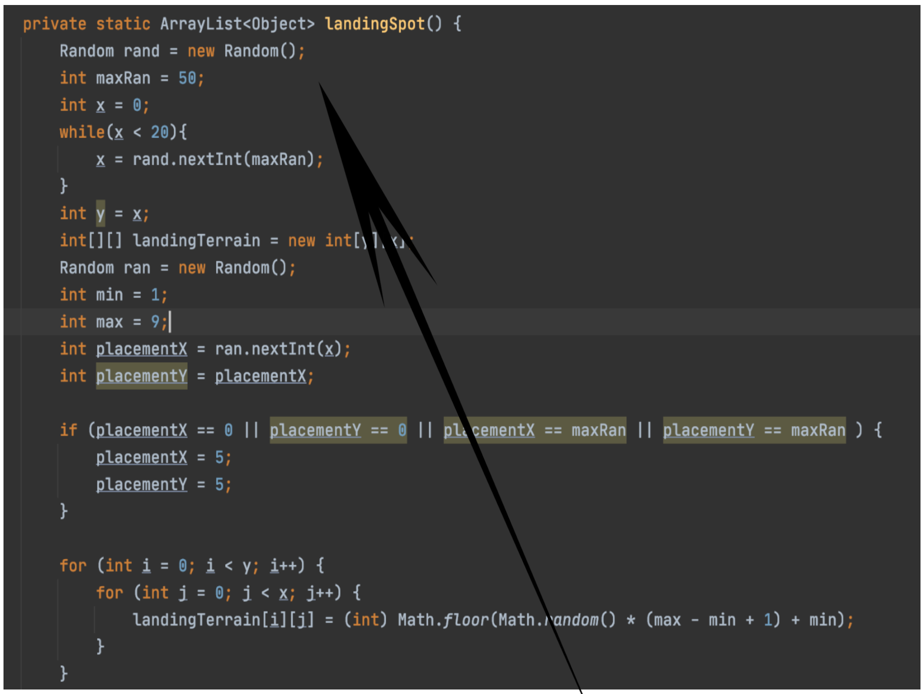
Criterion C: Development

Word Count: 1077

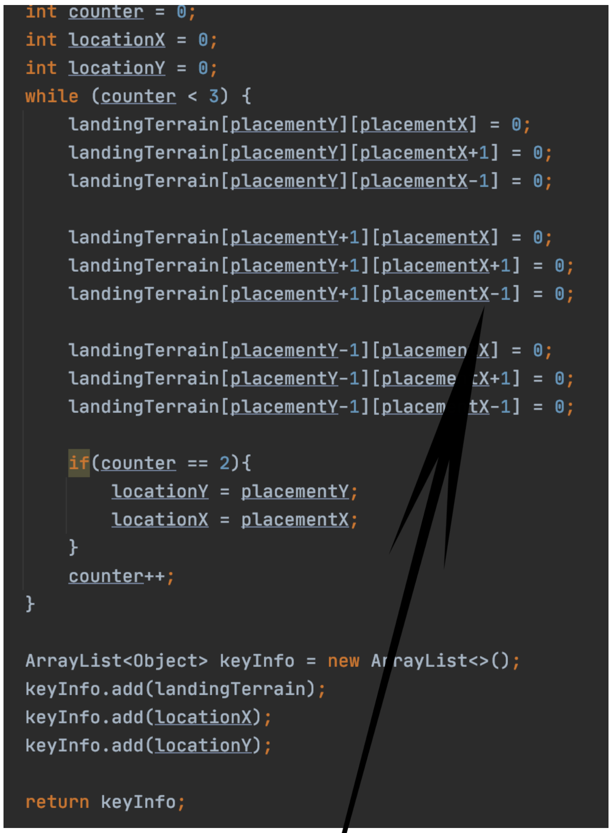
DELIVERABLE 1: SKILLS USED

Skill 1: 2D Array List

Figure 1: Screenshot of Controller Class code (Authors own, 2021)   Figure 2: Screenshot of Controller Class code (Authors Own, 2021)



```java
private static ArrayList<Object> landingSpot() {
    Random rand = new Random();
    int maxRan = 50;
    int x = 0;
    while(x < 20){
        x = rand.nextInt(maxRan);
    }
    int y = x;
    int[][] landingTerrain = new int[y][x];
    Random ran = new Random();
    int min = 1;
    int max = 9;
    int placementX = ran.nextInt(x);
    int placementY = placementX;

    if (placementX == 0 || placementY == 0 || placementX == maxRan || placementY == maxRan ) {
        placementX = 5;
        placementY = 5;
    }

    for (int i = 0; i < y; i++) {
        for (int j = 0; j < x; j++) {
            landingTerrain[i][j] = (int) Math.floor(Math.random() * (max - min + 1) + min);
        }
    }
```



```java
int counter = 0;
int locationX = 0;
int locationY = 0;
while (counter < 3) {
    landingTerrain[placementY][placementX] = 0;
    landingTerrain[placementY][placementX+1] = 0;
    landingTerrain[placementY][placementX-1] = 0;

    landingTerrain[placementY+1][placementX] = 0;
    landingTerrain[placementY+1][placementX+1] = 0;
    landingTerrain[placementY+1][placementX-1] = 0;

    landingTerrain[placementY-1][placementX] = 0;
    landingTerrain[placementY-1][placementX+1] = 0;
    landingTerrain[placementY-1][placementX-1] = 0;

    if(counter == 2){
        locationY = placementY;
        locationX = placementX;
    }
    counter++;
}

ArrayList<Object> keyInfo = new ArrayList<>();
keyInfo.add(landingTerrain);
keyInfo.add(locationX);
keyInfo.add(locationY);

return keyInfo;
```

As seen in figure 1, a randomised terrain had to be created so that the Perseverance rover could be landed; as seen, this is done by the method landingSpot(). The method presents a random variable being used from range 20-50, which is later set as an integer and used to prescribe the size of the 2D array list landingTerrain. Later on, I populate the array list with random integers from 1 to 9.

