**LunarX Final 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 right, and right to rotate the rocketship to the left. The rocketship has a limited fuel capacity. Fuel units are used up whenever the up arrow key for the thrusters is held down.

In order to land on a platform successfully, the player must land the rocketship in a vertical fashion and have a vertical speed of less than 60 m/s. 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 at the top left of the screen. 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

Highlighted below are the key data structures that represent different respective data:

|  |  |
| --- | --- |
| **DATA** | **Date Structure** |
| Rocket Pieces | LinkedList |
| Landing Zones | Map<Integer, LandingPoint> |
| Landscape y-values | int[] |
| Coordinates of Stars | int[][] |
| x-values of polygons | int[] |
| y-values of polygons | int[] |

For the rocket pieces, we have decided to use a LinkedList. LinkedLists offer quick add and remove methods. For purposes of our program, we utilize the fact that LinkedList provides quick add operations. Additionally, LinkedLists are easy to traverse through and access, which is important for our program because we need to edit information in certain elements of the linked list and traverse through the rocket pieces to get them to explode.

We have decided to store the landing zones in a Map linking an integer to a landing zone. This was useful since we needed the same key for a range of x-values to map to the same landing zone. This means that the when the rocket is in a certain zone its x value will match up to a certain landing zone. That way the rocket will only have to be checked against one landing zone at a time.

We have chosen to hold the y-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.

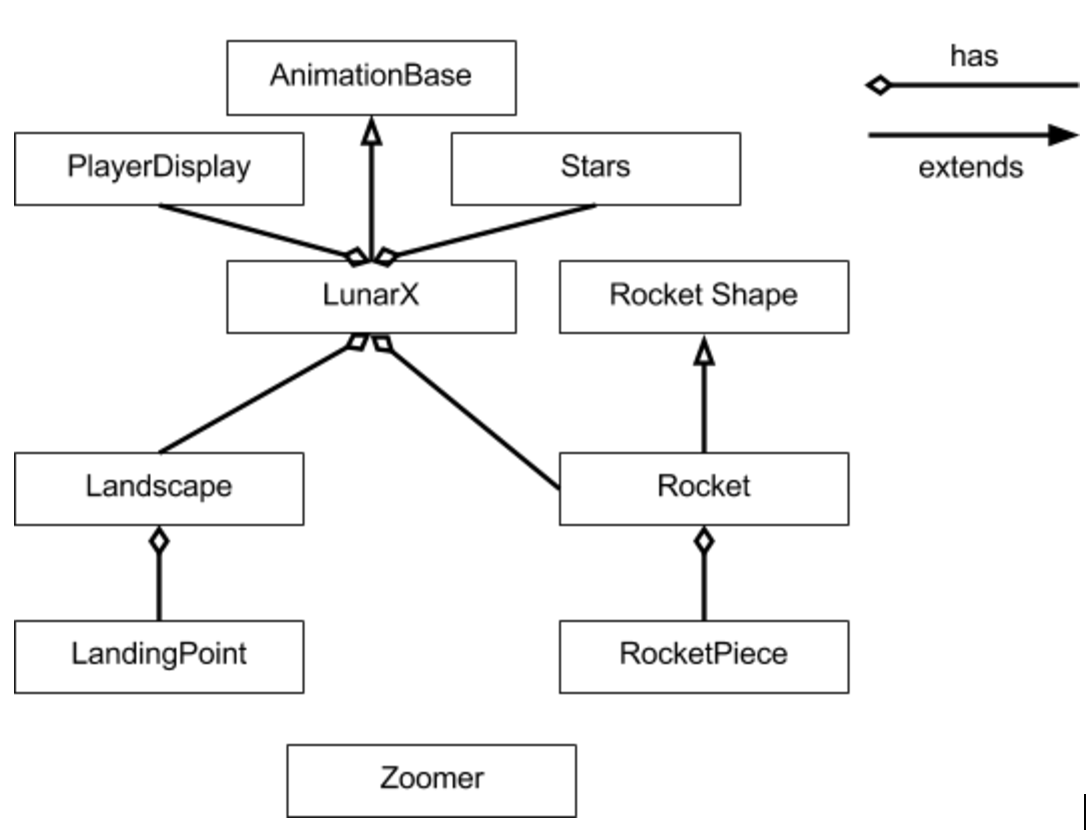
To store the landscape coordinates, we have decided to use an array. The index of the array corresponds to the X coordinate, and the value of the index in the array is the Y coordinate. We have decided to use an array because the length of the landscape is fixed, and there isn’t any need to add or remove elements in the array. In addition, arrays provide quick access to its elements, which is important for our game, which constantly compares the rocket’s coordinates to the coordinates of the landscape to detect a collision.

The coordinates of the stars are stored in a 2D array, with a width of 2. In the first column, the X coordinate of the star is stored, and in the second column, the Y coordinate of the star is stored. We decided to use a 2D array instead of a 1D array where the X coordinate is the index because that would leave several null locations in the array and waste space. There isn’t any removing or adding of stars in the program, so a 2D array served the purpose just fine.

We decided to store the X values of the rocket polygon in an array because there is a fixed amount of X coordinates for the rocket edges (it doesn’t change), and an array is easy to traverse through. No removal or addition is involved with the X coordinates of the rocket. An array wouldn’t waste space and served the purpose well, and it was easy to access certain elements in the array, We also decided to store the Y values of the rocket polygon in the array for the same logical reasons.

2. Object-Oriented Design

The diagram shows a visual representation of the classes for this project. There are at total of 10 classes: AnimationBase, LunarX, Landscape, Rocket, RocketShape, Stars, PlayerDisplay, LandingPoint, Zoomer, RocketPiece.



*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, landscape, player display, and the stars. 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 is in charge of calculating the position of the rocket based on the velocity, direction, and user input. Additionally, this class also handles the contact between the rocket and the landscape. If the landing was not safe the class will handle the explosion of the rocket.

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 angle of the rocket will also be calculated based on user input. This class will keep track of the rocket polygon. Finally this class in charge of drawing the rocket on the GUI.

The Landscape class contains the data for the terrain. It will keep track of the the location of the terrain based on x and y values. This class will randomly generate a landscape and is in charge of drawing the landscape every frame. Additionally this class handles the landing zones based on the landscape structure.

The LandingPoint class describes each landing point based on its features. The location of the start points and end point of the landing zone are stored. Additionally the point assigned to each landing zone is calculated within this class.

The PlayerDisplay class displays the points, fuel, altitude, and horizontal and vertical velocity. The class is in charge of drawing all this info within the GUI. Additionally this class handles the start menu of the game. Finally this class keeps track of the player’s points.

The Zoomer class helps zoom into portions of the GUI based on the rocket position. The class has a specific method that helps to zoom into the GUI. Also the class has a method that rotates the GUI objects about the rockets rotation.

The Stars class randomly generates stars within the landscape. The stars class is in charge of drawing the stars every frame with the Graphics object.

The RocketPiece class serves to represent the different rocket pieces when the rocket explodes. The class will generate random velocities of the pieces so that when the rocket explodes the effect is enhanced. This class is in charge of drawing the different rocket pieces to create the explosion effect.

3. Detailed Design

This projects detailed design has been generated from the Javadoc comment in the source files. The detailed design provides a detailed description of each class. Within each class all the fields and methods are described by their respective function.

4. Testing

Proper testing of the game is essential in order to phase out any bugs during the ending stage of the project. The testing was split up into two parts. The first part of the testing utilized the JUnit testing within Java. We tested every single method within the project to ensure that everything was working as intended. Although every method was testing within the JUnit testing the GUI component had to be tested differently. In order to test the GUI component of the project, multiple tests were made and we played the game to ensure that the tests were met sufficiently.