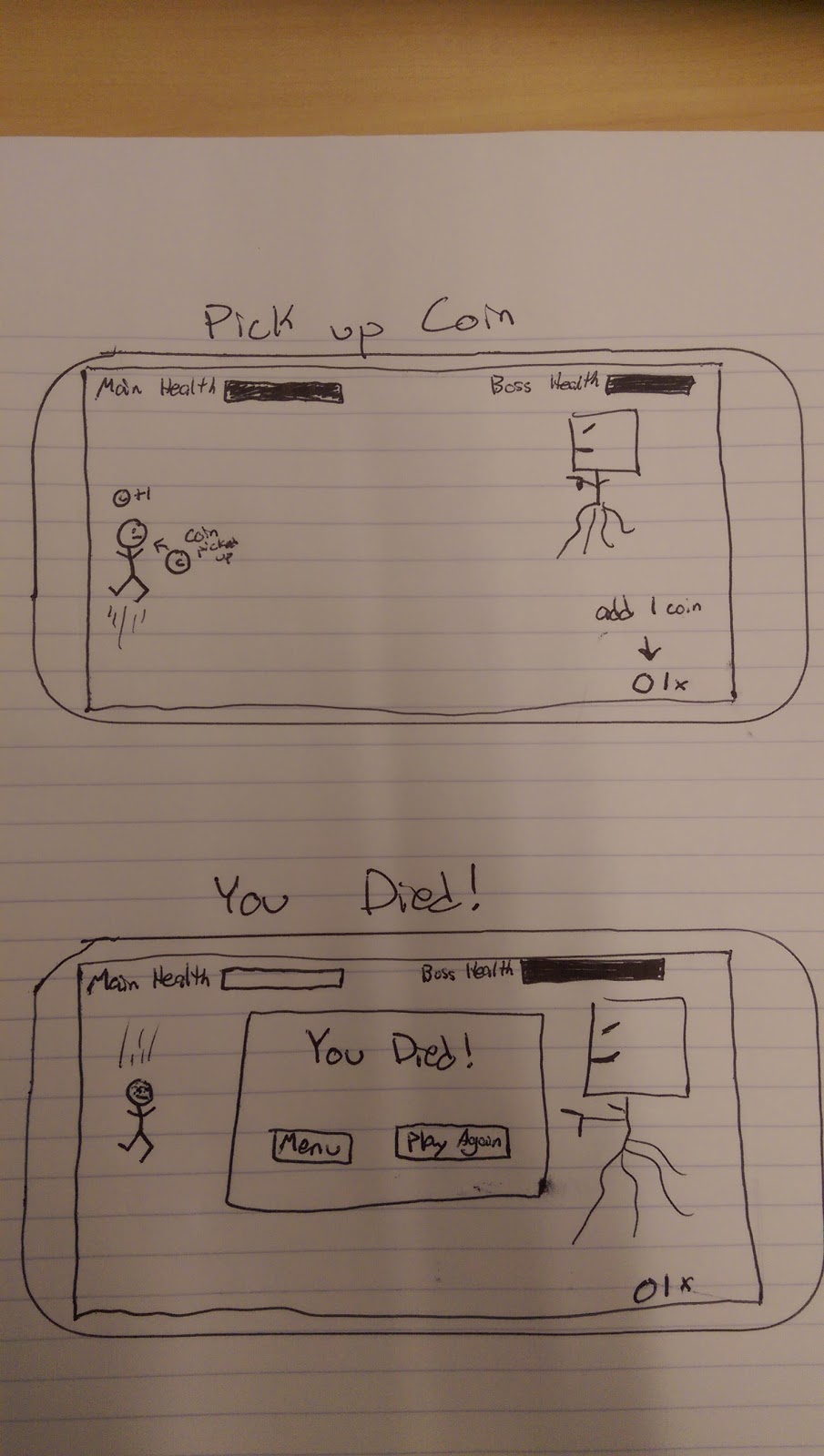
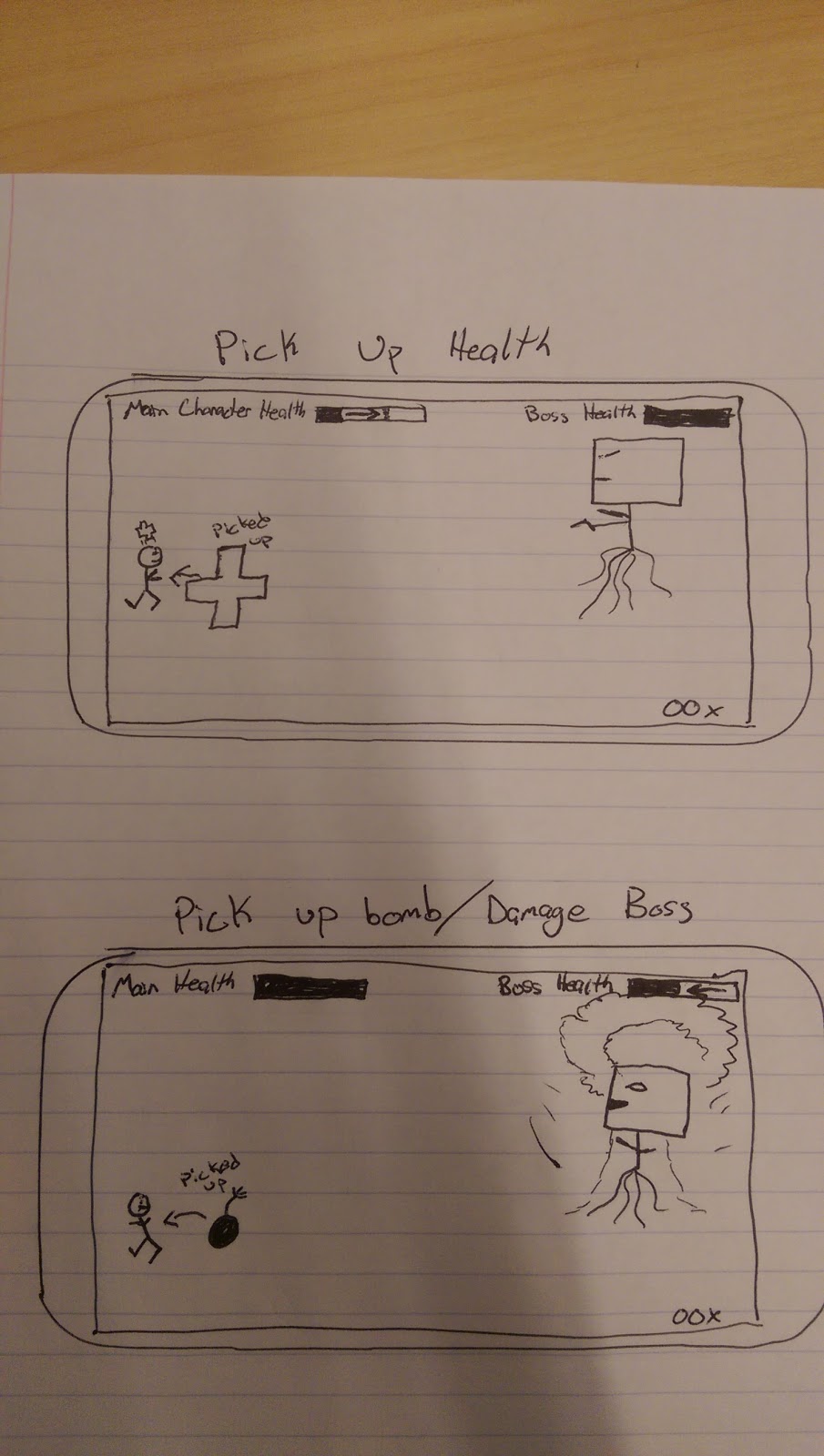
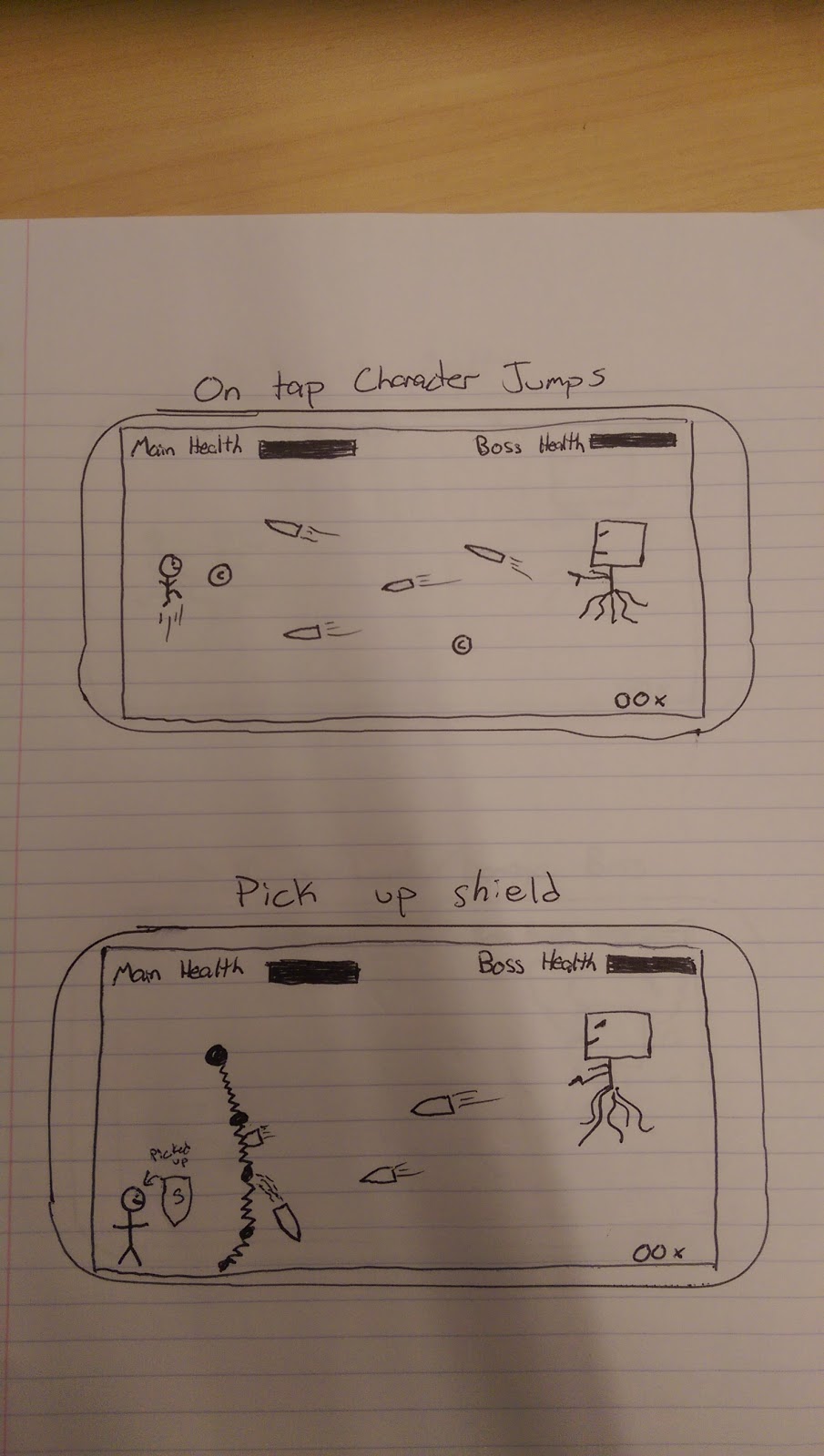
