Earl Haig Secondary School

HAMLET HERO

SOFTWARE ARCHITECTURE DOCUMENT (SAD)

Content Owner: Fei Dong and Anya Pechkina

DOCUMENT NUMBER:	RELEASE/REVISION:	RELEASE/REVISION DATE:
1	1	May 8th, 2015
1	2	May 26th, 2015

1. DOCUMENTATION ROADMAP

1.1. Document Management and Configuration Control Information

Revision Number: 2

Revision Release Date: May 26th, 2015

1.2. Purpose and Scope of SAD

This SAD is specific to the software architecture of the game Hamlet Hero. All information regarding the game's software architecture could be found in this document.

1.3. How the SAD is Organized

This SAD is organized into the following sections:

Section 1: Document Roadmap provides the roadmap and document overview to its intended audience.

Section 2: Architecture Background explains why the architecture is what it is. It provides a system overview, and establishes the context and goals for the development.

Section 3: Views specify the software architecture and its views, as well as the views' relationship with each other.

Section 4: Directory shows the index and glossary.

1.4. Stakeholder Representation

Manager

The manager is represented by the teacher, and their role is to oversee the operations and consider components such as time and financial management. Their viewpoint is to see the overall scope of the development and keep the rest of the employees on task. They are concerned with the final objective of the project and the productivity and interactions between team members. They are concerned with all aspects of the program, from its context, goals and requirements, to its organizational structure, and how it is deployed and used by the users. Their assessment is documented through the final assessment for the students through a grade mark. The views of their interest are the development, logical, process, and user-case view.

Architectural Developers

The developers are represented by the students working on the project, and are the team members who design the program. They are concerned with the flow of the program and how it makes logistical sense. They outline how the system will work and design solutions to the problems they face. They are the creative minds that solve obstacles and works towards development an end project to present to the acquirers and users. They are concerned primarily with the structure of the program and how well-organized it is in the development, logical, process, and user-case view.

Programmers

The programmers are also represented by the students working on the project, and are concerned with executing the plans of the manager and architectural developers. They work out the individual chinks and problems with the program as well as write the code that would execute as the final project. They are concerned with how each part of the program works and interacts with each other in order to operate, and the development and process view.

Users

The users are represented by the rest of the student body in the school, and are external from the team and are the people who actually use the program in the end. They are concerned with the playability of the game, its cost, and how fast it can be delivered. They may give feedbacks to maintainers in order to improve the game, but largely they are not concerned with the software programming of the game. They are concerned about the usability and deployment of the program, and the logical and user-case view.

Maintainers

The maintainers are represented by the students assigned this project, and their job is to listen to feedback from the users and use them to enhance and edit the game program. Their concern is with improving usability and response from the end users, and making sure that the code is sustainable and continues to work through revisions. They are concerned with each part of the

program and how they interact, so that they may detect if there's a problem to be fixed and work on it. They are also concerned about the development, process, and user-case view.

2. ARCHITECTURE BACKGROUND

2.1. Problem Background

System Overview and Goals

The general function of this program is to run the game Hamlet Hero successfully as planned by the development team. It is a game that includes an intro and outro, as well as several mini-games that gain points that culminate until the final boss level. It includes graphics and sounds developed by the art direction team.

The goal of this game is to present it to the manager team (teacher) and the users. The end-users should be able to use the program with full understanding of the instructions and does not produce any errors.

The program is coded using Python34 and PyGame.

Significant Driving Requirements

The requirements assigned by the manager team are to create a game which includes beginning and ending dialogue and animation, scoring process, sounds and graphics. It is required to have two or more levels in a group project which has a common theme.

The code should be written for maximum efficiency, using methods that the students learnt in class. Some examples include list and dictionary manipulation, input/output file, sprites, and collisions. In group projects, the work should be divided equally between both developer/programmer so that each person's ability can be properly measured. One person's level should not affect the others.

2.2. Solution Background

Architectural Approaches

The general approach of this game is to split it off into separate Python files, each including several major functions and sprites around a central theme, in order to better organize the large amount of code needed. This is efficient for changing the code as one part does not affect the other, and it will be easier to put several people's work together at the end.

