PNNSD

Kartikeya Sharma, Logan Stiles,
Claudia Shrefler, Jason Kang

Bibbin's Adventures (CSCI 205 Edition)
Design Manual

I. Introduction

Bibbin's Adventures (CSCI 205 Edition) is an RPG (role-playing game) featuring a storyline inspired by our CSCI 205: Software Engineering and Design class at Bucknell University. Inspired by games like Dragon Quest and Freddi Fish, our RPG allows users to join the class of CSCI 205 as a student at Bucknell, navigating the course by interacting with NPCs (non-playable characters) and completing tasks inspired by real-life aspects of the class. To successfully code a smooth-operating program as a team, we followed a clean object-oriented design and used Scrum and Agile frameworks for software development. Our program runs using a Model-View-Controller design to properly handle user interaction with the GUI.

The user first interacts with a JOptionPane screen that displays four buttons with the developers names assigned to each. By clicking on the button they want, the user is able to select the player they wish to play as in the game. After reading the introduction message that provides the user with their first task, the user sees the first room of the game with the player they chose standing in the center. From that point, the user is free to navigate the array of connected rooms by hovering over the doors in the center pane that represent where the next room is until an arrow appears. Clicking there moves the user into the next room. The user will also see NPCs in each room they travel into. When the user clicks on an NPC, the event handler in our program allows for the NPC's dialogue to be seen at the bottom right corner of the screen. In addition to speaking, the user is able to attack an NPC by clicking on the attack button on the screen. Once the NPC is dead, the user can click on them to have the system loot the NPC's body for items. The user can also trade items with NPCs by clicking the trade button. At any point throughout the game, the user is able to see their statistics (health, attack, defense) by looking at the top left-hand corner of the screen, in addition to seeing their inventory by clicking on the backpack icon. Our system also handles "drag and drop", which allows the user to drag items they see on the screen to the backpack icon to add them to their inventory or to their body to wear/consume them.

II. User Stories

SPRINT 1

As a player, I and NPCs should have health, attack, and defense statistics and have a fixed-sized inventory.	*As a player, I can have items that help me complete quests by improving my statistics, such as increasing health, attack, defense, or the size of my inventory.*	*As a player, I can click on doors leading to adjacent rooms if they exist.*	*As a player, I can trade items that will help me complete quests.*	*As a player, I can have different chances of successful attacks*
As a player, I have the ability to search NPCs for items and trade with NPCs.	*As a player, I can consume items to improve my statistics.*	*As a player, I can move left, right, up, or down into other rooms on the map.*	*As a player, I can talk with friendly NPCs during quest.*	*As a player, I can display what the NPCs say to me, so I know what to do next.*
As a player, I can talk with and trade/get an item from both non-hostile and hostile NPC characters.	*As a player, I can equip a variety of items, such as weapon, shield, and armor, to help me in battles.*	*As a player, I can follow a storyline to know how to complete subsequent quests.*	*As a player, I can fight hostile NPCs to complete quests.*	*As a player, I should be able to have a refreshing assortment of rooms every time I play the game.*
As a player, I can move between rooms to complete my quest.	*As a player, the room I am in is connected to adjacent rooms that allows me to move around the map*	*As a player, I can gather items from a room to add to my items list.*	*As a player, I can loot hostile NPCs to gain items to add to my items list.*	*As a player, buttons labeled with direction should indicate which rooms adjacent to the current room are available to be travelled to*

SPRINT 2

As a player, I can see my equipment and items, so I know what's available for me to use.	*As a player, I should be able to see the name of the room at the top of the screen.*	*As a player, I can see different images for different NPCs to distinguish between the rooms I'm in.*	*As a player, I can see the contents of the room to know where to go next in the story.*	*As a player, I can loot hostile NPCs to gain items to add to my items list.*
As a player, I can see my character's statistics, so I can better prepare myself for battling with the various NPCs in the game.	*As a player, I can see myself on the screen, so I know what my character is doing.*	*As a player, I can interact with NPCs to know what to do next in the game.*	*As a player, there should be some randomness involved in battles with NPCs so I can have a critical hit*	*As a player, I should not be able to kill or be killed by any enemy in one shot.*

SPRINT 3

As a player, I can see my updated statistics on the screen to know how much health I have left in the game.	*As a player, I have a visual of the items in the room to know what the item is.*	*As a player, I should be able to see my statistics and the NPC in the room's statistics refresh in the status bar when statistics change.*	*As a player, I should be able to see a visual on the screen representing that an attack is taking place.*
As a player, I can see doors and arrows on the screen to know where the rooms around me are.	*As a player, I should be able to see my statistics and the NPC in the room's statistics visually represented as a status bar*	*As a player, I should be able to drag and drop items into and out of my inventory*	*As a player, I should be able to hear sounds representing that an attack is taking place.*

SPRINT 4

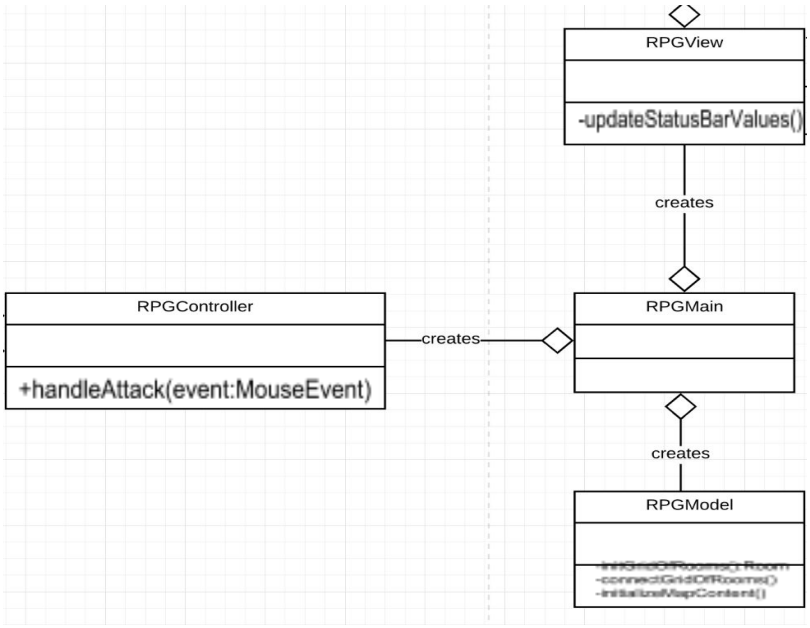
As a player, I shouldn't be able to click the attack button if there's already a battle happening so the story isn't confusing	*As a player, I can trade and search items to allow me to complete my task.*	*As a player, I can see myself on the screen, so I know what my character is doing.*	*As a player, I can follow a storyline to know how to complete my quest.*
As a player, I should hear sound effects throughout the game to enhance what's happening	*As a player, I am able to pick a player to allow me to portray a specific student*	*As a player, I can equip my items to help me in battles.*	*As a player, I can trade and search items to allow me to complete my task.*

SPRINT 5

As a player, I should be able to see the armor and weapons on my body so I know what I can use at the moment	*As a player, I should be able to see my first task via the initial popup when the game begins.*	*As a player, I should be able to select which player I would like to be from a given set of four characters.*	*As a player, I can see my name integrated with notifications that come from the game that address the user.*	*As a player, I can see the room background image to know what room I'm in.*

**NOTE: All user stories that are highlighted in yellow are user stories that are incomplete because they were low priority, and we did not have the time to finish them.

