Assignment 2

You are working for a company called **PipeDream** which produces desktop and mobile games. The company is currently developing a new game called **Invadem**. **Invadem** is a simple shoot'em up game where the player controls a tank that fights off a horde of invading space ships.

You have been given the task of building a demo. You will need to implement the tank, barriers, invaders and a single level that will be repeated if the player wins or loses. Additional requirements will be released after your milestone submission.

An artist has created a simple demonstration of the game and has posted it on your online forum (Ed). Use this demo to help layout your entities and develop your game mechanics.

Project and Build

You have been given a scaffold which will help you get started with this assignment. You can download the `scaffold` onto your own computer and invoke `gradle build` to compile and resolve dependencies.

You will be using the **Processing** library within your project to allow you to create a window and draw graphics. You can access the documentation from the following link

Game Objects and Mechanics

The project contains a number of entities that will need to be modelled within your application. You have been provided some basic test cases to help develop and implement their functionality.

Tank

This is a player controlled entity that can be moved by pressing the left and right arrow keys on the keyboard and moves at a rate of 1 pixel per a frame . The tank is 22x16 pixels, it starts at the bottom-middle of the display. A tank can fire projectiles which can hit the barriers or enemy invaders. It can shoot multiple projectiles towards the invaders. In addition to moving the tank with left and right arrow keys, the player can fire projectiles using the space key. If an enemy projectile hits the tank, it will lose a hit point, if the tank is hit 3 times, the game should transition to a **Game Over** screen since the tank has been destroyed.

Invader

Each invader has a unique starting position but move in time with every other invader. The invader swarm moves from the top-middle of the screen to the player's barriers. Once an invader has reached the barriers, the game should transition to a **Game Over** screen.

Invaders will move 30 steps in one direction before moving down and heading 30 steps in the other direction. Each sideways step will constitute a movement of 1 pixel, each step is made every two frames. When an invader moves downward, it will move 8 pixels down and transition to it's other sprite.

The invaders are part of a larger swarm, the swarm starts with 40 invaders (10 invaders per row, 4 rows). Each invader is 16x16 pixels, this will correspond with the size of their sprites. They will have the same collision area as their sprites.

Once an invader is hit, it is considered to have been destroyed and should no longer be rendered by the game. When all invaders have been hit, this will result in the player winning the game and transitioning to the **Next Level** screen.

Every 5 seconds, an invader will be randomly selected to fire a projectile downwards.

Barrier

A barrier is composed of 7 different components, each component can sustain 3 hits. Once a component has been destroyed, it no longer offers protection for the tank. When a barrier sustains a hit, it will change to a different sprite, indicating that it has been damaged. The player is provided with 3 barriers, each barrier, left barrier is at least 20 pixels away from the left boundary described in the application section, the center barrier starts in the centre of the screen, right barrier is at least 20 pixels away from the right boundary. Each barrier is at least 10 pixels above the tank's location.

A barrier can be hit by the tank and an invader.

Projectile

A projectile can be fired by both the tank and an invader, however, an invader's projectile will not hit any other invader (only the barrier and tank). The tank can hit the barrier as well as any invader. Once a projectile impacts with another entity, it will cease to exist.

The projectile is 1x3 pixels and travels upwards 1px per frame.

Game Conditions

The goal of the game is for the player to destroy all invaders before either the tank is destroyed or the invaders land.

The player wins when the following conditions have been met:

- All invaders are destroyed.

The computer wins when one of the following conditions have been met:

- An invader reaches the barriers (10px away from the barriers).
- The tank is hit 3 times and destroyed.

Application

Your application will need to adhere to the following specification

- 640 width 480 height window with a black background.
- Left boundary (tank cannot move past this point) at x 180, Right boundary at x 460.
- Must maintain a frame rate of 60 frames per second.
- Your application must be able to compile and run on any the university lab machines using `gradle build` & `gradle run`

  Failure to do so, will result in 0% for Final Code Submission.
- Your program must not exhibit any memory leak, try to load all assets prior to usage.
- You must use the processing library, you cannot use any other framework such as javafx, awt or jogl.

Assets

Artists within the company have produced **sprites** for your game. You have been provided a `/resources` folder which your code access directly. These assets are loadable using the `loadImage` method attached the `PApplet` type. Please refer to the processing documentation when loading and drawing an image.

Algorithms that may come in handy!

It is likely you will need to utilise the following algorithms and patterns within your project. You will need to identify where you will utilise these algorithms and document them in your report.