The outline of the code is as followed: one Python file is created to call upon the five mini-games, and the boss level. Each of these elements is assigned their own function or functions, but each can be executed by calling a single function. This allows the separate developers/programmers to work on their own part of the code and not have any compatibility issues when placed together.

When the user uses the program, it will start with the intro animation which showcases the story. There's a monster ravaging the town, and the user has volunteered to be a champion to slay the monster. To do so, they must gather supplies around the neighboring area. The user will then in a random order encounter all five mini-games, and then chose whether or not they wish to face the final battle. If they do not, they could play the round of mini-games again at a higher level of difficulty to gain more supplies, but the village might suffer more hits. If they do, they can face the boss, using their scores from previous mini-games as attributes such as attack, defense, and health. Whether or not the user wins or loses, an end animation plays out that ends the program.

Analysis Results

One version of a mini-game had been previously tested against end-users, and was found to be effective. Some criticisms documented include the need to increase difficulty of the game and organize the graphics better so that there's a visual cue for the end-users when they've accomplished a task.

Requirements Coverage

All of the requirements are met by the game Hamlet Hero. The game starts and ends using dialogue between the user and the townspeople through an animation. The game often uses sprite to represent the user, which interacts with other elements in the code through collision. The mini-games also increase in difficulty to create different levels as the user progresses, and give the user a score. The common theme of all of the mini-games is that they all required the user to deal with some type of element, such as metal, wool, berries, lumber, and honey, which represent their attributes such as attack, defense, and health when they face the boss level.

3. VIEWS

3.1. Logical View

This view describes the logical ordering of the different functions and elements of a system in this program and is concerned with how it functions and provides to the users. It is from the perspective of the manager, the architectural developers, and the users.

Figure 1 shows several classes and how they interact. The Player class determines whether it is the user (Me) or the computer generated avatar (Boss). Each type of Player has attributes such as attack, defense, health, and criticalAttack, as well as executable methods such as actionAttack(), actionDefense(), and healthChange(). These two Players are mostly shown at the end of the program during the boss level (see Figure 4 and 5). Further explanations on how the Mini-Game class fits into the Me class, refer to Section 3.2, using Figure 3.

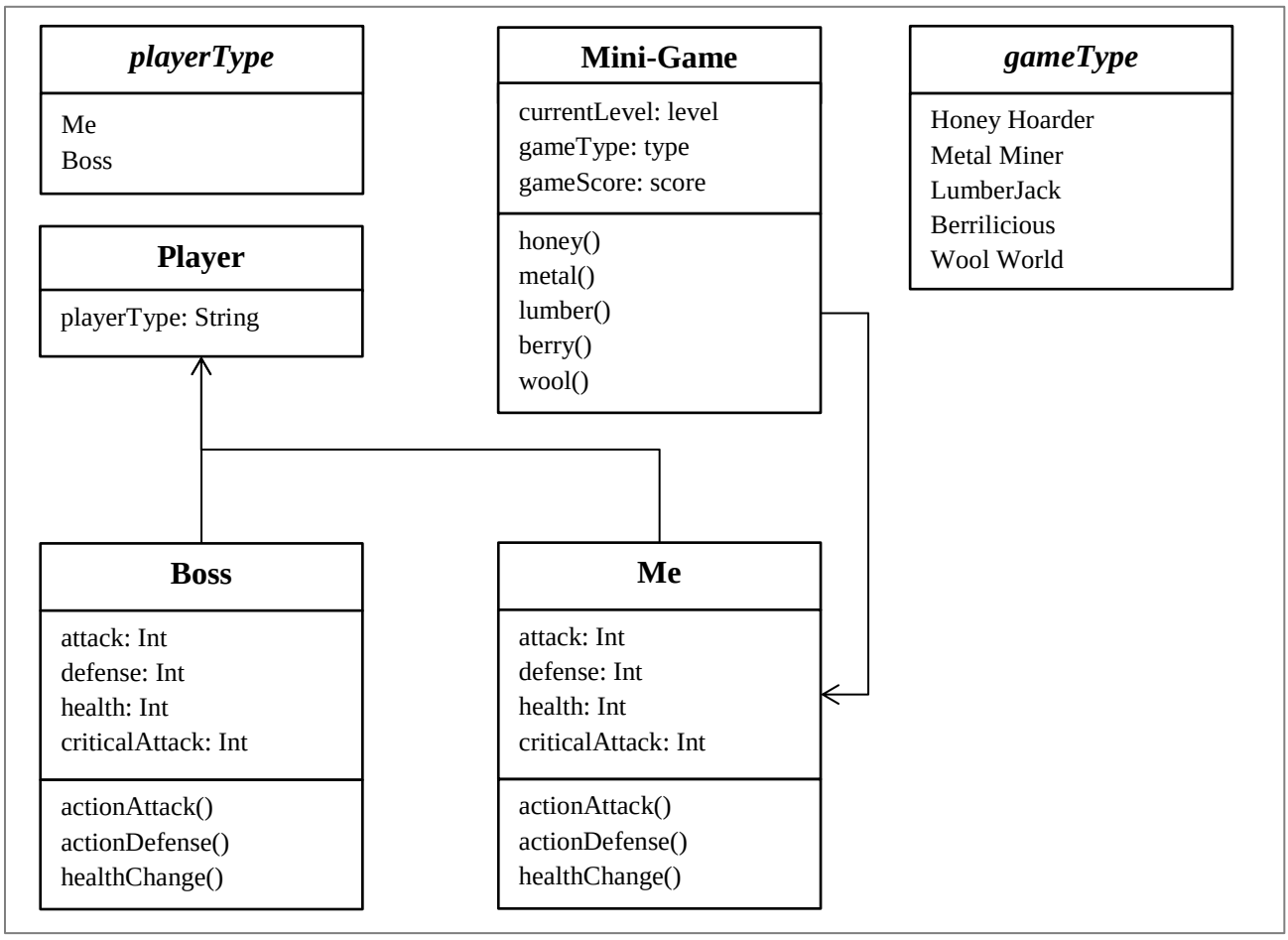

Figure 1: Class diagram showing different classes in the overall game and their attributes and functions

3.2. Development View

This view is concerned with the dependencies between different parts of the system, and is the concern of the manager, the architectural developer, the programmer, and the maintainer. Figure 2 shows how the different Python files are imported into the main Python file to be executed.

Figure 2 is shows how each of the Python files written by the programmers, which each consists of its own functions, are imported into the main game.py file in order to execute the game.

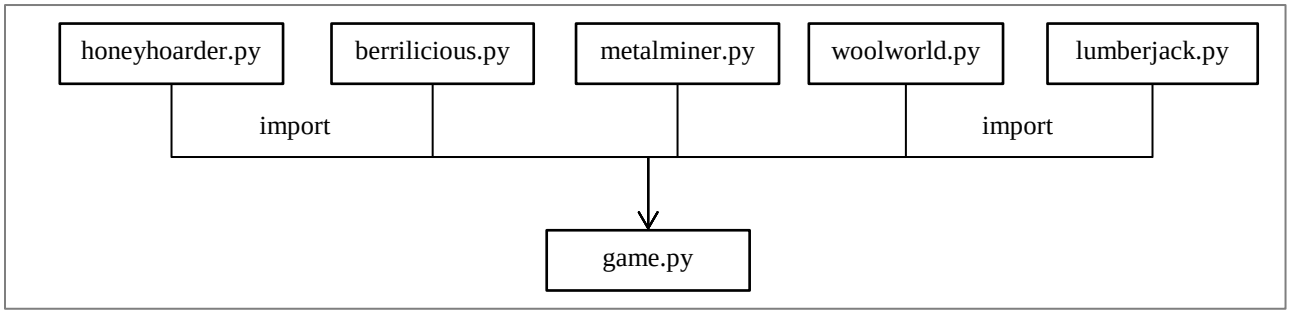

Figure 2: Package diagram shows how the different Python files are imported into the main Python file.

Figure 3 shows how each of the scores attained from the individual mini-games are taken into account to when deciding on the attribute of the Player (see Figure 1) such as attack, defense, health, and criticalAttack. Note that these attributes are only assigned to the playerType Me, as playerType Boss will receive different attributes that are hardcoded and unchangeable.

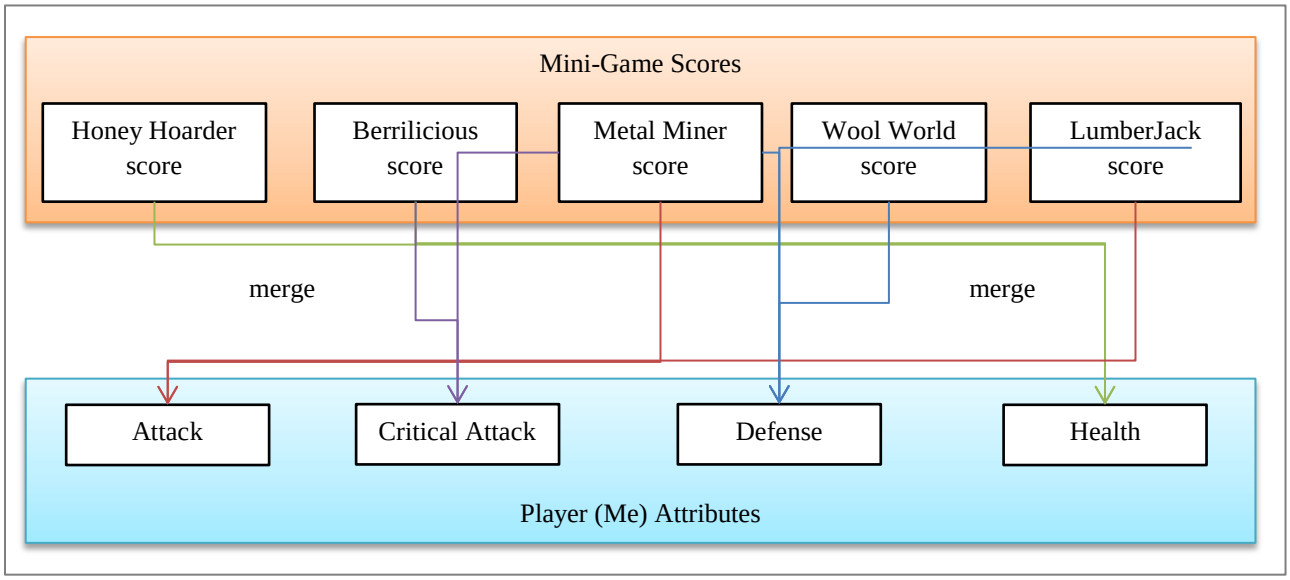

Figure 3: Package diagram shows how scores from mini-games are merged into attributes for the Player (Me) in the boss level.

3.3. Process View

This view deals with how dynamic the system is and focuses on concurrency, performance, scalability, and is from the perspective of the managers, architectural developers, programmers, and the maintainers.

Figure 4 shows how the entire game is lay out and the flow of the activities from start to end of the program. It shows all of the decision points and different ending that the user can encounter.