III. Object-Oriented Design



The part of the UML diagram featured on the right is the MVC design apparent in our program; it should be noted that the weaker associations have been removed. Our RPGMain class is responsible for creating the model, view, and controller, as well as the intro screen message the user sees. It then runs the entire program. Our RPGView class is where we create all the panes that represents the room that the user interacts with. In addition, it updates the status bar values that keep track of the user's health, attack, and defense. Our RPGModel class initializes the grid of rooms and connects them. This satisfies our user's ability to move between rooms to complete their tasks throughout the game. Our RPGController class is responsible for handling all mouse and drag events conducted by the user. When the user clicks on the NPC, the controller handles this event by displaying the NPC's dialogue. However, if the NPC is dead, the controller knows to loot the NPC's body. In addition, the controller handles the user moving between rooms by clicking on doors, dragging items around the room and into the inventory, as well as handling the attack, trade, search, and inventory buttons. Therefore, the MVC classes are really responsible for handling any user interaction with the elements on the screen view. Below are the CRC cards representing these classes:

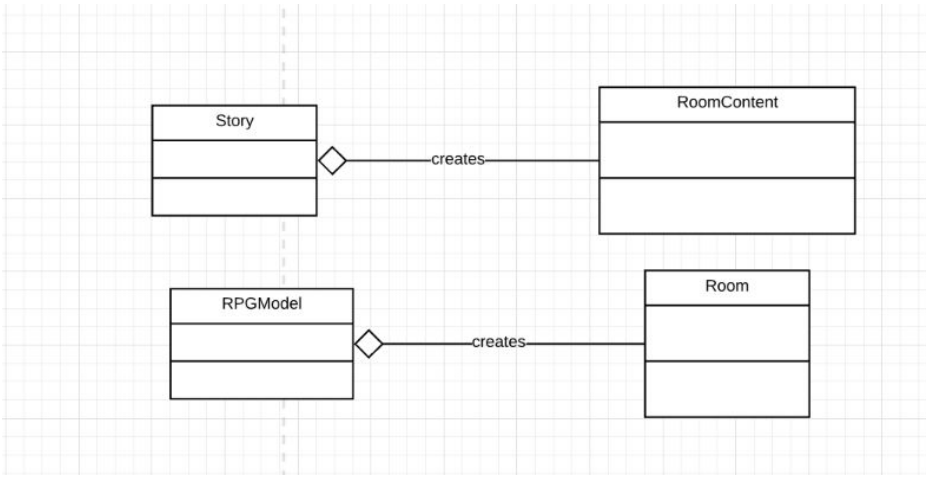
RPGView	
Knows story text Forms background panes Knows NPCs Maintains status bar	StatusBar RPGModel RPGController Room

RPGModel	
Knows Room contents Knows map contents	NPC Player Room

RPGController	
Knows NPCs in room Controls arrows Handles boss defeat Handles NPCs defeat Hande attack Handle buttons Handle mouse events	RPGView EventHandlers RPGMain RPGModel NPC

RPGMain	
Knows RPGModel Knows RPGController Knows RPGView	RPGModel RPGController RPGView