In figure 2, I create a while loop, choose a random location, and then create a square of 0s (this can be seen by the example below; however, this example is only to present the 0s in reality the 2D array would be bigger).

Figure 3: Non - realistic example of ArrayList (Authors own, 2021)

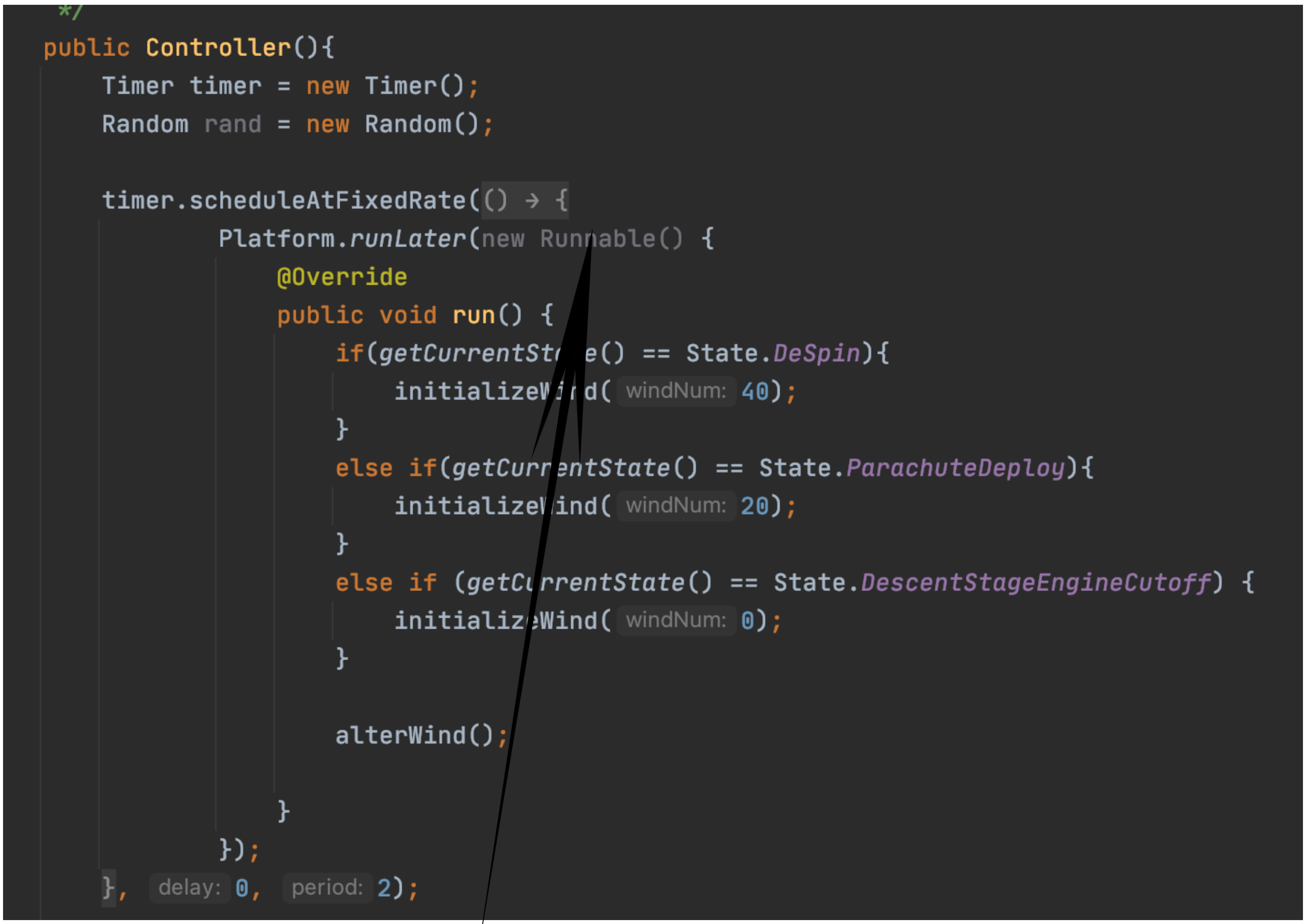
2	4	6	7	5
4	0	0	0	6
3	0	0	0	5
2	0	0	0	3

2	8	5	3	5

The intention of doing this is that later the user will have to search through the 0s since it is the only viable location where the rover can land. I later store the location of the 0s and the ArrayList in an ArrayList of objects. The purpose of doing this is to send both locations and the array to different classes.

