CE812 Assignment

Angry Birds Clone

RegNo. 2312089

# Game description

For this project, I decided to create an Angry Birds clone, so is a demolition game. Angry Birds is a mobile game where you use a slingshot and birds to knock down structures that house green pigs, that have stolen the birds’ eggs. My game consists of 3 levels that demonstrate the core gameplay of Angry Birds. As you will see, all visual and audio assets in this project are property of, the developers of Angry Birds, Rovio Entertainment (for more info see appendix). This was done to avoid wasting time making my own assets, and to try to replicate Angry Birds as closely as possible.

## How to play

The aim of each level in the game is to defeat all the piggies on screen. This is done by slinging birds from the slingshot to knock down structures or hit the piggies directly. A piggie will be defeated when it hits, or is hit by, an object above a certain speed.

To sling the bird, the user must use the mouse to pull the bird in the slingshot, giving the bird a force and making it fly. The user must place the mouse within the bird and hold the left mouse button. As the user drags the mouse, a line will appear on the screen that shows the rough trajectory of the bird, given the current pull of the user. Letting go of the left mouse button slings the bird in the direction set by the user. When a bird has done all the damage it can, and becomes still, the bird will be destroyed and a new one is placed in the slingshot. The user must wait for the bird to despawn, in certain situations, this may take a few more seconds than expected. Each time this happens, the bird will be randomly chosen from a set of 4 birds, each with their own properties. Red and Yellow are the birds that act as a normal projectile. Black is larger than these two and is consequently heavier. Blue is the smallest and lightest of the bunch.

Piggies are placed in structures that consist of 3 materials: glass, wood, and stone. Glass shatters upon any object/bird colliding with it above a certain speed. Wood requires multiple hits to be destroyed. Finally, stone cannot be destroyed, only moved. Structures also may have explosives or ropes. Ropes are destroyed upon hitting them at a certain speed, and usually are connected to a piggie. Explosives, upon being hit, explode and push back or destroy all nearby objects.

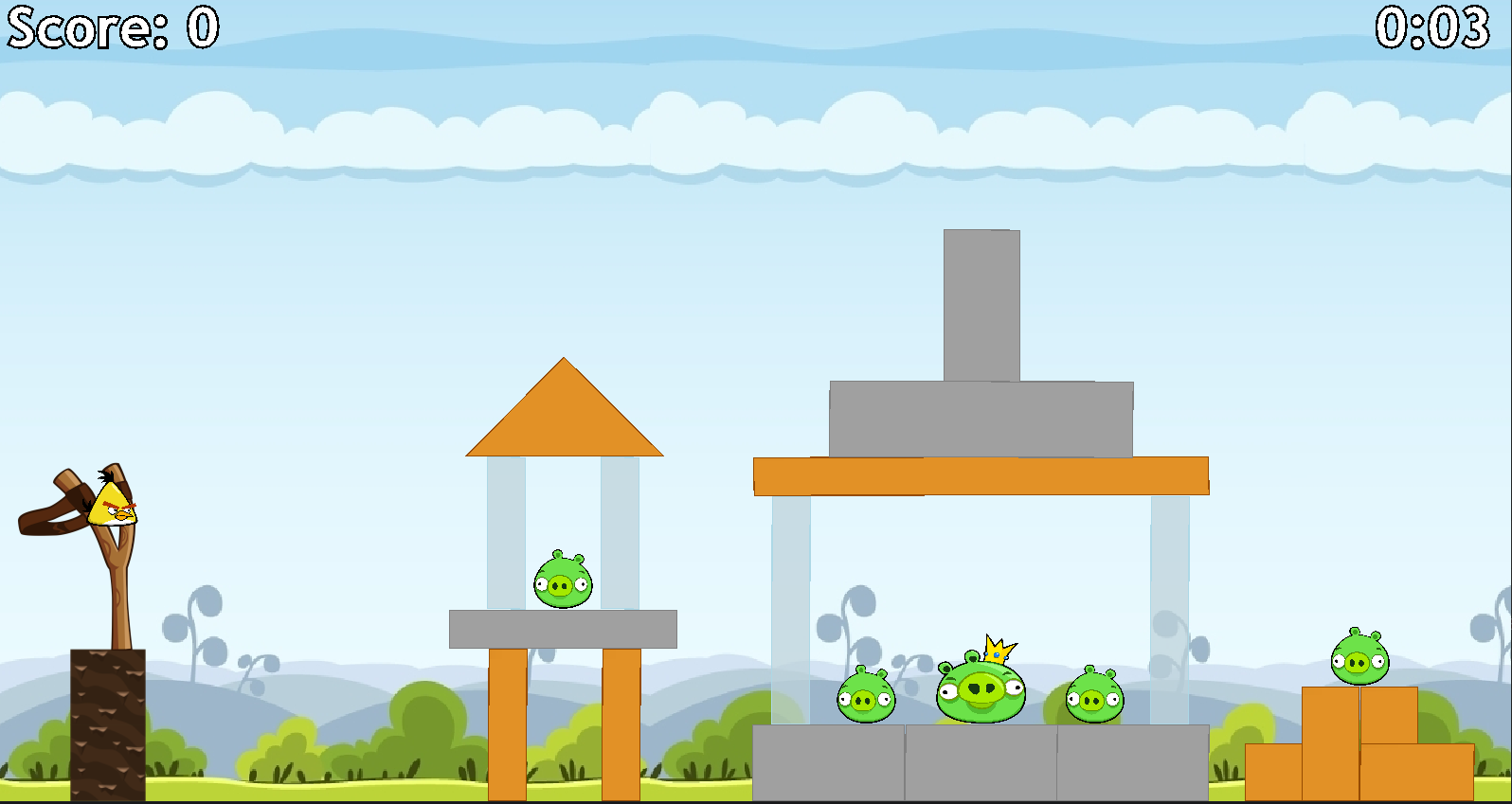
This game has a scoring system that the player can aim to maximise. Defeating a piggie adds a large value to the score, while each object destruction adds a much smaller score. However, each bird used subtracts from the score, so the user should aim to use as few birds as possible, while destroying the most objects and piggies. The game also has a timer for each level, which the user may want to use to beat levels in the fastest time possible.

