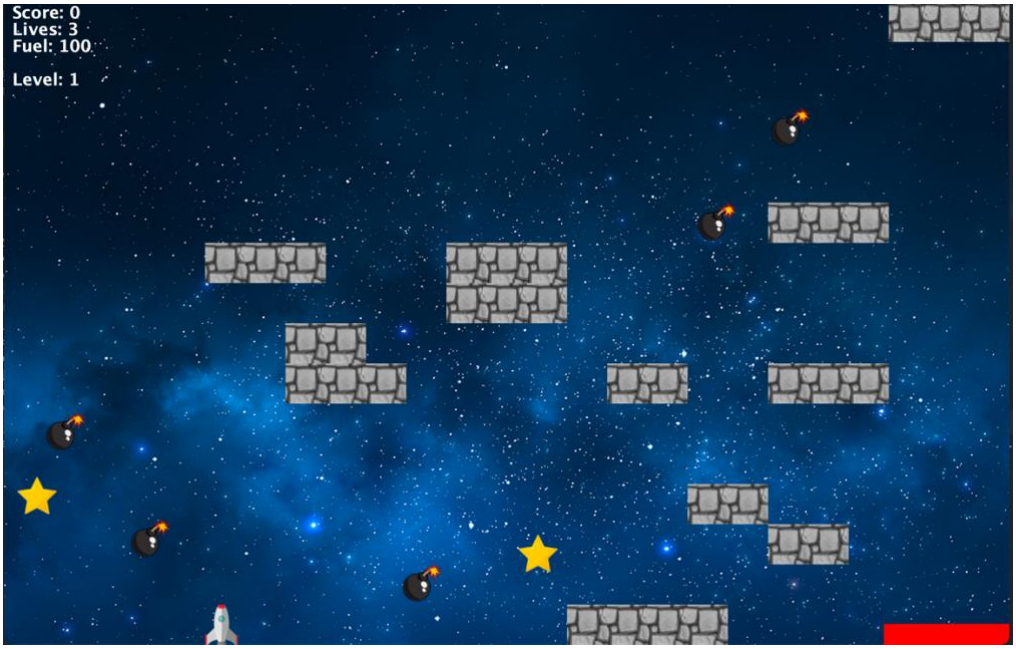
Rocket Blast

Group Members – Siddharth Narsipur

1 - About the game

My game is Rocket Blast, a slightly modified version of the Drone Pilot game described by the professor.



The player, using either the arrow keys or WASD, must control a rocket ship. The left and right keys rotate the rocket, and the up and down keys move it forward or backwards.

If no key is pressed, the rocket falls downward due to gravity.

There are two obstacles – rocks/walls which are pre-generated and bombs which are randomly generated. Hitting any of the obstacles reduces your life by 1 and returns the ship to the starting position.

There are also randomly generated stars that appear. Hitting a star gives the user 10 points.

The game ends when the rocket lands on the landing zone. The player needs to land with a low speed and the right orientation of the rocket (straight) to land successfully or else it crashes.

The game also tracks fuel. Fuel is consumed when the rocket rotates or moves forward or backward. If the rockets run out of fuel, the game ends.

2 – Flourishes

I implemented 2 flourishes.

Randomized stationary obstacles that destroy the drone – These are the bombs.

```
public void generateBombTiles(){
        for (int i = 0; i < lm.gameLevels[lm.currentLevel].bombNo;) {
            int randomX = random.nextInt( bound: gp.screenWidth-gp.tileSize + 1 - gp.tileSize) + gp.tileSize;
            int randomY = random.nextInt( bound: gp.screenHeight-gp.tileSize + 1 - gp.tileSize) + gp.tileSize;
            if (!CheckCollision.checkObjectCollision(new Point(randomX, randomY), objectSolidAreaArray, gp)) {
                bombTileArray.add(new Point(randomX, randomY));
                objectSolidAreaArray.add(new Rectangle(randomX, randomY, gp.tileSize, gp.tileSize));
                i++;
            }
        }
}
```

I generate random X and Y coordinates (within Screen width and height) and then checks if a bomb or wall already there. If there isn't, I add the bomb's coordinates to an array of Objects (walls and bombs) which I later use for drawing.

Non-stationary objects, like clouds, spaceships, or twirling stars, that score extra points (and vanish) if hit by the drone – These are the stars.