Skill 2: Timer

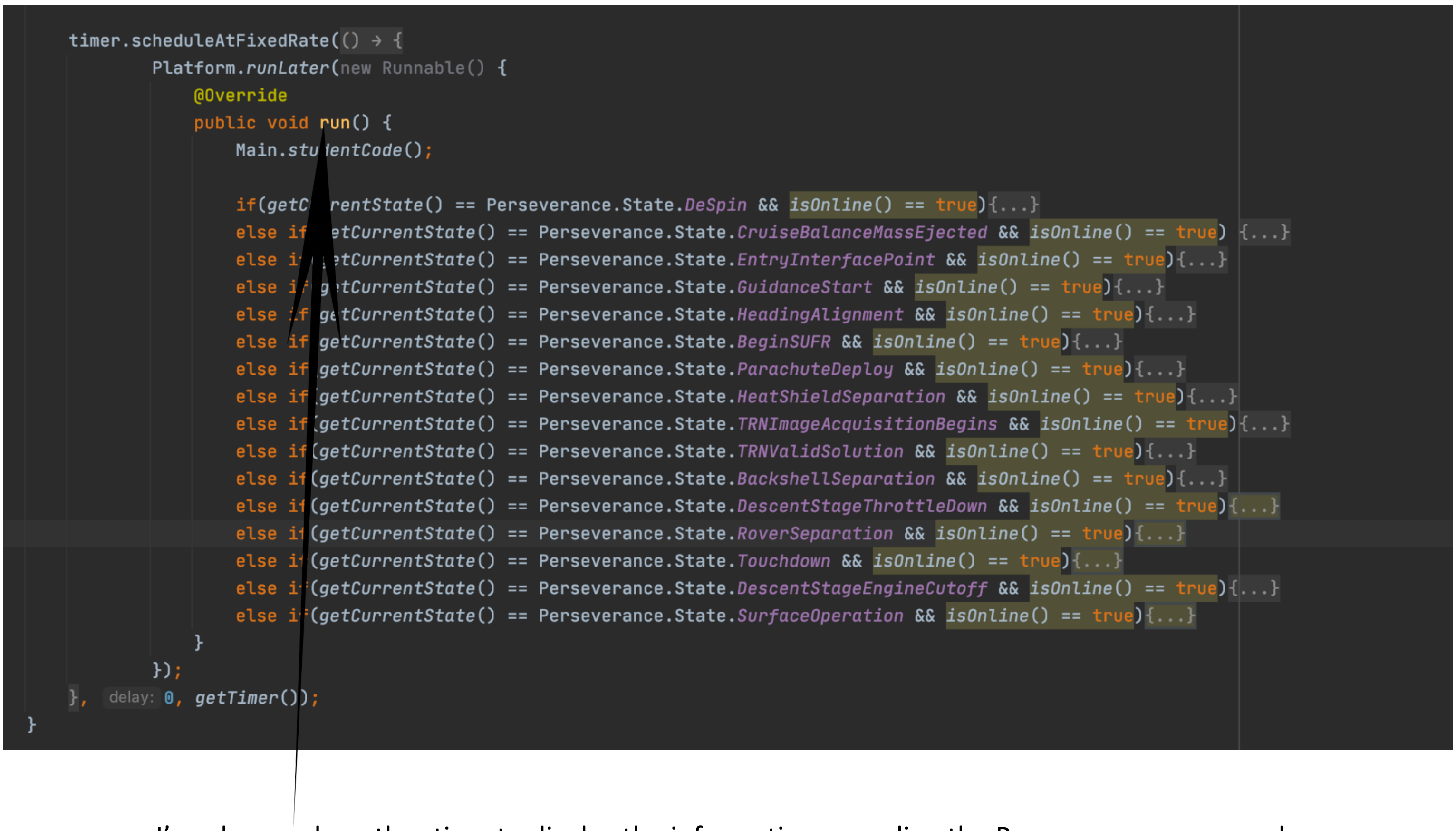
Figure 4: Screenshot of Controller Class code (Authors own, 2021)



```
*/
public Controller(){
    Timer timer = new Timer();
    Random rand = new Random();

    timer.scheduleAtFixedRate(() → {
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    if(getCurrentState() == State.DeSpin){
                        initializeWind( windNum: 40);
                    }
                    else if(getCurrentState() == State.ParachuteDeploy){
                        initializeWind( windNum: 20);
                    }
                    else if (getCurrentState() == State.DescentStageEngineCutoff) {
                        initializeWind( windNum: 0);
                    }

                    alterWind();

                }
            });
    }, delay: 0, period: 2);
```

As seen by figure 3 a timer was set on the controller class. When I first started implementing the GUI I noticed that once the GUI started running I was not able to update my code through the terminal which I was doing before putting everything into the GUI. The purpose of the timer was so that the GUI would be able to update every 2 milliseconds (as seen by the period). This was essential in order to give the GUI some form of animation and new information. In the example above the animation are the moving lines which represent the wind. This was necessary in order to create the illusion that the perseverance rover was indeed falling. As seen the timer is set on a fixed rate and uses the scheduledAtFixedRate which allows a program to run every set period with set delay.

Figure 5: Screenshot of Main Class code (Authors own, 2021)

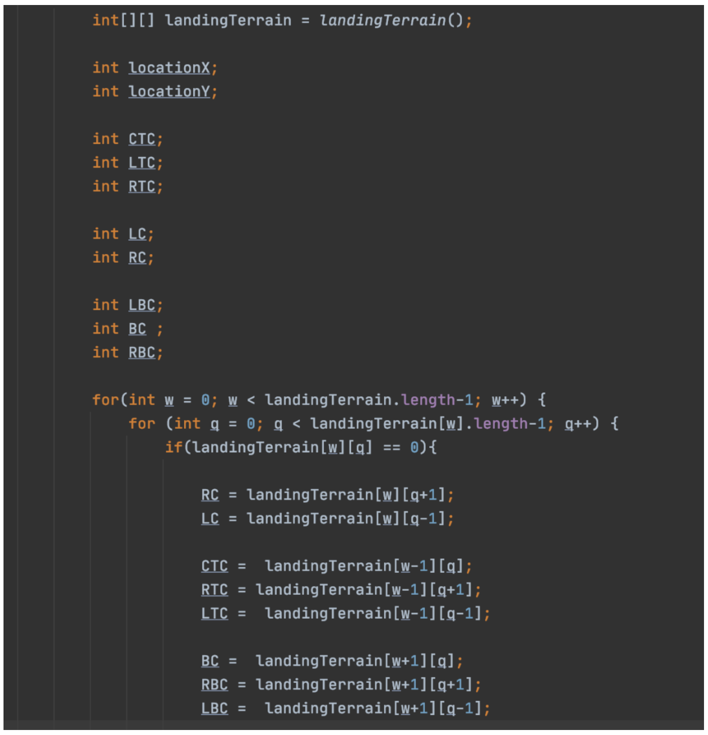


```java
    timer.scheduleAtFixedRate(() → {
            Platform.runLater(new Runnable() {
                @Override
                public void run() {
                    Main.studentCode();

                    if(getCurrentState() == Perseverance.State.DeSpin && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.CruiseBalanceMassEjected && isOnline() == true) {...}
                    else if(getCurrentState() == Perseverance.State.EntryInterfacePoint && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.GuidanceStart && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.HeadingAlignment && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.BeginSUFR && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.ParachuteDeploy && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.HeatShieldSeparation && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.TRNImageAcquisitionBegins && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.TRNValidSolution && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.BackshellSeparation && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.DescentStageThrottleDown && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.RoverSeparation && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.Touchdown && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.DescentStageEngineCutoff && isOnline() == true){...}
                    else if(getCurrentState() == Perseverance.State.SurfaceOperation && isOnline() == true){...}
                }
            });
    }, delay: 0, getTimer());
}
```