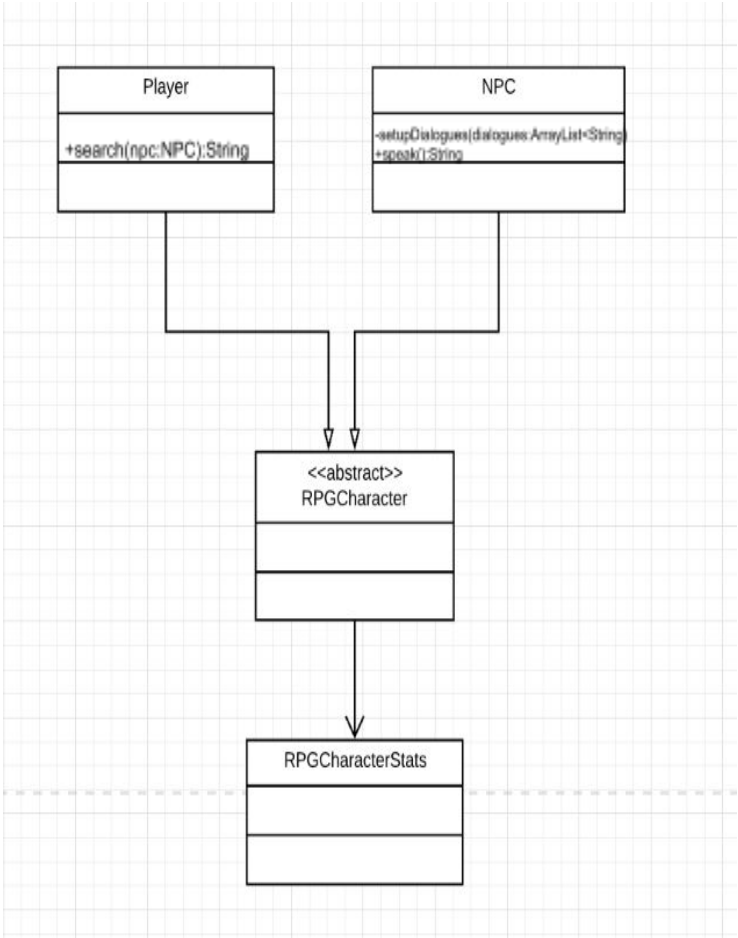
Featured on the right is how the RPGModel class interacts with the Room class, which contains the room's name, as well as knowing rooms to the north, south, east, and west. Hence on the diagram, the RPGModel "creates" the rooms. This, again, allows the user to enter other rooms from a current room. The RoomContent class contains the name, items, and NPCs in a certain room. This allows us to hardcode RoomContent objects in the Story class and assign them to rooms on the randomly-linked map. The Story class contains all the code that creates the story and the RoomContent objects, which in our case, is inspired by the class of CSCI 205. This class is the most dynamic of any class in the game because, if any developer wanted to take this RPG and create a different story, they can simply change the one we have coded in this class and make a new edition of the game. By changing images for NPCs and items, and changing the dialogue associated with each NPC (since the dialogue gives the user tasks), the entire storyline of the game can be altered. Since this idea is crucial to the dynamic characteristic of this program, we feel that, if we were to release a second version of this game, we would like to create a feature that allows a developer to easily change the storyline into whatever they would like it to be. Below are the relevant CRC cards:



RoomContent	
Knows name Knows NPCs Knows items	Item Story NPC wrapper

Story	
Knows room contents Story line	RoomContent

Room	
Knows name Knows NPC Knows items Holds background Knows surrounding rooms	Item NPCWrapper RPGModel