The levels of the game are split up by various menus that display the users score and time. The user must press the enter key on their keyboard to advance to the next level. This is on screen for the user if they forget.

The game has music, which you may know as the iconic theme song of Angry Birds, and has various sounds from Angry Birds, that are played during collisions and other interactions.

## Levels

Level 1 demonstrates the core gameplay of this game, and consists of 3 structures, housing 5 piggies.



Level 2 demonstrates ropes and a different terrain to level 1, consisting of 2 structures, housing 5 piggies.



Level 3 demonstrates explosives and tasks the player with defeating piggies that are housed under a structure. There are 5 piggies to defeat.



# Technical Aspects

This section includes some snippets of code, but I suggest you investigate the classes for the whole context. The entire project should be commented to a satisfactory degree. The features of note here are CollisionHandler and Trajectory Line.

## Engine

This game was developed with the JBox2D engine, so it uses a World object to simulate all physics of the instantiated objects. Each object has a body and fixtures that allow it to interact correctly. I chose JBox2D so that I could add interesting new physics-based interactions to the world, without having to create my own engine or world.

## Game Features

### Birds

Every bird, and even the piggies, are subclasses of the Bird class. This is so they are all similar and interact with the world in the same way. The Red, Yellow, Black, and Blue classes are used to identify which bird is in use and applies the correct attributes and sprites for that bird. The Piggie class overwrites some functions so that it acts slightly differently. For example, a piggie can’t despawn when sat idle, whereas birds will.

Birds despawn when the length of their linear speed vector is very small, simulating them being still. When this occurs, a new bird is spawned in the slingshot.

### Polygons

Every object in the world is a subclass of the Polygon class. Each of these classes have their own values for each attribute and can define different shapes. I have described the special interactions of these objects in the “How to play” section. Rope is a special case as it doesn’t directly inherit from Polygon, it holds 3 segments of rope, which are each an instance of Polygon.

### Flow of Game

The game uses an enum called LayoutMode to determine what the current state of the game is, and which state to move to next. So, the game starts in MENU\_1, and progresses to LEVEL\_1, which then goes to MENU\_2, and so on.

### Sounds

The game has various sounds that are played during certain interactions. This is done through the SoundManager class, which uses AudioInputStream and Clip to load and play sounds.

### Visuals

The game uses sprites, textures, and backgrounds. These are handled by the ImageManager class, which uses ImageIO to load images, and the View class draws them to the screen.

