Lab Session 7 – Introducing Dynamic Content to Arkanoid

.Objectives

The objectives for this week are:

- Add a thread to Arkanoid to produce dynamic visual content within the game.
- Adding a Block that moves around the game.
- Producing a Block Constructor that is separate to the Block and BreakBlock class.

.Tasks

Task 1

- A requirement of the Arkanoid game is to have a set of blocks to be destroyed. To be able to move forward, make sure that you have completed the previous weeks labs, so that you have a number of blocks constructed in rows.
- This pseudo code may help you construct an array of blocks:

```
Create an ArrayList of BreakBlocks

   For ( x = 0; While leftHandSideOfBlock is less than width
   of the screen, x++){

         leftHandSideOfBlock = x * (blockWidth +
         paddingBetweenBlocks);

         Add a new block to the array including
         leftHandSideOfBlock

     }

Reset the leftHandSideOfBlock to 0;
```

- There also needs to be a loop for the columns.
- Talk to your lab tutor, if you are unsure of any of the previous elements that have been covered.
- Open up your Arkanoid game that you have produced over the previous weeks.

Task 2

- The use of classes is fundamental to Androids structure. During the course of this lab we will construct a number of classes to aid the construction of the Arkanoid game.
- To produce dynamic content we need to create a thread within our Arkanoid game. The thread will sit within the **View.class** and will display the content that we place into the onDraw() method that we are overriding.
- A simple way to create a thread is to use a Handler. We discussed the use of a thread handler in the lecture of Week 5. If you are unsure of how this works, look through the lecture slides that accompany the lecture.
- Add to your GameView a Handler. This can take the below form:

```
//Create thread handler to update
private Handler hHandler;
```

- In the constructor of the GameView, create a new Handler object.

```
//Handler thread to update screen
hHandler = new Handler();
```

- We need now to have a Runnable object that can be controlled by our handler. This can be constructed using the following code:

```
private Runnable r = new Runnable() {
    @Override
    public void run() {
        invalidate();
    }
};
```

- To manipulate the thread with the handler, add the below code to the end of your onDraw method.

```
hHandler.postDelayed(r, 10);
```

- The handler runs the thread but places into it a delay (the second variable). This acts as a way of dictating the speed at which the thread runs.
- In order to see how this works, we need to **update** a part of the game. As a result, we need to create an **update()** method. The update is run each time that the onDraw

method is called. The invalidate() method within run() refreshes the screen and re-draws the components giving the sense of movement.

- Create an update method in your GameView, for example:

```
public void update (){
    //Update the positions of objects in here.
}
```

- Call this in the onDraw method.
- Create a single block that you can update the position of in the update method.
- Have the block move down the screen, and when it reaches the bottom it returns to the top.
- What could be the issues of running a thread in this way for a game loop? Why might we want to construct a game loop in a different way to this?
- **Next week we will look at the use of implementing a Thread more directly within a game loop.**

Task 3

- Create a separate BlockConstrutor class that can be used to construct an array of BreakBlocks.
- This will sit within the **model package.**
- Have the BlockConstructor return an ArrayList of BreakBlocks.
- Note: an ArrayList can be passed another ArrayList into it following construction.
- For instance see the below code. Here the break block constructor is created. The blockConstructor method returns an ArrayList of BreakBlocks that are passed to the bbArray .

```
//Create a blockconstructor
BlockConstructor bc = new BlockConstructor();

//Create an array of Breakblocks
ArrayList<BreakBlock> bbArray;
```

```
//Create new breakblock array and pass in break block
construction
bbArray = new ArrayList<BreakBlock>(bc.blockConstruct());
```

Task 4

- Optimising a game to support multiple different devices is a key part of developing mobile games. We will discuss this at length as we progress through this module.
- One element of this process is to be aware of differing screen sizes of different devices.
- Scale the blocks that you have in your array of blocks so that they fit to the size of the screen.
- How might you go about doing this? The width of the screen is supplied from the main activity. You also have a width of the block. The block width can be adapted to make the blocks scale to the size of the screen. How would adapt this value automatically?
- Add this process to the block constructor that you have created.
- The use of the remainder operator may be useful in this instance (%).
- **If you are unsure of this process, discuss this with your lab tutor.**
- Try out your method on different emulators to see if it works.