DOCUMENTATION GAMES

TREE GAMES

INDEX

START CLASSES

SPLASHACTIVITY

This class is just the activity that loads the start animation, is the first class loaded when the app is used. When the animation is over, the next Activity is charged.

MENU

This class is the main menu from the app, from this activity you can travel between the two different games and the ranking.

2048

GameCell

This class represents a single cell in the game grid of the app, and it has a single integer value that can be set and retrieved.
When the Cell is created it always has a 2 as its value.

GameCellView

The GameCellView class is a custom View used to display a GameCell object. It provides methods to set and update the GameCell object's value.

The GameCellView class has three constructors, which allow the view to be created with or without attributes and styles. The setCell() method allows a GameCell object to be set as the view's cell and calls updateValue() to update the value displayed on the view.

MainActivity2048

This is the main class for the 2048 game, this class has the following attributes:

- mCells: a matrix of gamecells, this is the board of the game.
- mCellView: a matrix of gamecellview, this is the visual representation of the board.
- mGestureDetector: this is the gesturedetector of the class, is used to know if the user has slipped in any direction.
- db: this is the Database of the application, is used to keep the scores.
- score: this is the score of the user.

METHODS

onCreate:

Inside the method, the layout for the activity is set using setContentView() method. A new instance of the DataBase class is created.

Then, a nested for loop is used to fill the mCellViews matrix, the invalidate() method is called for each cell view to force it to be redrawn.

Two random cells are added to the game board using the addRandomCell() method.

The updateCellsBackground() method is called to set the background color of each cell according to its value.

Finally, a GestureDetector is created to handle swipe gestures on the game board. The activity implements the GestureDetector.OnGestureListener interface to receive callbacks from the detector when a gesture is detected.

addRandomCell:

This method is responsible for adding a new random cell to the game board if there is at least one empty cell available.

The method first checks if there are any empty cells available on the game board. If there are, then it generates random row and column indices until it finds an empty cell.

Once an empty cell is found, a new GameCell object is created and assigned to the corresponding cell in the 2D GameCell array. Then, a corresponding GameCellView object is created and added to the GridLayout view of the game board with appropriate layout parameters.

Finally, the method updates the background color of all cells on the game board to reflect the current value of the cells.

updateCellsBackground:

This method updates the background of each cell in the game board according to its value. It first retrieves the GridLayout view where the cells are located and then iterates over each cell to assign a background image based on its value.

For each cell, it retrieves its corresponding GameCellView object, as well as its associated GameCell object, which contains the value of the cell. Based on the value of the cell, the method selects a specific background image to display using a switch statement.

If the cell is empty (i.e., its GameCell object is null), it sets the default background image for an empty cell. Finally, the method sets the background image to the GameCellView object and updates the score, as well as checks if the game has been won or lost.

Overall, this method is responsible for updating the visual appearance of the game board, based on the current state of the game.

MoveUp,MoveDown,MoveRight,MoveLeft:

This methods represents the moves action of the 2048 game, which is when all tiles on the board slide to the used direction of the board.

The method first initializes a boolean matrix merged to keep track of cells that have already been merged during the current move. Then, the method iterates over each column of the game board, and for each non-null cell in the column, it moves the cell as far up as possible by repeatedly swapping it with the cell on that direction until there are no more empty cells or cells with the same value.

If a cell is merged with another cell during this process, the method updates the score with the sum of the merged cells' values and marks the merged cell as already merged in the merged matrix to prevent it from being merged again during this move.

After all cells have been moved as far as possible, a new random cell is added to the board and the board is updated with the new cell values and background images using the updateCellsBackground() method. Finally, the method updates the cell views, score, and returns the score earned during this move.

UpdateCellView:

The updateCellView method is responsible for updating the view of the game cells. It iterates through the 2D array of GameCellView objects, and for each cell, it sets the corresponding GameCell object from the mCells array and then invalidates the view. Invalidating a view forces it to redraw itself with the updated data.

Essentially, this method ensures that the views representing the cells on the game board are up-to-date with the current state of the mCells array. This ensures that the user interface displays the current game state correctly to the player.

Loose:

This method checks whether the game has ended in a loss by first checking if there are any empty cells left on the board. If there are no empty cells left, then it checks if there are any adjacent cells that have the same value. If there are no adjacent cells with the same value, then the player has lost the game.

If the player is lost, an alert dialog box is shown with a title "You loose!" and a message asking the player to input their name to save their score. The method then retrieves the name entered by the player, creates a ContentValues object to store the name and score, and inserts the object into the "Score2048" table in the database using the insert method. Finally, the method finishes the current activity.

The method returns a boolean value of true if the player has lost the game, and false otherwise.

Win:
The win() method checks if the player has won the game by reaching the tile with a value of 2048. If such a tile exists, an alert dialog is displayed congratulating the player and asking them to enter their name to save their score in the database. If the player has not won yet, the method returns false.

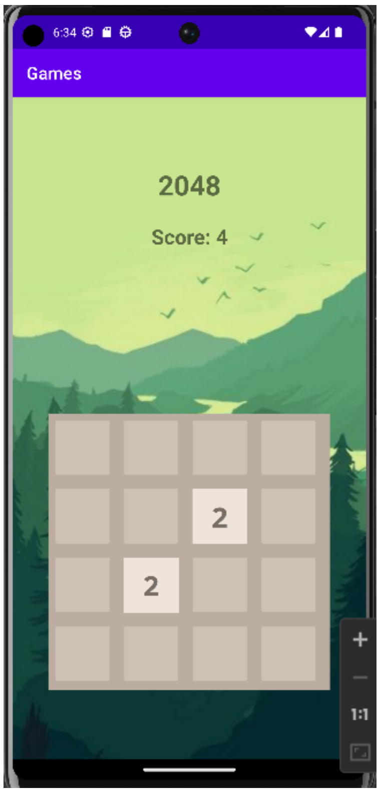
The method first initializes a boolean variable isWin to false. It then iterates through all the cells in the game grid using nested for loops. If the current cell is not null and its value is equal to 2048, the isWin variable is set to true. Then an alert dialog is displayed with the message "Congratulations! You win!" and an EditText field for the player to enter their name. When the player enters their name and clicks the OK button, a new row is inserted into the Score2048 table of the database with the player's name and score. Finally, the finish() method is called to close the activity.

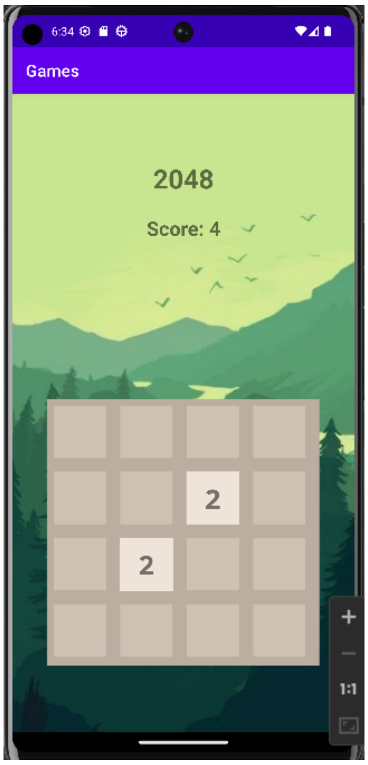
onFling:
This method is used to handle swipe gestures made by the user on the game board. It takes four arguments, two MotionEvent objects that represent the starting and ending points of the swipe, and two floats that represent the velocity of the swipe in the X and Y directions.

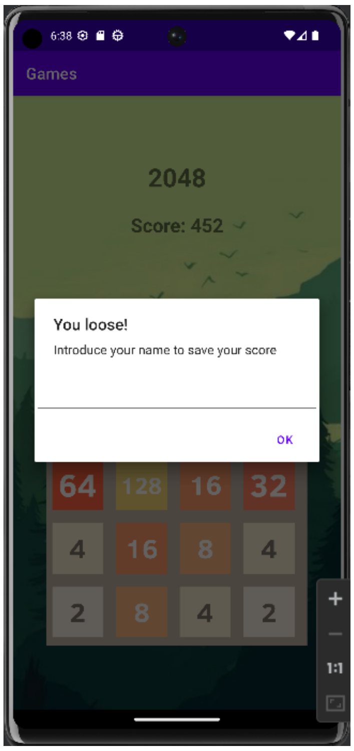
The method first calculates the difference in X and Y coordinates between the two MotionEvent objects. It then checks if the difference in X is greater than the difference in Y. If it is, the method checks if the swipe was to the right or to the left and calls the appropriate method to move the tiles on the game board in that direction . If the difference in Y is greater than the difference in X, the method checks if the swipe was up or down and again calls the appropriate method to move the tiles in that direction.

Finally, the method returns a boolean value indicating whether the event was handled or not.

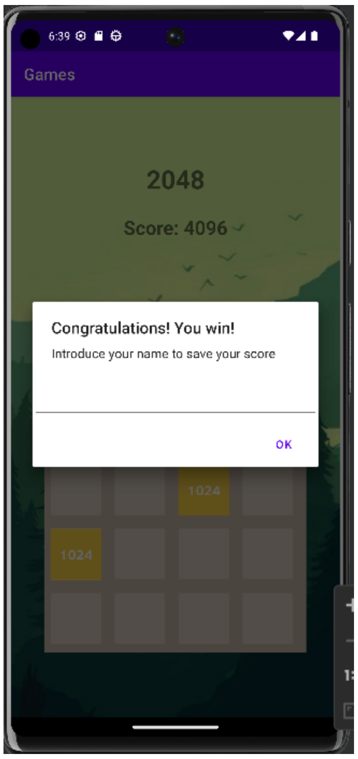
Appearance:






You loose!

Introduce your name to save your score


Congratulations! You win!

Introduce your name to save your score

LIGHTSOUT

GameCellLightsOut

This class represents a single cell in a Lights Out game. The GameCellLightsOut class has three boolean fields isOn, isClicked and ishelp. The isOn field indicates whether the cell is currently on or off. The isClicked field indicates whether the cell has been clicked or not. The ishelp field is used to indicate whether the cell is currently being used as a helper cell or not.

The GameCellLightsOut class has a constructor that sets the initial values of isOn, isClicked and ishelp to false.

The class also has a method named light() that toggles the value of isOn field. The isOn() method returns the value of the isOn field. The setOn() method sets the value of the isOn field.

The isClicked() method returns the value of the isClicked field. The click() method toggles the value of the isClicked field. The setClicked() method sets the value of the isClicked field.

The ishelp() method returns the value of the ishelp field. The sethelp() method sets the value of the ishelp field.

BoardLightsOut

The Board Lights Out class represents a board of cells for a Lights Out game. Each cell on the board can be on or off, and the objective of the game is to turn all cells off. The class contains a 2D array of GameCellLightsOut objects to represent the cells on the board.

The class has several methods to interact with the board:

BoardLightsOut(int h, int w): constructor that creates an empty board of height h and width w.

BoardLightsOut(BoardLightsOut copy): constructor that creates a copy of the board.

createEmptyBoard(int h, int w): creates a new empty board of height h and width w.

click(int h, int w): performs a click on the cell at position (h, w) and updates the state of adjacent cells.

randomize(): randomly sets cells on the board to be turned on.

win(): returns true if all cells on the board are turned off.

help(BoardLightsOut copy): if helps is false, marks cells that have a different state than the corresponding cells in copy as "helpful" to the user. If helps is true, clears all helpful markings.

flush(): clears all clicked cells on the board.

Overall, the BoardLightsOut class provides the functionality to create, manipulate, and play the Lights Out game on a board of cells.

ControllerLightsOut:

This class is used to control the game logic and interface of the Lights Out.

Board: An object of the BoardLightsOut class, which represents the game board of the Lights Out game.

CopyBoard: Another object of the BoardLightsOut class, which is used to store a copy of the game board, so that it can be used for the "help" and "retry" functions.

Buttons: A two-dimensional array of ImageButton objects, which represent the buttons on the game board.

ControllerLightsOut: A constructor method for the ControllerLightsOut class that takes a two-dimensional array of ImageButton objects as input. This method initializes the board and copyBoard objects, randomizes the board, and updates the view of the game board.

updateView: A method that updates the view of the game board based on the state of the board object.

click: A method that simulates a click on the game board at position (i, j). This method updates the board object and updates the view of the game board.

r**etryBoard**: A method that resets the board object to its original state (i.e., the state when the game started). This method also updates the view of the game board.

help: A method that sets the board object to a state where it has the minimum number of "on" buttons needed to win the game. This method also updates the view of the game board.

win: A method that returns true if the current state of the board object represents a winning state (i.e., all buttons are turned off), and false otherwise.

MainActivityLightsOut

This is the main class for the LightsOut game, this class has the following attributes:

- buttons: A matrix of ImageButtons, each position is a button of the game board
- controller: An object of the ControllerLightsOut class.
- textimer: The representation of the timer used to keep the score
- timerTask: The timer itself.
- db: The database of the application.

METHODS

onCreate:

This method initializes the UI elements and sets up the ControllerLightsOut object to manage the Lights Out game logic.

The method begins by calling the super.onCreate(savedInstanceState) method to initialize the activity's superclass. Then, it sets the activity's layout to the activity_main_lights_out layout file using the setContentView() method.

Next, the method initializes the DataBase object using the current activity context. Then, it finds the TextView element with the ID timer and assigns it to the textTimer variable.

The method then creates a 2D array of ImageButton objects with dimensions 5x5 and initializes each element with the findViewById() method, using the IDs of the ImageButton views in the layout file.

Finally, the method creates a new ControllerLightsOut object, passing the 2D array of ImageButton objects to its constructor. It then calls the startTimer() method to start the game timer.

onClick:

This is an onClick() method that handles the click events for several image buttons and three other buttons in a game called Lights Out.

The method first identifies which button was clicked using a switch statement and then calls the appropriate method in the controller object to perform the action associated with that button. The controller object is responsible for updating the game board and the view.

If the player wins the game by turning off all the lights, an AlertDialog is shown that prompts the player to enter their name. The player's name and the time taken to win the game are then inserted into a SQLite database table called ScoreLightsOut using ContentValues.

The method also resets and starts the timer if the player starts a new game or retries the current game board by clicking the New and Retry buttons, respectively. Finally, if the player clicks the Help button, the controller object displays a message to help the player understand how to play the game.

Start & Reset Timer:

This is a method that starts a timer to count the time elapsed in seconds. It uses the TimerTask and Timer classes provided by Java to perform a task repeatedly at a fixed interval.

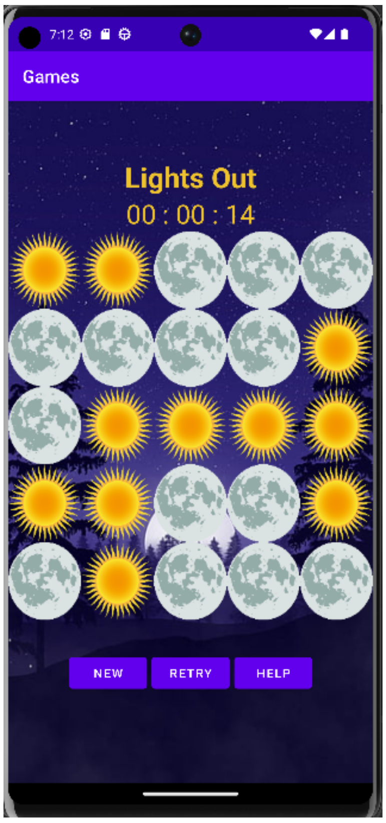
The TimerTask is an abstract class that represents a task that can be scheduled for one-time or repeated execution by a Timer. In this case, a new TimerTask is created and overridden the run() method to update the time and the text displayed in a TextView with the elapsed time.

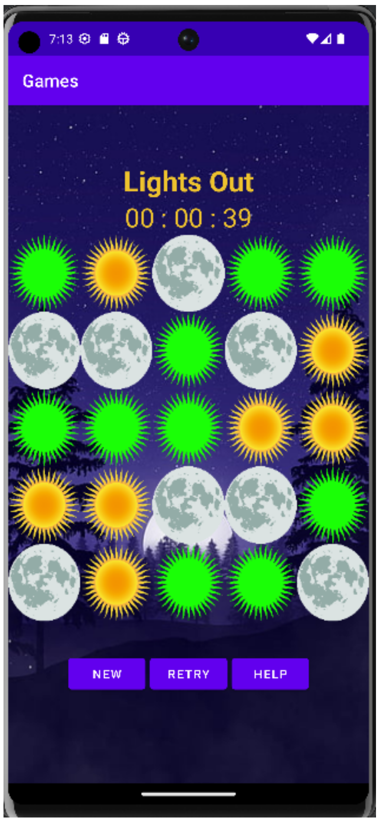
The runOnUiThread() method is called to ensure that the timer is updated on the UI thread, which is necessary when performing UI updates from a background thread.

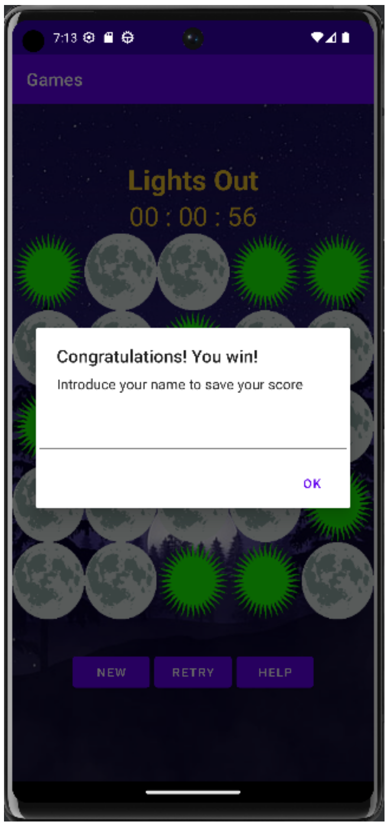
Finally, the timer is scheduled to run every second using the scheduleAtFixedRate() method of the Timer class. The method takes in the TimerTask object created and two parameters: delay and period. The delay parameter specifies the amount of time before the task is to be executed in milliseconds, while the period parameter specifies the interval between executions of the task in milliseconds.

The resetTimer reset all the parameters of the Timer.

Appearance:







Congratulations! You win!

Introduce your name to save your score

DataBase

DataBase

This Class is the Database of the application, it has 2 tables, one for the 2048 Scores and another one for the LightsOut Scores.

A parte from the creation it has 3 methods:

- **getAllScores:** this method takes all the scores from a table and return a list with all of them.

- **count:** Counts the number of scores on the table

- **query:** gets an specific score from the database

Score:

This class are the scores that are displayed on the RecyclerView, it has the name of the user and his score.

ScoreActivity:

This class is an activity that displays two RecyclerViews containing scores of players for two different games. It extends the AppCompatActivity class and overrides the onCreate() method to set up the layout and the RecyclerViews.

In the onCreate() method, the activity creates a new instance of a database helper class called DataBase. It then retrieves all scores from the database for the game "Score2048" and "ScoreLightsOut" using the getAllScores() method, and creates two instances of the ScoreAdapter class with these scores and the context of the activity.

Next, it sets up the two RecyclerViews by finding them by their respective IDs and setting a LinearLayoutManager and the appropriate ScoreAdapter to each of them using the setAdapter() method.

Finally, the onDestroy() method is overridden to close the database helper when the activity is destroyed.

In summary, this activity retrieves and displays scores for two different games using RecyclerViews and ScoreAdapters. It also manages the database helper to ensure that it is closed when the activity is destroyed.

ScoreAdapter:

This class is an adapter for a RecyclerView that displays scores of players in a game. It extends the RecyclerView.Adapter class and contains an inner class called ScoreViewHolder that extends RecyclerView.ViewHolder.

The ScoreViewHolder is responsible for holding references to the TextViews that display the name and score of a player in the RecyclerView. The onCreateViewHolder() method is called by the RecyclerView when it needs a new ViewHolder, and it inflates the layout for each player score.

The onBindViewHolder() method is called by the RecyclerView to bind the data to the ViewHolder. It gets the score at the specified position from a database using the query() method and sets the name and score TextViews in the ViewHolder with the data.

The getItemCount() method returns the number of scores in the database for the specified game. The ScoreAdapter constructor takes in a Context, a DataBase object, and the name of the game to display scores for.

Appearance: