Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project

Space Out: Retro sci-fi platformer game

Final Report

Doruk Çakmakcı, Anıl Erken, Umut Berk Bilgiç, Arda Can Atasoy

Course Instructor: Bora Güngören

# Table of Contents

# 1. Implementation Process

# 2. Status of the Implementation

# 3. Design Change Choices Based on the Current Implementation

**4. Appendices**

# 4.1 Appendix A

# 4.2 Appendix B

# 4.3 Appendix C

# 4.4 Appendix D

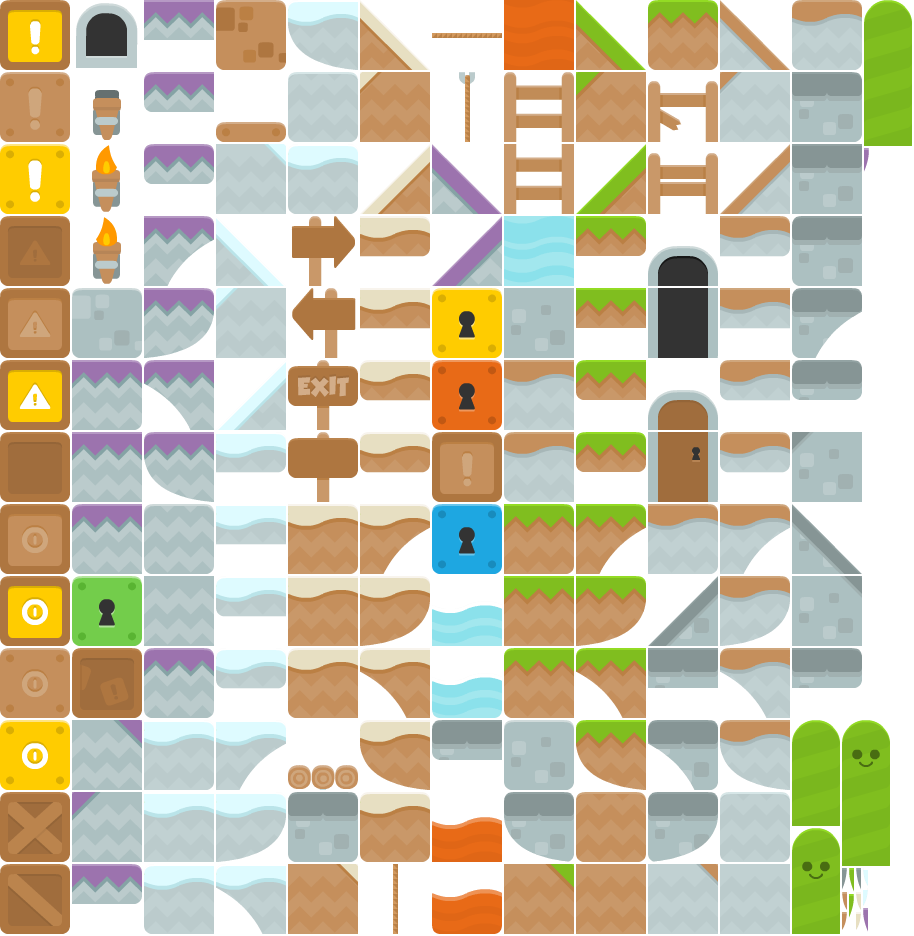
# Implementation process

During the first stages of the implementation we were in search for an exact way of doing it whether using default libraries from Java or using external 3rd party libraries of which rely on more efficient technologies. We have looked into Java2D (Swing and AWT), JavaFX, LWJGL and Slick2D. We have decided to go with Slick2D. More will be told about this at the dependencies part.

Slick2D uses state based applications. We will use this feature in order to implement several states to our application such as paused, in-game, in-menu, in-settings etc. The states shall dictate what the user sees and interacts with at that moment. **See appendix A.**

This state based application will follow class game development techniques such as the game-loop and three basic stages. First being the initialization of the game itself (not levels, players and such but the application as a whole.) Second being the update stage in which the game logic updates itself based on what happened during that frame. The third layer is the render stage which is required to separate game logic from GUI logic. **See Appendix B.** Within the game loop lies a state manager. This is crucial for the user to be able to move from menu to menu or menu to game and vice versa. When the user moves to the in-game state, a level is loaded from a map file and a player is placed in it.

Whilst researching about game mapping practices, we have found out that there is a universal way of saving and interacting with tile based maps. Since our game will use tile based maps we figured the best practice for this would be to create and use these standardized map format rather than creating proprietary map / tile system. We have ended up with using .tmx (Tile Map XML). These maps store the map tile and layer information as well as the useful meta-data for the map such as width, height, background color, version, render order of the tiles etc. These maps use things called tilesets i.e. image files containing every texture being used by the tiled map. Slick2D supports these maps with the TiledMap class. This will make level creation easier while not effecting our design flow. **See Appendix C**

Example TileSet sprite sheet that is used by the tiled map file. The tiled map file should be able to access this image in order to create the map. The maps are not saved as images but rather as references to individual tiles. [1]

Character models are not a part of this tileset and are handled in a different way.

