Klicka på cirklarna

Av Mellissa Qholizadeh, student ID: meqh4654

Sammanfattning

Denna applikation är ett spel som heter “Pop the circles” och spelet går ut på, precis som titeln lyder, att klicka på olika färgade cirklar. Det finns två olika spel-lägen, ett där användaren spelar tills den förlorar (endless mode) och ett där användaren ska få så mycket poäng som möjligt på 30 sekunder (time mode). Användaren får poäng när dom klickar på en cirkel, men man måste se upp för att inte klicka på en röd cirkel, för då förlorar man ett liv. Förlorar man tre liv då tar spelet slut.

Framtagande

Idé samling

När jag gjorde den frivilliga uppgiften 3.8 rita 2D-grafik fick jag en idé av att göra ett spel där användaren klickar på cirklar som kommer upp på skärmen. I uppgiften så skapade jag en canvasView och sen en knapp som visade cirklar i den canvasView:n varje gång användare tryckte på knappen:

Därefter började jag spinna vidare på det, att användaren skulle kunna få poäng om den klickar på cirklarna, men att man ska se upp för en viss färg, exempelvis röd. Till en början tänkte jag att spelet antingen skulle vara på tid eller att det skulle fortsätta tills användaren klickat på en röd cirkel tre gånger. Sedan tänkte jag att det kanske går att implementera båda, för om grunderna i spelet sitter bör det inte vara så svårt att implementera flera olika gamemodes.

Planering och start av projekt

Efter att jag bestämde mig för att göra detta spel så målade jag upp en form av plan, där jag ritade ut olika steg i projektet. Detta är ungefär så det såg ut:

1. Föra över all kod från uppgift 3.8 som start
2. Skapa cirklar continuerligt, istället för att klicka på knappen flera gånger
3. Kunna klicka på en cirkel, den försvinner och användaren får en poäng
4. Skapa 3 liv, användaren förlorar 1 liv varje gång en röd cirkel klickas på
5. När användaren förlorar skapa en gameover ruta
6. Ju längre spelet går desto snabbare ska cirklarna komma

7. Implementera time mode
8. Fixa till utseende

Denna lista hjälpte mig att skapa en bra struktur på mitt arbete, medan jag utvecklade spelet kunde punkter läggas till eller tas bort. När jag tänkt ut vad spelet ska innehålla så började själva utvecklingen.

Första stegen

Som jag tidigare nämnde så startade min utveckling med uppgiften 3.8 i åtanke, därav började jag arbeta med att kopiera över all den koden. Jag skapade utöver det en förstasida/ meny och de två olika aktiviteterna: endless och time mode. Själva funktionaliteten i spelet tänke jag var lättast att utveckla till fullo i en av aktiviteterna, så jag lade undan time mode aktiviteten till senare och började på endless aktiviteten.

Det första jag ville skapa var att istället för att behöva klicka in cirklar så ville jag skapa en kontinuerlig ritning av cirklar. Det kunde skapas med hjälp av en handler och en variabel, variabeln fick namnet spawnInterval och ska visa i millisekunder hur ofta det ska ritas en ny cirkel. Jag skapade även en remove funktion som ska ta bort den först skapade cirkeln om användaren inte tryckt på den, så att hela skärmen inte blir fylld med röda cirklar. Men jag märkte under spelets gång att den funktionen inte gjorde så mycket. Så jag lämnade den som den är, att den tar bort en cirkel var femte sekund.

I detta steg valde jag även att hålla mig till tre olika färger; blå, grön och röd. Jag fortsätter att variera storleken på cirklarna med hjälp av random, egentligen bara för att få lite variation. I samband med detta gjorde jag så att cirklarna inte kan skapas halvvägs utanför canvasen, det gjorde jag med hjälp av att minska den tillgängliga bredden och höjden där cirklarna kan skapas på.

Poäng och liv

För att börja implementera ett poäng- och livssystem behöver användaren kunna klicka på en cirkel för att ta bort den. Det går att göra med hjälp av Androids TouchEvent hantering, som fångar användarens touch-händelser på skärmen, i detta spelets fall ACTION_DOWN. Sedan med hjälp av koordinationerna på cirkeln går det att ta bort den specifika cirkeln som användaren klickade på. I samband med detta klicka och att en cirkel tas bort var det lätt att implementera en score variabel, som adderas varje gång en användare klickar på en cirkel.

