**LunarX Preliminary Specifications**

In this project, we create a single-player moon landing simulation game. The objective of the game is to gain points by successfully landing on the various platforms of the landscape which are marked with different point values depending on how difficult it is to land on the platform (the shorter the platform, the more difficult it is to land on it).

The controls are simple. The rocketship is controlled with three arrow keys - up for thrust, left to rotate the rocketship to the left, and right to rotate the rocketship to the right. The rocketship has a limited fuel capacity. 10 fuel units are used up for every second (or 1 fuel cells for every 0.1 seconds) that the thrust button is held.

In order to land on a platform successfully, the player must land the rocketship in a vertical fashion (with a leeway of around 10 degrees) and have a vertical speed of less than 10. If they fail to meet these conditions, the rocketship will break, the player gains no points for that run, and if they still have any fuel left they will reset onto the launch pad. If at any point the fuel runs out, the rocket will float freely through the map until it crashes, and the game will then reset.

1. Structural Design

|  |  |
| --- | --- |
| **DATA** | **class** |
| Terrain | Array |
| Points | Array |
| Rocket Position | Array |
| Controls | Queue |

We have chosen to hold the coordinates of the terrain in an array. Because the terrain will always remain the same, its length is fixed which allows us to use an array. An array provides us with the quickest access to its elements (O(1) time), which is important for our game which requires the constant comparison of the rocket position and the landscape line to determine if the rocket is landing or not. For similar reasons, we have all chosen to store the position of the rocket position, for purposes of being able to quickly access the coordinates to compare to the landscape. If the coordinates match, then the rocket has either landed or crashed.

Similarly, we have chosen to store the point values at the specific landing points in an array of ints. We know the length of the terrain, and for all parts of the terrain that are a landing point will have a int value corresponding to its point value. For the rest of the terrain, the value in the array is zero. An array provides the fastest access to these times, which is another reason why we have decided to use this data structure

For the controls, which are a list of the arrow keys entered, we have decided to use a queue to store the controls that are entered. An ActionListener, which receives the arrow key input, stores all of the keyboard input into an queue and processes them by the order which they are entered. A queue is standardly used to receive keyboard input.

2. Object-Oriented Design

The figure below shows a visual representation of the classes for this project. There are at total of 5 classes: Animation Base, LunarX, Landscape, Rocket, RocketShape.

Animation Base

LunarX

Rocket

Rocket Shape

Landscape

*Class Descriptions:*

The class AnimationBase provides a generic framework for applets that do simple animations. This framework is appropriate for an animation that runs continuously as long as the applet is active. It is assumed that each time a new frame of the animation is to be drawn, that frame will be drawn completely from scratch.

The LunarX class is the main class that runs the whole program and extends AnimationBase. LunarX initializes the rocket, background, and landscape. Once initialized the class will continually animate frames through the drawFrame() method.

The Rocket class handles all the motion components of the rocket. Rocket extends RocketShape. This class includes the following attributes: velocityX, velocityY, fuel, altitude, time etc. Additionally, the class will figure out the direction and speed of the rocket based on the user input.

The RocketShape constitutes of the rocket structure and size. This class with contain the rockets specific x and y values in order to locate the rocket. Additionally this class will be the blueprint for the shape of the rocket.

The Landscape class contains the data for the terrain. It will have a data structure that keeps track of the the location of the terrain based on x and y values. This data will then be used to compare with the rockets locations.

3. Detailed Design

Rocket:

Fields:

velocityX - the Rocket’s X coordinate velocity

velocityY - the Rocket’s Y coordinate velocity

RocketShape:

Fields:

X - the x coordinate of the rocket

Y - the y coordinate of the rocket

Radius - the radius of the rocket

Color - the color of the rocket

LandScape:

Fields:

Array yVals - an array containing the coordinates

LunarX - the main program

4. Testing

Proper testing of the game is essential in order to phase out any bugs during the ending stage of the project. In order to test this project, we will be using JUnit testing in java. The first part that needs to be tested is the Rocket class. The rocket will be tested for proper positioning and motion. Additionally, the Landscape should be tested. The landscape class will be tested for proper storage of data in the data structure. Lastly the score reports of the data on the screen will be checked for proper updates for the user.