

Java Asteroids application

MODULE 10

SHERIDAN,MORGAN

Contents

1. Project Overview	2
2. Project Requirements.....	2
2.1. Derived Requirements	3
3. Design Plans	3
3.1 Use case diagram.....	6
4. Implementation.....	10
4.1 Collision detection.....	10
5. Testing	12
Appendix A. Traceability matrix.....	12
Appendix B. Screens of the working app.....	13

1. Project Overview

This Application is my take on the asteroids video game. The main line is used to create the JFrame and start the Timer. I use a draw class to paint to the frame the objects I need. My player class contains the methods needed to run my game, and the information about ship (this may become 2 classes by completion of this project). The shot class contains positional information about the shots the sip can fire. The asteroid class contains positional information, and check for collisions.

2. Project Requirements

Start a document which will define a "Pong" application.

- Requirements Analysis
 - Define the requirements and derived requirements
 - Define **Use Cases**
 - Set up traceability matrix per all requirements
 - Show at least one Activity Diagram
- Implementation
 - Show design:
 - **Class diagram**
 - **Sequence diagram**
 - General screen layout
 - Show relevant code snippets
 - creation of objects
 - collision code
 - show that all requirements are met with code (traceability matrix)
- ~~Verification and Testing~~
 - ~~Show Unit testing of collision method.~~
 - ~~Show screens of working app.~~

Create a "Pong" application using OOP techniques learned so far, in particular, use base and derived classes for display objects on the screen.

Game Play (can really be anything you negotiate)

- keyboard and/or mouse inputs
- each player gets paddle on each side (could be one-player version)
- ball bounces back and forth as players use paddle to deflect ball back
- each time a player misses, the ball passes the paddle to score a point, and ball is lost, new ball is entered
 - After each point scored:
 - ball speed increases.
 - number of balls increases (balls bounce off each other, and paddle)
- Keep track of scores...20 points to win game (...or whatever you want).
- Keep track of 5 top high scores (save file on disk, ask for player's name if they are top 5)

- If player is in top score, they enter their name in top score ranking.
- Derived requirements detail the layout of the screen, flow of top score updates

Course Objectives for Game:

- You must use classes with inheritance, per assignments leading up to this one.
- You must use an Interface...somewhere...
- Implement JavaDocs for all code (complete this with minimal effort at the right time)
- Do complete Unit Testing only for the collision code method (as many collision code methods as you have)
- SAAD1001 => Define requirements, create UML diagrams, Use cases,per SAAD1001 assignment

2.1. Derived Requirements

In this project I will need Eclipse as a workspace, Git to upload my files, Object AID to build my class diagram and visio or draw.io to build my sequence and activity diagram. Junit to test my program.

3. Design Plans

The project at it's current stage has 6 classes (figure 1). I plan to split 1 of the classes and possibly implement a position interface. The main line starts the timer and makes the JFrame and the draw class. The draw class creates a player starts it and adds it to action listener. The player class contains a run element that makes the objects move and checks for collisions. Below you will find an activity diagram demonstrating the start sequence of my application (figure 2). Following that will be a sequence diagram demonstrating the collision detection (figure 3). (screen layout figure 8 & 9)

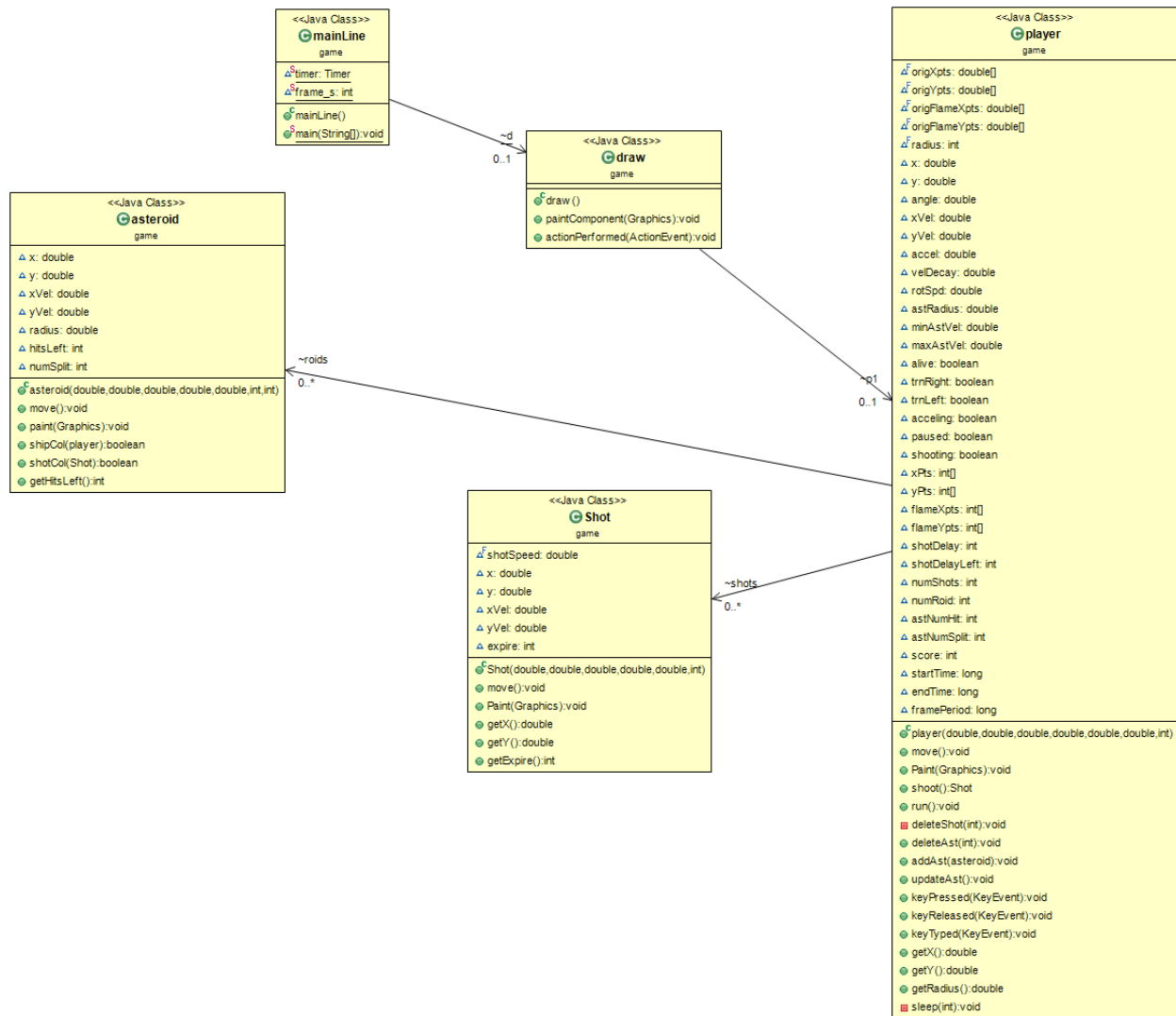


Figure 1 Class Diagram

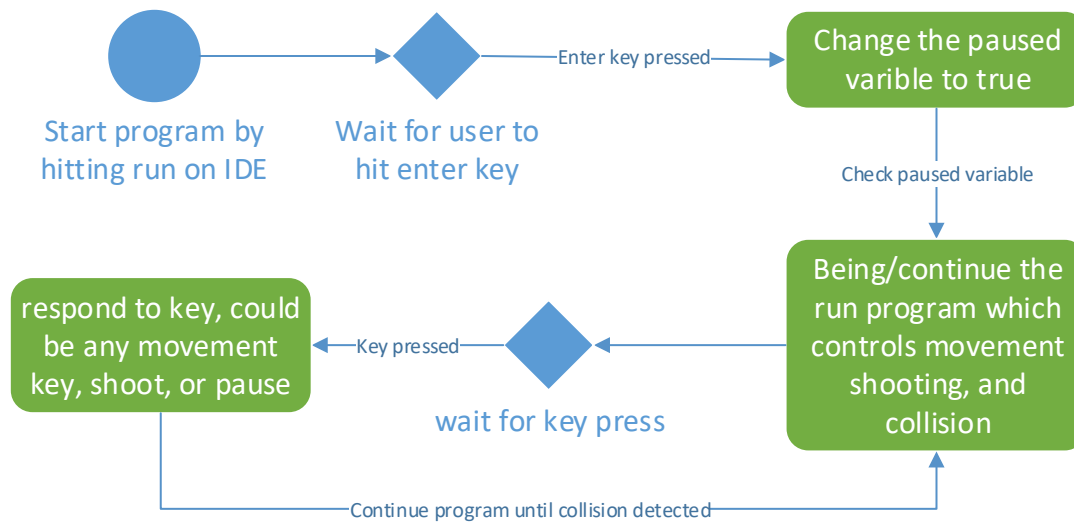


Figure 2 activity diagram of app start-up

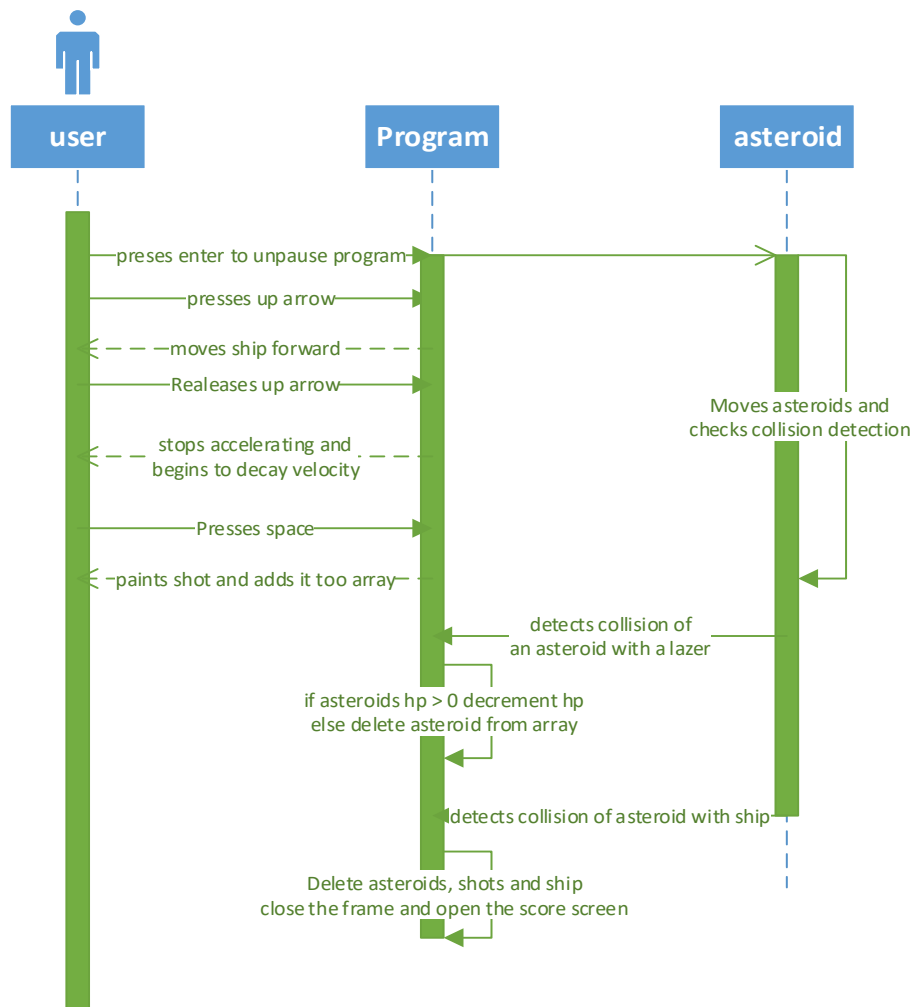


Figure 3 Sequence diagram of collisions

3.1 Use case diagram

Use case diagrams are an effective and dynamic tool used to demonstrate the way a user will interact with a system. A use case diagram should demonstrate what the program must do and how the user can affect that. They should be simple to understand without technical knowledge on the project. The simple nature of use case diagrams makes them ideal for showing stakeholders. Use case diagrams contain actors that represent people, companies, programs, etc. These actors have dependencies on various different use cases. Some use cases are contained within a subsystem. See my use case diagrams below (figures 4-7).

