**Project Document**

**VideoGame #2 Metal Sphere**

**1. Introduction**

The game we design is about the journey and adventure the ball comes across in different planets. It is a side-scrolling game, side-scroller or 2D is a video game in which the gameplay action is viewed from a side-view camera angle, and the onscreen characters generally move from the left side of the screen to the right to meet an objective. Super Mario is one of the typical side-scrolling game.

In our game, instead of a fat Italian pipe worker in Super Mario, the main character we design is a metal ball which aiming at a UFO which can transfer it to another adventure, and the game takes place on the surface of different planet. And bonus items and enemy will be encountered which may be helpful or deadly.

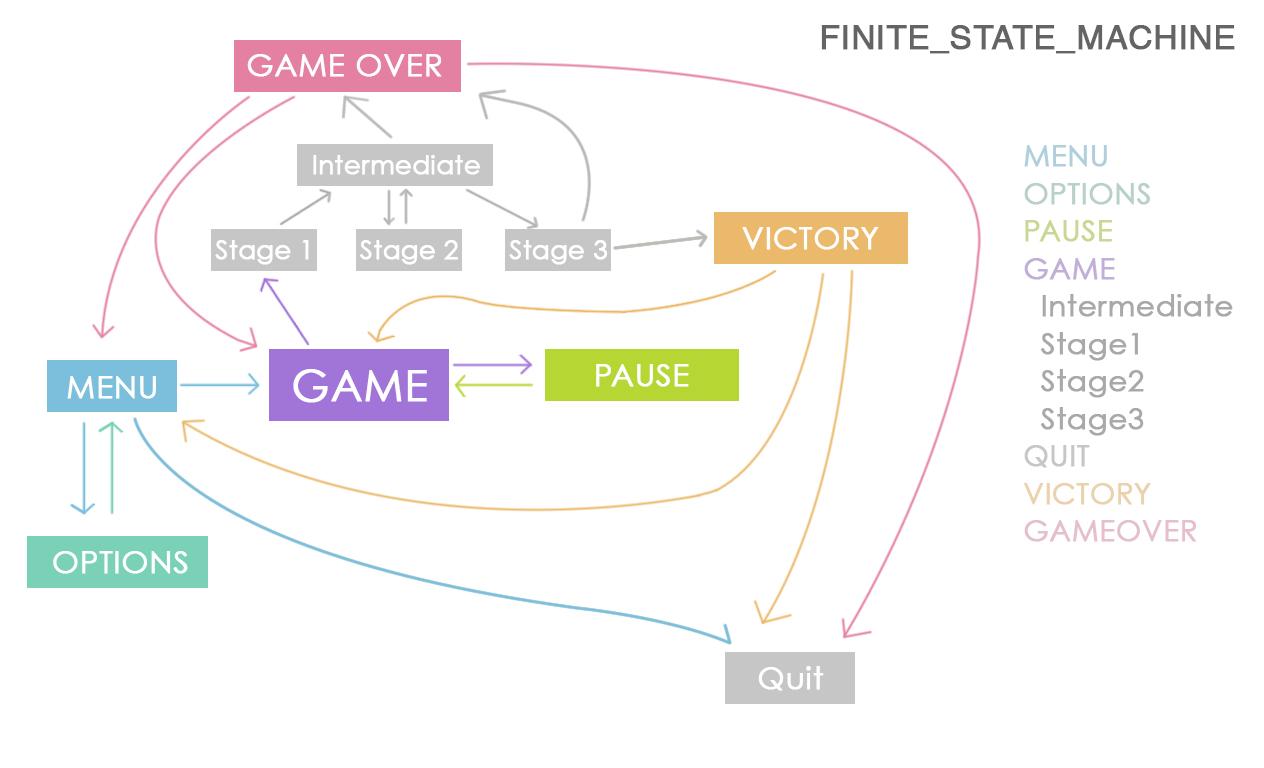
**2. Design**

We have 31 classes in 10 packages , which are categorized by its functionality. And use Lightweight Java Game Library and Slick (a 2D LWJGL Java Utility Library) as well.

The game is a finite state machine, the display mainly depends on what state the program currently in.

