Breaker

You are working for a company called **Courage** which produces desktop and mobile games. The company is developing a new game called **Breaker**. **Breaker** is a simple **Breakout** style game, the company has chosen java for the programming language and `gradle` for the build system to develop the game.

You have been tasked with writing the prototype for **Breaker**. It is a single player game where the aim of the game is break the the bricks within the stage and to continue until the player loses. The game is contained within a 520x400 (width,height) window.

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.

You will also be using a json library to load levels and enable other employees to construct levels to test. There is a simple tutorial of reading and using the `json-simple` library located here.

Game Objects and Mechanics

Paddle

The player controls a paddle which is in charge of launching the ball. The player will interact with the paddle using the keyboard arrow keys. When the game is launched, the paddle will be in the bottom-centre of the screen with a ball resting on top of it. The player can move the paddle in this state, using the left arrow key to move left and the right arrow key to move right. Once the space key has been pressed, the ball will launch.

The paddle is the determining factor for changing the vector of the ball. When the ball lands on the paddle it will interpolate and adjust the x and y component based on the centre of the paddle. When the ball lands in the centre of the paddle, the x-component will be `0` and the y-component will be `5`, when landing on the edge or corner of the paddle, the x-component will be `5` and the y-component will be `0` (it cannot result in 0 as the ball would continue to bound between the side of the wall and the paddle). You can refer to the algorithim section to assist with calculating the new x component and y component of the vector.

The paddle will be 40 pixels wide and 10 pixels high (40x10).

Ball

The ball is the primary object that determines if the player wins or loses the game. It is used to destroy **Brick** objects within the level and will determine if the player wins by destroying all bricks or loses when the ball y-coordinate exceeds the screen's height. The Ball has a vector which dictates its direction and velocity, you will need to use this vector as part of your rebound calculations.

We will provide a rebound algorithm to deal handle collision detection and rebounds.

To note, the collision detection component of this game is typically the most difficult part as there are *corner* cases to consider. Getting this component absolutely correctly is not necessary and we do encourage you to use a simple collision detection as noted in the algorithms section.

Brick

A level or stage is constructed with bricks. When a **Ball** contacts a **Brick**, it will be destroyed and will no longer be collidable by the ball. A **Brick** will have 20 pixels wide and 10 pixels hight (20x10). A Brick will have an assigned assigned sprite which you will be able to access from the `resources` folder in your gradle project.

Level

To allow other employees to work on the game after the prototype, you will need to use a `JSON` parser and load a level. `JSON` is a serialisable and human readable data format which can be communicated over a network and used as a file for applications. Using the `json-simple` package and the following documentation, you will be able to load the level and construct it prior to gameplay.

Each level will contain the following fields:

- name - String, will be the name of the level
- bricks - Array, will store individual bricks with their own x-y coordinates
- brick - Object, will hold an x and y coordinate and a sprite id. All brick objects will use maintain the same size as defined in the **Brick** section.

```
{
    "name" : "Level1",
    "bricks" : [
        {
            "x" : 0,
            "y" : 0,
            "id" : "brick_blue"
        },
        {
            "x" : 20,
            "y" : 0,
            "id" : "brick_red"
        },
        {
            "x" : 40,
            "y" : 0,
            "id" : "brick_pink"
        },
        ...
    ]

}
```

The walls of a level will be the boundary of the screen itself, if the ball exceeds screen width or ball x is 0, the ball will be rebound. If the ball hits the top of the screen it will also rebound, however the ball will not rebound if it hits the bottom of the screen.

Game Conditions

The goal of the is for the player to destroy all bricks. The player wins when the following conditions have been met.

- All bricks have been destroyed

The player will lose when

- The ball reches the bottom of the screen and cannot be hit by the paddle.

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 readme file (README.md or README.txt).

Interpolation

Given two known points on a number line, we want to extract a new point given some distance value from one of the points. `d` Is used as a percentage between these two points.

Example:

```
p1 = 7
p2 = 3

diff = abs(p1 - p2)

d = 0.25 //number between 0.0 and 1.0, distance from

new_point = p1 + (diff * d) //new point using p1 as a base
```

Collision Detection (AABB with modifications)

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

```
rebound(movingBox, rect):
    if (movingBox.x + movingBox.width + movingBox.speed_x > rect.x &&
        movingBox.x + movingBox.speed_x < rec.x + rect.width &&
        movingBox.y + movingBox.height > rect.y &&
        movingBox.y < rect.y + rect.height)
            movingBox.speed_x *= -1;

    if (movingBox.x + movingBox.width > rect.x &&
        movingBox.x < rec.x + rect.width &&
        movingBox.y + movingBox.height + movingBox.speed_y > rect.y &&
        movingBox.y + movingBox.speed_y < rect.y + rect.height)
            movingBox.speed_y *= -1;
```

Marking Criteria (12%)

Your final submission is due on 9th of February at 11:59PM AEST. Please make sure you submit your code
to Ed.

- **Final Code Submission (6%)**
    - Window launches and shows black background.
    - Paddle is rendered and can be moved by the user.
    - Bricks are rendered
    - Win condition can be reached by the player.
    - Loss condition can be reached when the player's ball goes below the screen.
    - Bricks are destroyed when hit by the ball.
    - Ball can collide with the paddle.
    - Ball correctly changes velocity as described.
    - 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 (2%)** will be announced after the milestone deadline which will need to be implemented.
- **Extension (2%)**

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 readme 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.

- **Milestone (2%)**
  To ensure progress has been made within on your project. You will need to submit a working submission
  of your project by Monday on February 3rd by 12:00pm AEST.

You must achieve the following:

- Draw the paddle on screen
- Load a level from file
- Display bricks on screen
- Ball is able to move
- Ball is able to rebound off the side of the screen
- Paddle is to be movable

- **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:

- Ball collision detection test cases
- Detecting if a brick has been hit
- Paddle movement and restriction to the screen environment

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.