Level 6
Mobile development
CRN: 35801
Coursework Semester one:
An interactive app for android

course	SOFTWARE ENGINEERING
Instructor	IAN DRUMM
Student ID	@00592983
Student Username	AGD626
Due Date	6TH JANUARY 2022.
Word Count	870

Table of Contents

1 Overview………...2
1.1 Background...2
1.2 Game Overview...2
2 Implementation....2
2.1 First Fragment.…..2
2.2 Second Fragment...3
2.3 Third Fragment.………...4
2.4 Fourth Fragment..5
2.5 Fifth Fragment.…………..6
3 UML Diagrams.….………..7
4 Android Studio and Kotlin critique..7

1.1 Background

The task was to create an Android game with a science-based topic suitable for Secondary students. This task was completed with knowledge gained from the implementation of module notes and without the use of third-party libraries, textbook material and online tutorials.

1.2 Game Overview

This game was developed using Kotlin in the Android Studio development environment. It utilises Object-Oriented Programming, interactive graphics and sensors. It is an interactive application making use of a navigation graph, 5 fragments and view binding. The application is connected using an array of buttons linking actions between fragments to allow for navigation. It contains 3 levels all being their own miniature activity focused on animals and the food chain. I used (.wav) for sounds and (.png) for images for consistency in this game.

2.1 First Fragment

Contains text view:

Displays text view that gives introduction to the game application.

Start Button:

Links first fragment and second fragment by invoking the action in nav graph xml.

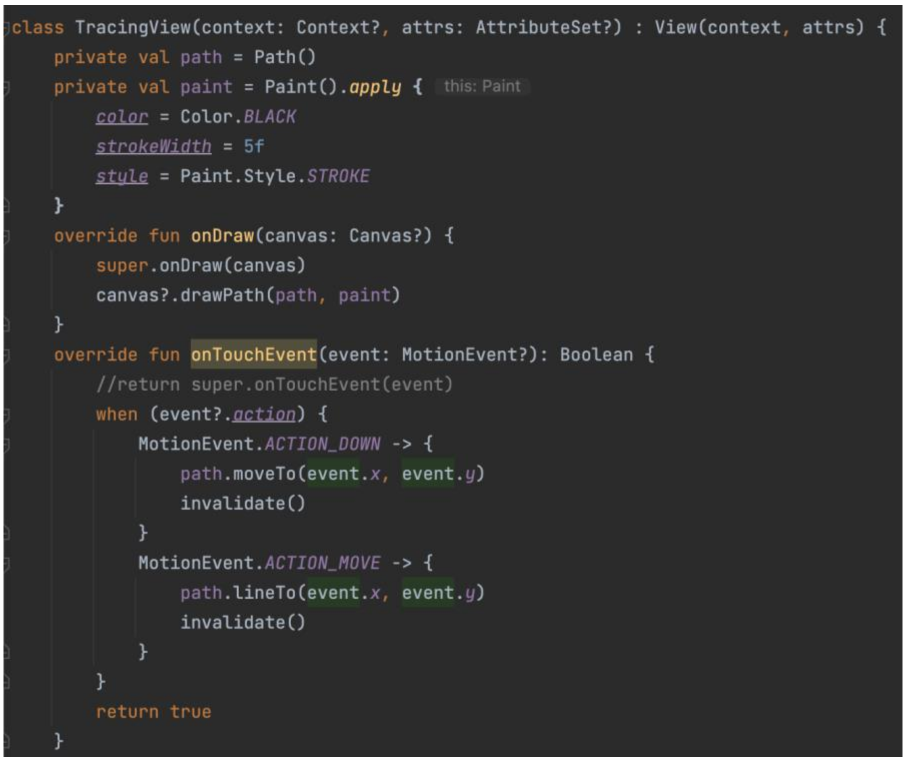
2.2 Second Fragment

Overview:

This level allows the user trace between blank images and animal pictures. Clicking on the blank image produces a sound and the user traces the right animal to the right sound.

Tracing View:

I created a view class called TracingView that uses the onTouchEvent and onDraw functions to allow the user the ability to draw lines.



```kotlin
class TracingView(context: Context?, attrs: AttributeSet?) : View(context, attrs) {
    private val path = Path()
    private val paint = Paint().apply {  this: Paint
        color = Color.BLACK
        strokeWidth = 5f
        style = Paint.Style.STROKE
    }
    override fun onDraw(canvas: Canvas?) {
        super.onDraw(canvas)
        canvas?.drawPath(path, paint)
    }
    override fun onTouchEvent(event: MotionEvent?): Boolean {
        //return super.onTouchEvent(event)
        when (event?.action) {
            MotionEvent.ACTION_DOWN -> {
                path.moveTo(event.x, event.y)
                invalidate()
            }
            MotionEvent.ACTION_MOVE -> {
                path.lineTo(event.x, event.y)
                invalidate()
            }
        }
        return true
    }
```

I then added a view in my fragments xml and changed its type to TracingView to implement the tracing features to the fragment.