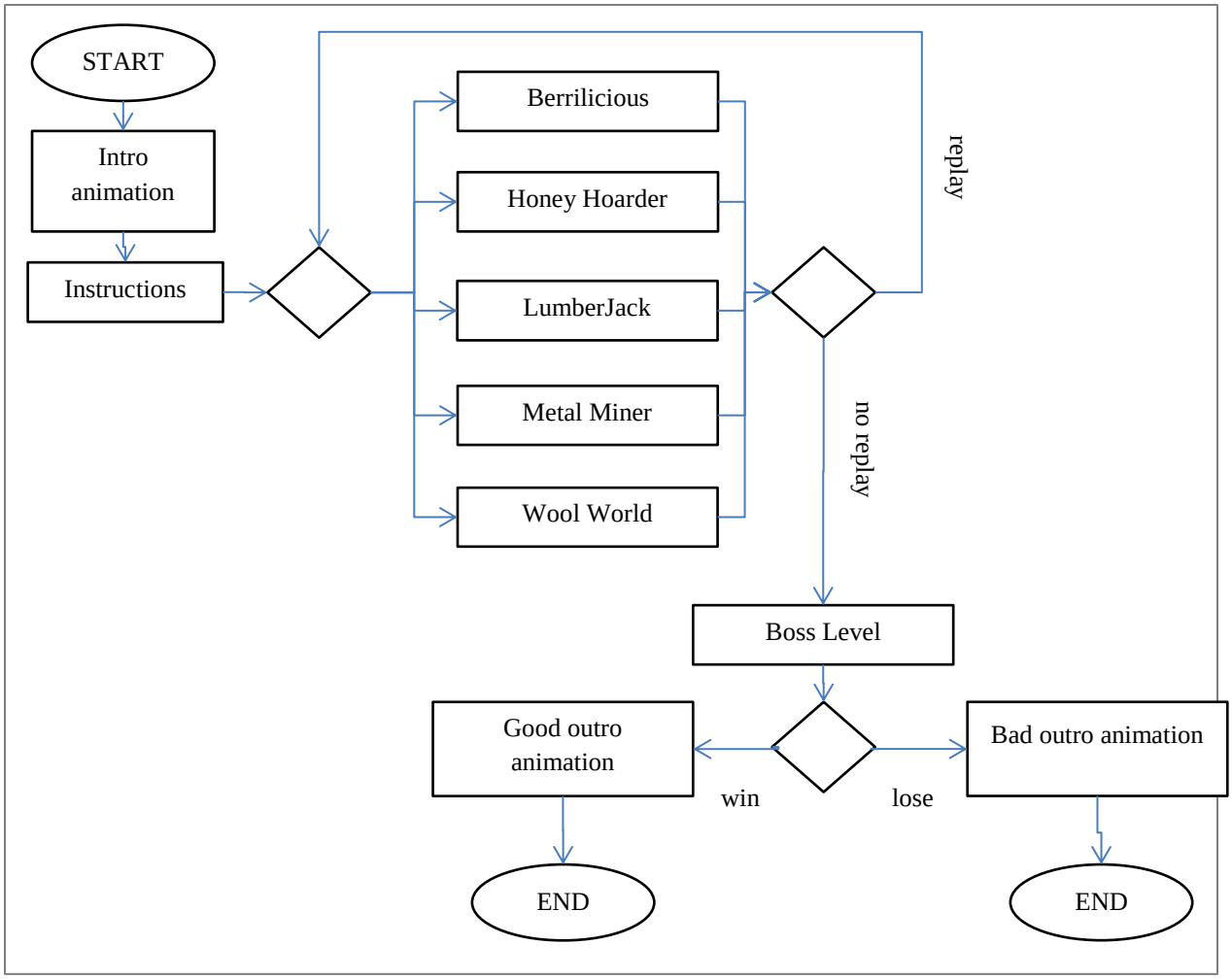

Figure 4: Activity diagram showing the flow of the game.

This program is shown to be dependent on a few major components, but is still flexible in terms of being continuous developed by the maintainers and programmers. A scalable aspect of the program is that more mini-games can be added simply to the program thanks to its use of importing separate Python files into the main Python file (see Figure 2)

3.4. User-Case View

This view describes the type of scenarios that could be encountered by the user that are significant to the function of the program. It is from the perspective of the users and is being taken into considering in all stages of development.

Figure 5 is a user-case diagram that demonstrates the scenarios that the user could encounter when using the program.