To develop the game, we have designed several states to help realizing the game function. The state switch will be triggered by mouse input, or other condition has been fulfilled such as the character falls off the cliff. Each stage will have its own background music and appearance such as terrain and images.

Figure 1 shows a demo of our state machine.

Figure 1 Game State Machine

**2.1 Classes and Packages**

**Controller.java**

The Controller class handles and controls all of the game objects. There are four main functions.

**tick()** updates all the coreobjects in the array. (For example, as the player walks, all of the blocks will move to the left or right depending on direction).

**render()** makes the core objects from array appear.

**add Objects()** adds a new instance to the arraylist to be ticked and rendered

**removeAll()** Removes all objects after that stage is completed

**StageGenerator.java**

StageGenerator class is a class that we create all the stuff in different levels.There are two main functions.

**BuildTerrain()** is about the way to build our map, we created the object named block. and we arrange the different position of them to set different types of terrain.

**Init()** remove all the old stuff in the last stage and set the initial stage in the game.

**Game.java**

The Game class is the start of the game, the source of everything. There are three main functions

**init()** run as the first function to provide full support of the program.

**run()** connect two stages continuously.

**Start()** creates a thread and call the thread. start() which will call the run function to start the program.

**Packages:**

**core**

including class Coreobjects

**Coreobjects**: the base class of any foreground object such as Blocks, enemy, player, items.

**entity**

including class Player

**Player:** check the position of the metal ball and check its situation(fall of collision).

**enums**

including class Gamestate and StageState, which contain each game state and stage state of game.

**gfx**

including class Renderer, Texture

**Renderer:** draw the objects in the game.

**Texture:** read all the images for objects like the ball, coin etc.

**input**

including class KeyInput, MouseInput

**KeyInput:** controls the movement of the ball like A to left, D to right, space to jump, ESC to pause during the game. Collision detection is also in consideration of the movement of the ball.

**MouseInput:** controls every buttons in each screens.

**libs**

including class Audio, Fonts, Identities, Images, Reference. It is a package which set the name and relative path for all the resource we need in the game.

**objects**

including class Block, Desti, Enemy, Item, Ufo. Each object has their own collision test method.

**Block**: the basic model in our map, it composes the terrain.

**Desti:** check whether the ball comes to the destination of current stage.

**Enemy:** create a new object called enemy, set the ball die if touches the enemy.

**Item:** create a new object called coins, use to get points.

**Ufo**: the logo in the end of the stage.

**screens**

including class Help, Intermediate, Menu, Option, Over, Pause, Play, Victory

**Menu:** the first interface when entering the game.

**Option:** including some setting buttons like help, volume up and down also mute / unmute.

**Help:** show help page.

**Play:** the core screen the all the game stages will running in.

**Intermediate:** show scores between two stages.

**Pause:** show when press ESC during the game

**Over:** when the ball touches an enemy or falls off the cliff.

**victory:** show when finally wins.

**utils**

including class AudioPlayer, BufferedImageLoader, Button, ResourceLoader, SpriteSheet

**AudioPlayer:** load the music and sound into two hashmap, set playSounfEffect, playMusic and stop method.

**BufferImageLoader:** load the image.

**Button:** set draw rectangle button method used to draw buttons in screens, also realize different appearance of the button under different mouse input action.

**ResourceLoader:** load all images, sound, music and fonts we need from hard drive into memory.

**SpriteSheet:** loads the block image into the game from a sprite sheet image.

**2.2 Program Procedure**

Here is how the game works: the main while loop locates within the run function. The isRunning (a boolean variable) is set to indicates the game is running or not, in case of multiple opening. It is set true within the start() function.

Within the main loop, two significant functions will be called to both draw the whole picture and update the status of each object. They will call the corresponding function in individual class/objects according to the current state. All the process runs according to the state machine in *Figure 1*.

**3. Conclusion & Future Work**

We design the game in a great frame structure, it is easy to add more features and functions into the game. Also, we may allow users to build their own map with a map editor. More further, the frame structure we use may applied to other game.

**4. Source Reference**

The music and sounds are from [www.opengameart.org](http://www.opengameart.org)