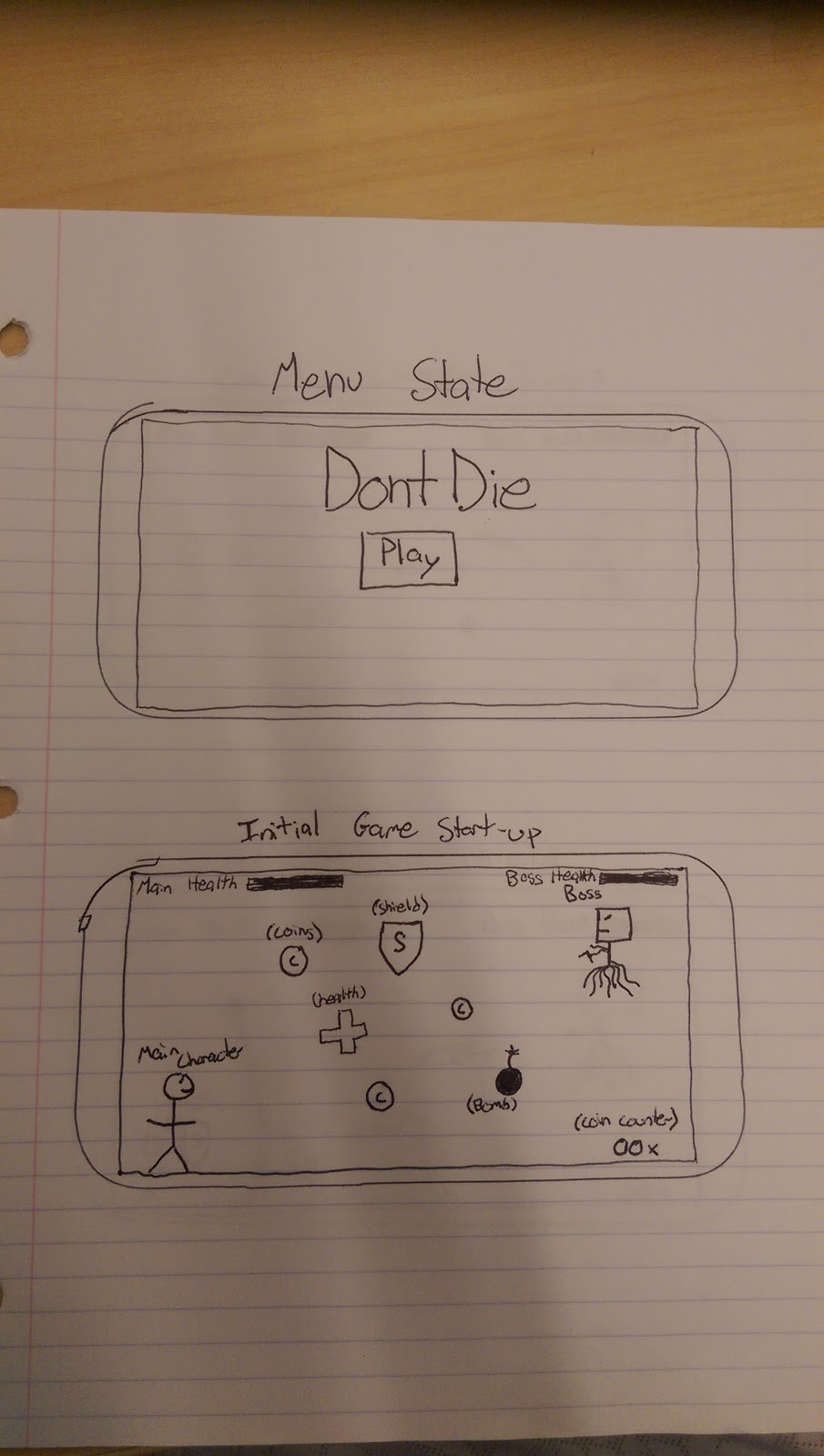
### **Context**