### Scores and Times

During a level, the user can see, at the top of the screen, their score and current time, which is updated constantly. At the end of a level, the game saves the score and time from the level. These are then displayed on the screen.

### Trajectory Line

When aiming the bird, I chose to draw a line that shows the trajectory that the bird will take. This is done in the View class, in the drawTrajectoryLine method. This method takes the force that would be given to the bird if it was slung, and sets the start position, velocity, and acceleration of an imaginary bird. It then simulates 70 Euler updates using the improved Euler equation and plots the position at each time step as a dotted line. This updates in real time.

### Controls

The user can control the game with the mouse, this is handled by the MouseControls class. This class is instantiated twice, as a MouseListener and a MouseMotionListener, to allow for mousePressed, mouseDragged, and mouseReleased to all work properly.

The user may user the enter key to navigate from menus to levels. This is handled by the KeyControls class, which is basic.

### Barriers

Barriers determine the boundaries of the world. They do not move but still allow object collisions. By design, these are not drawn to the screen, but do still exist in the world. However, if you want to see the barriers, see the View class and its drawEntities method.

### Text

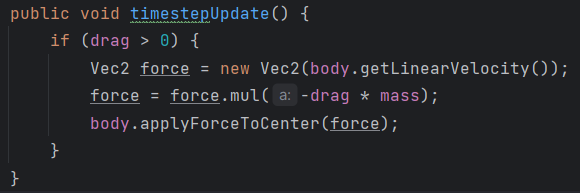
To draw text on the screen, I use GlyphVector, which allows for text to be turned into a shape and then drawn to the window. This means that I can give each letter an outline, such that they can stand out on the screen.

## Physics

To reiterate, the game uses the physics of JBox2D, but I have added additional features to manipulate this.

### Additional Updates

Each entity has a method called timestepUpdate. This method adds drag to the object’s movement, so that objects move similarly to real life physics. This method is invoked every time-step/frame.



This just applies a small force, based on the mass of the object and the drag coefficient, to the centre of an object. This force acts in the opposite direction the object’s linear velocity.

### CollisionHandler

The CollisionHandler class is an implementation of ContactListener from JBox2D. It has an attribute that is the speed of which a collision should elicit a specific reaction. This class doesn’t change any interactions in the world, it just adds additional properties to certain collisions. I only needed to implement the beginContact method from ContactListener for this game, but the other 3 methods need to be present for it to work, so they are ignored.

My beginContact implementation does 2 things, the first being finding the correct entity for the given body. This uses the userdata of the body to find the entity, which I assign in each entities class.

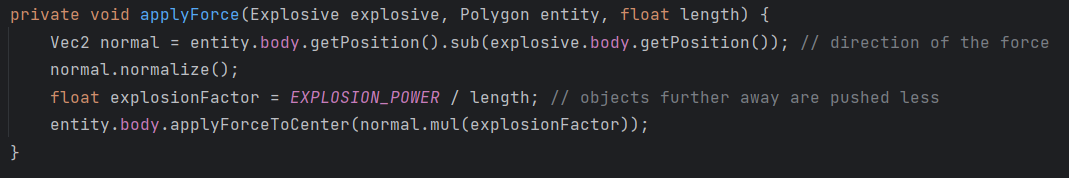
Once the entities are found, specific conditions are checked to apply some special interactions. When Wood is included in a collision, it takes damage, if it is currently vulnerable to damage, or is destroyed. When Glass, Rope, Piggie, or Explosive are included in a collision, they are destroyed.

CollisionHandler doesn’t do any actual destruction, but it flags the objects that need to be destroyed. During any collision, an appropriate sound is played.

This class has many methods, one for most entity types. This is to avoid casting the Object as the specific entity class constantly.

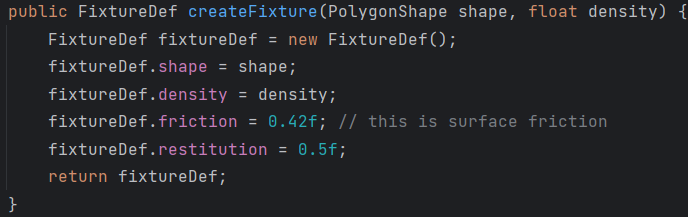
### Explosions

When an explosive is hit, the game detects which objects are in its blast radius. These objects are then either destroyed, take damage, or are forced away from the centre of the explosion. The force is calculated by finding the normal vector of the line between the centre of the explosive and the centre of the object and multiplying it by a factor that is based on the distance from the object. This will apply the most force to objects close to the explosion, and a lesser force to those on the edge of the blast radius. The method below demonstrates this. It takes the distance from the explosive to the object as a parameter.



### Complex Shapes

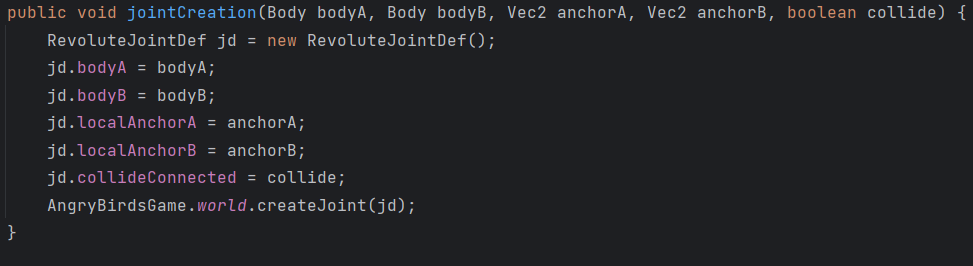
Complex shapes are defined by two fixtures for one body. To create the shape of these objects, two rectangles are combined, and, in JBox2D, act as two objects stuck together, similarly to joints, but are one entity.



To do so correctly, the method above is used twice, once for each rectangle.

### Joints

For a Rope to act similarly to real life, the class uses 4 joints. 2 that link the 3 segments together, and 2 that attach each end to the objects given.



This method is invoked for each joint. It takes the two bodies, that are to be joined, as parameters, as well as the local body coordinates of the joint. I have used RevoluteJoints to simulate the swinging of rope.

## Tuned Constants and Values

The Constants class holds many of the tuned values for this game. These include the densities or sizes of entities, physics constants, and the screen/world dimensions. All these values were tuned to generate a game that simulates semi-real physics.

Additionally, there are a few values that aren’t saved in Constants that needed tuning. These include the registerSpeed in CollisionHandler, and the factor that is multiplied by the distance from the mouse to the bird to sling the bird at the correct speed.

To save space in this document, I will not list all of them here, so Constants is important to investigate and has been commented properly. However, I should mention DELTA\_T and the values associated with it. I have assigned DELAY as 20 milliseconds and DELTA\_T is this value in seconds. The number of EULER\_UPDATES\_PER\_FRAME is set to 1000. These should allow the game to perform smoothly.

## Bugs

There is only one bug that I have observed with this game. Rarely, an error will be printed to the command line that says that the graphics cannot draw the bird. This only happens when the graphics tries to draw the bird while it is being initialised. This bug doesn’t stop the game from running or terminate the game. I have mentioned this here so that if it is observed, just ignore it.

# Reflection

I believe that this project went quite well. I have made a very good recreation of Angry Birds, which is what I set out to do. The difficult parts were creating each version of Polygon and all their interactions, especially CollisionHandler, and manipulating the images that I load. The easy parts were the sounds and the maths-heavy parts of physics. I am especially proud of the CollisionHandler class, and its use in AngryBirdsGame, and the trajectory line.

## Future Work

If I were to continue working on this project, there is potential to add a special effect to each bird to make them more unique. I could also add more levels and more materials for building structures. I could even go as far as adding a story to the game, or even changing all the visual/audio assets to those created by myself, changing this from a clone to my own new IP.

# Appendix

Even though it shouldn’t matter, this whole project was made in IntelliJ.

The following classes have code that was adapted from the lab work, any classes not included are solely created by me. I have made a comment at the top of each class that is on this list.

* Rope
* Constants
* MouseControls
* AngryBirdsGame
* Barrier
* Bird
* Polygon
* View
* Window

## Acknowledgements

All visual/audio assets are property of Rovio Entertainment. I obtained all the assets from the following links.

* [Search - The Spriters Resource (spriters-resource.com)](https://www.spriters-resource.com/search/?q=angry+birds&c=8&o%5B%5D=s&o%5B%5D=ig&o%5B%5D=g)
* [Mobile - Angry Birds - Sound Effects - The Sounds Resource (sounds-resource.com)](https://www.sounds-resource.com/mobile/angrybirds/sound/659/)
* [Category:Galleries | Angry Birds Wiki | Fandom](https://angrybirds.fandom.com/wiki/Category:Galleries)