Därefter ville jag implementera så att användaren kan se i realtid hur mycket poäng den har. Det gick att göra med hjälp av att skapa en listener som uppdaterades varje gång en cirkel blev klickad på, sedan skickades uppdateringen till Endless-Activity, där själva UI:n uppdateras. I och med att jag testade att klicka runt lite på mitt spel så märkte jag att om två cirklar låg på varandra så blev klickade på den underliggande, när det ska vara på den översta (den närmast skärmen). För att lösa det behövde jag iterera över hela listan av cirklar för att ta bort den som senast blev tillagt i listan (av två cirklar på varandra).

Att införskaffa 3 liv var inte allt för komplicerat, jag behövde en ny variabel som höll koll på användarens liv. Variabelns värde (3) får minus ett om användare klickar på en röd cirkel och när användaren förlorar tre liv är spelet över. För att visa att spelet var över behövde jag skapa en dialogruta, som visar hur mycket poäng användaren fick, alternativet att spela igen och alternativet att gå till huvudmenyn. För att kunna spela igen behövde jag även skapa en resetGame metod, där alla variabler nollställs. Till en början så fortsatte spelet i bakgrunden när min gameover dialog var uppe och sedan användaren tryckte på try again så blev det dubbla hastigheten. För att lösa det skapade jag två flaggor isSpawning och gameOver, för att visa funktionen startSpawning att det bara ska vara en tråd av den.

Öka hastighet

När jag kände att allting fungerade med klicken och cirklarna så ville jag implementera att “spawn” hastigheten skulle öka. För att skapa det behövde jag en till handler och en variabel som dynamiskt ändrar på sig (dynamicSpawnInterval), den variabeln har samma startvärde som den tidigare spawnInterval variabeln. Så i den tidigare spawn loopen byter jag ut den tidigare spawn variabeln med den nya dynamiska. Sedan skapar jag en till loop som minskar intervallen med en viss andel procentenheter, detta uppdateras exempelvis var åttonde sekund. Sedan ville jag även begränsa det så att intervallet inte kan bli mindre än noll.

Time mode

När jag var nöjd med hur endless mode-aktiviteten såg ut och mitt spel fungerade så som jag tänkte ville jag implementera min time mode aktivitet. Den enda skillnaden är bara att spelet har en timer på 30 sekunder som räknar nedåt, för att göra det behövde jag skapa en Countdown-timer och två variabler: InitialTime och TimeLeft. De fungerar lite som de tidigare variablerna spawnInterval och den dynamiska varianten. Sen gjorde jag bara en startTimer och resetTimer funktion. Men jag ville göra så att livssystemet fortfarande fanns med här så jag har två olika game over screens, en när användaren förlorar tre liv och en när tiden tar slut. I denna fas började jag även inse att jag behöver ha en onDestroy funktion för att navigera runt i appen utan att det krashar.

Sista fixarna

När jag kände att all funktionalitet var klar så ville jag lägga till lite ändringar, jag ville göra så att titeln på de båda nivåerna var dynamiska med den nivån, trots att de delar på samma layout. Jag ville även skapa en liten tips text i början så användarna ska förstå ungefär vad spelet går ut på. Lade även till lite royalty free hissmusik på varje nivå så att användarna inte ska vara alltför understimulerade, med hjälp av MediaPlayer.

Form

Det färdiga projektet består av tre olika aktiviteter och en CanvasView, i CanvasView utförs den stora delen av hela spelet.

CustomCanvasView

I denna klass sker skapandet av cirklar, hanteringen av användar input och återställning av spelet. I samma fil befinner det sig en annan klass vid namn Circle och den är till för att Canvas-klassen ska kunna skapa objekt av cirkel-klassen med följande attribut: x, y, radius och color.

Den första funktionen i denna klass är startSpawning:

```
fun startSpawning(){
    if (gameOver || isSpawning) return
    isSpawning = true

    handler.post(object: Runnable{
        override fun run() {
            addRandomCircle()
            invalidate()
            if (!gameOver){
                handler.postDelayed(this, dynamicSpawnInterval)
            }
        }
    })

    difficultyHandler.postDelayed(object: Runnable{
        override fun run() {
            if(!gameOver){
                dynamicSpawnInterval = (dynamicSpawnInterval * 0.9).toLong()
                if(dynamicSpawnInterval < 50){
                    dynamicSpawnInterval = 50
                }
                difficultyHandler.postDelayed(this, 5000)
            }
        }
    }, 5000)
}
```

Den fungerar så att när användaren trycker på startgame knappen i vardera aktivitet anropas denna. Den gör först och främst så att cirklarna kan ritas kontinuerligt, intervallet de ritas på beror på dynamiccSpawnInterval variabelns värde, som är 500 (alltså en halv sekund). Så den ritar alltså¨en cirkel varje halv sekund, men samtidigt som denna handler körs så körs det en annan. Dens funktion är att minska intervallets värde var femte sekund. Tills den får värdet 50, alltså 0,05 sekund.
Som jag nämnde tidigare finns det också en remove funktion som ser ut exakt som den första handlern i startSpawning, men jag känner inte att den uppfyller så mycket funktion.