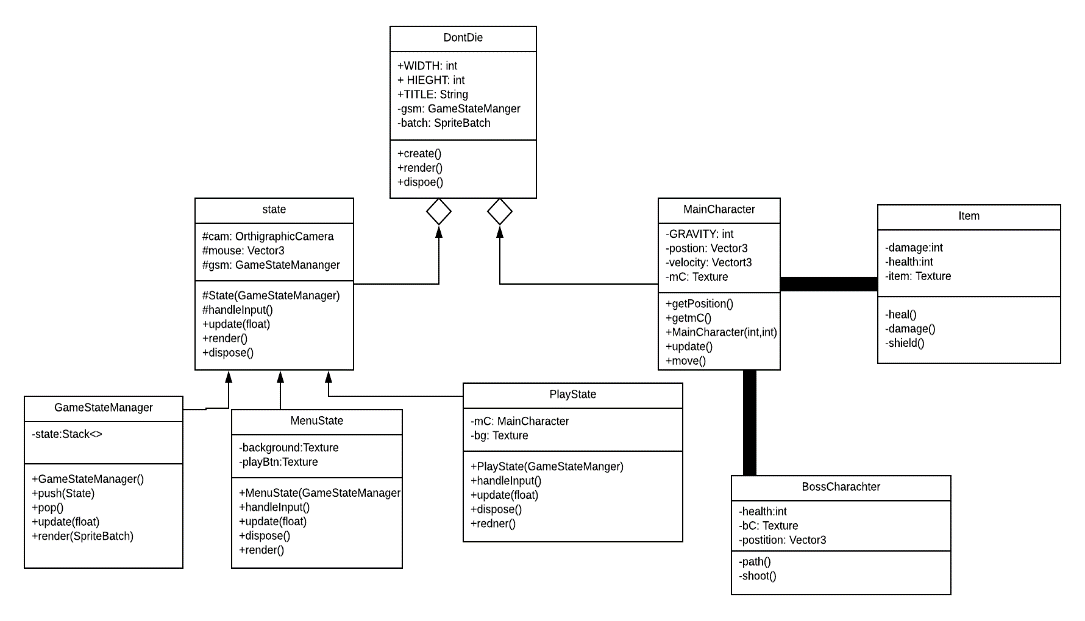
DontDie will have a menu state. This is where the user will touch play in order to start the game. When the user touches play, the system put on the stack the play state causing the game DontDie to launch. Users will then be taken to the main game of DontDie. The user will be the main character who is trying to avoid being shot by the boss. By tapping the screen, the main character will jump in the air to avoid the shots that the boss is shooting. The tap can be anywhere on the screen. The system will recognize the screen has been tapped and cause the system to make the main character to move vertically a set amount and then fall back down to the ground. If the screen is repeatedly tapped, the main character will continue to go vertically to the top of the screen. The user will see that the boss character will be predicting where the main character will be and shoot at the location based on its prediction. The boss will be able to predict where the main character will be by looking at its coordinates through the system and shoot near that coordinate. If a shot hits the main character, then the character will die and it will be game over. The user will also see that there will be items and coins popping up on the screen. If the main character comes in contact with these items, it will pick them up automatically. The system will keep track of how many coins that the user acquires. These coins will be used to buy unlockables such as skins and backgrounds

