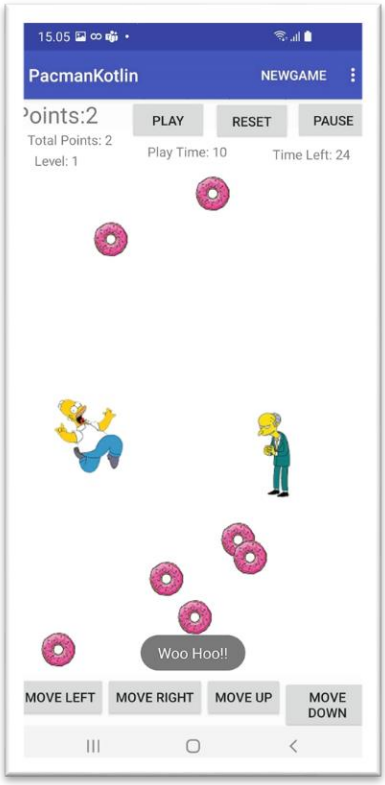
Homer Simpson PacMan

By Ruth Moritz Mikkelsen - Android Development Spring 2021

Github : https://github.com/RMikkelsen/Simpsons-Pacman

Intro



The app I created was based on a PacMan template. I changed the theme so that instead of PacMan picking up gold coins and having ghosts as the enemy, it's Homer Simpson picking up donuts and Mr. Burns as the enemy – which I find highly amusing.

The current functionality allows for the user to press "play" to start Homer moving. Mr. Burns is on the same timer as Homer so he moves as Homer moves, although with opposite directions. The user can press one of four directions to move Homer. When Homer "eats" a donut, a toast pops up saying "Woohoo!". If Homer runs into Mr. Burns there are a few things that happen starting with a toast popping up saying "Doh!", next the game stops preparing to start a new game, and a second toast pops up saying "Game Over".

I also added a timer that counts down and stops the game with a toast that says "Time's Up Game Over!". There are also New Game buttons to start a new game, "reset", which has the same functionality as new game, and pause to pause the timers. I also added the extra features of creating new levels with Finally, I tested the app on a virtual emulator as well as on 2 different devices.

Control Buttons

The template we were given started out with a button to move to the right. I was able to add the 3 additional direction buttons in the activity_main.xml file using the design view, set up the constraints, and add the string texts, and ids. To get the buttons to move to the bottom of the screen, I switched to the code view and wrapped the <LinearLayout> in a <RelativeLayout>, allowing me to align the buttons with the bottom.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/and
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    xmlns:tools="http://schemas.android.com/tools"
    tools:context="org.pondar.pacmankotlin.MainActivity">

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:id="@+id/buttons"
    android:orientation="horizontal"
    android:layout_alignParentBottom="true">
    <Button
        android:id="@+id/moveLeft"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Move Left" />

    <Button
        android:id="@+id/moveRight"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Move Right"/>
```

```
fun movePacmanRight(pixels: Int) {
    if (pacx + pixels + pacBitmap.width < w) {
        pacx = pacx + pixels
        doCollisionCheck()
        gameView!!.invalidate()
    }
}
fun movePacmanLeft(pixels: Int) {
    if (pacx > 0) {
        pacx = pacx - pixels
        doCollisionCheck()
        gameView!!.invalidate()
    }
}
fun movePacmanUp(pixels: Int) {
    if (pacy > 0) {
        pacy = pacy - pixels
        doCollisionCheck()
        gameView!!.invalidate()
    }
}
fun movePacmanDown(pixels: Int) {
    if (pacy + pixels + pacBitmap.height < h) {
        pacy = pacy + pixels
        doCollisionCheck()
        gameView!!.invalidate()
    }
}
```

Within the code, inside the game class, I edited the function to move pacman (or Homer) Right, and created new functions for him to move in the other directions. These are later called in the MainActivity file and the code there is edited to work with the timer.

Moving Homer (PacMan) via timer

A timer was created in the MainActivity file to move Homer automatically (in my app when the user pushes play) and the onClick methods were adjusted to work with the direction associated with a number rather than calling in the direction methods from the game class the whole time. This allows the user to have Homer moving and only use the buttons (with the onClick) when he needs to change direction. Within the "if else" statements, the directions are defined by how many pixels Homer should move and calls the methods from the game class.

```
private var myTimer: Timer = Timer()
```

```
game?.running = true //should the game be running?
//We will call the timer 5 times each second
myTimer.schedule(object : TimerTask() {
    override fun run() {
        timerMethod()
    }
}, delay: 0, period: 200) //0 indicates we start now, 200
//is the number of miliseconds between each call
```

```
moveRight.setOnClickListener { it: View!
    game?.direction = 0
}
moveLeft.setOnClickListener { it: View!
    game?.direction = 1
}
moveUp.setOnClickListener { it: View!
    game?.direction = 2
}
moveDown.setOnClickListener { it: View!
    game?.direction = 3

}
```

```
if (game?.direction == 0) { // move right
    game?.movePacmanRight( pixels: 40)
    game?.moveEnemyLeft( pixels: 40)

} else if (game?.direction == 1) {
    game?.movePacmanLeft( pixels: 40)
    game?.moveEnemyRight( pixels: 40)

} else if (game?.direction == 2) {
    game?.movePacmanUp( pixels: 40)
    game?.moveEnemyDown( pixels: 40)

} else if (game?.direction == 3) {
    game?.movePacmanDown( pixels: 40)
    game?.moveEnemyUp( pixels: 40)
}
}
```

Adding Donuts & Collision Detection

To add donuts (referred to as gold coins in the code for simplicity's sake), I created a gold coin class, and upon initializing them in the code, added them with a random function so there would be a random number of donuts with each new game/level from 0-8 (so max 9).

```
class GoldCoin(
        var coinX: Int, var coinY: Int, var taken: Boolean= false) {
```

```
fun initializeGoldcoins() {
    //DO Stuff to initialize with some coins.
    //DO Stuff to initialize the array list with some coins.

    var minX: Int = 0
    var maxX: Int = w - coinBitmap.width
    var minY: Int = 0
    var maxY: Int = h - coinBitmap.width
    val random = Random()
    for (i in 0..8) {
        var randomX: Int = random.nextInt( bound: maxX - minX + 1) + minX
        var randomY: Int = random.nextInt( bound: maxY - minY + 1) + minY
        coins.add(GoldCoin(randomX, randomY,  taken: false))
    }
    coinsInitialized = true

}
```

To allow little Homer to "eat" the donuts, the majority of that functionality was created in the collision detection. This was a function that could be called in each direction Homer moves. I went with the Bounding Box option as seen below. Here it starts with a for loop targeting each coin in the coins, then the if statement goes through and checks the coordinates starting from the upper left corner to bottom right corner. Then the Toast makes the text "woohoo" if the coins are taken. With each coin taken, the points and total points increase by 1. The text views are called in to display the points and total points. The next if statement is if all of the coins are taken, the level increases by 1 and displays in the text view and the new level is returned (which previously was the newGame).

```
fun doCollisionCheck() {
    for (coin in coins) {
        if (pacx + pacBitmap.width >= coin.coinX && pacx <= coin.coinX + coinBitmap.width && pacy + pacBitmap.height >= coin.coinY && pacy <= coin.coinY + coinBitmap.height && !coin.taken) {
            Toast.makeText(this.context,  text: "Woo Hoo!!", Toast.LENGTH_SHORT).show()
            coin.taken = true

            points += 1
            totalPoints  += 1
            pointsView.text = "${"Points:"}$points"
            totalPointsView.text = "${"Total Points:"} $totalPoints"
        }
        if (points == coins.size && coin.taken) {
           // Toast.makeText(this.context, "YOU WON!!", Toast.LENGTH_SHORT).show()
            level += 1
            levelsView.text = "${context.resources.getString(R.string.Levels)} ${level}+1"
            return newLevel()
        }
    }

}
```

Adding Enemies & Collision Detection

Adding enemies was similar to the coins in that it required an additional bitmap and the related functionality. However, because the enemies move, directions were created for each enemy (like Homer) and the enemy was added to Homer's timer so the enemy (Mr. Burns) could move in relation to Homer.

```
fun moveEnemyLeft(pixels: Int) {
    for (enemy in enemy) {
        if (enemy.enemyx > 0) {
            enemy.enemyx = enemy.enemyx - pixels
        }
    }
    doCollisionCheck2()
    gameView!!.invalidate()
}

fun moveEnemyUp(pixels: Int) {
    for (enemy in enemy) {
        if (enemy.enemyy > 0) {
            enemy.enemyy = enemy.enemyy - pixels
        }
    }
    doCollisionCheck2()
    gameView!!.invalidate()
}

//works
fun moveEnemyDown(pixels: Int) {
    for (enemy in enemy) {
        if (enemy.enemyx + pixels + enemyBitmap.height < h) {
            enemy.enemyy = enemy.enemyy + pixels


        }
    }
    doCollisionCheck2()
    gameView!!.invalidate()
}
```

```
if (game?.direction == 0) { // move right
    game?.movePacmanRight( pixels: 40)
    game?.moveEnemyLeft( pixels: 40)

} else if (game?.direction == 1) {
    game?.movePacmanLeft( pixels: 40)
    game?.moveEnemyRight( pixels: 40)

} else if (game?.direction == 2) {
    game?.movePacmanUp( pixels: 40)
    game?.moveEnemyDown( pixels: 40)

} else if (game?.direction == 3) {
    game?.movePacmanDown( pixels: 40)
    game?.moveEnemyUp( pixels: 40)
}
```

For the collision detection, my app kept crashing when Homer ran into Mr. Burns. This was solved by moving the collision checks ouside of the for loop in the enemy movements and making a var playerDead = false within the collision check, as seen below.

```
//check if the pacman touches an enemy
fun doCollisionCheck2() {
    var playerDead = false
    for (enemy in enemy) {
        if (pacx + pacBitmap.width >= enemy.enemyx && pacx <= enemy.enemyx + enemyBitmap.width && pacy + pacBitmap.height >= enemy.enemyy && pacy <= enemy.enemyy + enemyBitmap.height && !enemy.alive) {
            Toast.makeText(this.context,  text: "DOH! GAME OVER!", Toast.LENGTH_SHORT).show()
            playerDead = true
            enemy.alive = false
        }
    }
    if (playerDead == true) {
        return newGame()
    }
}
```

Pause/Play/Reset Buttons

Aside from the direction and new game buttons, I also added a play (to start the game), pause (to pause the game and timers), and reset. The reset has similar functionality to the new game button, but since I added levels, it should actually just reset the level. In the onCreate function, I've assigned the click listener and later on, in the onClick function below, I've assigned the functionality of each button. If play, then the game is running, if pause then not running, if reset, the first counter returns to zero, the countdown timer starts back at 26, and a new game is called. Running should be stopped so the user has to press play to start.

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    //makes sure it always runs in portrait mode
    requestedOrientation = ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
    setContentView(R.layout.activity_main)
    play.setOnClickListener(this)
    pause.setOnClickListener(this)
    reset.setOnClickListener(this)
```

```
override fun onClick(v: View) {
    if (v.id == R.id.play) {
        game?.running = true
    } else if (v.id == R.id.pause) {
        game?.running = false
    } else if (v.id == R.id.reset) {
        game?.counter = 0
        game?.counter2 = 26
        game?.newGame() //you should call the newGame method instead of this
        game?.running = false
        textView.text = "Play Time: {game?.counter}"
    }
}
```

Timer 2 (Countdown)

I created a second timer to create a countdown in the MainActivity file. This is connected to the pause, play, reset, and new game functionality. This runs in intervals of 1000 to equate to 1 second. Within the timerTick2 method, I set the time to decrease in time and start at 26 (I chose this because it seemed like a good amount of time upon testing). The first if statement means that if the second counter (specifically for the second timer and called from the game class) is running (== true) then the game counter should decrease (--), and the text view associated will show how much time is left and display the time remaining. The second if statement means that if the second counter (called from the game class) and is less than or equal to zero, then to make a Toast that the time is up and

stop the game running.

```
//this timer for countdown
Timer2.schedule(object : TimerTask() {
    override fun run() {
        timerMethod2()
    }
}, delay: 0, period: 1000) //0 indicates we start now, 1000 is for each second
```

```
// this is for 2nd timer
private val timerTick2 = Runnable {
    //This method runs in the same thread as the UI.
    // so we can draw
    if (game?.running == true) {
        game!!.counter2--
        //update the counter - notice this is NOT seconds in this example
        //you need TWO counters - one for the timer count down that will
        // run every second and one for the pacman which need to run
        //faster than every second
        textView3.text = "Time Left: {game?.counter2}"
        if (game!!.counter2 <= 0) {
            Toast.makeText( context: this, text: "TIME'S UP! GAME OVER!", Toast.LENGTH_SHORT).show()
            game?.running = false
            game?.newGame()
        }
    }
```