Next Button:

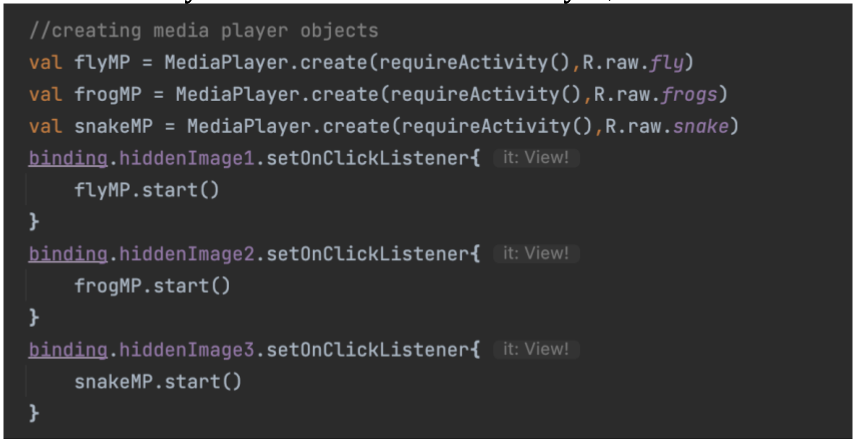
Links second fragment and third fragment by invoking the action in nav graph xml.

Back Button:

Links second fragment and first fragment by invoking the action in nav graph xml.

Image Buttons (Blank Image):

Gives the blank images for tracing. Also gives sound for identifying where to trace to this is done by utilisation of the MediaPlayer, OnClickListener functions in the fragment.



```kotlin
//creating media player objects
val flyMP = MediaPlayer.create(requireActivity(),R.raw.fly)
val frogMP = MediaPlayer.create(requireActivity(),R.raw.frogs)
val snakeMP = MediaPlayer.create(requireActivity(),R.raw.snake)
binding.hiddenImage1.setOnClickListener{  it: View!
    flyMP.start()
}
binding.hiddenImage2.setOnClickListener{  it: View!
    frogMP.start()
}
binding.hiddenImage3.setOnClickListener{  it: View!
    snakeMP.start()
}
```

Image Buttons (Animal Image):

Gives the animal images for tracing.

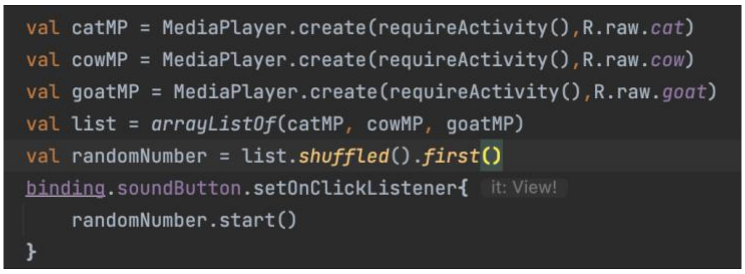
2.3 Third Fragment

Overview:

This level allows the user draw animal pictures based on a randomly generated sound it also allows users change the colour of paint being used with a colour changer button.

Generate Sound Button:

Utilises onViewCreated function in the fragment by creating a list of MediaPlayer objects adding them to an arrayList and using shuffled function to return a random sound to the OnClickListener.



```
val catMP = MediaPlayer.create(requireActivity(),R.raw.cat)
val cowMP = MediaPlayer.create(requireActivity(),R.raw.cow)
val goatMP = MediaPlayer.create(requireActivity(),R.raw.goat)
val list = arrayListOf(catMP, cowMP, goatMP)
val randomNumber = list.shuffled().first()
binding.soundButton.setOnClickListener{  it: View!
    randomNumber.start()
}
```

Back Button:

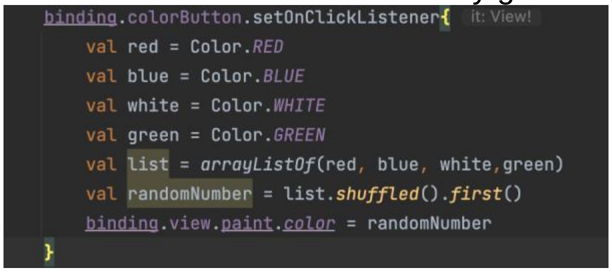
Links third fragment to the second fragment by invoking the action in nav graph xml.