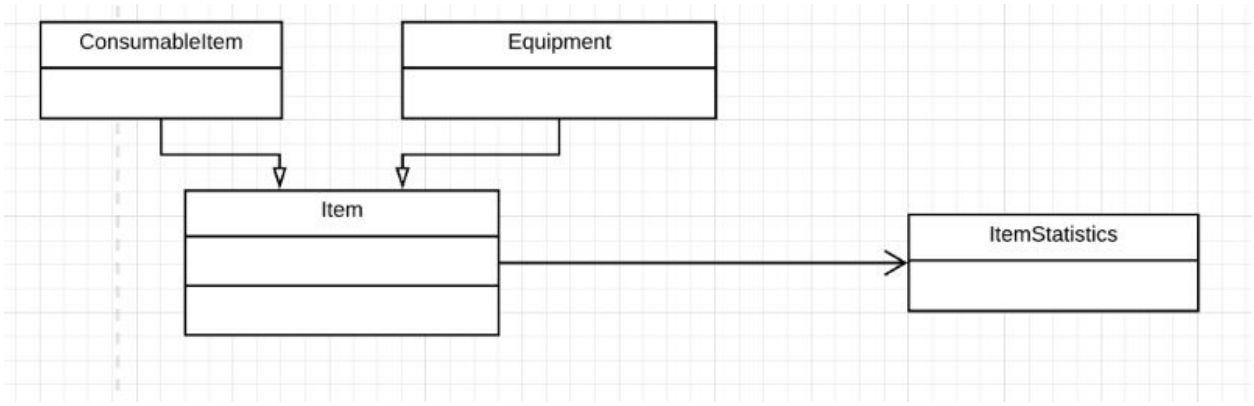
This portion of the UML Class diagram indicates the relationships between RPGCharacter, Player, NPC, and their stats. RPGCharacter is an abstract class that creates the methods that players should have, such as allowing the user to use an item, equip/unequip items, consume items, swap out the items that they're using, attack, calculate damage done to NPC during battle, etc. We decided to make this class abstract because both the player and the NPCs will require these methods, so they both implement RPGCharacter in our project. Leveraging the idea of data encapsulation, RPGCharacterStats contains the max health, health, attack, and defense attributes of a character in the game. Other than implementing the methods from RPGCharacter, our Player class also has a search() method which allows the player to search an NPC's body for items once the NPC is dead. In the NPC class, we have setupDialogues() and speak() methods. The setupDialogues() method allows us to set up the two sets of dialogues for each NPC, one dialogue being a hint for the user to complete a task and the other being regular/funny dialogue. Since we only wanted one NPC to be giving a task to the user at any given time, we checked if a certain task (e.g. giving time to Martin) was completed (does Martin have time?). If that task was completed, then we switch Martin's dialogue to being funny and switch the next NPC to giving the next task or have a JOptionPane message pop up on the screen to give the user the next task. Below are the relevant CRC cards:

RPGCharacter	
Knows name Knows stats Knows inventory Knows location Knows alive Knows dialogue Knows damage Can equip Can unequip Can swap equipment Can consume	Item Room Equipment RPGCharacterStats

NPC	
Knows dialogue Knows desired items Knows is friendly Can speak	Item RPGCharacter

RPGCharacterStats	
Knows health Knows attack Knows defense	

Player	
Knows defaults Can search	RPGCharacter NPC

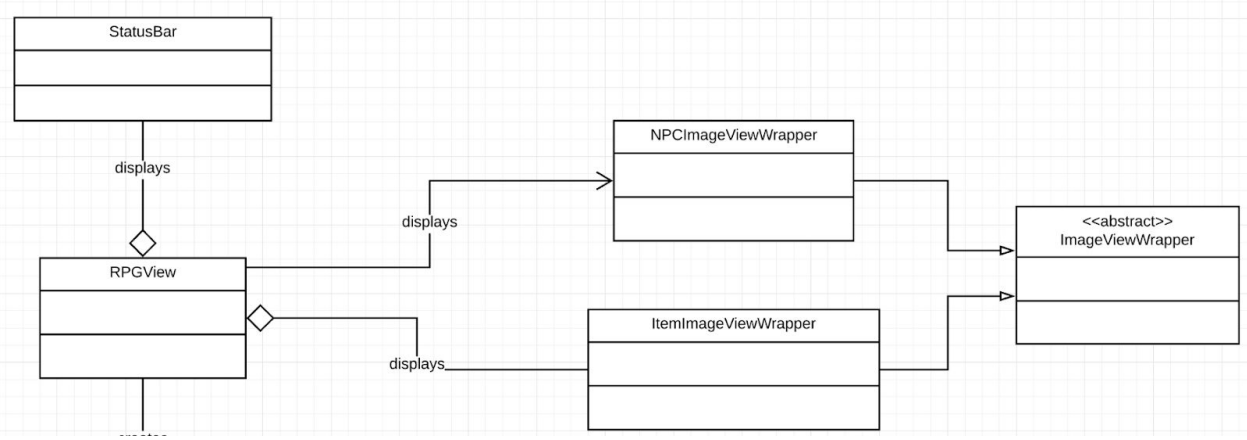


Above is the portion representing items in the game. The Item class initializes an item with a name, item statistics, and its image view key. So in the UML Class diagram, the Item class uses ItemStatistics (data encapsulation), which instantiates how much the item changes the player's statistics by, since its item stats are parameters in the constructor for the Item class. ComsumableItem and Equipment extend the Item class because they are types of items the user could have. Consumable items are items that the user can consume to improve their stats, and equipment items include weapons, armor, and shields. Below are CRC cards:

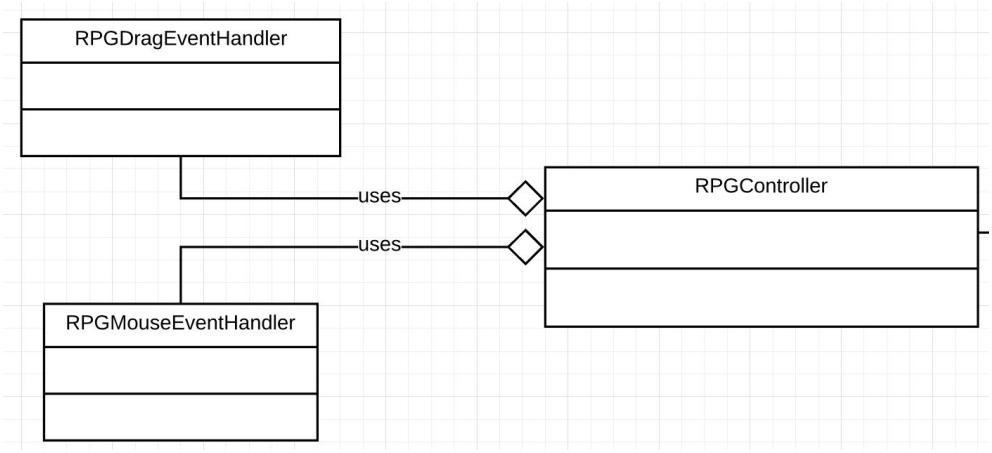
Item	
Knows name Knows stats Knows image	ItemStatistic ImageKey

Equipment	
Knows name Knows stats Knows image key	Item ItemType

ConsumableItem	
Knows name Knows stats Knows image key	Item



Items of type ItemImageViewWrapper are created at startup and are extension of ImageView so they can be rendered. The primary purpose of the ItemImageViewWrapper is pair a location of an ImageView on the view to the Item object that they represent. The NPCImageViewWrapper and ItemImageViewWrapper inherit from the parent class ImageViewWrapper. The RPGView displays both the NPCImageViewWrapper and ItemImageViewWrappers in each Room. The enumeration class ItemKey was used to specify types of items, e.g. PEN_AND_PAPER, which is a ‘level 1 weapon’. Each of these ImageView items are assigned a key defined in the enumeration ImageKey. This enumeration property was primarily leveraged to create a EnumMap of ImageKey to ItemImageViewWrapper, the view representation of an item image. Moreover, the ImageKey was used to specify the tooltip message for particular types of items.



The RPGController creates and uses RPGDragEventHandler and RPGMouseEventHandler to handle DragEvents and MouseEvents on the screen. RPGDragEventHandler handles detecting events related to dragging over an object, dragging an object, and dropping an object, which handles transferring items among the inventory, player, and the room space. The RPGMouseEventHandler, on the other hand, primarily handles the user clicking on action buttons, clicking on ImageViews representing NPCs on the screen (to talk with them).

Threading was implemented in the attack methods within the the controller to successfully show the POW visual after attacking the NPC and, after, play one sound effect after attacking the NPC in JavaFX. Then, threading was subsequently implemented within the counter-attack methods within the controller to display the BAM visual after the NPC counter-attacks the player and, after, play another sound effect after the NPC counter-attacks the player.