Although not required, these are also randomized. The direction variable controls whether it starts on the right or left side of the screen.

```
public void generateStars(){

    int direction = random.nextInt( bound: 2 + 1 - 1) +1;
    int randomX;
    int randomY;

    switch (direction) {
        case 1 -> {

            randomX = 0;
            randomY = random.nextInt( bound: gp.screenHeight-gp.tileSize + 1 - gp.tileSize) + gp.tileSize;
            starTileArray.add(new Object(randomX, randomY,  speed: 1, direction));
        }
        case 2 -> {
            randomX = gp.screenWidth-gp.tileSize;
            randomY = random.nextInt( bound: gp.screenHeight-gp.tileSize + 1 - gp.tileSize) + gp.tileSize;
            starTileArray.add(new Object(randomX, randomY,  speed: 1, direction));
        }
    }
    }
```

The function below check if the rocket has collided with the star. If it has, the star is removed from the star array (making it vanish) and the player's score is increased by 10.

```
public static boolean checkStarCollision(PlayerControlledObject object1, GamePanel gp){
    boolean flag = false;
    for(int i = 0; i < gp.gameObjects.starTileArray.size(); i++){
        if(object1.solidArea.intersects( gp.gameObjects.starTileArray.get(i).solidArea) ){
            gp.gameObjects.starTileArray.remove(i);
            i--;
            flag = true;
        }
    }
    return flag;
}
```

The function below changes the position of the star before I draw them again.

```
public void changeStarSpeed() {
    try{
    for(Object object : starTileArray) {
        if (object.direction == 1) {
            object.x += 2;
            object.solidArea = new Rectangle(object.x, object.y, gp.tileSize, gp.tileSize);
        }
        if (object.direction == 2) {
            object.x -= 2;
            object.solidArea = new Rectangle(object.x, object.y, gp.tileSize, gp.tileSize);
        }
        if(object.x >= gp.screenWidth || object.x <= 0){
            starTileArray.remove(object);
        }
    }}catch (Exception ignored){}
}
```

3 – Requirements

There is animation – The main player-controlled rocket and the star objects are both animated and change position.

It is interactive (i.e., keyboard/mouse.) – The player controls the rocket using arrow keys or WASD.

There is a scoring mechanism and it is shown to the player – The score is shown on the screen. The player gets 10 points if they hit a star and bonus points at the end of the level depending on the time it took to complete the level.

There is a definitive ending mechanism (i.e., you can win or lose) – The player loses a life if they crash into an object (rock, bomb) or into the landing pad. The game ends if they run out of lives. The player can finish a level if they land the rocket on the landing pad. The player wins and completes the game if they finish each level.

There is a physical mechanism (i.e., velocity, mass, motion, gravity) – The rocket has velocity, motion, and gravity.

There is collision detection (i.e., walls, items, rewards, hazards.) – There is edge detection with the rocket so it cannot fly of screen. There is also collision detection between the rocket and the rocks, bombs, and the stars.

It is creative – I cannot judge this ☺.

You implement one flourish per team member (randomized collidables, two-player mode, sound effects, high scores list, it looks really good, or some other tricky thing.) – Done, as mentioned above.

4 – How to run

Go to the terminal and with RocketBlast/src as the root folder, run:

javac RocketBlast.java
java RocketBlast

5 – Class Descriptions

CheckCollision – Checks for collision between objects or between objects and the rocket

GamePanel – The JPanel on which the game and menu is drawn. The class controls the game thread that runs the game. The class also has functions that control moving between levels.

KeyHandler – A simple keyhandler and mouselistener that checks if WASD/Arrow is clicked for rocket movement or if menu buttons are clicked.

Level – My class that describes a level and its instance variables.

LevelManager – Stores an array of levels that are used for the games and has variables for the current level and menu messages.

RocketBlast – The class with a main method. Creates a JFrame and adds GamePanel, then starts GamePanel's game thread.

Object – My class that describes an object and its instance variables.

ObjectManager – Handles the creation and drawing of all the objects (rocks, bombs, stars) in the game. Bombs and stars are created randomly. Rocks are created based on a map that I drew.

PlayerControlledObject – My class that describes a PlayerControlledObject and its instance variables.

Rocket - Extends PlayerControlledObject. Handles the creation, drawing and movement of the rocket. Uses the classes above to check for object collisions, edge collision and level completion.

GameScreen – Draws the score, fuel, lives, and level on the game screen.

Menu – Draws the menu screen before, between and after the game.