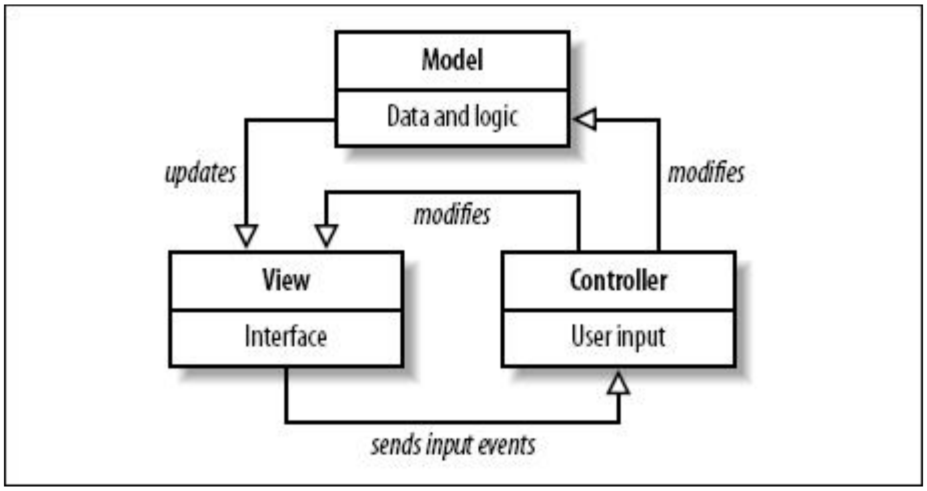
System Design Document
CSCE 361--- Group 14 JJP Game

1. Introduction

In this design document, we will give a high level architecture and class diagrams that show the relation between each classes and objects. People have some computer science background should be able to implement this system by looking at those http://www.moock.org/lectures/mvc/ diagrams show below.

2. Architecture

2.1 Introduction



At a high level, the architecture of our system will resemble a model-view controller system. The view will be responsible for receiving the user's inputs, as well as rendering data to the screen. Unity will handle most of the heavy lifting for this area. The controller will contain all the various scripts that will determine the game logic. The view will also process the user input and send the data to be rendered to the view. The model will be strictly in charge of holding the game data.

2.2 Modules

2.2.1 Controller

Inside Unity, various scripts will be attached to each game object. These scripts will make up the controller. All the game logic will be defined by these scripts. The controller will also interpret the user's inputs and figure out how to move each of the various objects. This data will be sent to the view to be rendered. These scripts will be written in C#. The Unity game objects will be linked through these scripts.

2.2.2 View

To interact with the game, the user will send inputs using either an xbox controller or their mouse and keyboard. The view will receive these inputs, parse them out and then send them to the controller. The view will also handle displaying the data to the screen. Inside the Unity editor the game objects will be modeled inside the game view. Unity will handle rendering each of the game scenes and displaying them to the monitor. 3d models of the player, AI, weapons, and environment will be either created in Unity, or imported from some external program.

2.2.3 Model

All of the games data will be held inside the model. This includes the various user settings, health, items held, player and AI health, environment data, etc. The model only houses this data, and does nothing more.

3. Class diagrams

C# will be used to implement all classes we used in Unity. The system will contains five classes. One for player, one for NPC, one for Items, and two classes derived from Items: Armor and Weapon. Each class will interact other classes. For example, when the NPC sword collides with a player hitbox, the health variable in the player class should be auto changed by this activity.

3.1 Class Information

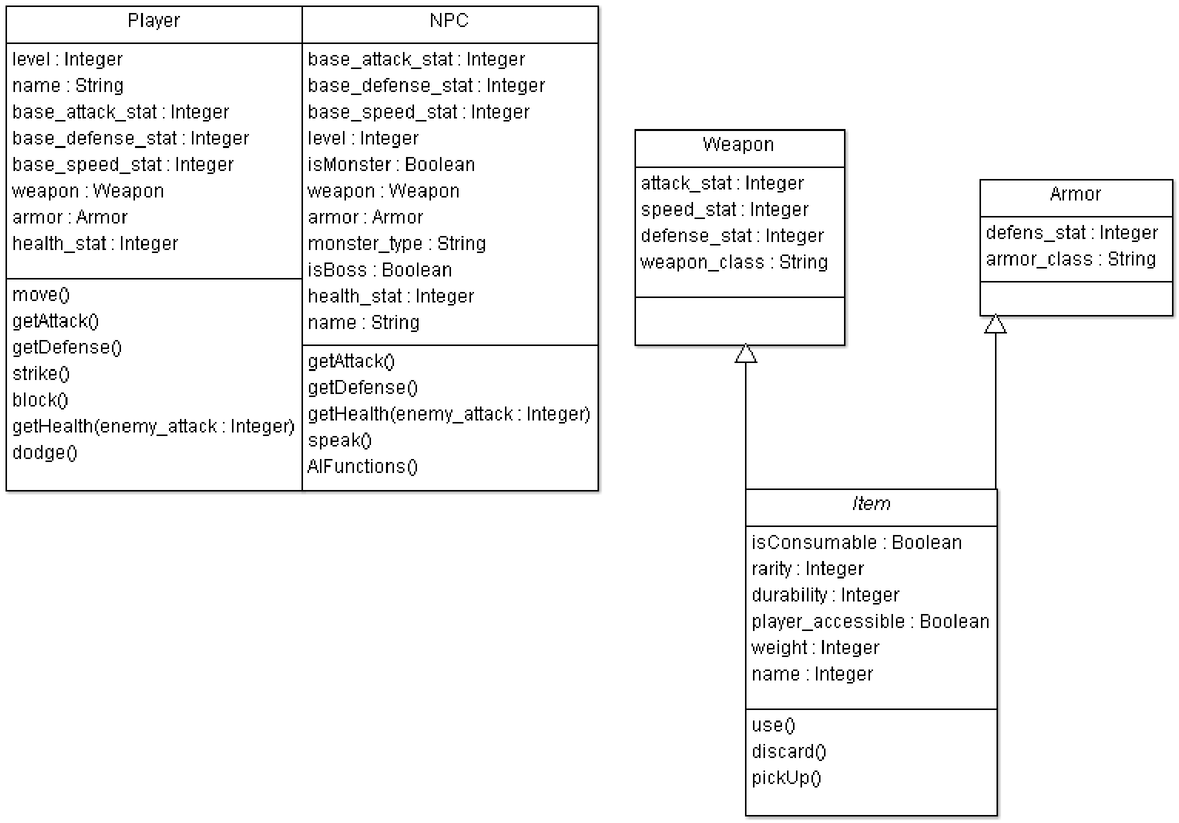
Player: This is the class information that will constitute the player's character. The various functions available will be called to update the player's position, or perform a certain movement such as an attack or block. The player class also contains a getHealth function which can be used by other components of the game. The player class also includes different variables which contain information about the player.

NPC: This is a class that holds NPC’s characters, such as the name of that NPC, is he or she a monster or boss, or what kind of weapons does he or she uses. It also contains getAttack, getDefense, getHealth functions. Those functions will update the current status.

Item: This is an abstract class that sets the framework for all items throughout the game, weapons, armor, etc. Each item class created will inherit this class.

Armor: This class is used to coordinate with the wearable armor/clothing items for both npcs and players in the game. It provides information to be used when determining a true defense stat for both players and npcs.

Weapon: This class is used to coordinate with equipable weapons in the game. It provides information to be used when determining a true attack/damage dealt for both players and npcs.



4. References

http://www.moock.org/lectures/mvc/