Extra Functionality

For my extra app component, I created additional levels with 2 new text views that show the level and the total points (adding the points for each level). To start, I added new text views as properties to the game class, one for the level and one for the total points, and later created variables for them (see below). I used the majority of data from the newGame() method for the newLevel() method so I could use that in my collision detection to return either a new game or new level. With the levels, the number of Mr. Burns increases by 1 for each level.

```
class Game(private var context: Context, view: TextView, view2: TextView, view3: TextView) {
```

```
var level = 0
var levelsView: TextView = view2

var totalPoints = 0
var totalPointsView: TextView = view3
```

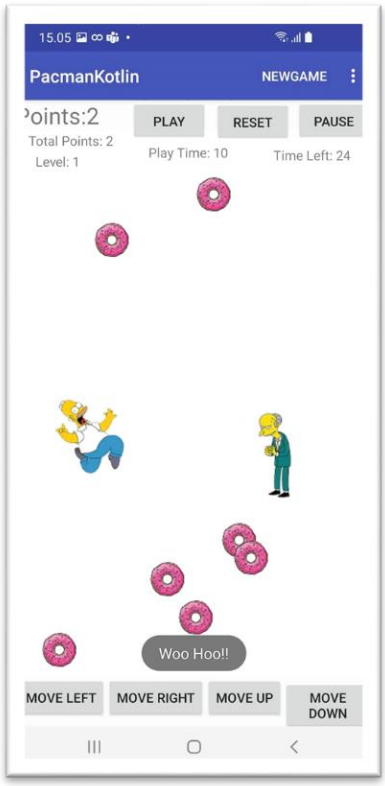
```
fun newLevel() {
    pacx = 100
    pacy = 400 //just some starting coordinates - you can chan
    //reset the points
    coins.clear()
    coinsInitialized = false
    enemy.clear()
    enemyInitialized = false
    points = 0
    pointsView.text = "${"Points:"} $points"
    levelsView.text = "${"Level:"} ${level + 1}"
    totalPointsView.text = "${"Total Points:"} $totalPoints"
    counter = 0
    counter2 = 26
    running = false
    gameView?.invalidate() //redraw screen
}
```

Testing

To test my Homer app, I ran it on an emulator, a real device, and checked my .apk file by sending it to another student who tested it on their device. The emulator was a pixel 3 API 30. I didn't test it on older versions however because by the time it would (if ever) go live, I expect phones to be much more advanced than the current standard. The device I tested it on was a Samsung Galaxy S10, SM-G973F (as shown in Android Studio). The device my .apk file was tested on was a Huawei p30 pro. On both devices, the app ran as it should, properly going to the next level, toast showing up, and not crashing. The app also worked fine on the emulator; however, the toasts didn't seem to show up. I thought perhaps they weren't working originally until I connected to an actual device.

Issues & Conclusion

I really enjoyed working in android studio and making this app, even though certain aspects of the process were more challenging. I would have really like to add in sound effects associated with the collisions (I downloaded DOH and WooHoo sound files) and some background music. I also wanted to have some animations (donuts spinning, Homers legs running, etc.). I also prepared additional enemies (Flanders, Marge's sisters) with the intent to add them in later. Some issues in my code are that some code may be redundant, and possibly misplaced in the file. Some bugs that occur are after a few levels it might start a new game, and I only realized this before hand-in so haven't yet had a chance to figure it out. Mr. Burns also is able to move partially off screen so I would like to fix that later on as well. Finally, I don't have the center of Homer interacting with the center of the donuts or Mr. Burns, so the game isn't very accurate, but overall functionality works and I personally find the Simpsons theme pretty cute.



The Final Look 😊