Lua project- Corona game

Introduction

The purpose of this project was to develop a game for the Corona simulator, coded in Lua. The game was to include a ball that the user clicks to set it moving up the screen towards the goal. An obstacle should move horizontally across the screen, bouncing off the sides of the screen. The score will be displayed at the bottom of the screen. If the ball is scored, the score will go up and if the ball hits the obstacle, the score will be reset. The obstacle will change colour and increase in speed as the score goes up to a max of 5 but will reset if the ball hits it. When the user reaches a score of 5, the text "you win" will be displayed to signify that the game will be ended.

To achieve these criteria, 10 functions will be created:

- Loadbackground()
- Loadobjects()
- Loadborders()
- Counter()
- Tap()
- Reset()
- Goal()
- Miss()
- VoidBorderCollision()
- Endgame()

The game will start by initialising the physics before calling the loadbackground(), loadborders(), counter(), reset() and loadobjects() functions to set up the game. The loadobjects() function will have an event listener for when the ball is tapped to set it in motion. Various functions will be called from other functions based on the events that have been triggered throughout the game until the max score is reached.

Methodology

The first step was to initialise the physics. To do this, the game was programmed to require physics with no gravity:

```
require("physics")

physics.start()

physics.setGravity(0,0)
```

The first function to be coded was to load in the background (loadbackground()) with the dimensions 360, 570. The background was displayed in the centre of the x and y axis:

```
function loadbackground()

	local background = display.newImageRect( "background.jpg", 360, 570 )

	background.x = display.contentCenterX

	background.y = display.contentCenterY
```

```
end
```

The second and main function was to load in all the objects (loadobjects()) and set the behaviour of the objects. The two objects to be called in were the ball and the obstacle, both of which were dynamic, and had physics applied to them and a colour set. The obstacle was set to have friction of 1 and bounce of 1 so it bounces back and two off the borders. The obstacle had a fixed rotation, so it didn't spiral after collisions. A set of if statements were used to change the velocity and colour of the obstacle based on the number of goals scored:

```
function loadobjects()
	ball = display.newCircle( 160, 250, 20 )
	ball:setFillColor(255, 255, 255)
	physics.addBody(ball, "dynamic", {friction=0, bounce = 0, radius=20})
	obstacle= display.newRect(150,50,100,10)
	obstacle:setFillColor(255, 255, 255)
	physics.addBody(obstacle, "dynamic", {friction=1, bounce = 1})
	obstacle.isFixedRotation = true
	if (goalcount==0) then
		obstacle:setLinearVelocity(100,0)
	elseif (goalcount==1) then
		obstacle:setLinearVelocity(200,0)
		obstacle:setFillColor(0, 255, 0)
	elseif (goalcount==2) then
		obstacle:setLinearVelocity(300,0)
		obstacle:setFillColor(0, 0, 255)
	elseif (goalcount==3) then
		obstacle:setLinearVelocity(400,0)
		obstacle:setFillColor(255, 0, 0)
	elseif (goalcount==4) then
		obstacle:setLinearVelocity(450,0)
		obstacle:setFillColor(0, 0, 0)
	else
		reset()
	end
	ball:addEventListener("tap",tap)
	leftborder:addEventListener("collision", VoidBorderCollision)
	rightborder:addEventListener("collision", VoidBorderCollision)
	obstacle:addEventListener("collision", miss)
end
```

Within the loadobjects() function, event listeners were added for collisions and taps. The first event listener was to detect a tap on the ball. When called, this function sets an upward linear velocity to the ball:

```
function tap()
	ball:setLinearVelocity(0,-800)
end
```

Another event listener, within the loadobjects() function, was to detect the collision of the obstacle. This calls the miss() function. This function, sets the score back to zero and reloads the objects unless x=1, then it will set x back to 0:

```
function miss()
	if (x~=1) then
	goalcount=0
	display.remove(goaltext)
	goaltext=display.newText(goalcount,display.contentCenterX,350,
native.systemFont,60)
	reset()
	timer.performWithDelay(1000, loadobjects)
	else
	x=0
	end
end
```

The other function in the loadobjects() function is to detect collisions on the left or right borders. This calls the VoidBorderCollision() function which simply sets x to one so the miss() function can differentiate between the ball colliding with the obstacle and the border colliding with the obstacle. This was necessary because without it, the game would reset every time the obstacle collided with the borders:

```
function VoidBorderCollision()
	x=1
end
```

Another function was to load borders to the top, left and right of the screen. These borders were all static objects. The purpose of the left and right borders was to allow the obstacle to bounce off them, so the bounce was set to 1 for both borders. The purpose of the top border was to detect if the ball hits it to signify a goal:

```
function loadborders()
	leftborder= display.newRect(0,0,1,display.contentHeight)
	leftborder:setFillColor(255, 255, 255)
	physics.addBody(leftborder, "static", {friction=0, bounce = 1})
	rightborder= display.newRect(display.contentWidth,0,1,display.contentHeight)
	rightborder:setFillColor(255, 255, 255)
```

```
	physics.addBody(rightborder, "static", {friction=0, bounce = 1})
	topborder= display.newRect(0,-12,display.contentWidth*2,1)
	topborder:setFillColor(255, 255, 255)
	physics.addBody(topborder, "static", {friction=0, bounce = 0})
end
```

The counter() function was to initialise the goal count to 0 and display it at the bottom of the screen:

```
function counter()
	goalcount=0
	goaltext=display.newText(goalcount,display.contentCenterX,350,
native.systemFont,60)


end
```

the reset() function was to remove the ball and object from the display:

```
function reset()
	display.remove(obstacle)
	display.remove(ball)
end
```

The purpose of the goal() function was to add 1 to the goalcount after a goal and then reload the objects unless the goalcount is equal to 5. In this case, the goaltext changes to “you win” and the endgame() function is called with a delay to signify the user has won the game:

```
function goal()
	goalcount=goalcount+1
	display.remove(goaltext)
	if (goalcount==5) then
		goaltext=display.newText("you win",display.contentCenterX,350,
native.systemFont,60)
		timer.performWithDelay(1000, endgame)
	else
		goaltext=display.newText(goalcount,display.contentCenterX,350,
native.systemFont,60)
	end
	reset()
	timer.performWithDelay(1000, loadobjects)
end
```

The endgame function exits the game:

```
function endgame()
	native.requestExit()
end
```

After all these functions have been declared and the physics is initialised, the following functions are called to start up the game:

```
loadbackground()

loadborders()

counter()

reset()

loadobjects()
```

and also, a collision event listener is set for the top border to listen for “goals”. These functions and event listener start the game loop. When the game loop starts, functions are called from other functions based on what is happening in the game until the goal count reaches 5 and the game ends.

Results and conclusion

In conclusion, the game functioned how expected. The game ran smoothly with no unexpected actions happening. Once the game was completed, it was ran multiple times, testing every possible situation to ensure the game reacted how expected. The game looked professional and was fun to play. To test the game fully, it was built as a .APK application and tested on an android device. Once built, it ran exactly the same as in the simulator apart from the resolution appearing to be slightly lower and the borders being slightly visible unlike in the simulator. Below is an image of the interface of the final version of the game running in the Corona simulator: