

# Project 1: Space Invaders

---

*DUE DATE: Tuesday, July 11 at 6pm*

***Please read the entire project description before you begin coding.***

## Overview

In this project you will modify and extend an implementation of the classic game Space Invaders. In this game, the player controls a space ship that is under attack by alien monsters. The goal is to avoid dying while destroying as many monsters as possible. Using the arrow keys, the player can maneuver the ship to avoid monsters and their laser fire and use the space bar to shoot bullets at the alien invaders. The player can also detonate bombs using the 'B' key. A blue bar in the lower left corner of the screen indicates remaining shield strength. Shields are depleted as the player incurs hits from enemy lasers or crashes into enemies. The score increases with each monster destroyed by the player.

Although the game compiles and runs as given, it will be your job to enhance game functionality. Part 0 will guide you in setting up your project and running the program in its primitive state. Part I will guide you through some basic changes, including modifying fields and methods. Part II instructs you on extending the game by adding additional features. Part III requires you to further modify and enhance the game however you would like. It is important to compile and test your program after every change in order to diagnose and repair errors and bugs as you go.

You are provided with 11 Java classes, each defined in a Java file. You are also provided with a *resources* folder. The *resources* folder contains image and sound files used by the program for the user interface. The classes interact to give the game its functionality. Make sure you have all of the following java files:

- Actor.java
- Bomb.java
- Bullet.java
- Invaders.java
- Laser.java
- Monster.java
- Player.java;
- ResourceCache.java
- SoundCache.java
- SpriteCache.java
- Stage.java

### ***Invaders.java***

The entry point (main method) for the project is found in the Invaders.java file. This file not only contains the main method, but it also initializes the game world through the `initWorld` method. Actors (see the next section) are created and placed in the world. As of now, actors include the player (spaceship), monsters, lasers, bullets and bombs. These actors can perform actions on each round of the game and interact with each other. In addition to adding these objects to the world, the `initWorld` method sets up all the graphics for the game.

## ***Actor.java***

An actor is the most basic interactive object in the game world. The player, monsters, lasers, bullets, and bombs all inherit properties and behaviors from the Actor class. You will notice that each of these classes, for example `Player.java`, has the code `extends Actor` following the class header. The fields of an actor objects describe its physical presence, including its x-y position and its size. While there are many methods in the Actor class, one method to note is the `act()` method which allows objects to move and interact with other objects.

## ***Player.java***

The Player class represents the spaceship that the player controls when playing the game. In addition to the fields inherited from the Actor class (position and size) this class has fields including values for velocity and shields.

The Player `act()` method allows the player to move and interact with objects in the game world. The method looks at the player's current x-position and updates it by the current x-velocity and similarly for the y-position. This method also establishes boundaries on where the player's ship can move, preventing the spaceship from moving off the screen.

The `fire()` methods shoots Bullets, and the `fireCluster()` shoots Bombs. Player only has a limited number of Bombs, so every time a bomb is shot, the `clusterBombs` count will decrease.

The `collision()` method is responsible for deciding how a Player will react with its surroundings. The method is called when the Player collides with another object. Damage is incurred or points acquired depending on whether the object that collides with the player is a Monster or a Laser.

## ***Monster.java***

Objects of type Monster attempt to kill the player's spaceship while the player fires back at them. Monster has many of the same methods as Player, but it is necessary to make the motion of Monster objects autonomous (independent of user input). You will be instructed to make changes here and you may be able to come up with many more enhancements for this class to make the game more difficult.

## ***Laser.java***

Objects of type Laser are fired by Monsters. If a laser hits (collides with) the player's spaceship, the shield is reduced. The preexisting implementation should already work; take a look at the code if you curious to see how movement is controlled.

## ***Bullet.java***

The player fires Bullets at the monsters. A monster is destroyed when a Bullet hits (collides with) the monster.

Three of the other java files, `ResourceCache.java`, `SoundCache.java`, and `SpriteCache.java` deal with resources, such as the loading of images and sound files for the project. They are fully implemented and there is no need to edit any of them. Note that the sound has been disabled for this project.

Each part of the project will guide you through modifying code in the rest of the files and adding new java files. By following the steps and performing some analysis on your own, you will come to understand how the files work together and will be able to add new features to the game.

## **Part 0**

You have been given a folder called *Invaders* containing .java files, a *resources* folder, and this document. Rename the folder *Invaders\_group#*. If you are not in a group, use your name instead of *group#*. Follow these directions to get your game running:

1. Open JCreator and create a new project; name it the same name as you renamed the folder and choose the location to be that folder. The java files should automatically load to the project.
2. Select 'Project' from the menu bar and make sure this project is set as the active project by clicking 'Set Active Project' and selecting the project we just created.
3. Select 'Project' from the menu bar, select 'Project Properties...'. Click the button to the right of the 'Run' box and select 'Invaders' as the run file. Press 'OK'.
4. Now go to the 'Build' menu and select 'Compile Project' (not 'Compile File'). If there are no errors (there should not be any if you have followed directions correctly), select 'Execute Project' in the 'Build' menu.
5. Take a few minutes to get familiar with the game, its layout and controls. Yes, this is the only time we are allowing you to play computer games in class.

## **Part I**

Step 1: The world starts off with 5 monsters. In the `initWorld` method in `Invaders.java`, change one line of code so that we start off with 10 monsters in our world. (*hint: Look in the `for` loop*)

Step 2: Initially the score is set at 100, but we need it to start at 0. Change one line in `Player.java` so that the score is initially set at 0.

Step 3: Monsters should shoot at the player. Add code to the `act()` method that allows monsters to fire lasers. It is not sufficient to simply add a call to the `fire()` method (Try it and see what happens!). One way to implement firing is to call `Math.random()`, which returns a random number between 0 and 1 and compare it to the variable `FIRING_FREQUENCY`.

Step 4: As of now, monsters are stationary. In the `act()` method of the monster class, devise a way to make the monsters move horizontally back and forth through the game world. Monsters should neither move off the edge of the screen nor should they teleport (i.e. their motion should be smooth). Looking at the `Player act()` method will help give you an idea of how to complete this step.

Step 5: When a player kills a monster using a bullet or a bomb, the monster reappears somewhere else on the screen. We call this respawning. You may have noticed that when the player's spaceship collides with a monster, the monster dies and does **not** respawn. Modify the `collision(Actor a)` method in the `Monster` class so that a monster respawns upon collision with the player as it already does upon collision with a bullet or bomb.

## **Part II - Extending the Game**

We will now add a new kind of object to the game, a Star. If the player's spaceship collides with a Star object, the shields will be increased by 10, the player will receive one more clusterbomb, and the ship's speed will increase slightly. It is your job to implement these features.

Step 1: Make a new class Star.java which extends Actor. Copy the code below to give you a start on creating the class.

```
public class Star extends Actor {

    public Star(Stage stage) {
        super(stage);
        setSpriteNames( new String[] {"star1.png"});
    }

    public void act() {
        super.act();
    }
}
```

Step 2: Modify the `initWorld()` method in Invaders.java so that on creation of the game world, a star will also be created. Refer to the code which adds the player to the world to help you complete this step.

Step 3: Modify the `collision(Actor a)` method in the Player class so that the player's spaceship knows how to interact with a star upon collision (increase shields, add clusterbombs, and increase velocity).

Step 4: Add a `spawn()` method and a `collision(Actor a)` method to Star.java so that the Star object can be replaced upon collision with a player. Refer to these same methods in the Monster class for a model of how to do this.

## **Part III - Adding your own part to the game**

Now that your game is more or less complete, it's time to add something that will set it apart. Think of any feature that would make the game more entertaining and/or more challenging. Perhaps you want to add a new kind of object to the game, such as a Boss. On the other hand, you could take an existing class, such as Monster, and make it more complex, such as adding more advanced navigation and attacking. You could even add "homing" capabilities to the player's bullets. It's completely up to you. Please keep your idea straightforward and well-defined in your submission, but feel free to play around and make interesting changes to the game (Gleb and Lawrence spent many hours playing and extending Invader. They implemented spiraling bombs and a particle swarm that protects the player). You must describe your idea to one of the AITI staff before you begin coding.