Regarding the Characters within our game, we use simple keyboard listeners to control them and the Slick2D built in moving animations to animate them. This way, we are not reinventing the wheel and can focus on making a better game without worrying too much about losing a lot of development time on developing inefficient versions of things that are already pretty well understood and built. To animate our character, we enumerate (we don't need to this, it is just a readability upgrade) a Facing for characters; left and right. Depending on the last move made by the user the facing variable of the character is changed. We also create HashMaps in order to map different stages of the movement to different character images. We put these images in correct order fast enough so it looks like our character is moving. **See Appendix D.**

C:\Users\UmutBerk\AppData\Local\Microsoft\Windows\INetCache\Content.Word\player_right.png**C:\Users\UmutBerk\AppData\Local\Microsoft\Windows\INetCache\Content.Word\player_right.png**



These are the four stages of movement currently being used in our test builds. We use built in Java methods to flip these images during the game to produce left orientation.

## Status of the Implementation

Currently we are still implementing the core aspects of the game. The aspect that is taking most of our development time is surely collision detection and how to translate TiledMaps to "collidible", solid tiles which can be interpreted by our physics handler. Some of the other features and design goals of our game such as Cloud based achievement and leaderboard system and world-based game mechanics are yet to be implemented. Currently we are focusing on getting our core base mechanics up to a level we are comfortable to work with. Since, in order to implement up on the game, we believe we must have strong core that handle such additions without breaking down and crashing constantly.

## Design Change Choices Based on the Current Implementation

Even though we have moved from Swing and AWT development to Slick2D development, we did not need any major design changes due to it. If anything, it made our design choices more clearer and made us more confident about these choices we have made. Because with using this 3rd party technology we can focus on actually implementing extra features by getting through the core aspects much faster, these core aspects being things such as maps, levels, game states, characters, level design and such.

Space Out uses native java libraries as well as Slick2D, 2D Java game library which consists of tools and utilities wrapped around LWJGL (Light-weight Java Game Library). Slick2D is used to make implementation of Space Out easier and not to re-invent the wheel but to use the formerly invented wheel and create a more enhanced game. Therefore, using Slick2D doesn’t change the design of Space Out specified in former documents.

Space Out will be distributed with a program with .jar extension. It will consist of the dependencies of the source code and the source code itself. Using .jar extended program will increase the portability of Space Out. Since the .jar extended program runs on JVM, the user is expected to download the latest JRE in order to run Space Out on any operating system.

**APPENDICES**

**APPENDIX A** (from Game class)

**public** **class** **Game** **extends** **StateBasedGame**{

//set the window width and then the height according to an aspect ratio

**public** **static** **final** **int** ***WIDTH*** = 1920;

**public** **static** **final** **int** ***HEIGTH*** = ***WIDTH*** / 16 \* 9;

**public** **static** **final** **boolean** ***FULLSCREEN*** = **true**;

// tiles are 70x70 but we want them rendered to be 60x60

// . . . 60/70 = 0.85714285714

**public** **static** **final** **float** ***SCALE*** = (**float**) 0.85714285714;

**public** **static** **final** **String** ***TITLE*** = "Space Out";

**public** **Game**(){

**super**(***TITLE***);

}

**public** **void** **initStatesList**(**GameContainer** gc) **throws** **SlickException**{

// create a new level state which will handle in game stuff

// . . . empty is a level file in our game data files

addState(**new** LevelState("empty"));

**this**.enterState(0);

}

**public** **static** **void** **main**(**String**[] args) **throws** **SlickException**{

**AppGameContainer** **application** = **new** AppGameContainer(**new** Game());

application.setDisplayMode(***WIDTH***, ***HEIGTH***, ***FULLSCREEN***);

application.start();

}

}

**APPENDIX B** (from LevelState class)

**public** **void** **init**(**GameContainer** container, **StateBasedGame** sbg) **throws** **SlickException** {

//after init we load the level given

level = **new** Level(startinglevel);

//at the start of the game we don't have a player yet

player = **new** Player(0, 0);

level.addCharacter(player);

// create a input handler

playerController = **new** InputHandler(player);

}

**public** **void** **update**(**GameContainer** container, **StateBasedGame** sbg, **int** delta) **throws** **SlickException** {

//every update we have to handle the input from the player

playerController.handleInput(container.getInput(), delta);

}

**public** **void** **render**(**GameContainer** container, **StateBasedGame** sbg, **Graphics** g) **throws** **SlickException** {

g.scale(**Game**.***SCALE***, **Game**.***SCALE***);

//render the level

level.render();

}

**APPENDIX C** (from Level class)

**public** **class** **Level** {

**private** **TiledMap** map;

// character list holds all character on this map

**private** **ArrayList**<Character> characters;

**public** **Level**(**String** level) **throws** **SlickException**{

map = **new** TiledMap("data/levels/" + level + ".tmx", "data/img");

characters = **new** ArrayList<Character>();

}

**public** **void** **addCharacter**(**Character** char){

characters.add(char);

}

**public** **void** **render**(){

// first render the map

map.render(0, 0);

// than place all the characters on top of the map

**for**(**Character** **c** : characters)

c.render();

}

}

**APPENDIX D**

**private** **void** **setAnimation**(**Image**[] images, **int** time)

{

// right facing animations are out the right part of the hash map

movingAnimations = **new** HashMap <Facing, Animation>();

right = new Animation(images, time);

movingAnimations.put(*Facing*.***RIGHT***, right);

// reverse the right facing images to make left facing ones

**Animation** **left** = **new** Animation();

**for**(**Image** **i** : images){

facingLeftAnimation.addFrame(i.getFlippedCopy(**true**, **false**), time);

}

movingAnimations.put(*Facing*.***LEFT***, left);

}