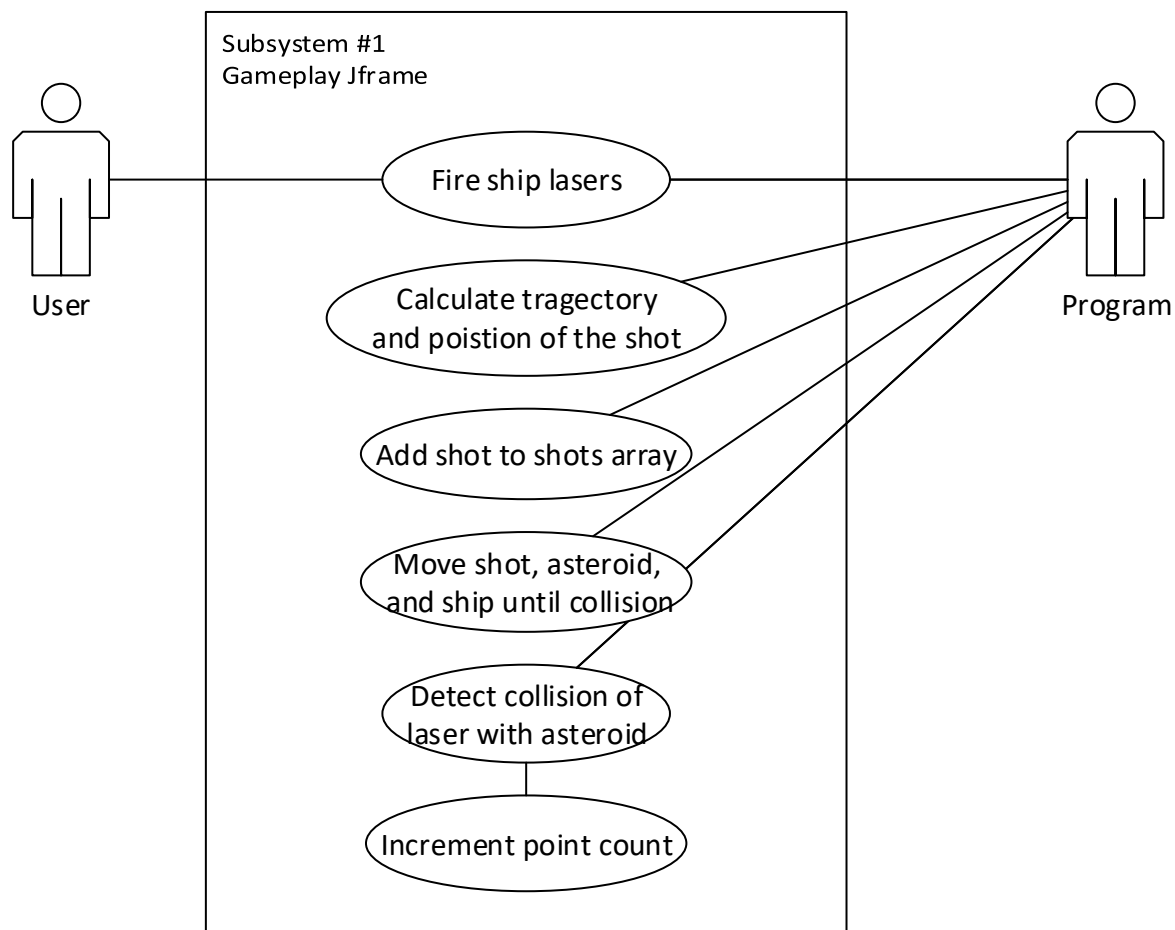


Figure 4 Scoring a point use case

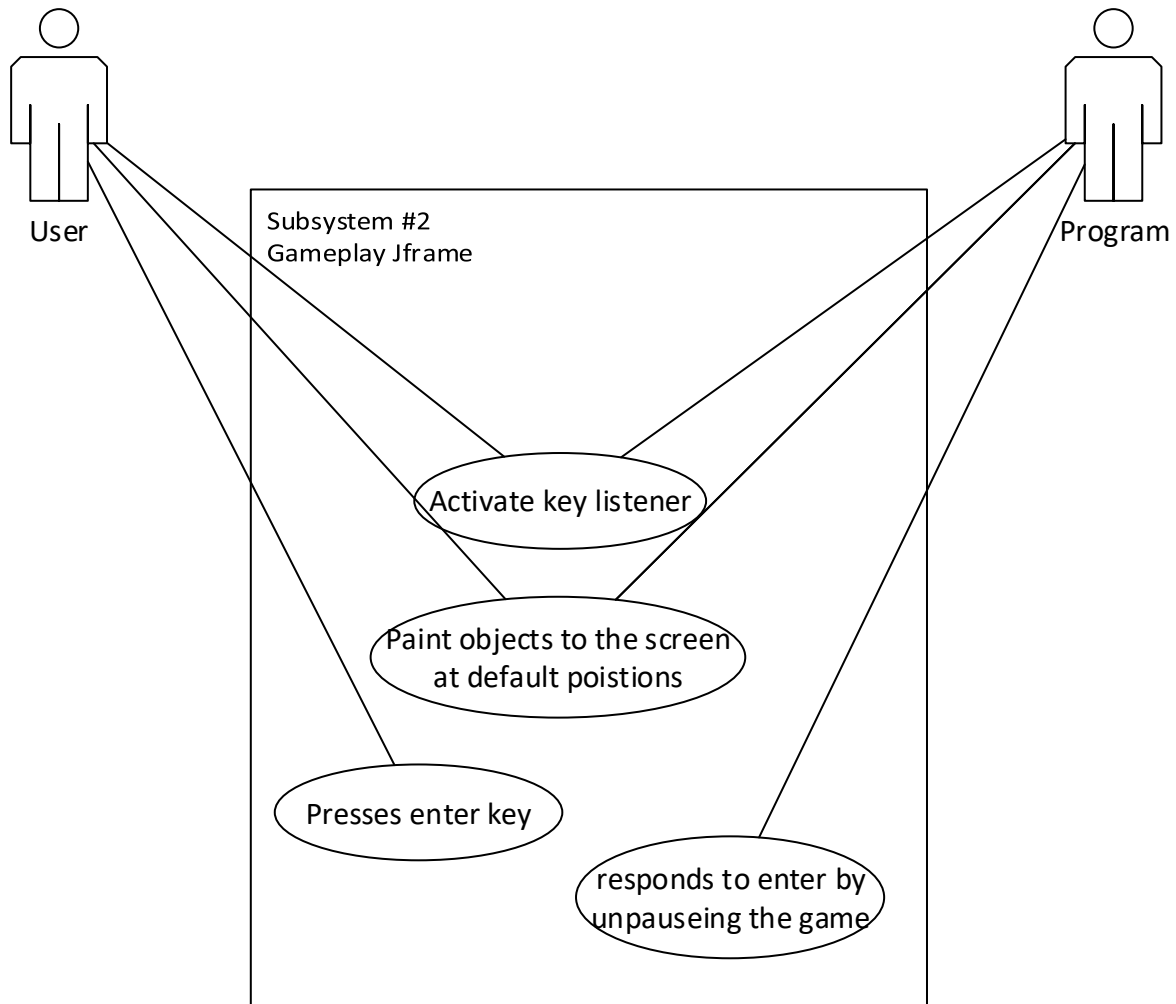


Figure 5 Start game use case

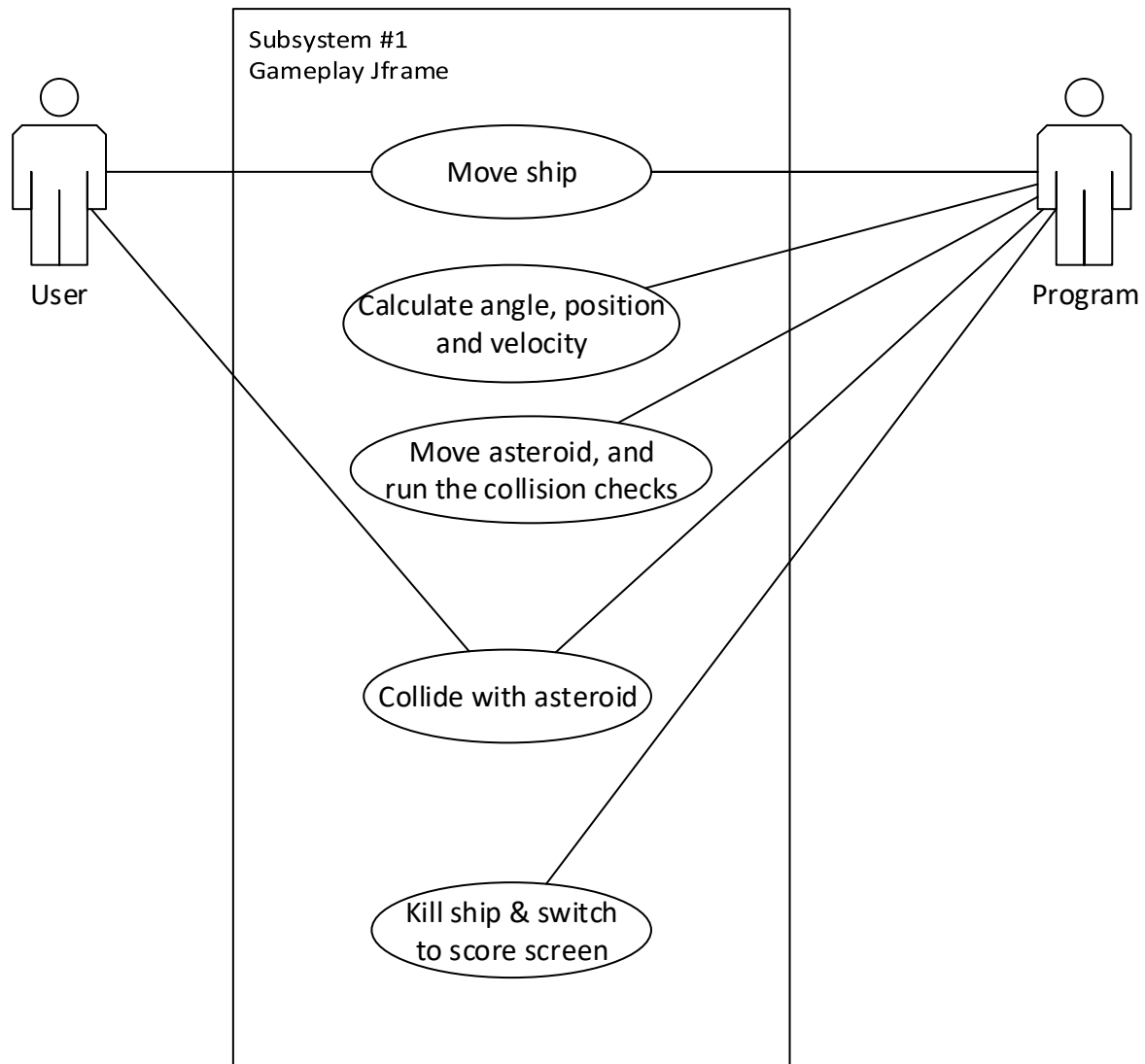


Figure 6 Collide with asteroid use case

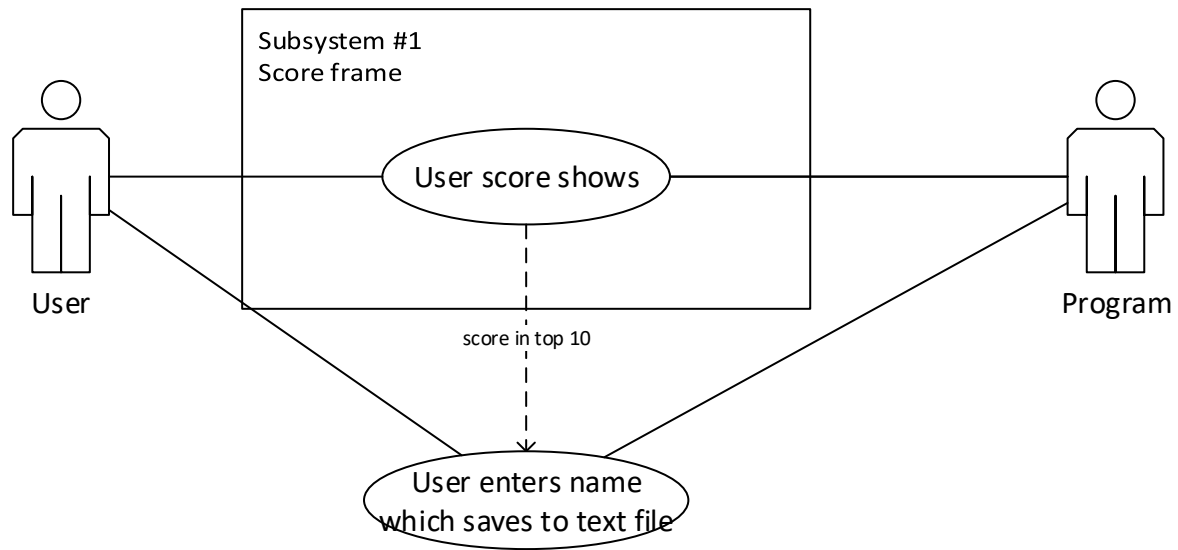


Figure 7 Score screen use case

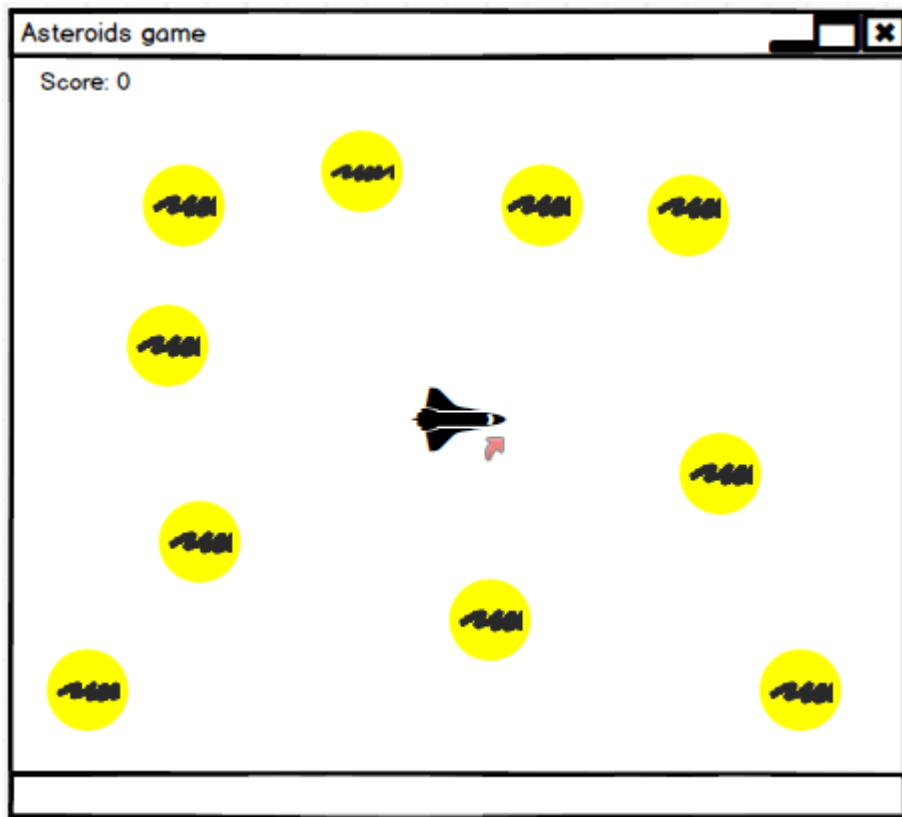


Figure 8 Main game screen

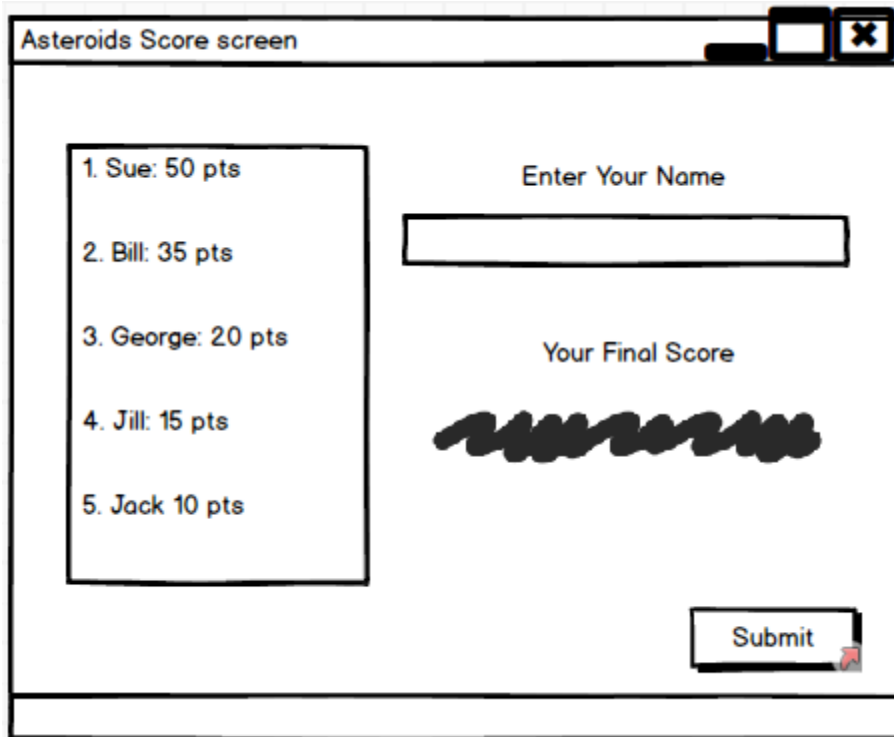
The image shows a Java Swing window titled "Asteroids Score screen". On the left, a rectangular box contains a list of the top 5 high scores: "1. Sue: 50 pts", "2. Bill: 35 pts", "3. George: 20 pts", "4. Jill: 15 pts", and "5. Jack 10 pts". To the right of this box, the text "Enter Your Name" is positioned above a single-line text input field. Below the input field, the text "Your Final Score" is displayed above a large, dark, scribbled-out area representing the score. In the bottom right corner of the window, there is a "Submit" button with a small red mouse cursor icon pointing at it. The window has standard OS window controls (minimize, maximize, close) in the top right corner.

Figure 9 Score screen layout

4. Implementation

This Project is implemented using a variety of java packages. The run method runs until the player dies; However, it only moves objects if the game is not paused. The base move method (Table 1) of the ShapeBase class uses math functions to calculate velocity band position. The angle is the determine the direction of the ship shaped polygon that rotates based on key input. The high scores are tracked through a high score class. This class contains an array of the 5 highest scores and an array of names to match the scores. It also contains functions for checking scores printing them and putting them into a file called highscore.ser (table 4).The classes are created in draw class (see table 3).

4.1 Collision detection

My collision detection methods are contained in my asteroid class, because all the collisions involve asteroids. The collisions create a hit-box around the ship and the asteroid based on radius. If these boxes meet the ship will be destroyed and the game will end. The shot is just treated as a coordinate as it is very small, and if an asteroid collides with a shot it's hp is decremented by 1 being destroyed at hp = 0. (table 2)

Table 1 Move function

```

public void baseMove() {
    x += xVel;
    y += yVel;
    if (x < 0) {
        x += 600;
    } else if (x > 600) {
        x -= 600;
    }
    if (y < 0) {
        y += 600;
    } else if (y > 600) {
        y -= 600;
    }
}

```

Table 2 collision code

Shot collision <pre> public boolean shotCol(Shot shot) { if (Math.pow(radius, 2) > Math.pow((shot.getX() - x, 2) + Math.pow(shot.getY() - y, 2)) { this.hitsLeft--; return true; } return false; } </pre>	Used in game <pre> if(roids[i].shotCol(shots[j])) { deleteShot(j); if (roids[i].getHitsLeft() < 1) { deleteAst(i); p1.score++; } } </pre>
Ship collision <pre> public boolean shipCol(player ship) { if (Math.pow(radius + ship.getRadius(), 2) > Math.pow(ship.getX() - x, 2) + Math.pow(ship.getY() - y, 2)&& ship.isAlive()) { return true; } return false; } </pre>	Used in game <pre> if (roids[i].shipCol(p1)) { p1.alive = false; numRoid = 0; numShots = 0; startLvl(); } </pre>

Table 3 Class Creation

```

player p1;
GameApp g1;

public draw() {
    p1 = new player(250, 250, .35, .98, 12);
    g1.getPlayer(p1);
    addKeyListener(p1);
    p1.start();
    this.setFocusable(true);
    this.requestFocusInWindow();
}

```

Table 4 High score checking and file writing

Checking scores <pre> public void checkHighScore(player p, String username) { int temp = 0; int check = p.score; String checkUser = username; String tempUser = ""; for (int i = 0; i < numScores; i++) { if (check > highScores[i]) { tempUser = user[i]; temp = highScores[i]; user[i] = checkUser; highScores[i] = check; check = temp; checkUser = tempUser; } } user[numScores - 1] = ""; highScores[numScores - 1] = 0; } </pre>	Writing to file <pre> public void write_to_ser() { try { FileOutputStream fout = new FileOutputStream("HighScore.ser"); ObjectOutputStream oos = new ObjectOutputStream(fout); oos.writeObject(this); oos.flush(); oos.close(); } catch (IOException e1) { e1.printStackTrace(); } } </pre>
---	--

5. Testing

I performed unit tests of both my ship and shot collision methods. I performed one for each that I knew would succeed and one that would fail (see table 5). These are curated circumstances where my objects do and don't meet.

Table 5 Junit tests

Success <pre>@Test void col_test_01() { asteroid a = new asteroid(200, 200, 6, 0.1, 3, 3, 1); player p = new player(200, 200, .5, .1, 12); boolean test = a.shipCol(p); assertEquals(true, test); }</pre>	Success <pre>@Test void col_test_02() { asteroid a = new asteroid(200, 200, 6, 0.1, 3, 3, 1); Shot s = new Shot(200.0, 200.0, 1.0, 1.0, 0, 25); boolean test = a.shotCol(s); assertEquals(true, test); }</pre>
Failure <pre>@Test void col_test_03() { asteroid a = new asteroid(Double.MIN_VALUE, Double.MIN_VALUE, 6, 0.1, 3, 3, 1); Shot s = new Shot(Double.MAX_VALUE, Double.MAX_VALUE, 1.0, 1.0, 0, 25); boolean test = a.shotCol(s); assertEquals(true, test); }</pre>	Failure <pre>@Test void col_test_04() { asteroid a = new asteroid(50, 250, 6, 0.1, 3, 3, 1); player p = new player(0, 1, .5, .1, 12); boolean test = a.shipCol(p); assertEquals(true, test); }</pre>

Appendix A. Traceability matrix

This is a traceability matrix for the requirements of this assignment as showing in 2.0.

Paragraph Define	Requirements text	Paragraph implemented	Paragraph Tested
P. 3.1	Define Use Cases	Figures 4-7	untested
P. 3.0	Creation of classes	Figure 1-2/ table 3	
P. 3.0	Collision code	P. 4.1 Figure 3/ table 2	Table 5
	Highscore	P. 4 Table 4	

Appendix B. Screens of the working app

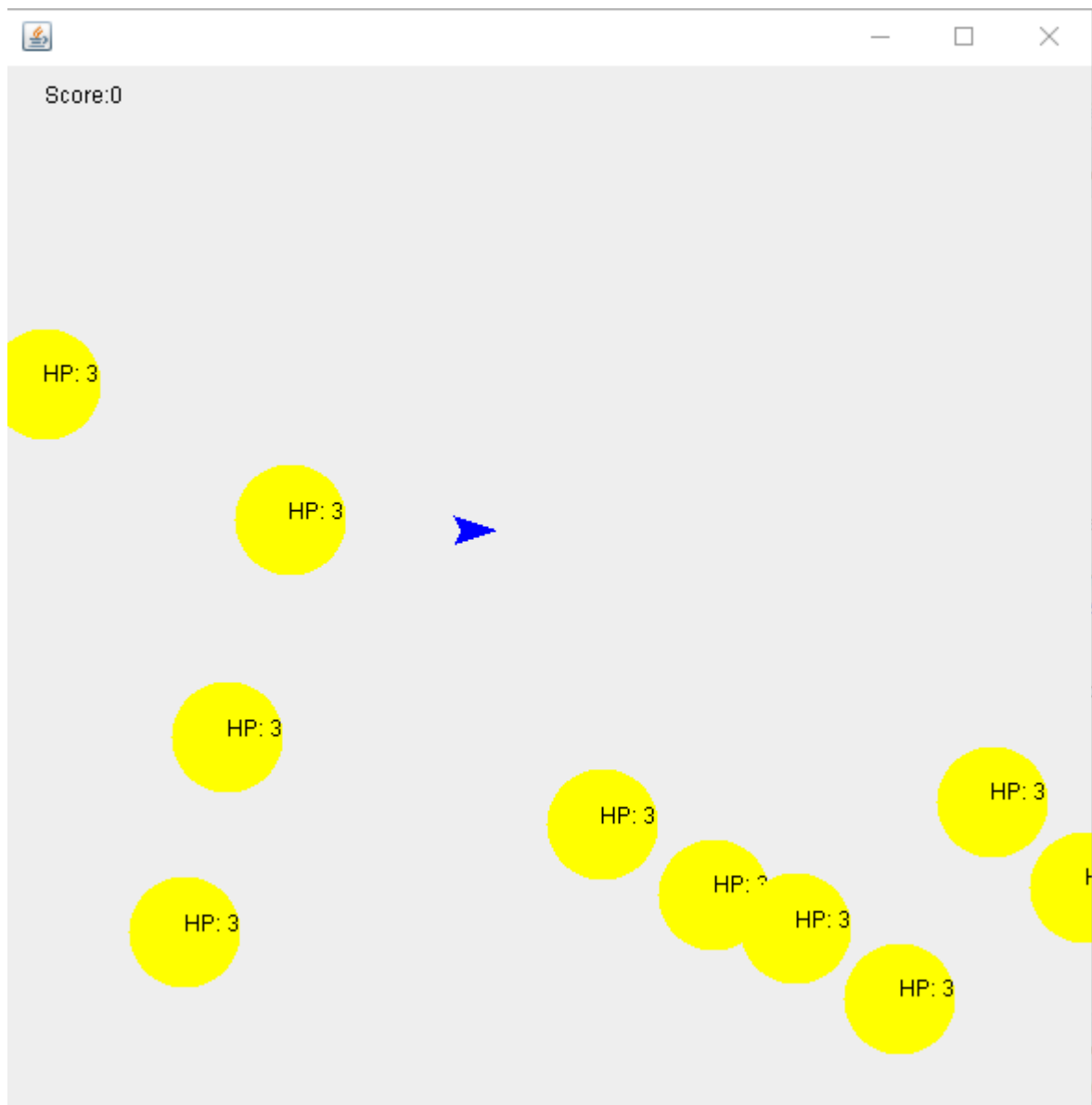


Figure 4 Working game screen

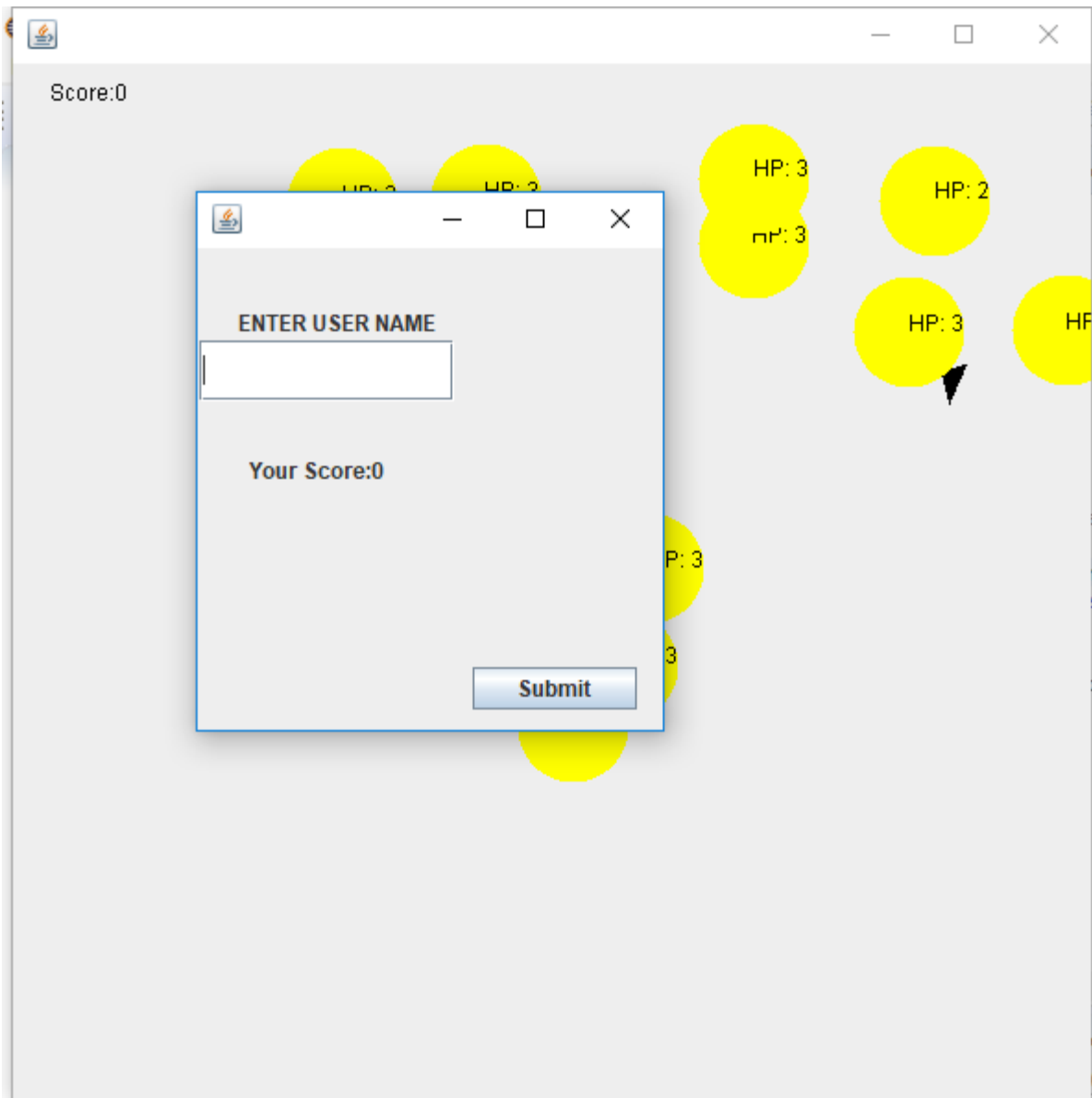


Figure 5 wokring score screen