Den finns två andra viktiga funktioner som används, addRandomCircle och onDraw:

```
fun addRandomCircle() {
    val radius = (60 ≤ .. ≤ 120).random().toFloat()
    val x = random.nextFloat() * (width - 2 * radius) + radius
    val y = random.nextFloat() * (height - 2 * radius) + radius
    val color = colors.random()
    val newCircle = Circle(
        x = x,
        y = y,
        radius = radius,
        color = color
    )
    circles.add(newCircle)
}
```

```
override fun onDraw(canvas: Canvas) {
    super.onDraw(canvas)
    if (gameOver) return
    for (circle in circles) {
        val paint = Paint().apply {
            color = circle.color
            style = Paint.Style.FILL
        }
        canvas.drawCircle(circle.x, circle.y, circle.radius, paint)
    }
}
```

Det är funktionen addRandomCircle som skapar cirklarna och deras egenskaper, för att sedan lägga till dem i en lista. Radien, färgen och koordinaterna slumpas, sedan skapas cirkel objektet med de egenskaperna. onDraw funktionen ansvarar för att rita alla cirklar som finns i listan: circles, rita dem på canvas. Denna funktion anropas när någon tidigare metod använder sig av invalidate().

Min onTouchEvent metod ser ut såhär:

```
override fun onTouchEvent(event: android.view.MotionEvent): Boolean {
    if (event.action == android.view.MotionEvent.ACTION_DOWN){
        val touchedX = event.x
        val touchedY = event.y

        for (i in circles.size - 1 ≥ downTo ≥ 0){
            val circle = circles[i]
            val distance = Math.sqrt(
                ((circle.x - touchedX).toDouble().pow(2.0)) + ((circle.y - touchedY).toDouble().pow(2.0))
            )

            if (distance <= circle.radius){
                circles.removeAt(i)
                invalidate()

                if (circle.color == Color.RED){
                    lives--
                } else {
                    score++
                }
                onScoreChangeListener?.invoke(score)
                onLivesChangeListener?.invoke(lives)

                if(lives <= 0){
                    gameOver = true
                    onGameOverListener?.invoke(score)
                }

                break
            }
        }
    }
    return true
}
```

Denna funktion utgår utifrån ett MotionEvent som kan vara exempelvis att swipa eller att trycka på skärmen. I detta spelets fall är det att trycka på skärmen, vilket ACTION_DOWN syftar på. Sedan hämtas koordinaterna vart användaren tryckt, sedan startas en loop. Den är till för att först prioritera rit-ordningen så att de som är närmast skärmen försvinner om två cirklar är på varandra. Därefter beräknas cirkelns touch area, när det beräknas och funktionen hittar en cirkel som användaren trycker på så försvinner den och canvasen uppdateras. Om användaren trcker på en röd cirkel förlorar den ett liv, klickar den på en cirkel med en annan färg får användaren poäng. Poäng- och liv variabeln skickar

uppdatering till aktivitetsklasserna hela tiden, för att uppdatera UI:n. Om användarens liv blir 0 avslutas spelet. Efter allt detta skett, bryts loopen med en break.

Aktiviteter

Som jag tidigare nämnt så är de två olika aktiviteterna Endless mode och Time mode generellt väldigt lika, de sköter till stor del det grafiska. Uppdaterar poängen, tiden och användarens liv i realtid. De har varsin dialog-ruta som ser ut ungefär såhär (från Endless Mode):

```
private fun showGameOverDialog(score: Int){
    val dialog = AlertDialog.Builder(this)
        .setTitle("Game Over")
        .setMessage("Score: $score")
        .setPositiveButton("Try Again") {
            dialog, _ ->
            startGame()
            dialog.dismiss()
        }
        .setNegativeButton("Back to menu") { _, _ ->
            finish()
        }
        .setCancelable(false)
        .create()

    dialog.show()
}
```

Det finns även en liten funktion som jag lade in på slutet för att spela upp bakgrundsmusik:

```
private fun playBackgroundMusic(){
    mediaPlayer = MediaPlayer.create(this, R.raw.background_music_2)
    mediaPlayer?.isLooping = true
    mediaPlayer?.start()
}
```

Pseudo-kod

Här kommer en förklaring av hur helheten i mitt projekt fungerar i pseudokod:

FUNKTION startSpawning():
  Om gameOver ELLER isSpawning ÄR TRUE:
    RETURNERA
  isSpawning = TRUE

 UPPREPA med handler var dynamicSpawnInterval:
  OM gameOver ÄR FALSE:
    Kalla på addRandomCircle()
    Rita om skärmen (invalidate)

```
 ÖKA svårighetsgrad var 5000 ms med difficultyHandler:
      dynamicSpawnInterval = dynamicSpawnInterval * 0.9
      OM dynamicSpawnInterval < 50 ms:
         dynamicSpawnInterval =  50 ms

FUNKTION startRemoving():
    OM gameOver ÄR TRUE:
      RETURNERA
    UPPREPA med handler var lifespan:
      Ta bort den äldsta cirkeln
      Rita om skärmen (invalidate)


FUNKTION addRandomCircle():
   Radius = slumpmässigt tal mellan 60 till 120
   X = slumpmässig position inom bredden
   Y = slumpmässig position inom höjden
   Color = slumpmässig färg från colors
   Lägg till ny cirkel till circles

FUNKTION removeCircle():
  OM circles INTE är tom:
    Ta bort första cirkeln
    Rita om skärmen (invalidate)

FUNKTION onTouchEvent(event):
   OM event ÄR ACTION_DOWN:
     touchedX = event.x
     touchedY = event.y

   FÖR varje cirkel FRÅN slutet till början:
      Distance = avstånd mellan touch och cirkelns centrum
      OM distance <= cirkelns radius:
         Ta bort cirkeln från circles
         Rita om skärmen (invalidate)
         OM cirkeln ÄR RÖD:
           Lives = lives - 1
         ANNARS:
           Score = score + 1

         Uppdatera lives och score via lyssnare
         OM lives <= 0:
           gameOver = TRUE
           Kör game over-lyssnaren
         AVBRYT loopen
 RETURNERA TRUE
```

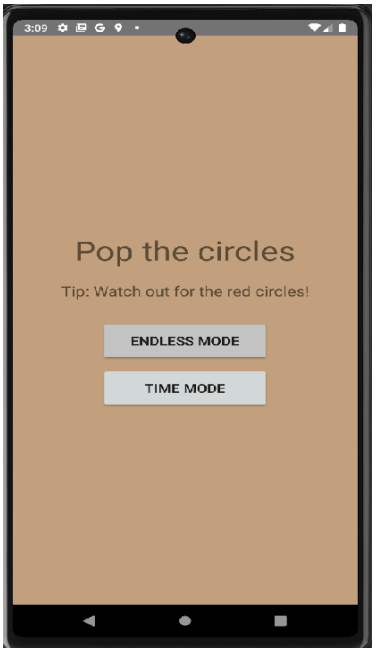
```
FUNKTION onDraw(canvas):
    OM gameOver ÄR TRUE:
      RETURNERA
    FÖR varje cirkel i circles:
      Rita cirkel på canvas
```

Sen en liten pseudo-kod för aktiviteterna:

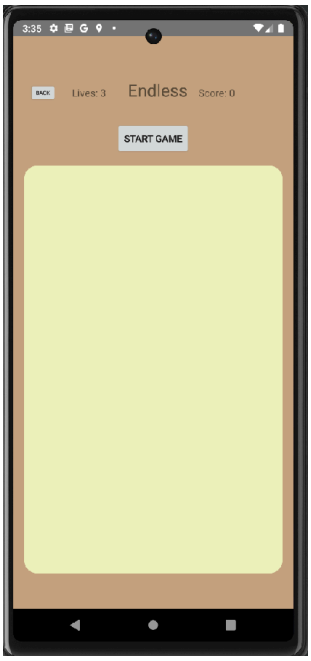
```
VID start (onCreate):
   Sätt upp layout från activity_game
   Spela bakgrundsmusik

   START-knapp:
     Kalla på startGame()

   BACK-knapp:
     Gå till första sidan

   canvasView ÄNDRINGS-LYSSNARE:
      VID poängändring -> Uppdatera scoreTitle
      VID livsändring -> Uppdatera livesTitle
      VID game over -> Visa game over-dialog
```