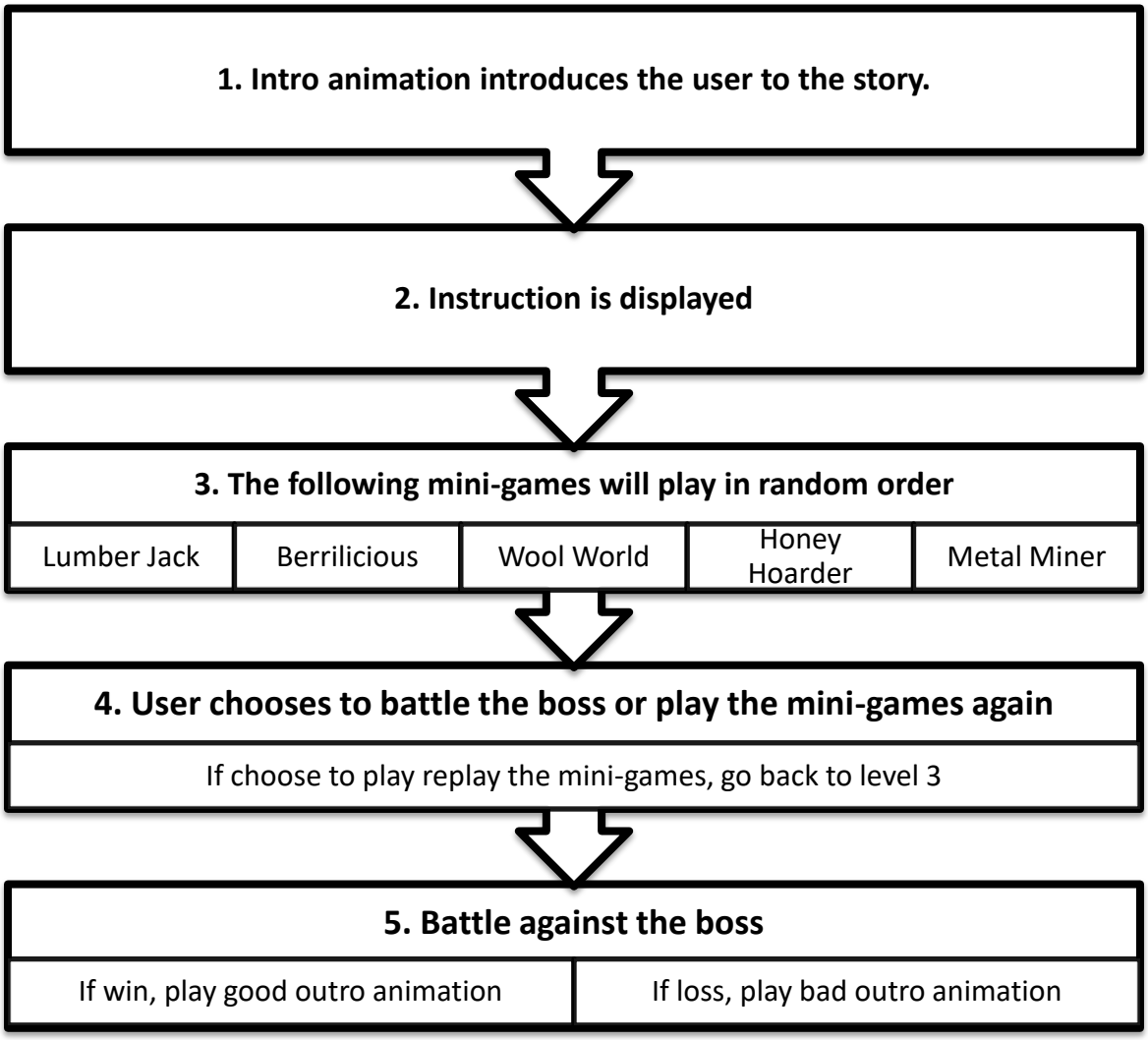

Figure 5: User-case diagram showing the flow of the program

4. Directory

4.1. Index

4.2. Glossary

Attributes: quality or characteristics which defines an object

Class: a template used in object-oriented programming to group attributes and methods

Software architecture: the structure or structures of a software system, their externally visible properties, and relations among each other.

Methods: a procedure in object-oriented programming associated with a class

View: a representation of a whole system from the perspective of related set of concern, seen from the perspective of a viewpoint.

Viewpoint: the outlook and concerns of a stakeholder