PROGRAMMING IN JAVA – SEMESTER 2 GAME PROJECT

Christopher Michael-Stokes

Table of Contents

Additional Features

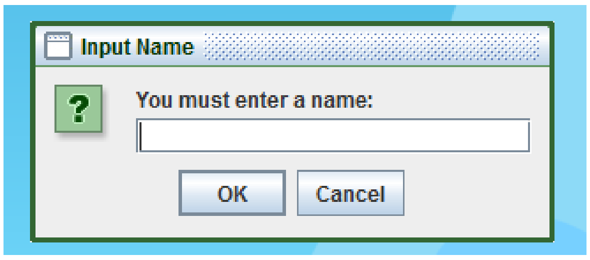
Feature 1 – High Score table

The first of my features is a high score system. In order to facilitate this, I made a high score object that stores player score and a list of previous scores.

Scores are stored in a hash map, in a file “data/scores.txt”, with the key being a unique player name, and the value being their score. When a high score object is instantiated, it tries to read the scores from the file. If for some reason the scores cannot be read, the scores will be reset. Once scores are read, they are then sorted into descending order, and inserted into a linked hash map.

As the game time increases, the score will decrease, in order to reward players who are quick to complete the game. The score will also be effected by the player dying, completing a level, taking damage and damaging an enemy.

When the game is over, either by the player running out of lives or completing the game, an internal prompt will show up, asking the player to input their name (see below).



The name that the player inputs will be checked to make sure that it is not already in the high scores map, and that it is not completely made from spaces.

When the user submits a valid name and clicks ok, the players name and score is added to the score map, then serialized back to the scores file. The scores are then sorted and the top twenty scores are added to a JTable component. The scores are shown to the user, with the user’s scores being highlighted.

If the player chooses not to input a name and clicks cancel, the scores will be shown anyway, but the players score will not be in the list.

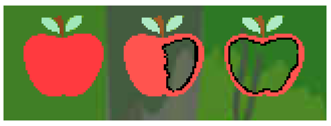
After the player sees their score, they will be asked if they would like to restart the game, or would rather quit. If they choose to restart, their score will be reset and the game will go back to the first level.

Feature 2 – GUI

The GUI for the game was completely coded using swing components. The game was made up of multiple JPanels added to a JLayeredPane. This is to be able to keep components separate from each other. The background and foreground of the level are kept on separate JPanels, meaning that one can be changed independently of the other. In addition, the pause menu is kept in a JPanel that is in the last position, and when the game is paused, the panel is brought to the front to allow it to overlay the other panels.

Health and Projectile Display

Displaying the players health is most simply done by showing the user a number, and whenever they take damage, their health goes down by 1. However, I chose to do things a little differently, rather than having health stored as a number, it is stored as a number of hearts with a number of states. 1 heart is effectively two health, therefore each heart is represented by an integer between 0 and 2, with 2 being a full heart, 1 is a half heart and 0 is an empty heart.



This image shows each state the heart can be in.

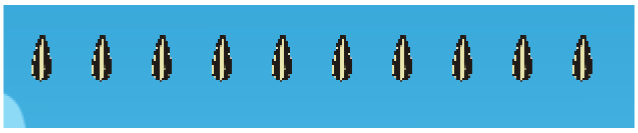
This arrangement of hearts will be displayed as the string "210".

If the health value was stored as an integer, deducting one from the value would be simple to show a loss of health. However, my health value is in a string format, with multiple integers. To deduct a health point from the player, the value on the far right must first be checked. The character at the end of the string is parsed to an integer, then is checked if it is greater than zero; if it is zero, the value to the left is parsed to an int and this is checked. The same cycle is repeated until a value greater than zero is found, or there are no more values to find. Once the value greater than zero is found, one is subtracted from that. Now the value has been modified, it must be put back into the string. There are three positions that the value can be placed, either at the beginning, at the end or in between both. If the value goes at the beginning, the value is cast to a char, then a substring of the health string, starting after the first index, is found and the two are concatenated.

After the health value is calculated, it is then drawn on the screen. In the draw method, each character of the heart value is looked at, and the corresponding image of a heart is added to the heart panel.

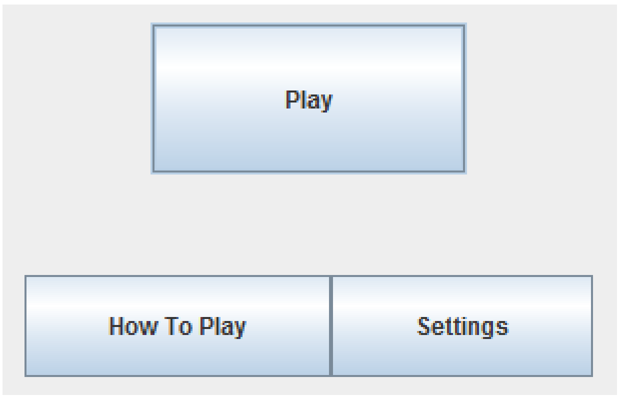
Player projectiles are displayed similarly to hearts, the image of the players projectile will show up as the ammo count to the player, for example, say the player was to have 10 shots; ten scaled versions of the players current projectile will show up under the players health display. Whenever a shot is taken, a projectile will disappear from the display.

When the display is called to be drawn, the players projectile image is obtained, and so is the players shot count. Then the appropriate amount of projectile icons are added to a JLabel, which is then added the player projectile panel. The projectile display is shown below.



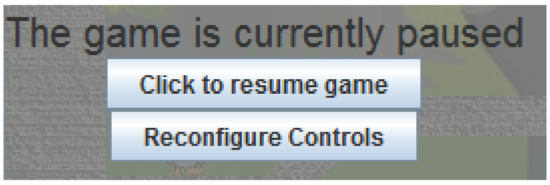
Menus

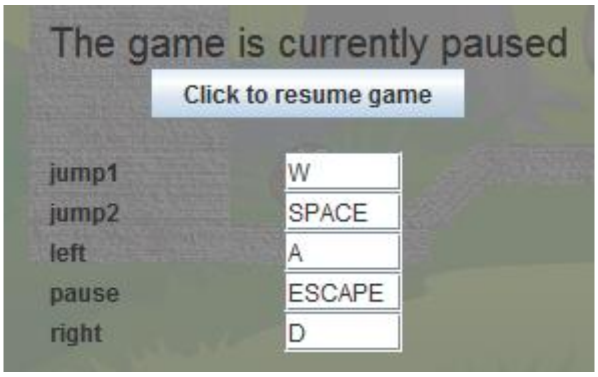
Before the game starts, the user is directed to a starting screen. There are three options in that screen for the user to select; go to the settings menu, go to the how to play menu or play the game.



All of the formatting was made with GridBagConstraints

At any point during the game, the user can press escape (ascii char 27) to pause the game and bring up a menu. On this menu the user has the option to rebind keys, or to continue the game. If the user chooses to rebind keys, the current keybinds will show up and the player can click on any key, then a dialog will show up, then they can type in any key to set the key bind.





The game is currently paused

Click to resume game

jump1	W
jump2	SPACE
left	A
pause	ESCAPE
right	D

Writing the code to allow for the keys to get rebound required the use of jdk8 specific features. This was to use the streams api to make sure that when the user bound a new key, the key was not already bound to something else.

Side feature

Scripted polygon editor

In order to make bodies with an accurate polygon shape, it is necessary to use the given polygon editor. However,
when using the polygon editor, every time you wish to model a new image, you have to open a text editor and replace the line of code that specifies an images directory and you have to set a new scale. While this can be made slightly easier with the use of the net beans ide, since it will compile code on the fly, there is a better way to do it. I chose to make a bash script that will look for an image in a given directory, then ask the user to choose the scale of the image. The script will then edit the PolygonEditor.java.file, compile it then run it with the output of coordinates being stored into a file. The script was written in bash, meaning that it will only work in a Linux environment, or a Windows one with the Linux sub system installed.

Programming challenges

Changing level design

In the Levels.java class, each of the levels are stored as an enum type with a value corresponding to their number, as the enum LevelNumber.LEVEL1 will have a value of 1. The reason for this is to enforce consistency when specifying a level. It is much more apparent when supplying the parameter LEVEL1 as opposed to 1 to tell that I am specifying a level. One of the other reasons of using enumeration was to allow levels to be incremented.

To change the level, a switch statement is called. The statement expects a LevelNumber enum type as a parameter, and will return no output. Most of the code that takes place when levels are changed has been generalized into separate methods in order to reduce code duplication.

When designing this aspect of the game, I thought that the best way to lower cohesion of the level changer would be to make an event to signal that a level is finished. To implement this, I made an event called ChangeLevelEvent, and a listener for said event. Then I made methods in the abstract Level class to allow any level to raise a change level event. In the Levels class, I made a listener for the event, when the listener is triggered, the level number is incremented, and the next level is loaded.

Implementing the listeners requires the use of synchronized methods. These methods were used to add a listener to the list of listeners, and to remove a listener. It is important that these methods are synchronized, otherwise thread errors might arise if listeners are being instanced in separate threads.

Shooting projectiles

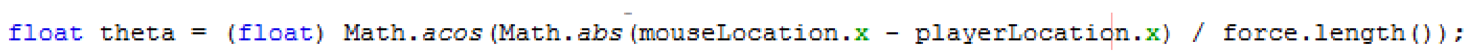
There are two parts to making a character be able to shoot projectiles, the first part is constructing a projectile and the second is working out the direction and speed necessary to shoot the projectile.

The projectile that is shot from the player is a subtype of the DynamicBody object. An attribute that has been added to the class is called shootingBody and is also of type DynamicBody. This is important since, during a collision with another body, if the other body also handles collisions, then it is important to know where the projectile came from to be able to distinguish the action that occurs when it collides. In addition, each projectile object will handle its own collisions. If a projectile hits the body it is shot from, or hits another projectile then the projectile will just bounce. However, were the projectile to hit an enemy or a destructible object or an enemy, the projectile will decrement their health and destroy itself.

The player object will shoot a projectile when the player clicks the mouse in the game window. This allows me to use to position of the mouse to be able to create a vector to direct the projectile. When getting the mouse location, the direction of that vector is relative to the origin of the axis of the point system, however I need the vector to be relative to the player's position, as the projectile will be shot from the player.
The first thing that had to be done was to store the position of the player and the mouse into Vec2 objects. The force of the projectile is stored in a Vec2 object and

is made from the result of the mouse position sub the player position. The force is then normalized to get a unit vector, which can then be multiplied by a constant to ensure that the mouse position and the player position will not affect the speed of the projectile. If I did not do this, then as the distance between the player and the mouse increases, the speed of the projectile would also increase.

Now that the projectile is positioned, I need to correctly set the rotation of the projectile image. To get an angle between the horizontal and the force, I need to use the following formula:



```
float theta = (float) Math.acos(Math.abs(mouseLocation.x - playerLocation.x) / force.length());
```

This formula will only give an angle between zero and pi radians, so the angle must be shifted along the cosine graph to get the correct orientation.