no idea how long certain tasks would take to accomplish. This meant that many of our earlier time estimates were inaccurate, and it caused us to shuffle around the tasks assigned to certain milestones. This issue resolved itself as we became more familiar with Pygame.

2.2 Refactoring

Refactoring was continuously being performed throughout the process. Initial refactoring took place during code review, where a second developer would attempt to refactor the proposed branch as much as possible and remove any code smells they might catch. Further refactoring then took place throughout all iterations of the project. Sometimes, this was done through explicit issues and commits. Other times, it was simply done while working on a section of related code.

2.3 Testing

Testing was a continuous process throughout the process. One place testing took place was during the code review process. When a developer reviews a branch, they test it on their local machine to ensure that is works properly and that there are no noticeable regressions. We also had a set of automated unit tests that helped us test the functionality of our code in a consistent and automated fashion. In addition to this, we had a set of Lettuce tests for testing the behavior of the code. These tests allowed us to use a plain English description of the behavior of a section of code, and then have the tests run and produce a nice

clear output. Finally, we had a script providing code coverage metrics to help guide us in writing our tests and verify that we were testing all of our code.

2.4 Collaborative Development

In Extreme Programming, it is typical to pair program with another developer as much as possible. As students, this proved very difficult in CS 427 due to all of us having very different work schedules. As a result, we decided to drop pair programming in favor of code review. This means that all development work is performed in separate git branches. When the changes are functioning properly and ready to be merged into master, a pull request is opened. At this point, one or more other developers review the code changes and test out the branch. This provides us with the benefit of having multiple sets of eyes reviewing all code, but does not require us all to have the same work schedule.

3. Requirements & Specifications

We successfully implemented the majority of our user stories for this project. At this point, we have a functioning game where the player can move around the world collecting items and killing enemies before battling the boss to win the game.

3.1 Collision Detection

The TMX map format that we are using has support for multiple layers. This allows us to put things like grass that the characters is able to walk on in a background layer and keep rocks and walls in a separate foreground layer. When doing collision detection between the character and a new square, we simply check if there is anything present on the map in the foreground layer. Collision between the player and enemies or bullets and enemies is done with `Pygame`'s built-in collision detection functionality.

3.2 Maps

We decided to use a free application called Tiled to create the maps for our game. It uses an XML-like TMX format, and supports multiple layers. This allowed us to take advantage of an open source TMX library for reading the maps into our `Pygame` application. In order to be able to store information about enemies and items, we created JSON files that stored their coordinates on the map. We also use a special JSON file for specifying how the different maps connect to each other. This is what allows the player to move from one map to the next.

3.3 Saving and Loading

When saving the game, we make a copy of the JSON files uses to start a new game, and place these in a new directory. We automatically update the JSON file for a map when the player leaves that particular map. By using this approach, we are able to use nearly identical code for starting a new game and loading a game. Saving the game can be done with simple JSON and OS file manipulation libraries.

3.4 Menus

All menus in the game use nearly identical code. We simply create a new screen on top of any previously existing screen. Menu entries are drawn as regular strings on the screen, and we color the selected entry a different color. Scrolling up or down using the keyboard triggers an event that we listen for, and we update the color of the menu entries to reflect the newly selected option.

Most of our menus simply involve selecting one of many options. However, in the case of starting a new game, we needed a way to also acquire a name for the game. For this, we used the `Tkinter` library, as it has pre-built widgets for text input and it proved trivial to integrate with our existing code.

3.5 Enemies and Boss

Enemies share a lot of their code with the player. As a result, both classes extend the `CreatureSprite` class. This allows our `EnemySprite` class to focus on the enemy-specific code. Our enemies currently wander randomly around the screen. If they collide with the player, as detected using `Pygame`'s collision detection code, the player is bounced backwards and damaged. We also have a `ShooterSprite` class that extends `EnemySprite`. This special enemy will sometimes fire at the player if it is lined up horizontally or vertically. Finally, the `BossSprite` class is simply an extension of the `ShooterSprite`, but it will occasionally move two spaces instead of one at a time, and it fires three bullets instead of one. The boss also has a button next to it that kills the boss when pressed (similar to classic Mario games). By heavily utilizing inheritance, it made it very easy to have different types of enemies and test them.

4. Architecture & Design

4.1 UML Diagram

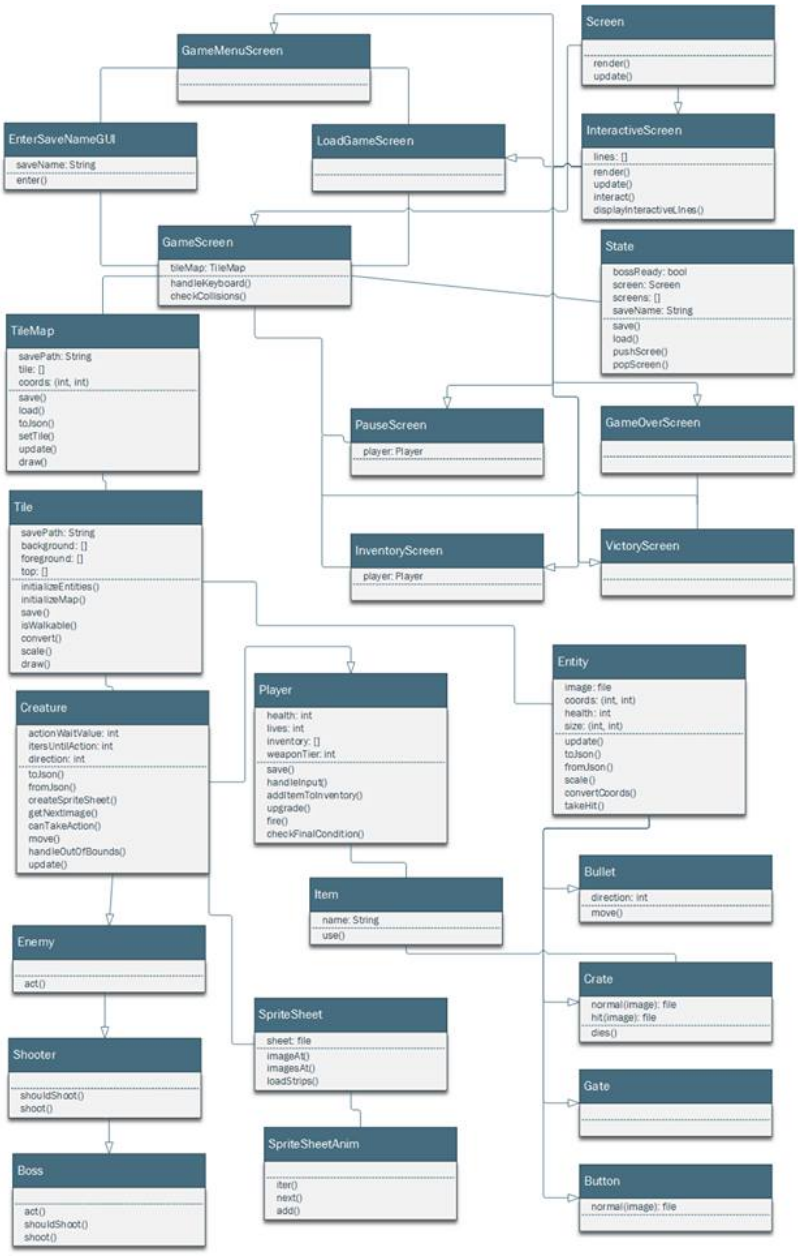

Figure 4.1a

4.2 Architecture of System

Pygame operates on an event-driven system, but it also has a concept of clock ticks. This allows us to react to user input, such as a key press, as it happens, but also cause enemies to move around the screen on their own. The game itself runs in an infinite loop that is spawned by `main.py`.

We rely pretty heavily on class inheritance in this game. One of our basic classes is our `EntitySprite` class. This class merely provides the logic for loading and saving itself, updating its position on the screen, and taking a hit. From here, our `CreatureSprite` class extends it to provide code for loading images from a sprite sheet, actually moving, and collision and boundary detection. This class is then further extended by our `PlayerSprite` and `EnemySprite` classes. The player knows how to interact with items, receive movement instructions from the keyboard, and shoot a gun. The enemy knows its specific movement logic. The enemy is further extended by a `ShooterSprite` which is extended by a `BossSprite`; both of these classes are just special types of enemies that know how to move and attack in different ways. The full class structure of our project can be seen in Figure 4.1a.

4.3 How Choice of Framework Influenced Design of System

`Pygame` operates by drawing objects and sprites on screens. As a result, when deciding how to abstract out code, we often did so by moving screens and sprites to their own classes. This had the benefit of making our code very modular and reusable, but it had the disadvantage of requiring a lot of imports from many different files to perform even the most simple of tasks. This made tracing and debugging problems take a bit more time.

5. Future Plans

5.1 Going Forward

We currently have no plans to continue development of this project after the class ends. This project was meant to be a learning experience for us, and it allowed us to experiment with `Pygame`. We will make the source code, documentation, and all associated resources available to the public under a XXX license, which will allow any interested parties to continue development. We will also probably use the skills and knowledge gained from this project to help us develop additional games in the future.

5.2 Personal Reflections

5.2.1 Nathan Handler

This was my first time developing a game. I quickly learned that it has several challenges not present in traditional application development. First, the user interface plays a very important role. When developing a tool, functionality tends to be more important than a pretty interface. With a game, an ugly or unusable interface makes the game useless. Second, a game is a continuously running program. This means that the state is constantly changing, so all algorithms need to be able to be run quickly enough to prevent noticeable lag in the game. Overall, this was a positive experience for me. While I do not foresee myself doing much game development in the future, I am still glad to be able to say that I have developed a game on my own.

5.2.2 Kyle Nusbaum

This was my first major project using python, but I was familiar with SDL and OpenGL before we began working on this project. Using a completely dynamic language like python presented certain challenges you don't find in the more static languages I was used to working with, and required a certain amount of discipline in order to keep the code sane, such as keeping class members documented within the class definition, even though they could be added at runtime. Sometimes a grep was necessary to see

all of the members I'd added to a class throughout the lifetime of the program, which is not a very good thing. Forcing myself to modify a class definition when doing something like this, even though python doesn't require it was something I wasn't used to. On the other hand, the dynamic nature of the language allowed us to do some neat things like creating mock classes for testing, since duck-typing allowed us to just implement stubs of the necessary methods.

I'd programmed a few small games before in C++, and was worried when we started about our ability to write a game in an interpreted language with the necessary performance, but the pygame library turned out to be plenty fast for our purposes, and by consistently refactoring our code, we were able to create nice abstractions that allowed us to easily separate our code, and run only what needed to run at any given time. Working with libraries and targeting different platforms are some of the more frustrating parts of any project I've worked on, and this was no exception. Fortunately, most of the libraries were well documented, but where we encountered those that weren't, we ended up with some weird display issues across different platforms.

I don't plan on working with pygame again, but the experience working with a team on a project from the ground up was great fun, and very educational. I'd never had the opportunity to work with as many people on a project that was entirely our creation before, and I'd like to do it again in the future.

5.2.3 Jonathan Albrecht

This was my first time developing anything that had to do with graphics in any language. One of the challenges that I noticed was that making the game appear to be smooth so that the player can actually tell what was going on. This meant that we had to be careful with how we handled user input. For instance we had to adjust the speed of the character if the user was holding down the movement keys instead of tapping them. The other challenge that we faced during the project is coming up with an idea for the game that would actually be interesting for someone to play.

5.2.4 Andrew Roth

Being traditionally a backend systems developer, I had never developed something so UI-intensive before. You have to think very carefully about how things are displayed on the screen, or the user will find the game unplayable. To avoid working with a camera and offscreen elements, we decided for our game to exist on a grid of tiles, which each tile being comprised of a grid of squares. This lead to many challenges, such as how to make entities slide nicely between squares, and how to transition properly between tiles. Another challenge we faced was determining how to properly pass around all the data we needed, since many of our screens relied on data from other screens. Overall, while the project would certainly need more polish before a public release, I am proud of what we created, and I would certainly work on a similar project with this group again.

5.2.5 Ian Canaday

I too have never worked on a game before. This has been the most UI intensive project I have been apart of. It has also been the the project with the most user interaction that I have ever worked on. The vast majority of everything I have done up to this point has been backend, data work. The few times I have done any sort of user input or UI have been very basic. Making a clean, usable UI that a user wants to actually use has been an important experience for me that I am sure we will be helpful in my future work. I have learned a great deal about taking in user input and applying it to an interactive program and then properly displaying all of the actions. This seems like an invaluable skill as most programs require a certain degree of interaction.

5.2.6 Matt Verzak

This is the first game that I have worked on. Like many of the other team members, I had little experience working with a UI before this project. Most of the work I did on this project ended up involving the UI. I wrote the code for most of the menus that players use while playing the game such as the inventory menu. I also worked on the main menu as well as menus to save and load the game. Writing these involved using two UI libraries (`Pygame` and `Tkinter`). It has been an important experience for me to finally write a

UI that wasn’t riddled with bugs and unpleasant to use. This has been important for me because I may write a program for use by the public and if I do they will want solid interactive menus to use with the program.