Next Button:

Links third fragment and fourth fragment by invoking the action in nav graph xml.

Change Colour Button:

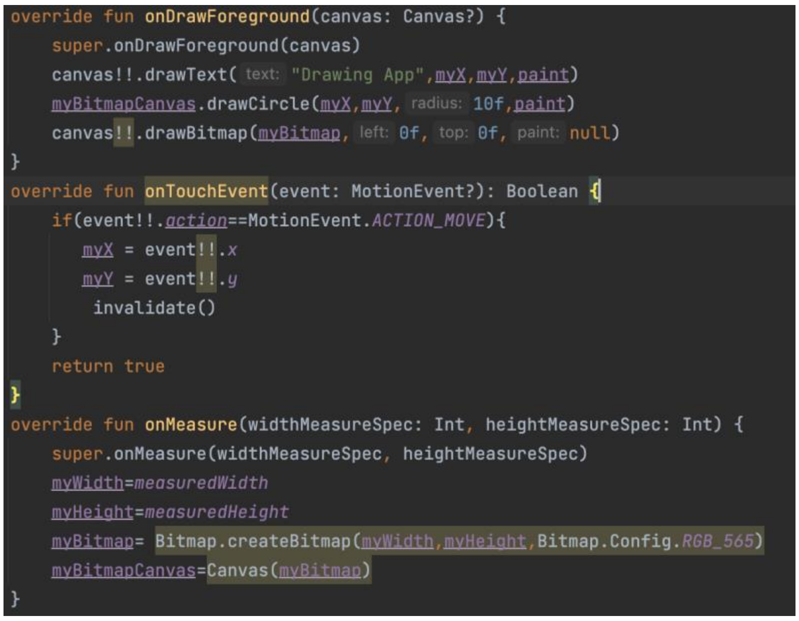
Uses OnClickListener to randomly generate colour when button is clicked.



```
binding.colorButton.setOnClickListener{  it: View!
    val red = Color.RED
    val blue = Color.BLUE
    val white = Color.WHITE
    val green = Color.GREEN
    val list = arrayListOf(red, blue, white,green)
    val randomNumber = list.shuffled().first()
    binding.view.paint.color = randomNumber
}
```

LevelTwoView:

This is a view Class I created to allow the user to draw the animal for the generated sound. This was achieved using the onDrawForeground, onMeasure and onTouchEvent functions.


```
override fun onDrawForeground(canvas: Canvas?) {
    super.onDrawForeground(canvas)
    canvas!!.drawText( text: "Drawing App",myX,myY,paint)
    myBitmapCanvas.drawCircle(myX,myY, radius: 10f,paint)
    canvas!!.drawBitmap(myBitmap, left: 0f, top: 0f, paint: null)
}
override fun onTouchEvent(event: MotionEvent?): Boolean {
    if(event!!.action==MotionEvent.ACTION_MOVE){
       myX = event!!.x
       myY = event!!.y
        invalidate()
    }
    return true
}
override fun onMeasure(widthMeasureSpec: Int, heightMeasureSpec: Int) {
    super.onMeasure(widthMeasureSpec, heightMeasureSpec)
    myWidth=measuredWidth
    myHeight=measuredHeight
    myBitmap= Bitmap.createBitmap(myWidth,myHeight,Bitmap.Config.RGB_565)
    myBitmapCanvas=Canvas(myBitmap)
}
```
I then added a view in my fragments xml and changed its type to LevelTwoView to implement the drawing features to the fragment.

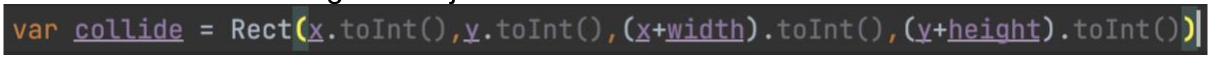
2.4 Fourth Fragment

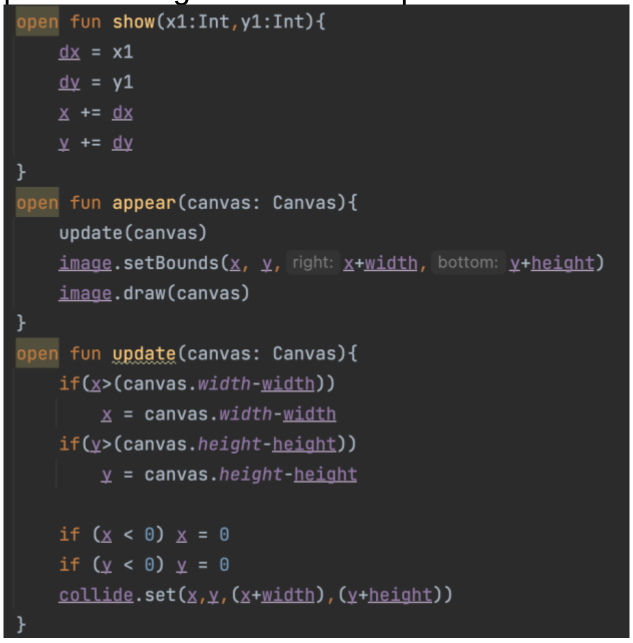
Overview:

This level allows the user feed the frog by tilting the phone to collide with flies. Upon collision/intersection the flies disappear.

LevelThreeObject:

I created a game Object class called LevelThreeObject, where a rectangle tracks collisions between the game objects.

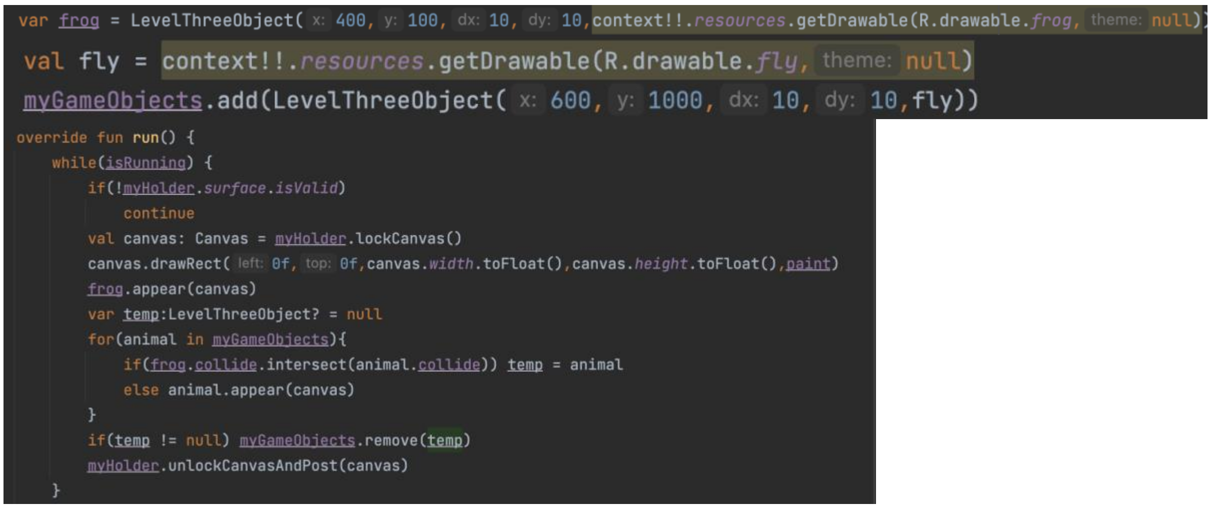

```
var collide = Rect(x.toInt(),y.toInt(),(x+width).toInt(),(y+height).toInt())
```
I used show appear and update functions to control the position of the objects and put a rectangle at the same position of the object with the same size.


```
open fun show(x1:Int,y1:Int){
    dx = x1
    dy = y1
    x += dx
    y += dy
}
open fun appear(canvas: Canvas){
    update(canvas)
    image.setBounds(x, y, right: x+width, bottom: y+height)
    image.draw(canvas)
}
open fun update(canvas: Canvas){
    if(x>(canvas.width-width))
        x = canvas.width-width
    if(y>(canvas.height-height))
        y = canvas.height-height

    if (x < 0) x = 0
    if (y < 0) y = 0
    collide.set(x,y,(x+width),(y+height))
}
```

MySurfaceView:

This is a SurfaceView Class I created to allow the user to move the animal by tilting .
 This was achieved using the run function and creating LevelThreeObjects.

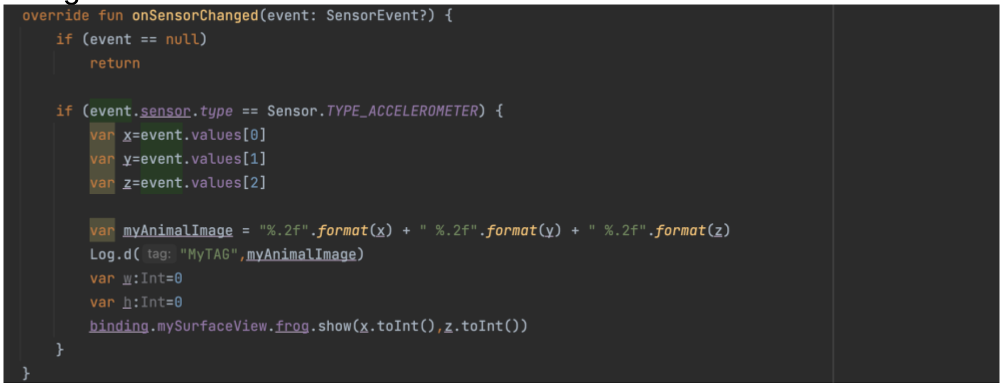


```
var frog = LevelThreeObject( x: 400, y: 100, dx: 10, dy: 10,context!!.resources.getDrawable(R.drawable.frog, theme: null))
val fly = context!!.resources.getDrawable(R.drawable.fly, theme: null)
myGameObjects.add(LevelThreeObject( x: 600, y: 1000, dx: 10, dy: 10,fly))
override fun run() {
    while(isRunning) {
        if(!myHolder.surface.isValid)
            continue
        val canvas: Canvas = myHolder.lockCanvas()
        canvas.drawRect( left: 0f, top: 0f,canvas.width.toFloat(),canvas.height.toFloat(),paint)
        frog.appear(canvas)
        var temp:LevelThreeObject? = null
        for(animal in myGameObjects){
            if(frog.collide.intersect(animal.collide)) temp = animal
            else animal.appear(canvas)
        }
        if(temp != null) myGameObjects.remove(temp)
        myHolder.unlockCanvasAndPost(canvas)
    }
```

The for loop allows animal GameObjects to be destroyed upon intersection by making it null.

Fragment:

In this fragment I used the accelerometer sensor by using the sensor manager. Then in the onSensorChanged function I called my frog levelThreeObject to show using the show function and the x and z values from the accelerometer to move it.



```
override fun onSensorChanged(event: SensorEvent?) {
    if (event == null)
        return

    if (event.sensor.type == Sensor.TYPE_ACCELEROMETER) {
        var x=event.values[0]
        var y=event.values[1]
        var z=event.values[2]

        var myAnimalImage = "%.2f".format(x) + " %.2f".format(y) + " %.2f".format(z)
        Log.d( tag: "MyTAG",myAnimalImage)
        var w:Int=0
        var h:Int=0
        binding.mySurfaceView.frog.show(x.toInt(),z.toInt())
    }
}
```

Finish Game Button:

Links fourth fragment and fifth fragment by invoking the action in nav graph xml.

Back Button:

Links fourth fragment to the third fragment by invoking the action in nav graph xml.

2.5 Fifth Fragment

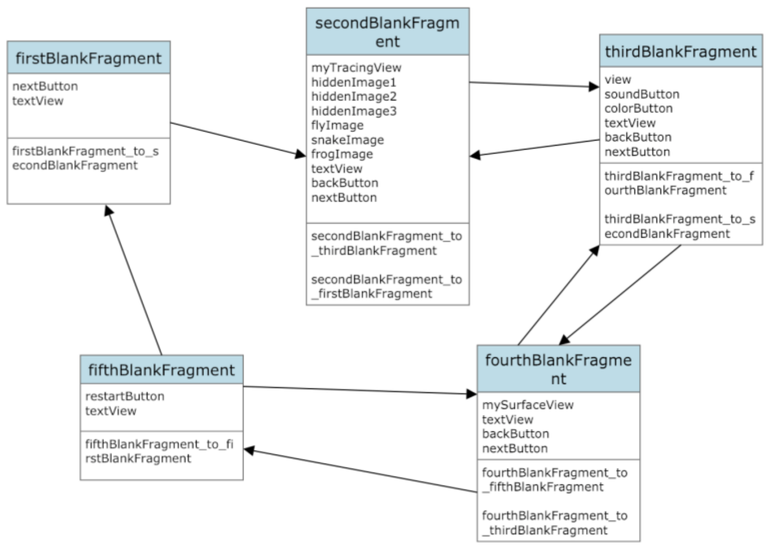
Contains text view:

Displays text view that gives thank you for playing the game application.

Restart Button:

Links fifth fragment to the first fragment by invoking the action in nav graph xml.

3 UML Diagram



4 Android Studio and Kotlin critique

Below I share my experience developing an android app against my experience developing java applications.

Pros:

- The experience using Android Studio was pleasant as I have enjoyed utilizing many programming practices.
- The design view for the xml files is very helpful.
- Kotlin shares many similarities with java making it a very smooth transition.

Cons:

- The variety of screen sizes that uses android makes it even more tedious to develop android applications
- Android Studio IDE gave general problems while developing.