KING FAHD UNIVERSITY OF PETROLEUM & MINERALS

### Information and Computer Science Department

# ICS-201 Introduction to Computer Science

**Lab Project**

**Semester 141**

## I. Introduction:

The project task is to develop a simple 2D game using the Java Swing/AWT libraries. This document will explain the concept/mechanics of the game, features, functionalities, programming process, and deliverables. You will gain experience in GUI, event-driven, and game programming paradigms. You might also decide to use multi-threading to make your game more responsive.

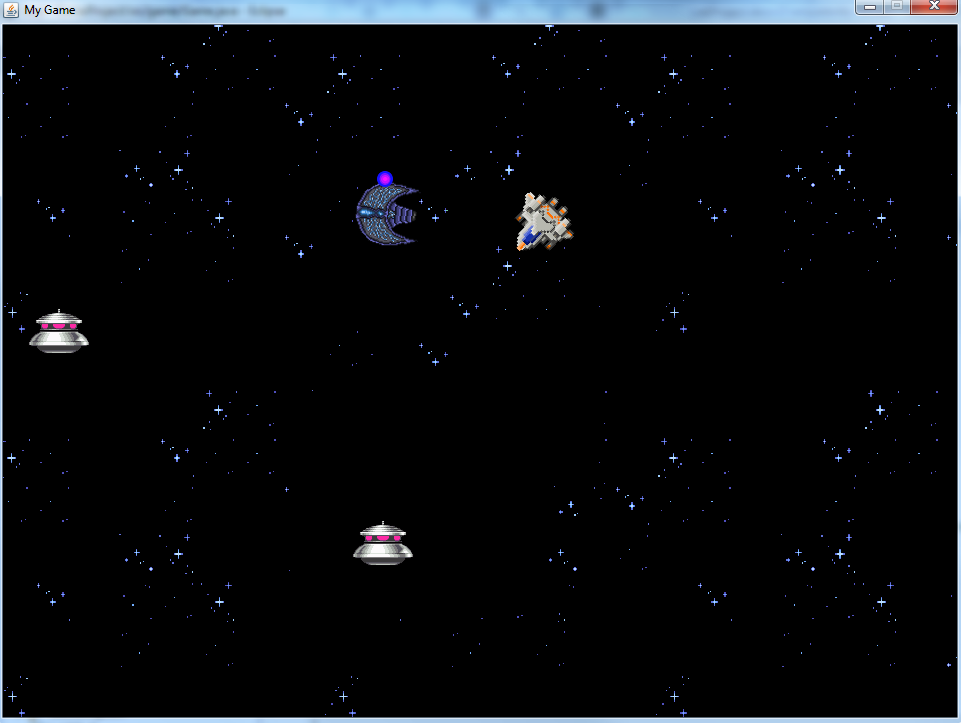


Figure 1. Space ship shooter concept example.

## II. Game Concept & Mechanics:

The game will be a rendition to the old classical game of a space shooter. The player will control a space ship around the screen and shoot down enemies. Once all the enemies are dead the game is over. The following are the mechanics of the game in detail:

1. The user will control the **Player’s** ship which has a total of **3 health bars.**
2. The player’s ship can move in all directions using the WASD keys as input.
3. The player’s ship will adjust its orientation according the mouse’s location.
4. The player’s ship can shoot bullets using the Left-Mouse-Button.
5. There are two types of **enemies** that will target the player:
   1. **Follower**: This ship will continue following the player until it smashes into him.
   2. **Shooter**: This ship stands in place and shoots bullets at the player.
6. If a player’s bullet **collides** with any enemy entity, then the bullet and the enemy entity will be destroyed from the game.
7. If any enemy entity collides with the player, one health bar is subtracted from the player’s total life.
8. When a level starts, the player’s ship will be placed in the middle of the map. Enemies can be either randomly spawned using a random spawn system (recommended) OR can be programmatically predefined prior to the start of the level (note this can be tedious).
9. A level ends when either one of the following happens:
   1. The player’s life reaches 0.
   2. All enemies have been destroyed.
10. Once a level ends, the user will be shown a result screen with the total score and he may choose to end the game or restart.

**Bonuses:**

1. Create a Main Menu for our game with your logo, a start game button, a help button, and a quit button.
2. The game starts by pressing the start game button.
3. When a level ends, you display the result screen with an additional button that will return the user to the main menu.
4. (Extra Bonus) Add a difficulty option that modifies your spawning system to spawn more and faster enemies.

## III. Programming Process:

This section will cover some recommended approaches/techniques to aid you in your programming journey. Again, this is just recommendations, you can choose to selectively go over this section, or completely ignore it all together if you want to do it another way.

## 1- Framework:

My advice is to try to make your game framework using a divide and conquer approach. Break away your core framework elements down and implement each separately. One big example is that you create a class that represents the JPanel that you will draw your game objects on (We can call this the renderer). Another class can be used to manage your gaming objects in terms of creating, deleting, moving, etc (Game Manager or Game Engine). Also, creating a good GameObject class hierarchy for your gaming objects/entities will help you organize and manage your code much more easily. You can also create a controller class for managing all the user input in a more compact manner. Finally, you will have a class that will encapsulate all these classes/managers for easy communication. This idea will make much more sense if you read the next section.

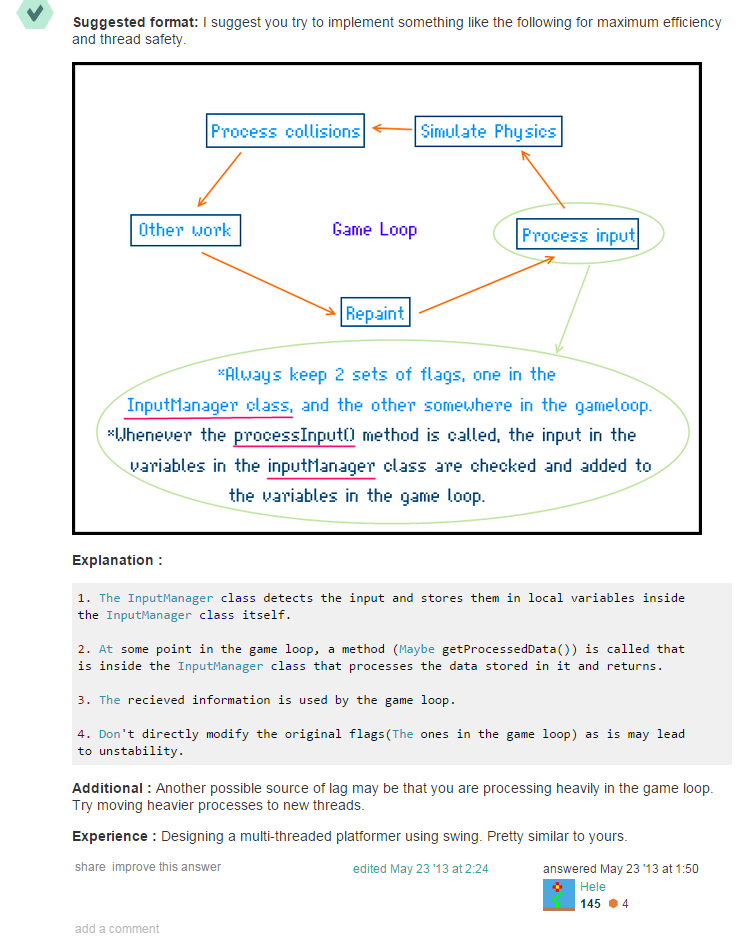
## 2- Game Loop:

Every GUI or game has a loop that will loop until the user decides to quit the game. Up until now, this looping mechanism was not clear to you because it is handled automatically by java when you use the Swing/AWT library. But for our application we need to explicitly run a gaming loop that will loop until a quit condition is met. What happens on each iteration is the key, and can be described as follows:

1. First thing we process the user input from our controller. To get smoother controls, we should use flags to indicate each iteration which buttons are being pressed and update accordingly.
2. Then we update our gaming world via the engine or object manager. This will move each of our objects each iteration with regards to their velocity.
3. Then we process collisions via our game engine and update our world accordingly (destroy objects, enemies, etc.).
4. Finally, we call our rendered or whichever is responsible for painting our objects, like the JPanel, to paint the updated gaming world. One way to do this is to pass your updated objects as an array to the renderer and then it will loop on the objects and draw them one by one.

We call each iteration a frame, and for our application we will be shooting for 60 frames per second (fps). To do this, you will need to create a timer that will only do the steps above if enough time has passed. We can further discuss the implementation of the looping and timing in class. Also, I will provide you the source code for the Game Engine that you may decide to use to make your task easier. You can also opt to make your own game engine for experience purposes.

Here is an explanation of this concept that I found on stackoverflow.com by the user Hele:



## 3- Game Objects:

As mentioned earlier, it is recommended to create classes in a hierarchical fashion that will represent the state of your game objects in the game. Let us discuss this further. Our task is to make a 2D game, therefore, it makes sense if we make a super class that encapsulates all properties that every 2D game object should have. For example, every 2D object will have a location (x, y) and velocity (vx, vy). Each frame we will update the position of each object to have the new location (x + vx, y + vy). There are many other properties you may wish to consider such as sprites/images, orientation/direction, adding methods like move, drawObject, etc.

## 4- Game Panel:

All your drawing and rendering will take place on a JPanel that we will call the Game Panel. Of course your JPanel will be contained within a JFrame and you need also consider the resolution of your game. A good resolution will adhere to the 4:3 ratio like 640 x 480 or 960 x 720. For simplicity, pick a resolution that you like and store it as a static variable for reference throughout your project.

## 5- Player Orientation and Shooting

One of the mechanics you will implement is that your player’s ship should orient itself so that it matches the mouse’s location each frame. To do this, you will need to apply some math to get the angle of rotation. I will not explain the entire process, but I will mention that it involves using trigonometry and right angle triangle rules. Once you find the angle of rotation, you will keep track of it and adjust the image before drawing to the screen. For adjusting graphics/images in Java Swing consider looking at the AffineTransform class in the Java API. The idea is that you should use AffineTransform to translate (move) and rotate (adjust orientation) of each object’s image before drawing it on the screen. We can discuss the implementation details in the lab.

## IV. Deliverables:

1. The minimum deliverable that I expect to see is a GUI window that should implement all the features in the Game Concept & Mechanics section excluding the bonuses.
2. There will be a project demonstration followed by Question/Answer session. I expect you, and only you, to code your project. Therefore, I will be asking question to test your knowledge and experience with the project.
3. I will also take a look at the code and expect some level of adequate enforcing of good coding practices: good identifier naming, commenting, spacing, indentations, etc.
4. Extra bonus will be given if you implement the bonus features.
5. Extra bonus will be given if you document your code using javadocs.