Collision Detection (AABB)

Since all entities within the game can contain an axis aligned bounding box. You may implement the following collision detection method.

Assume the following variables (r1, r2) have the properties: x, y, width, height.

```
check_collection(r1, r2):
    if ( r1.x < (r2.x + r2.width) ) and
       ( (r1.x + r1.width) > r2.x ) and
       ( r1.y < (r2.y + r2.height) ) and
       ( (r1.height + r1.y) > r2.y ):

       //Collision has been detected
      return true;
    else:
      return false
```

Animation

You have been given sprites to associate to entities. Due to the simplicity of the game there isn't a lot of animation that is required but you may find it beneficial to maintain an animation state within each entity.

Each entity will contains a **list** of sprites which can be swapped to.

```
AnimationData {
  sprites[];
  currentSpriteIndex;
  delay;

  tick() {
    //Action to take on a draw call
    //use information to delay a sprite transition
    //Loop around if index >= length of sprites
  }
}
```

Marking Criteria (12%)

Your final submission is due on **10th of November at 11:59PM AEST**. Please make sure you submit your code to Ed and your report to Canvas.

- **Final Code Submission (5%)**

  You will need to have implemented and satisfied requirements listed in this assignment. Make sure you have addressed the following and any other requirements outlined previously.

  - Window launches and shows black background
  - Tank is rendered and can be moved by the user
  - Invaders are rendered, move according to the pattern described and change sprites on movement
  - Barriers are rendered, can be hit by both tank and invader
  - Win condition can be reached by the player
  - Loss condition can be reached by the computer
  - Ensure your application does not repeat large sections of logic, try to minimise repetition.
  - Ensure your application code exhibits good OO principles (Utilising inheritance, interfaces and classes effectively)
  - **Additional requirements will be announced after the milestone deadline which will need to be implemented**
- **Milestone (1%)**

  To ensure progress has been made within on your project. You will need to submit a working submission of your project by Tuesday on October 29th by 11:59pm AEST.

  You must achieve the following:

  - Draw the tank on the screen.
  - At least one invader is drawn in its starting position.
  - At least one barrier is set up.
  - The tank can be moved using left and right arrow keys.

  However, try to aim to complete as much as you can so your tutor can provide as much feedback on your milestone submission.

- **Additional Test Cases (2%)** During development of your code, add additional test cases to your project and test as much functionality as possible. You will need to construct unit test cases within the `src/test` folder.

  To test the state of your entities without drawing, implement a simple game loop that will update the state of each object but not draw the entity.

  Some suggestion for test cases you should create:

  - Tank (movement, change sprites, fires projectile, intersection, state changes)
  - Projectile (checks for intersection, travel velocity)
- Invader (checks movement pattern, projectile firing, state change)
- Barrier (It can be hit, changes state after hit, can be hit by different entities, it's current placement in the application)

Code coverage, common and corner cases

- Ensure your test cases cover over 90% of execution paths (Use jacoco in your gradle build)
- Ensure your test cases cover common cases.
- Ensure your test cases cover edge cases.
- Each test cases must contain a brief comment explaining what it is testing.

- **Design, Report and Comments (3%),** (400 words minimum, 1200 words maximum) You will need to submit a report that elaborates on your application design and development progress. Please include:
  - Documentation on your progress, highlighting what you have identified from the problem description, requirements gathering and how you have represented concepts within your application.
    - Provide a high level overview of your implementation
    - For each class created, state what you used it for and why it was incorporated into your design.
  - Highlight changes you have made after the milestone submission, identifying how your initial design at the milestone hindered or helped your final submission.
  - Reflection, highlight issues that you encountered while modelling your application and any improvements you can identify.
  - Your code should be clear, well commented and concise, try not to repeat yourself and utilise OOP constructs within your application.

    Your code should be easy to follow, help the reader and avoid distraction. The code and design should aim provide a cohesive layout and concise comments; well chosen names; and idiomatic use of the Java language
- **Extension (1%)** You really want to impress! Add an additional feature to the game. We have provided some extension ideas but you are free to implement your own extension, please check with your tutor for approval.

  You will need to outline your extension idea within your report so the marker knows what to look for.

  Some ideas that you can implement:
  - 2 Player mode, two players can play the game at the same time on the same computer, this tank will be using a different sprite to the one provided.
  - Network play, Your game can create a server where two people can play the game from different computers.
  - Add sound effects to your game, make those visual pop with some sound effects.

Warning : Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.

Academic Declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.