JANUARY 2023

TEB1043: Object Oriented Programming

PROJECT: SAVE THE BUNNY ANDROID APP

BACHELOR'S DEGREE IN COMPUTER SCIENCE

NAME	STUDENT ID
ZULAIKHA SYAMIMI BINTI ROSLI	21001290
ISABELLA JOY JOSEPH	22007192
NUR ALYA SYAZA BINTI MOHD AZRIN	22005220
CHAN VI VEN	22007206
AMIERA JANNAH BINTI MOHAMMAD KHIR	21001117
AZRA RHEA BINTI RADZLAN	21001239
NUR ZAKIAH BALQIS BINTI MOHD FAUZI	21001372

Video Presentation Link: https://youtu.be/b5lu_IRS588

PROJECT DESCRIPTION

"Save the Bunny" is a mobile game that requires players to control a rabbit character, helping it avoid falling spikes while attempting to stay alive for as long as possible. While the spikes are a danger to the rabbit, carrots also fall from the sky, which provide points to the player and help the rabbit progress to the next level. The player's primary objective is to keep the rabbit character alive while collecting as many carrots as possible to gain points and advance through the game's levels.

ARRANGED TASKS

NAME	UML AND DESIGN
ZULAIKHA SYAMIMI BINTI ROSLI	Carrot
ISABELLA JOY JOSEPH	DatabaseHelper
NUR ALYA SYAZA BINTI MOHD AZRIN	GameView
CHAN VI VEN	SignupActivity & LoginActivity & Explosion
AMIERA JANNAH BINTI MOHAMMAD KHIR	MainActivity & Level
AZRA RHEA BINTI RADZLAN	GameOver
NUR ZAKIAH BALQIS BINTI MOHD FAUZI	Spike

UML FUNCTIONS

MainActivity

The MainActivity consists of two methods, onCreate() and startGame(). The onCreate() method is in charge of configuring the app's layout and initializing the user interface. It inflates the XML layout and sets the window flag to keep the screen on.

The startGame() method is in charge of handling user input, specifically when a button in the layout is clicked. It makes a new instance of the GameView class and assigns it the layout. When the user presses the "Start Game" button in the activity_main.xml layout, this function is invoked.

SignupActivity

The SignupActivity class is used when new users register for a new account at the start of the game. The class will have a function that will ask for input from the user for 3 variables - email, password and confirm password. It also helps to further validate the user's information (e.g. prevent reuse of email). Suitable messages will be displayed to indicate whether the account set is a success or a failure.

LoginActivity

The LoginActivity class is used for pre-existing users when they enter the game. It contains a function that will verify the user's information by comparing the user input with the existing data from our database. If it matches, the program will let the user continue to play the game, otherwise the program will display a suitable message to the user, letting them know that their login has failed.

DatabaseHelper

This is the DatabaseHelper class, which is used to manage a SQLite database. 'DatabaseHelper' class: This is the main class being represented. It has several methods and an attribute which are 'databaseName': This attribute is a private string that holds the name of the database being managed. Besides, 'DatabaseHelper'(context: Context): This is the constructor for the class. It takes a Context object as input and creates a new instance of the 'DatabaseHelper' class. Next, 'onCreate(MyDatabase: SQLiteDatabase): void': This method is called when the database is first created. It takes a 'SQLiteDatabase' object as input and creates a new table within the database. 'onUpgrade(MyDB: SQLiteDatabase, i: Int, i1: Int): void': This method is called when the database needs to be upgraded to a new version. It takes a 'SQLiteDatabase' object as input and deletes the existing table(s) before creating a new one. Furthermore, 'insertData(email: String, password: String): Boolean': This method is used to insert new data into the database. It takes an email and a password as input and returns a boolean value indicating whether the insertion was successful. Then, 'checkEmail(email: String): Boolean': This method is used to check if a given email exists in the database. It takes an email as input and returns a boolean value indicating whether the email exists in the database. Lastly, 'checkEmailPassword(email: String, password: String): Boolean': This method is used to check if a given email and password combination exists in the database. It takes an email and a password as input and returns a boolean value indicating whether the email and password combination exists in the database. So, this UML diagram shows the class structure of the 'DatabaseHelper' class, which is used to manage a SQLite database. It has several methods for creating, upgrading, inserting, and checking data in the database.

GameView

The GameView class is the main class and represents the game view. The ‘GameView’ class overrides the ‘onDraw()’ method to draw the game graphics on a canvas object, which is provided as an argument. The GameView class has several fields, such as ‘background’, ‘ground’, and ‘rabbit’, which are ‘Bitmap’ objects that represent the images to be displayed in the game. The ‘rectBackground’ and ‘rectGround’ fields are ‘Rect’ objects that define the position and size of the background and ground images. The ‘handler’ field is a ‘Handler’ object that is used to schedule tasks to be run on the UI thread. The ‘runnable’ field is a ‘Runnable’ object that is used to update the game graphics on each frame. The ‘textPaint’ and ‘healthPaint’ fields are ‘Paint’ objects that are used to render text and the player's health bar on the screen. The ‘points’ and ‘life’ fields keep track of the player's score and remaining lives, respectively. The ‘dWidth’ and ‘dHeight’ fields represent the width and height of the game's display, which are obtained from the device screen size. The ‘random’ field is a ‘Random’ object that is used to generate random numbers for certain game mechanics. The ‘rabbitX’ and ‘rabbitY’ fields represent the position of the player character. The ‘spikes’, ‘carrots’, and ‘explosions’ fields are ‘ArrayList’ objects that represent the game's obstacles and other objects. The ‘levels’ field is an array of ‘Level’ objects that represent the game's levels. The ‘currentLevelIndex’ and ‘currentLevel’ fields keep track of the current level in the game. The ‘generateCarrots()’ method is a private helper method that is used to create a new ‘Carrot’ object and add it to the ‘carrots’ ArrayList. The ‘onDraw()’ method first draws the background and ground images on the canvas using the ‘drawBitmap()’ method. It then draws the rabbit image on the canvas at the position specified by the ‘rabbitX’ and ‘rabbitY’ fields. The method also draws all the carrot images on the canvas. The ‘carrotFrame’ field of each ‘Carrot’ object is incremented, which causes the image to animate. Finally, the method also draws the player's score, remaining lives, and current level on the screen using the ‘drawText()’ method of the ‘Canvas’ object.

Carrot

The class imports Context class, Bitmap class and Bitmap Factory. Within the constructor, the code initializes the carrot bitmap array by loading two bitmap images from the app resources using the BitmapFactory.decodeResource() method. The resetPosition() method is called to set the position of the Carrot object at random position. The getCarrot() method returns a Bitmap object from the carrot array at the index specified by the CarrotFrame parameter. The getCarrotWidth and getCarrotHeight returns the width and the height of the first Bitmap object in the carrot array. The method generates random x and y coordinates for the Carrot object using the random.nextInt() method. The method also generates a random CarrotVelocity value between 35 and 50, which will be used to determine how quickly the Carrot object will move down the screen.

Spike

The class imports Context class, Bitmap class and Bitmap Factory. Within the constructor, the code initializes the spike bitmap array by loading three bitmap images from the app resources using the BitmapFactory.decodeResource() method. The resetPosition() method is called to set the initial position of the Spike object. The getSpike() method returns a Bitmap object from the spike array at the index specified by the spikeFrame parameter. The getSpikeWidth and getSpikeHeight returns the width and the height of the first Bitmap object in the spike array. The method generates random x and y coordinates for the Spike object using the random.nextInt() method. The method also generates a random spikeVelocity value between 35 and 50, which will be used to determine how quickly the Spike object will move down the screen.

Explosion

In the game, there are different styles for the explosion effect. We use the Explosion class to initialize and store the different explosion styles, each in its own position in the array. So that when the game runs, an array that is declared in this function will be able to return a suitable explosion style when objects explode.

Level

The Level class has two private variables, levelNumber and requiredScore, which are used to represent the level number and the score required to reach that level, respectively. The class has a public constructor that takes the level number and required score as arguments and initializes the corresponding instance variables.

There are also three public methods in the class: getLevelNumber(), getRequiredScore(), and getCurrentLevel(playerScore: int). The getLevelNumber() method returns the level number, getRequiredScore() returns the required score, and getCurrentLevel(playerScore: int) is a static method that takes a player's score as an argument and returns the current level that the player has reached based on the required score for each level.

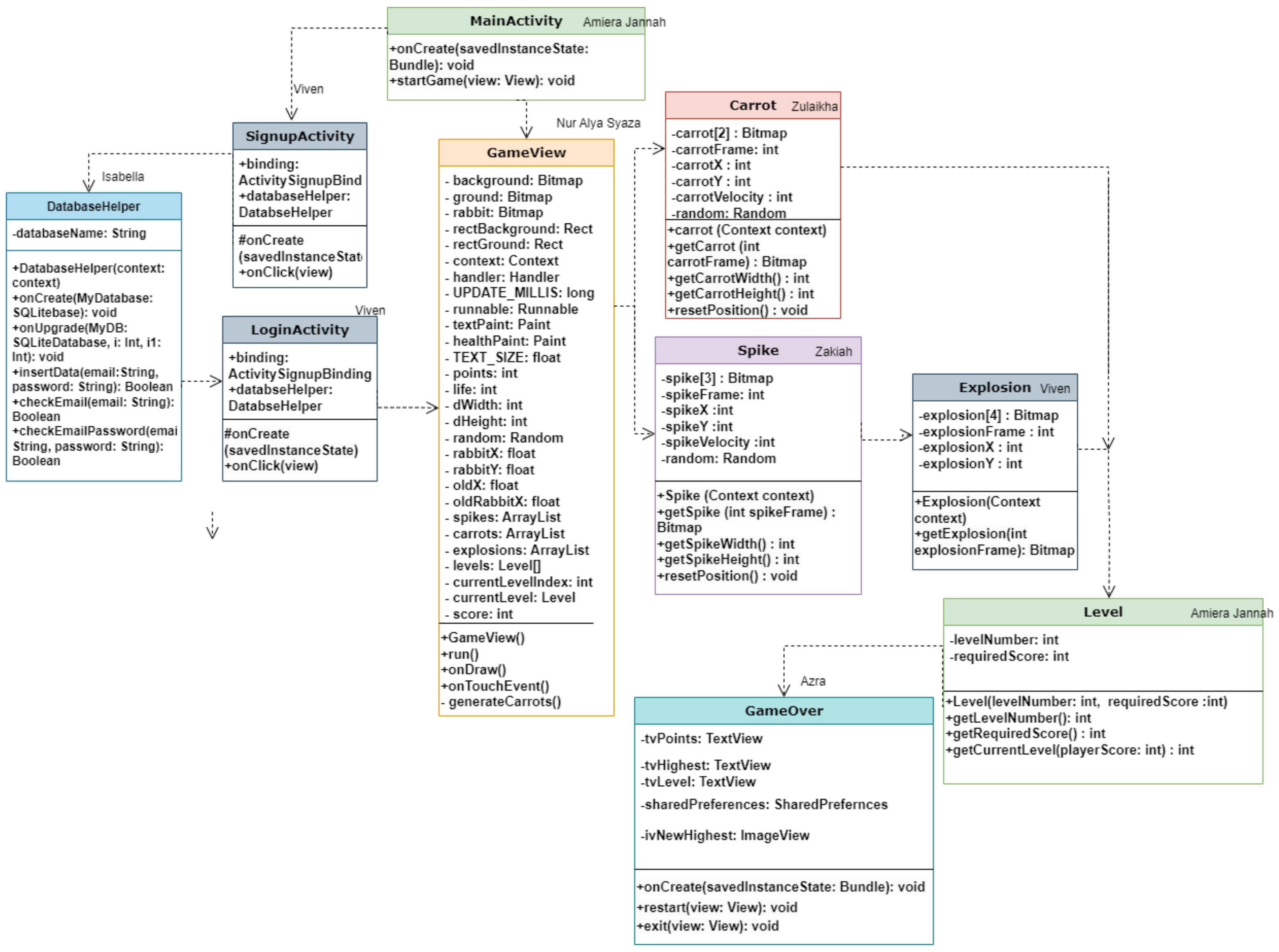
GameOver

The class contains five private fields: tvPoints, tvHighest, tvLevel, sharedPreferences, and ivNewHighest. These fields are represented as attributes of the class and are denoted by the **-** symbol, indicating that they are private. The GameOver class also has three public methods: onCreate, restart, and exit. These methods are represented as operations of the class and are denoted by the **+** symbol, indicating that they are public. The onCreate method takes a savedInstanceState parameter of type Bundle and is responsible for initializing the UI elements of the game over screen. The restart method takes a view parameter of type View and is responsible for restarting the game by creating a new instance of the MainActivity class. The exit method takes a view parameter of type View and is responsible for exiting the app.The GameOver class extends the AppCompatActivity class, which is represented as a box at the top of the diagram. The AppCompatActivity class is a part of the Android Framework and provides basic functionalities for implementing Android activities. Overall, the UML class diagram provides a high-level overview of the GameOver class and its relationships to other classes in the

application.

Video Presentation Link: https://youtu.be/b5lu_IRS588

UML DIAGRAM


MainActivity
Amiera Jannah
+onCreate(savedInstanceState: Bundle): void
+startGame(view: View): void
Viven
SignupActivity
+binding: ActivitySignupBind
+databaseHelper: DatabseHelper
#onCreate (savedInstanceStat
+onClick(view)
Isabella
DatabaseHelper
-databaseName: String
+DatabaseHelper(context: context)
+onCreate(MyDatabase: SQLitebase): void
+onUpgrade(MyDB: SQLiteDatabase, i: Int, i1: Int): void
+insertData(email:String, password: String): Boolean
+checkEmail(email: String): Boolean
+checkEmailPassword(emai String, password: String): Boolean
Viven
LoginActivity
+binding: ActivitySignupBinding
+databseHelper: DatabseHelper
#onCreate (savedInstanceState)
+onClick(view)
Nur Alya Syaza
GameView
- background: Bitmap
- ground: Bitmap
- rabbit: Bitmap
- rectBackground: Rect
- rectGround: Rect
- context: Context
- handler: Handler
- UPDATE_MILLIS: long
- runnable: Runnable
- textPaint: Paint
- healthPaint: Paint
- TEXT_SIZE: float
- points: int
- life: int
- dWidth: int
- dHeight: int
- random: Random
- rabbitX: float
- rabbitY: float
- oldX: float
- oldRabbitX: float
- spikes: ArrayList
- carrots: ArrayList
- explosions: ArrayList
- levels: Level[]
- currentLevelIndex: int
- currentLevel: Level
- score: int
+GameView()
+run()
+onDraw()
+onTouchEvent()
- generateCarrots()
Carrot
Zulaikha
-carrot[2] : Bitmap
-carrotFrame: int
-carrotX : int
-carrotY : int
-carrotVelocity : int
-random: Random
+carrot (Context context)
+getCarrot (int carrotFrame) : Bitmap
+getCarrotWidth() : int
+getCarrotHeight() : int
+resetPosition() : void
Spike
Zakiah
-spike[3] : Bitmap
-spikeFrame: int
-spikeX :int
-spikeY :int
-spikeVelocity :int
-random: Random
+Spike (Context context)
+getSpike (int spikeFrame) : Bitmap
+getSpikeWidth() : int
+getSpikeHeight() : int
+resetPosition() : void
Explosion
Viven
-explosion[4] : Bitmap
-explosionFrame : int
-explosionX : int
-explosionY : int
+Explosion(Context context)
+getExplosion(int explosionFrame): Bitmap
Level
Amiera Jannah
-levelNumber: int
-requiredScore: int
+Level(levelNumber: int, requiredScore :int)
+getLevelNumber(): int
+getRequiredScore() : int
+getCurrentLevel(playerScore: int) : int
Azra
GameOver
-tvPoints: TextView
-tvHighest: TextView
-tvLevel: TextView
-sharedPreferences: SharedPrefernces
-ivNewHighest: ImageView
+onCreate(savedInstanceState: Bundle): void
+restart(view: View): void
+exit(view: View): void