I've also used another time to display the information regarding the Perseverance rover and its current state. Two timers where needed because they both have different periods meaning they update at different times. The second timer is updated by the users choice, it decides how fast the simulation will go while the other one updates at a fixed rate because of the animation.

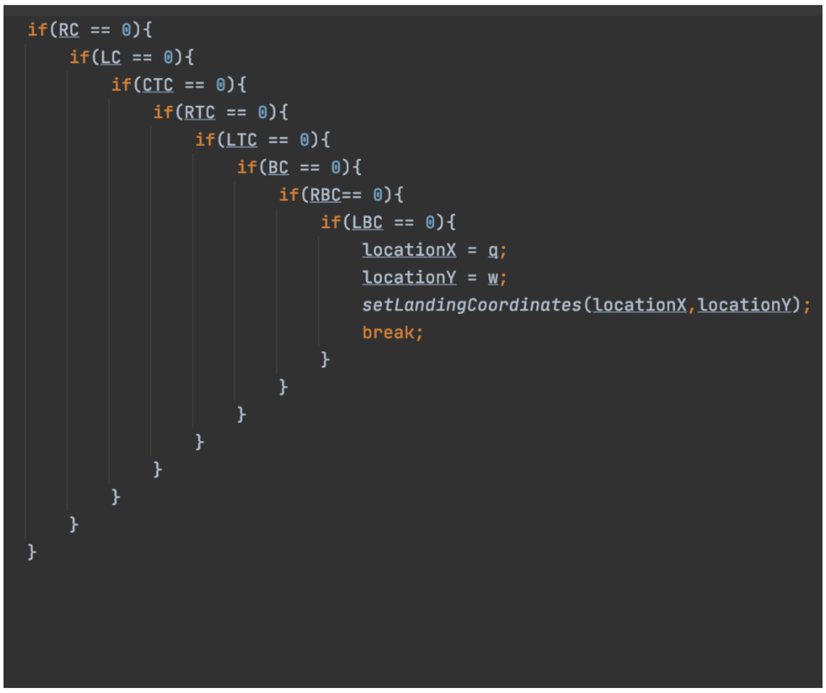
Skill 3: Searching

Figure 6: Screenshot of Main class (Authors own, 2021)



```
int[][] landingTerrain = landingTerrain();

int locationX;
int locationY;

int CTC;
int LTC;
int RTC;

int LC;
int RC;

int LBC;
int BC ;
int RBC;

for(int w = 0; w < landingTerrain.length-1; w++) {
    for (int q = 0; q < landingTerrain[w].length-1; q++) {
        if(landingTerrain[w][q] == 0){

            RC = landingTerrain[w][q+1];
            LC = landingTerrain[w][q-1];

            CTC =  landingTerrain[w-1][q];
            RTC = landingTerrain[w-1][q+1];
            LTC =  landingTerrain[w-1][q-1];

            BC =  landingTerrain[w+1][q];
            RBC = landingTerrain[w+1][q+1];
            LBC =  landingTerrain[w+1][q-1];
```

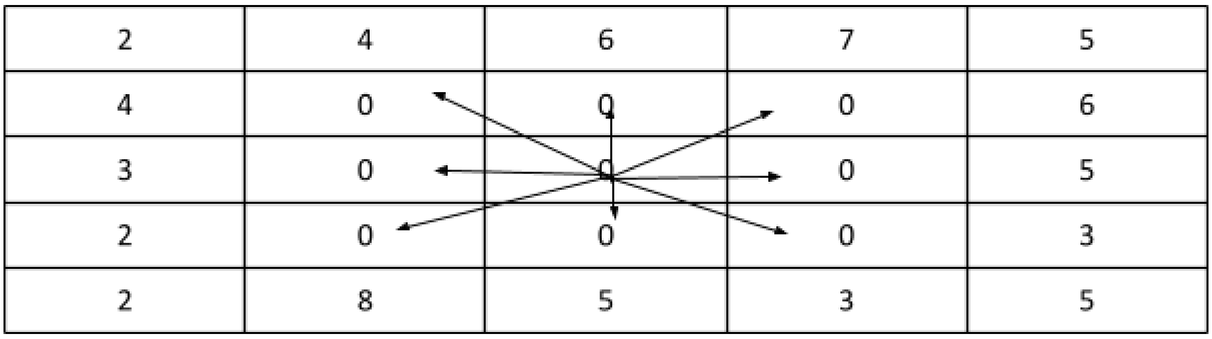
Figure 7: Screenshot of Main class (Authors Own, 2021)



```
if(RC == 0){
    if(LC == 0){
        if(CTC == 0){
            if(RTC == 0){
                if(LTC == 0){
                    if(BC == 0){
                        if(RBC== 0){
                            if(LBC == 0){
                                locationX = q;
                                locationY = w;
                                setLandingCoordinates(locationX,locationY);
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}
```

As mentioned in the 2D Array skill set the randomised ArrayList has a square of 9 0s; in order to find them, the user has to perform a searching algorithm. As seen, firstly, there are a lot of int variables initialised (CTC = Center Top Center, LTC = Left Top Center, etc.) After that, I begin looping through the randomised 2D ArrayList and search for a 0. As seen in figure 7 once I find the zero, I check if the numbers surrounding that zero are zero as well (presented in the diagram below).

Figure 8: Non - realistic example of ArrayList (Authors own, 2021)

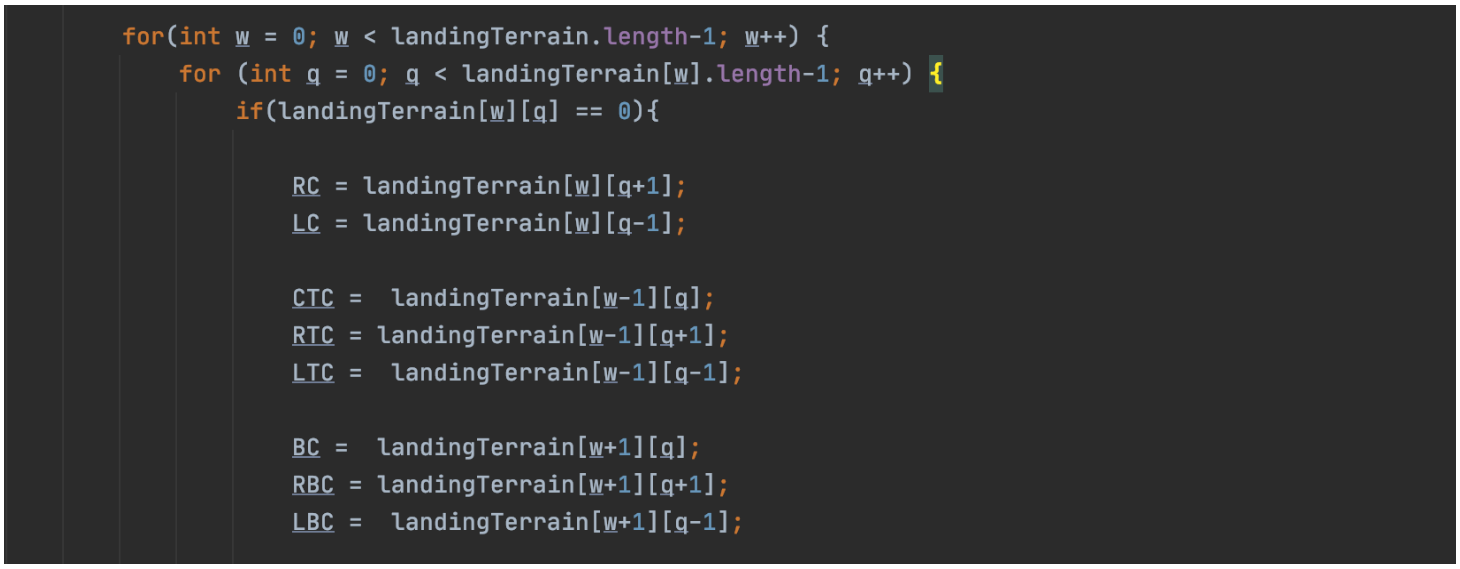


2	4	6	7	5
4	0	0	0	6
3	0	0	0	5
2	0	0	0	3
2	8	5	3	5

Once the check is complete the user calls a method and gives the locationX and locationY of the 2D arrayList as parameters. This will then allow the Perseverance rover to land on a flat surface and to move into the next stage.

Skill 4: Nested Loops

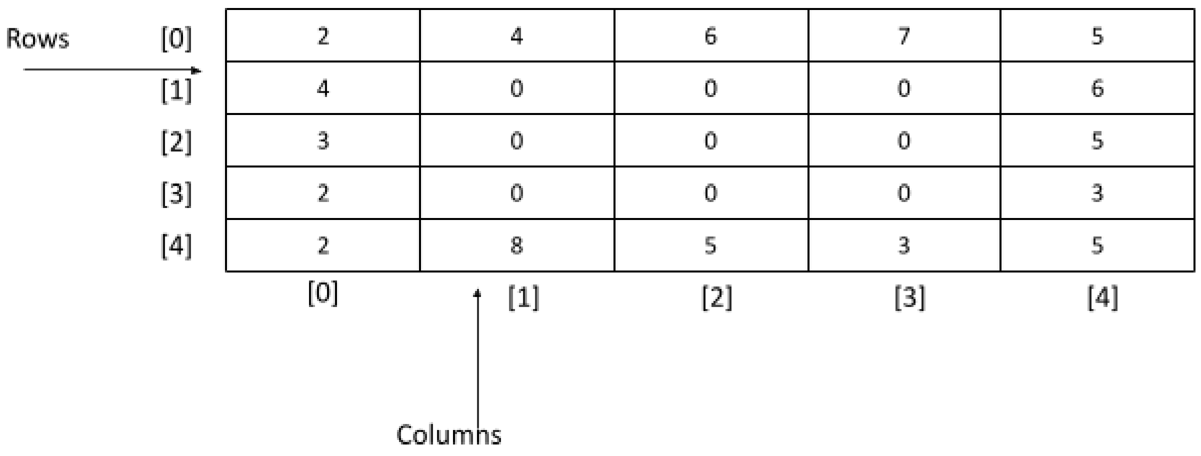
Figure 9: Screenshot of Main class (Authors own, 2021)



```
for(int w = 0; w < landingTerrain.length-1; w++) {
    for (int q = 0; q < landingTerrain[w].length-1; q++) {
        if(landingTerrain[w][q] == 0){

            RC = landingTerrain[w][q+1];
            LC = landingTerrain[w][q-1];

            CTC =  landingTerrain[w-1][q];
            RTC = landingTerrain[w-1][q+1];
            LTC =  landingTerrain[w-1][q-1];

            BC =  landingTerrain[w+1][q];
            RBC = landingTerrain[w+1][q+1];
            LBC =  landingTerrain[w+1][q-1];
```

As seen by figure 9 nested for loops were used in order to access the landingTerrain[][] 2D arrayList. As previously explained this was implemented so that the user would be able to search through the 2D arrayList and check for a flat square landing zone so that the Perseverance Rover can land properly. Since the 2D arrayList in this instance is square then iterating through it should not be too complicated as seen above. The example above demonstrates a simple example of how the nested loop first looks at the row and then at the column. This can be seen in the example below.

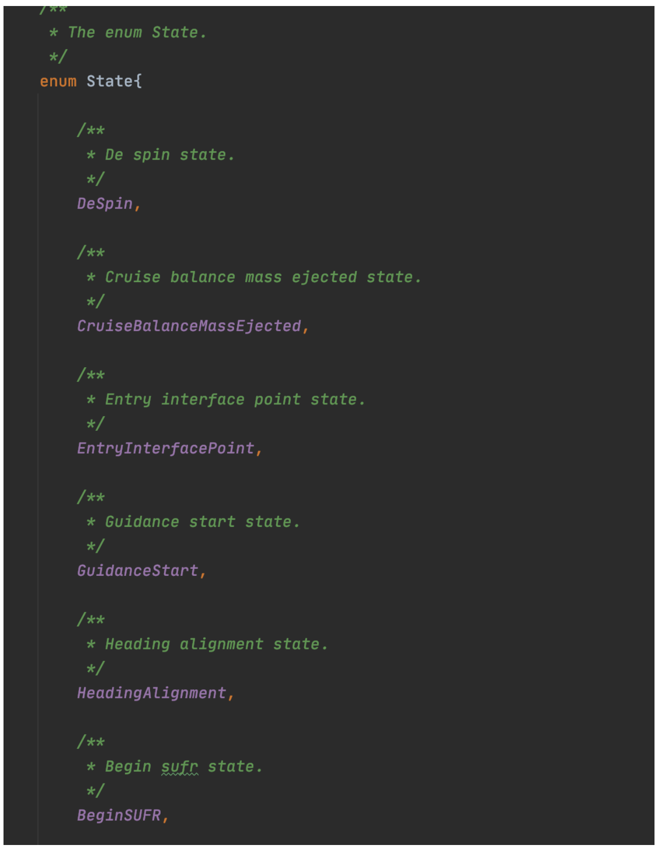
Figure 10: Non - realistic example of ArrayList (Authors own, 2021)



Rows					
[0]	2	4	6	7	5
[1]	4	0	0	0	6
[2]	3	0	0	0	5
[3]	2	0	0	0	3
[4]	2	8	5	3	5
	[0]	[1]	[2]	[3]	[4]

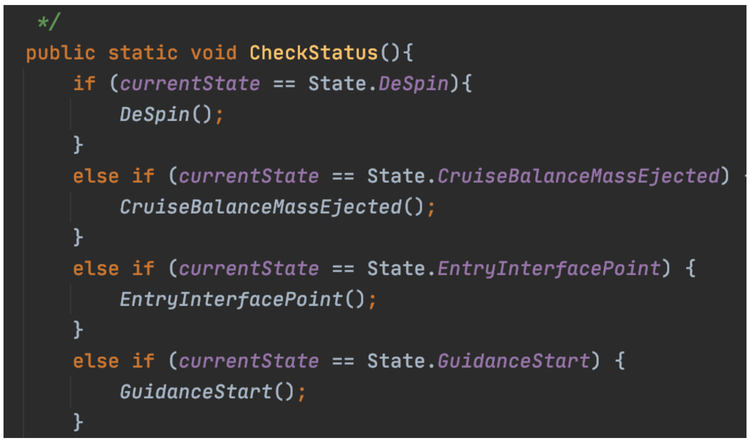
Skill 5: Enum State

Figure 11: Screenshot of Perseverance class (Authors own, 2021)



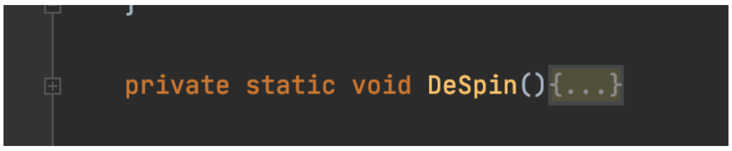
```
/**
 * The enum State.
 */
enum State{

    /**
     * De spin state.
     */
    DeSpin,

    /**
     * Cruise balance mass ejected state.
     */
    CruiseBalanceMassEjected,

    /**
     * Entry interface point state.
     */
    EntryInterfacePoint,

    /**
     * Guidance start state.
     */
    GuidanceStart,

    /**
     * Heading alignment state.
     */
    HeadingAlignment,

    /**
     * Begin sufr state.
     */
    BeginSUFR,
```

Figure 12: Screenshot of Perseverance class (Authors own, 2021)



```
 */
public static void CheckStatus(){
    if (currentState == State.DeSpin){
        DeSpin();
    }
    else if (currentState == State.CruiseBalanceMassEjected)
        CruiseBalanceMassEjected();
    }
    else if (currentState == State.EntryInterfacePoint) {
        EntryInterfacePoint();
    }
    else if (currentState == State.GuidanceStart) {
        GuidanceStart();
    }
```

Figure 13: Screenshot of Perseverance class (Authors own, 2021)

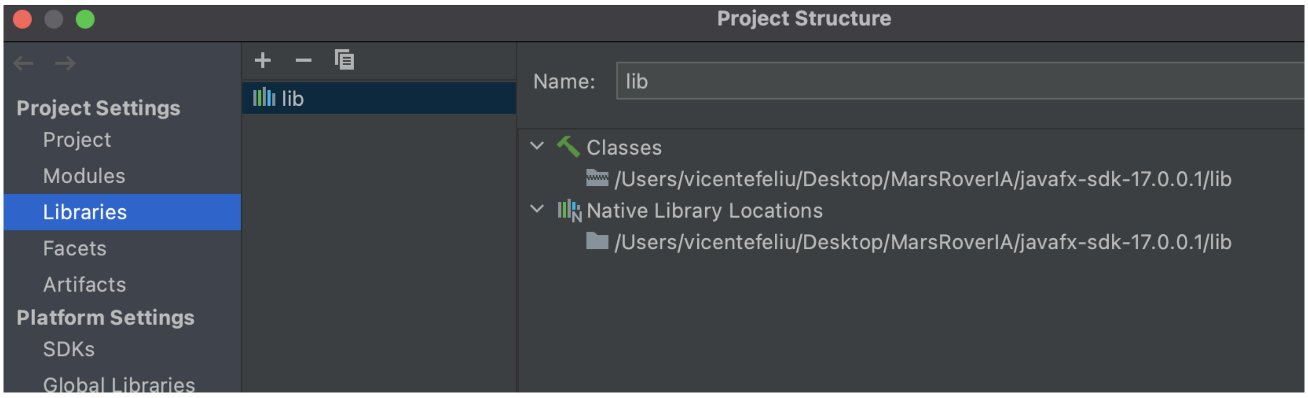


```
    private static void DeSpin(){...}
```

As seen by figure 11 enum class is what is used as the base of the whole simulation. I used enum in order to state all the landing stages of the Mars Perseverance landing simulation. Since enum states have only one instance of “constants in the JVM” (bealdubg, 2021) it further allowed me to create a function checkStatus() where the currentState gets compared to the different states (seen in figure …). If the currentState was the same as the state which it was being compared to then a function would be called, these functions were the landing stages themselves. Inside these functions I was able to include information/actions about that stage, and then move on to the next one by setting the currentState to the next stage. I also compare states in both Controller and Main class because depending on the state certain information must be shown or certain action must be taken.

Skill 6 & 7: GUI and Imported Libraries

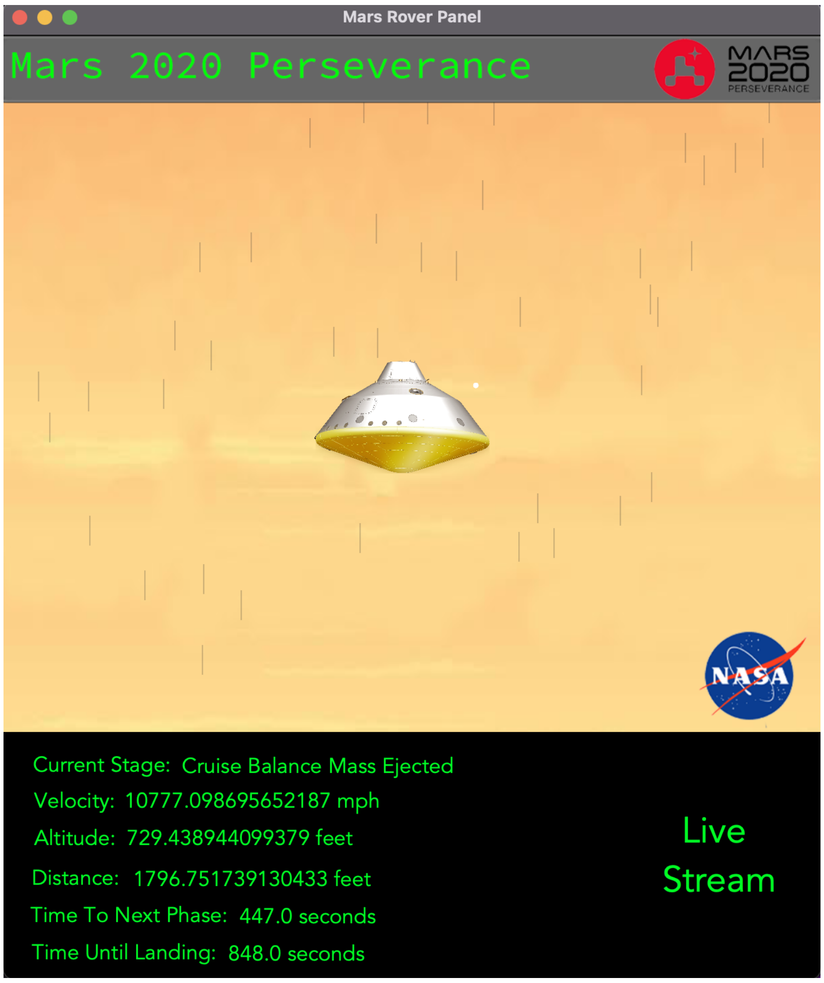
Figure 14: Screenshot of Project Structure (Authors own, 2021)



Name: lib

Classes
/Users/vicentefeliu/Desktop/MarsRoverIA/javafx-sdk-17.0.0.1/lib
Native Library Locations
/Users/vicentefeliu/Desktop/MarsRoverIA/javafx-sdk-17.0.0.1/lib

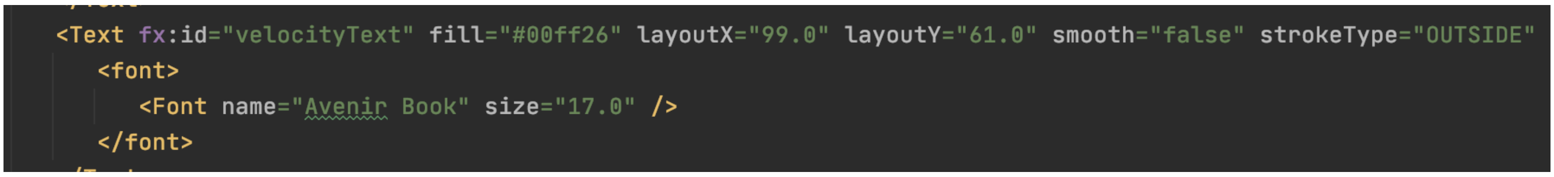
Figure 15: Screenshot of Perseverance class (Authors own, 2021)



Current Stage: Cruise Balance Mass Ejected
Velocity: 10777.098695652187 mph
Altitude: 729.438944099379 feet
Distance: 1796.751739130433 feet
Time To Next Phase: 447.0 seconds
Time Until Landing: 848.0 seconds

The graphical user interface served the purpose of demonstrating a 2d visual representation of the perseverance mars rover simulation landing. As seen the GUI presents to the user the Mars perseverance rover and a set of key information which gets updated every second as previously mentioned on the timer skill. This is important to the user because it presents information which helps them on their challenge to programme the actions which the rover must take, additionally it provides the user to know what stage in their code does not work. I was able to achieve such a GUI with the use of an imported Javafx SDK. The Javafx SDK library allowed me to connect the IA project to SceneBuilder. SceneBuilder is an application that allows developers to design user interfaces in a simple way. As seen most of my application is built with textLabels and imageViews put on top of eachother.

Figure 16: Screenshot of Project Structure (Authors own, 2021)

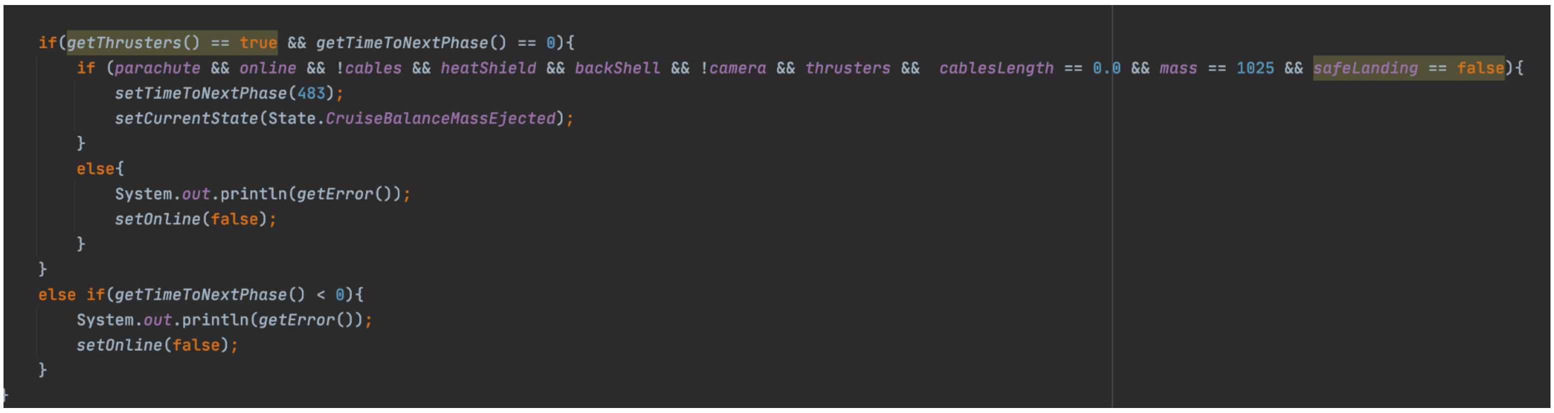


```
<Text fx:id="velocityText" fill="#00ff26" layoutX="99.0" layoutY="61.0" smooth="false" strokeType="OUTSIDE"
   <font>
      <Font name="Avenir Book" size="17.0" />
   </font>
```

Scenebuilder then allows the developer to connect these textLabels and imageViews to the project by giving them code fx:id, as seen by figure 16. Which then gets connected to the controller class where it is then coded.

Skill 8: Complex Selection & Nested if statements

Figure 17: Screenshot of Perseverance class(Authors own, 2021)



```java
    if(getThrusters() == true && getTimeToNextPhase() == 0){
        if (parachute && online && !cables && heatShield && backShell && !camera && thrusters &&  cablesLength == 0.0 && mass == 1025 && safeLanding == false){
            setTimeToNextPhase(483);
            setCurrentState(State.CruiseBalanceMassEjected);
        }
        else{
            System.out.println(getError());
            setOnline(false);
        }
    }
    else if(getTimeToNextPhase() < 0){
        System.out.println(getError());
        setOnline(false);
    }
}
```

As seen in figure 17 I had to use multiple conditions in the if statement in each stage of the landing throughout the landing of the perseverance rover. The reason for this was that I had to check that the student coding the challenge does not change the value of a variable since that would mean that the rover would crash. Additionally, I also used nested if statements in order to make sure that certain changes were made on that stage. Although it would have been more efficient to code everything in one if statement, this allowed the code to be more organised in terms of what should be constant in that stage and what should be changed.

REFERENCES

baeldung (2016) *A Guide to Java Enums*, *Baeldung*. Available at: https://www.baeldung.com/a-guide-to-java-enums (Accessed: 10 January 2022).

javaTpoint Team (no date) *Java Timer scheduleAtFixedRate() method - javatpoint*, *www.javatpoint.com*. Available at: https://www.javatpoint.com/post/java-timer-scheduleatfixedrate-method (Accessed: 10 January 2022).