.

User Interface:

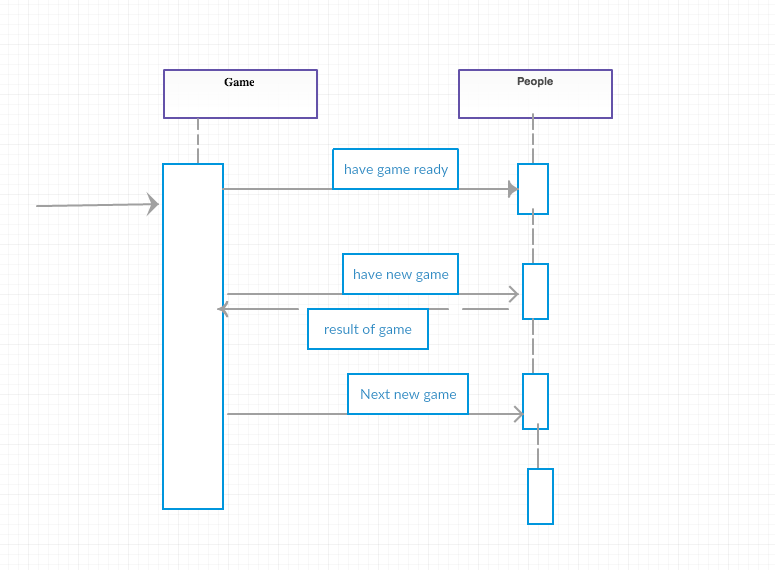
The user interface for this game will allow players to play mainly on their phones. The game has one basic control “tap to jump” the screen will keep scrolling and the boss character will move by itself. The player entity can pick up items which will either deal damage to the boss character or provide a shield.

**Class Diagram:**



For DontDie there are two packages that contain the majority of the classes, the state package and sprite package. The state package has a Factory pattern using the State class as the parent and MenuState, GameStateManager, and PlayState are all children. GameStateManager is straight forward, it managers the state the game is in. MenuState is what allows the user to choose and navigate which the state the user wants to be in. PlayState is the state where the game is being played. The most common method found in each is render() and update(), which are used to show the user the graphics of the game and to update those graphics. The sprite package will contain everything that would be considered an entity. The MainCharacter class contains everything that pertains to the playable character, like the gravity the character falls at, its position, and all of its textures. The next class is the BossCharacter class, which will contain the script for the path of the boss character, its textures, and how the boss is going to shoot at the player. The Item class will contain everything that the player can pick up, such as coins, items to damage the boss, items that will protect the player and other items that effect the game. The sprite package overall has more unique classes that are all things the user can interact with in the game environment.

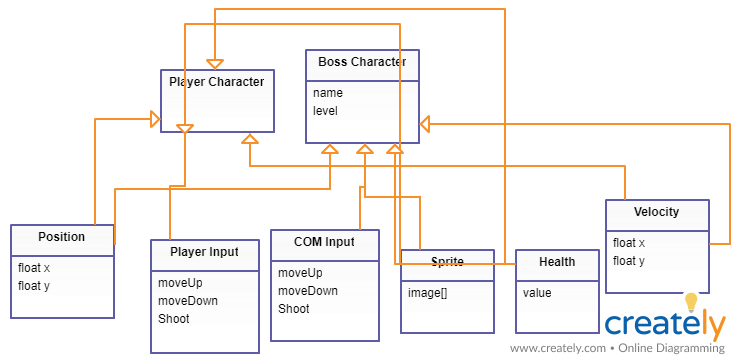
**Design Model**

In our game, we have some UML model in your game design. The first I want to talk about is use case diagram. People will play the game and system will accept the order to start the game. The developer will check the code and fix the bug. The second diagram is that state diagram. In our game, we have lots of state diagrams. We have a state called state it is the main track method. Then, we have three other state diagrams, such as GameStateManager, MenuState and PlayState. MenuState is for people to see the main screen of game and have the background image inside the code. PlayState is for people can run the game and control character to move up and down. GameStateManager is to track people who try to play the game and get game feedback all the time to track. So in our game these four game state diagrams have relationship with each diagram. We can track the game and run the system to see the bugs in the game. The third diagram is that Sequence diagram. This is the image for the sequence diagram.

In the architecture part, we first try to do the state diagram to track the game activity. The four state diagrams connected with each other and the relationship connect with them just each class. This is the basic design idea for us to develop this game. Each member all work together with the state code at first step and we use version control system to track the stage of project. In the development part, we discuss the all idea of the state diagram for us to understand what should be in the game. In the development view, we think the most important is state diagram because the game is basic on state. How people play in the game and How was game showed on the android. We follow the state diagram to get our game finishing.

**System Architecture:**

The game DontDie the UML of the entities looks like this. We have two entities, Player and Boss entities. They have different attributes within themselves. They have position and velocity also with an input attributes depending whether it's player defined or computer controlled.



**Timeline**

Week 7

* Meet Thursday from 8-9pm
* Get prototype finished (character jumping up and down) **(COMPLETED)**
* Figure out classes that will be used **(COMPLETED)**
* Turn in Project 2

Week 8

* Meet Thursday from 8-9pm
* Add boss class  **(COMPLETED)**
* Have boss character moving with a scripted path  **(COMPLETED)**
* Turn in Assignment 2

Week 9

* Meet Thursday from 8-9pm
* Fix bugs related to boss character and main character **(COMPLETED)**
* Add shooting  **(COMPLETED)**

Week 10

* Meet Thursday from 8-9pm
* Collision Detection **(COMPLETED)**
* Remove scripted path from boss and have it trail main character (implement Boss AI) **(COMPLETED)**

Week 11

* Meet Thursday from 8-9pm
* Add items in game, coins and skins  **(COMPLETED)**
* Work on bugs with collision detection
* Turn in Project 3 **(COMPLETED)**

**---->** Week 12 **(WE ARE CURRENTLY ON SCHEDULE!)**

* Meet Thursday from 8-9pm
* Work on bugs with collision detection
* Finish inserting skins
* Add Pause State, Game Over State and Unlockable State
* Turn in Assignment 3

Week 13

* Meet Thursday from 8-9pm
* Test boss character and main character
* Fix bugs revealed by testing

Week 14

* Meet Thursday from 8-9pm
* Test collision testing
* Fix bugs revealed by testing

Week 15

* Meet Thursday from 8-9pm
* Make game more appealing visually
* Test and fix bugs
* Turn in Project 4
* Turn in Assignment 4

Week 16

* Meet each day from 8-9pm
* Complete Project