Funktion

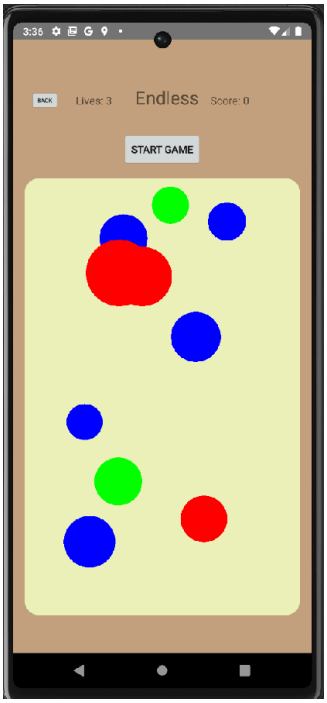
När användaren startar programmet får den upp en förstasida som är en meny:



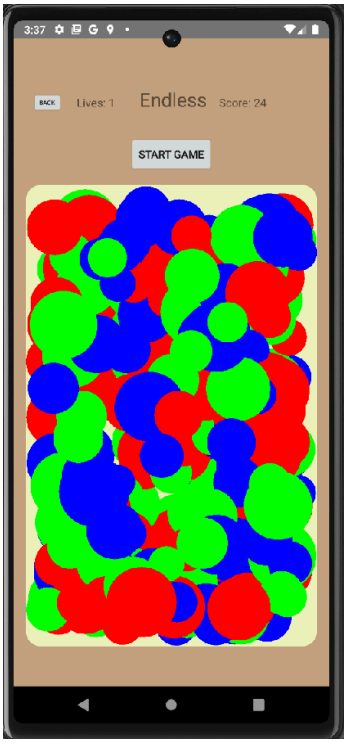
Användaren har två nivåer att välja på, Endless mode eller Time mode. På första sidan får användaren även en hint om vad spelet kan vara, både på grund av titeln på sidan men även på grund av tipset. Om användaren väljer endless mode kommer den till följande sida:



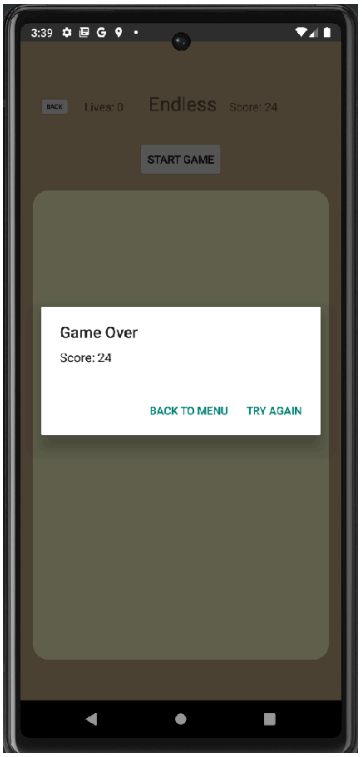
Då har användaren två alternativ att göra, antingen klicka på back och komma tillbaka till menyn eller starta spelet. När användaren startar spelet så ser det ut så här och användaren startar med 0 score och 3 liv:



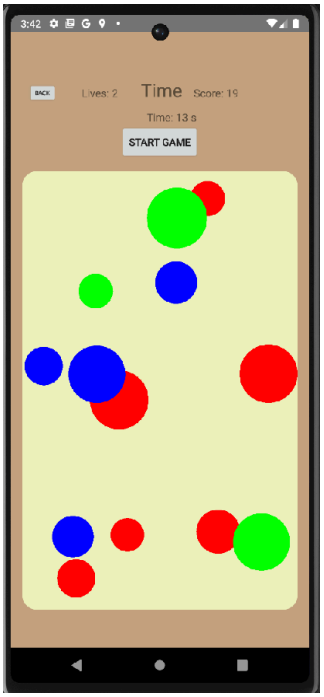
När användaren har spelat ett tag och hastigheten för cirklarna att ritas har öka kan det se ut såhär:



När användaren förlorar sina tre liv kommer det upp en game over dialogruta där användaren kan välja att spela igen eller gå tillbaka till menyn, rutan ser ut såhär:



Om användaren väljer att spela time mode istället så är själva layouten lik, den enda skillnaden är att användaren har 30 sekunder på sig att samla så mycket poäng som möjligt:



I denna nivå finns det två olika dialogrutor som kan komma fram, en för om användaren förlorar sina tre liv innan tiden är ute eller om tiden går ut: