MEMORY GAME PROJECT

IN JAVA

CONTENTS

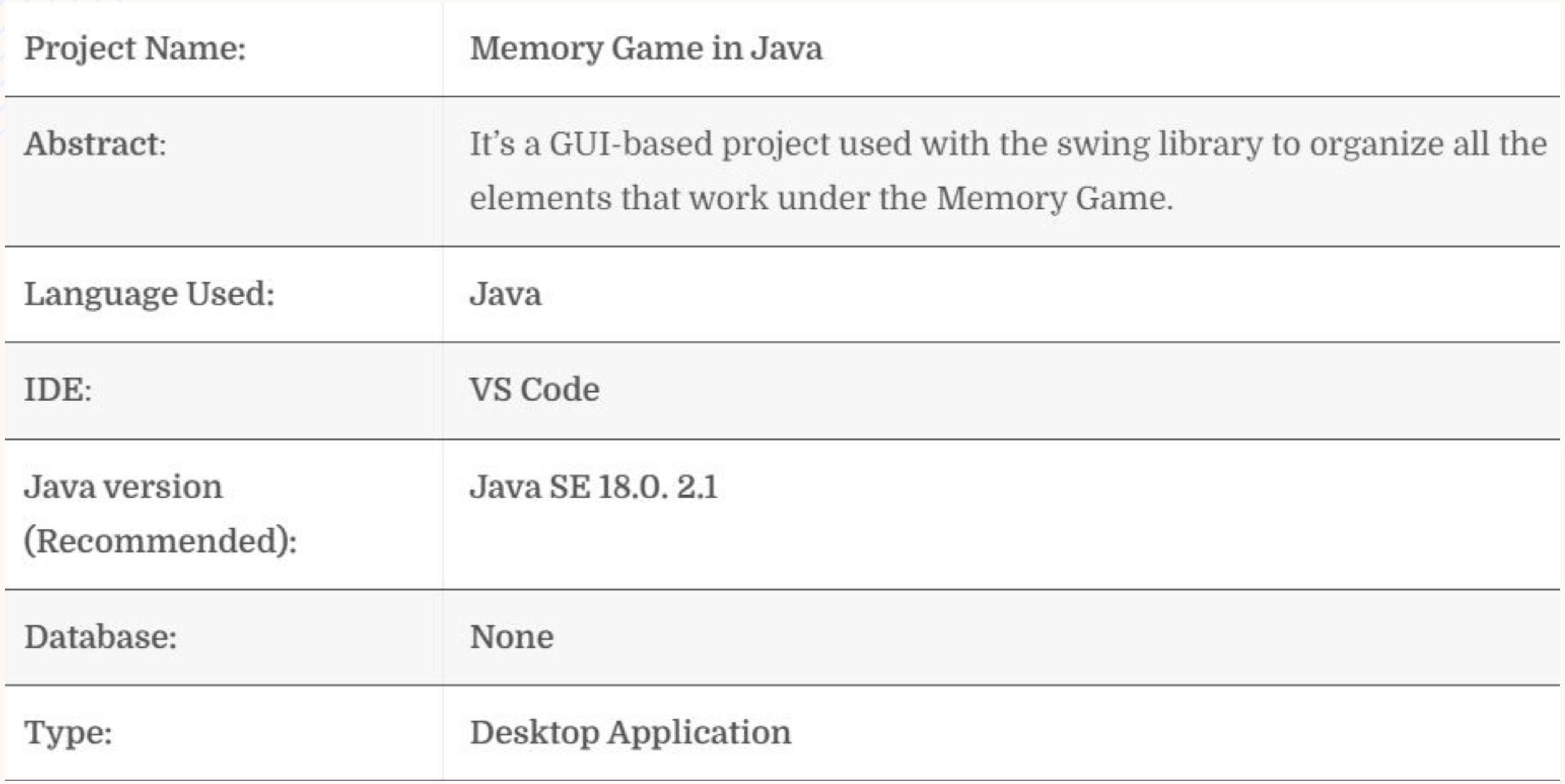
INTRODUCTION

Playing memory games can improve attention, and concentration and enhance critical thinking.

The memory game is going to be made in Java Language.

This game requires the player to remember the position of different images and try to match two similar images within a limited number of attempts.

PROJECT DETAILS

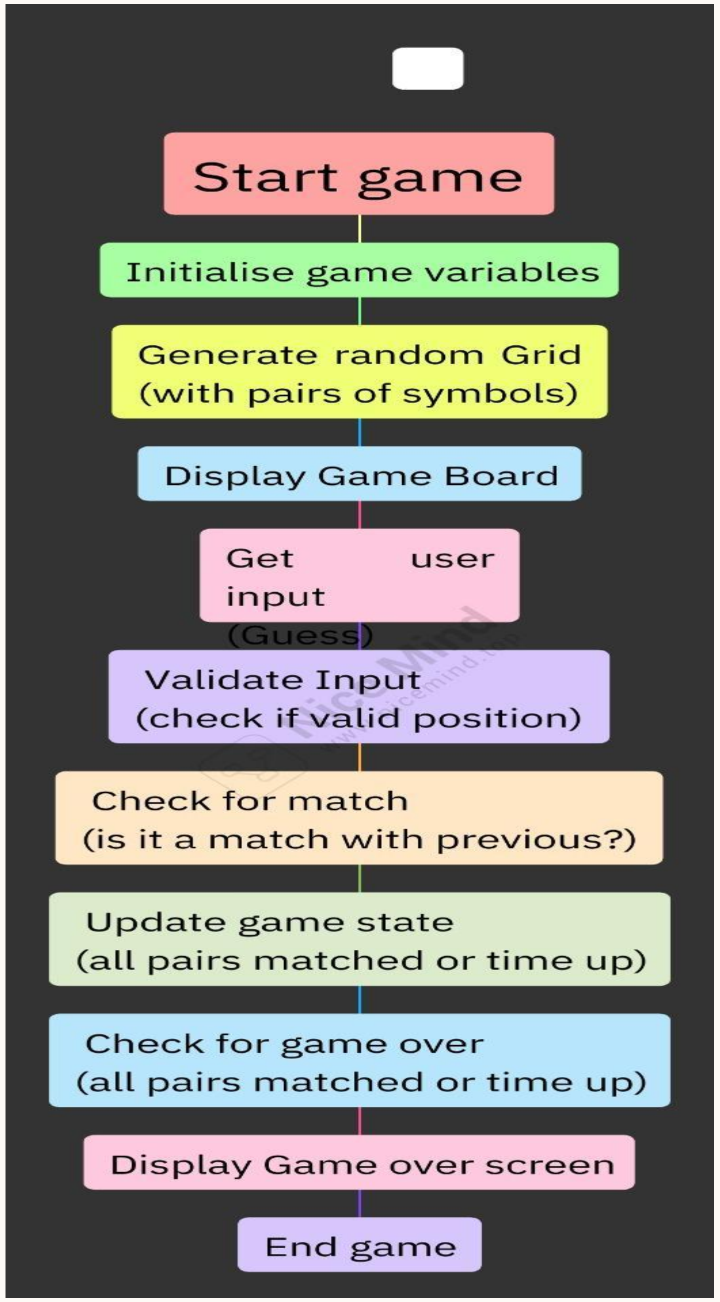


Project Name:	Memory Game in Java
Abstract:	It's a GUI-based project used with the swing library to organize all the elements that work under the Memory Game.
Language Used:	Java
IDE:	VS Code
Java version (Recommended):	Java SE 18.0. 2.1
Database:	None
Type:	Desktop Application

FEATURES OF THE GAME

- The player will be able to flip the pieces and check the image on it
- If two images do not match then the number of tries left decreases
- A total of 10 tries are given initially to increase the difficulty of the game
- 4*4 images are shown and the player needs to match all the images correctly to win
- For a fresh start, the images are picked and placed randomly in the window

FLOW CHART


Start game
Initialise game variables
Generate random Grid (with pairs of symbols)
Display Game Board
Get user input (Guess)
Validate Input (check if valid position)
Check for match (is it a match with previous?)
Update game state (all pairs matched or time up)
Check for game over (all pairs matched or time up)
Display Game over screen
End game

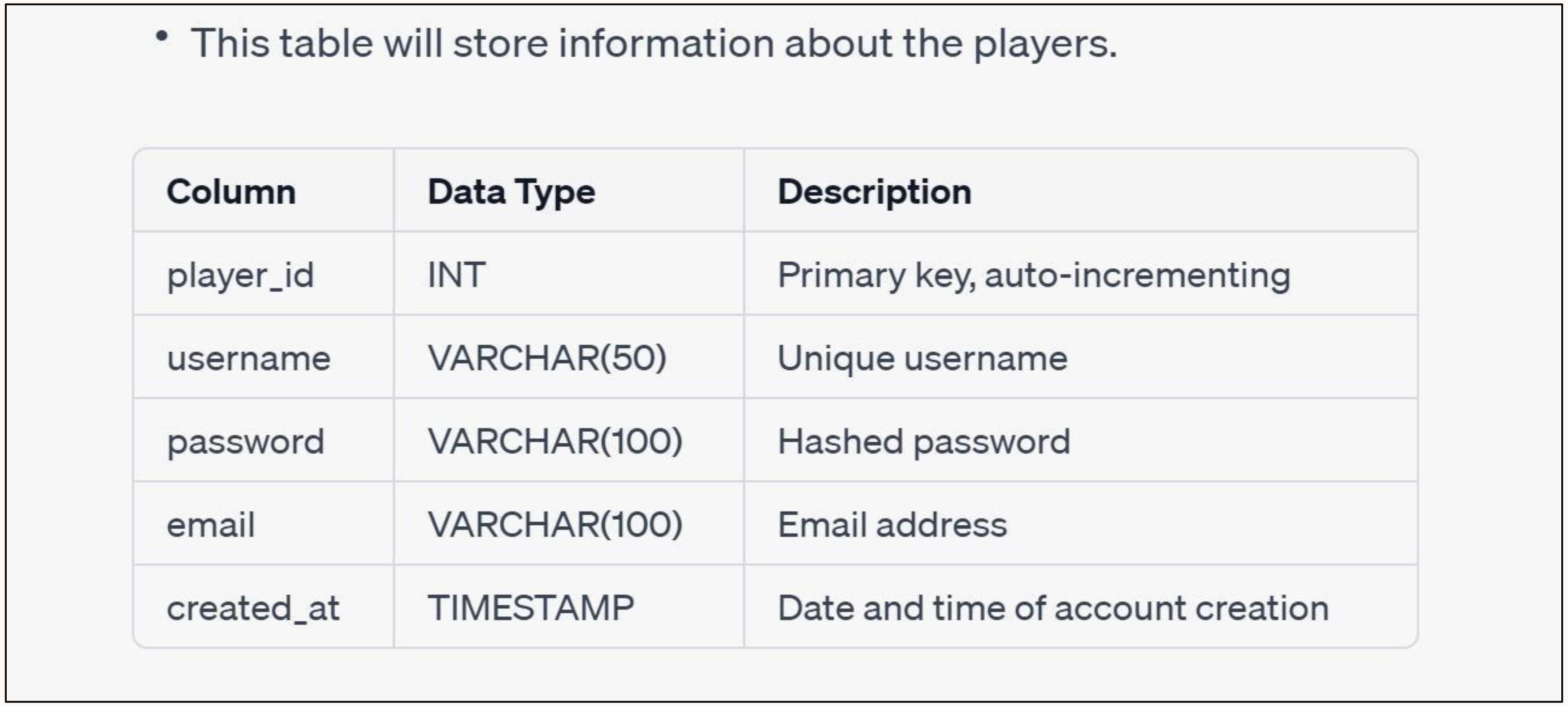
Step for the game:

1. Start Game: This is where the game begins.
2. Initialize Game Variables: Set up any necessary variables like the grid size, symbols, score, timer, etc.
3. Generate Random Grid: Create a grid with pairs of symbols randomly arranged.
4. Display Game Board: Show the grid to the player.
5. Get User Input (Guess): Wait for the player to make a move (select a pair of tiles).
6. Validate Input: Check if the user's input is valid (i.e., within the grid bounds and not already matched).
7. Check for Match: Determine if the selected pair of tiles is a match.
8. Update Game State: Adjust variables based on the outcome (e.g., flip tiles, update score).
9. Check for Game Over: Determine if the game is over (either all pairs matched or time is up).
10. Display Game Over Screen: Show the final score or message indicating the game is over.
11. End Game: Terminate the game.

DATABASE DESIGN

Tabular data

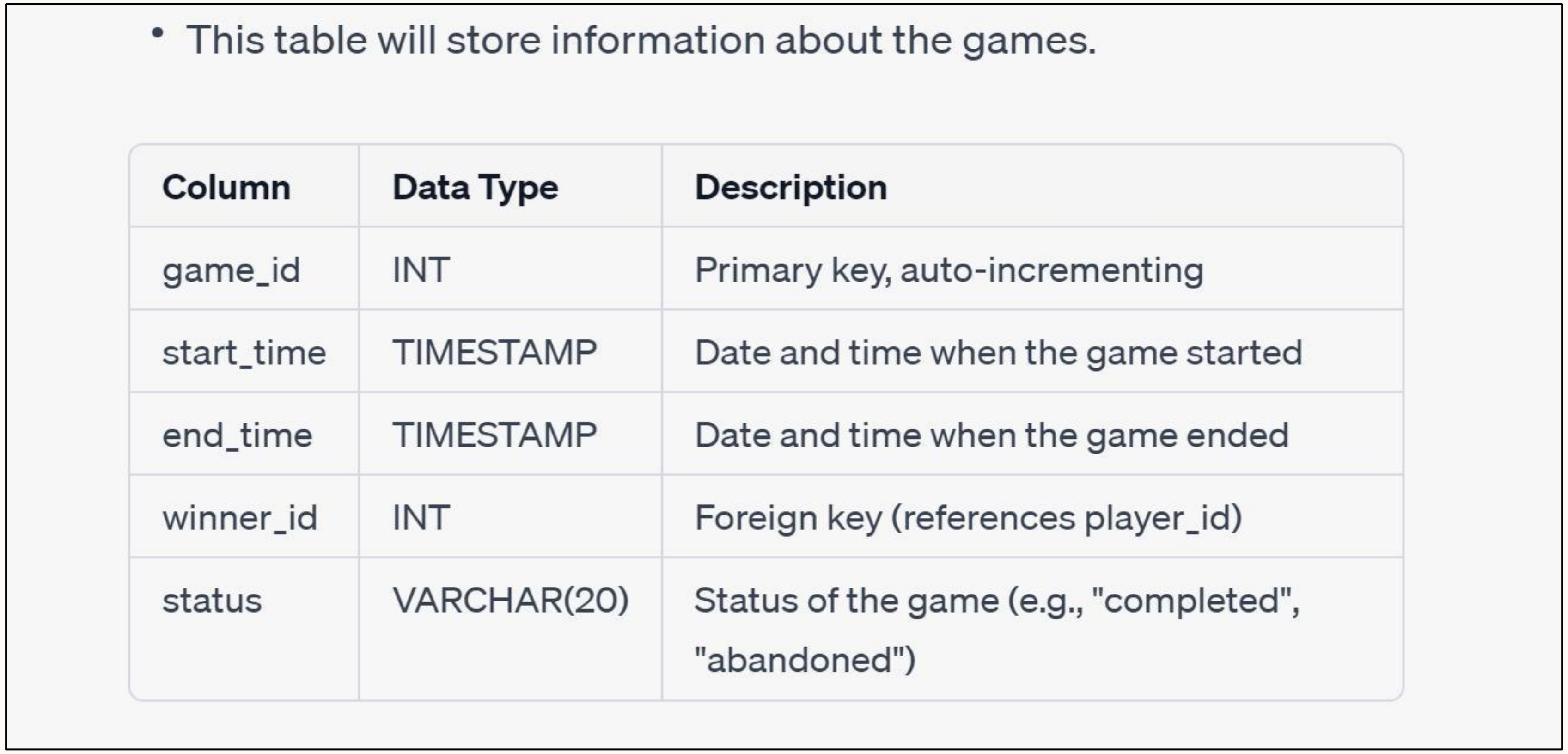
1. PLAYERS TABLE



- This table will store information about the players.

Column	Data Type	Description
player_id	INT	Primary key, auto-incrementing
username	VARCHAR(50)	Unique username
password	VARCHAR(100)	Hashed password
email	VARCHAR(100)	Email address
created_at	TIMESTAMP	Date and time of account creation

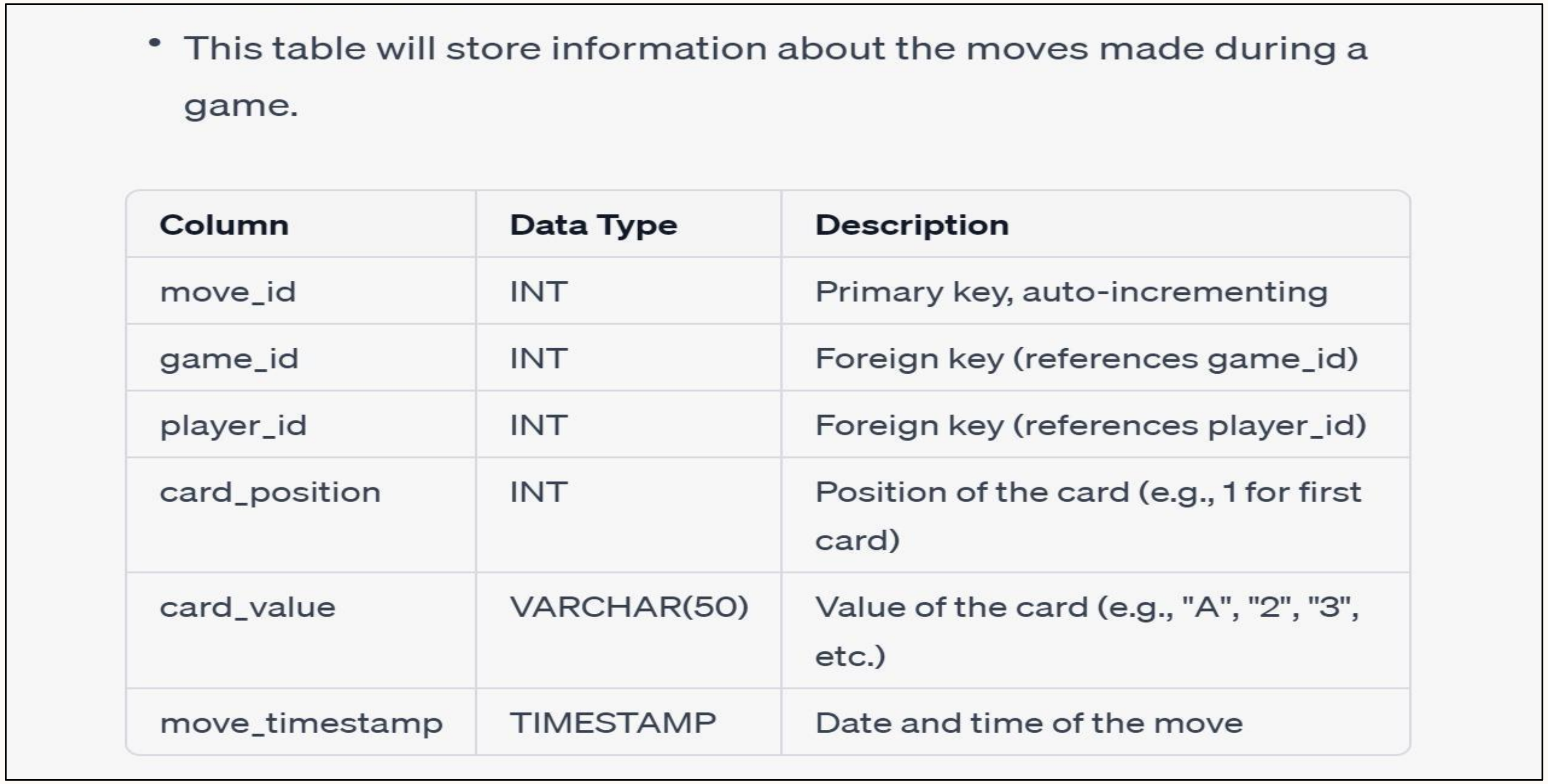
2. GAMES TABLE



- This table will store information about the games.

Column	Data Type	Description
game_id	INT	Primary key, auto-incrementing
start_time	TIMESTAMP	Date and time when the game started
end_time	TIMESTAMP	Date and time when the game ended
winner_id	INT	Foreign key (references player_id)
status	VARCHAR(20)	Status of the game (e.g., "completed", "abandoned")

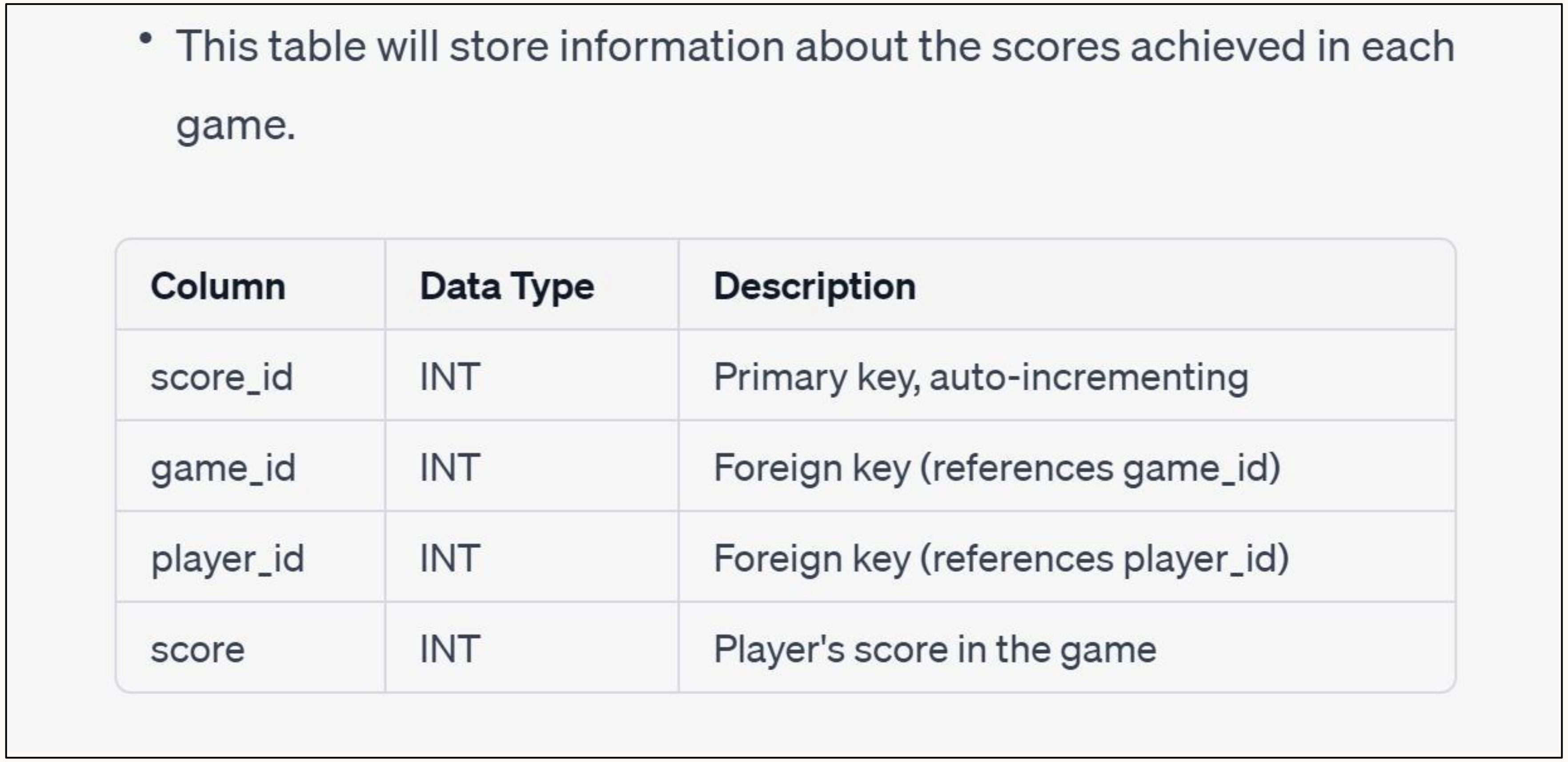
3. GAMES MOVES TABLE



- This table will store information about the moves made during a game.

Column	Data Type	Description
move_id	INT	Primary key, auto-incrementing
game_id	INT	Foreign key (references game_id)
player_id	INT	Foreign key (references player_id)
card_position	INT	Position of the card (e.g., 1 for first card)
card_value	VARCHAR(50)	Value of the card (e.g., "A", "2", "3", etc.)
move_timestamp	TIMESTAMP	Date and time of the move

4. SCORES TABLE



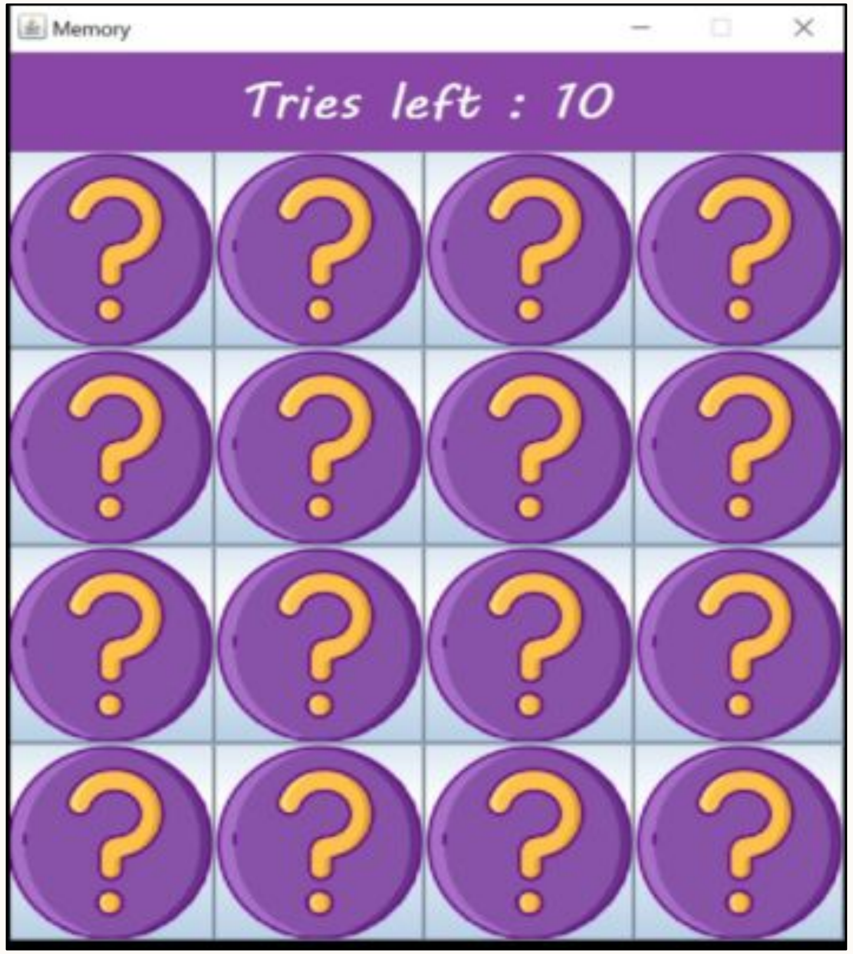
- This table will store information about the scores achieved in each game.

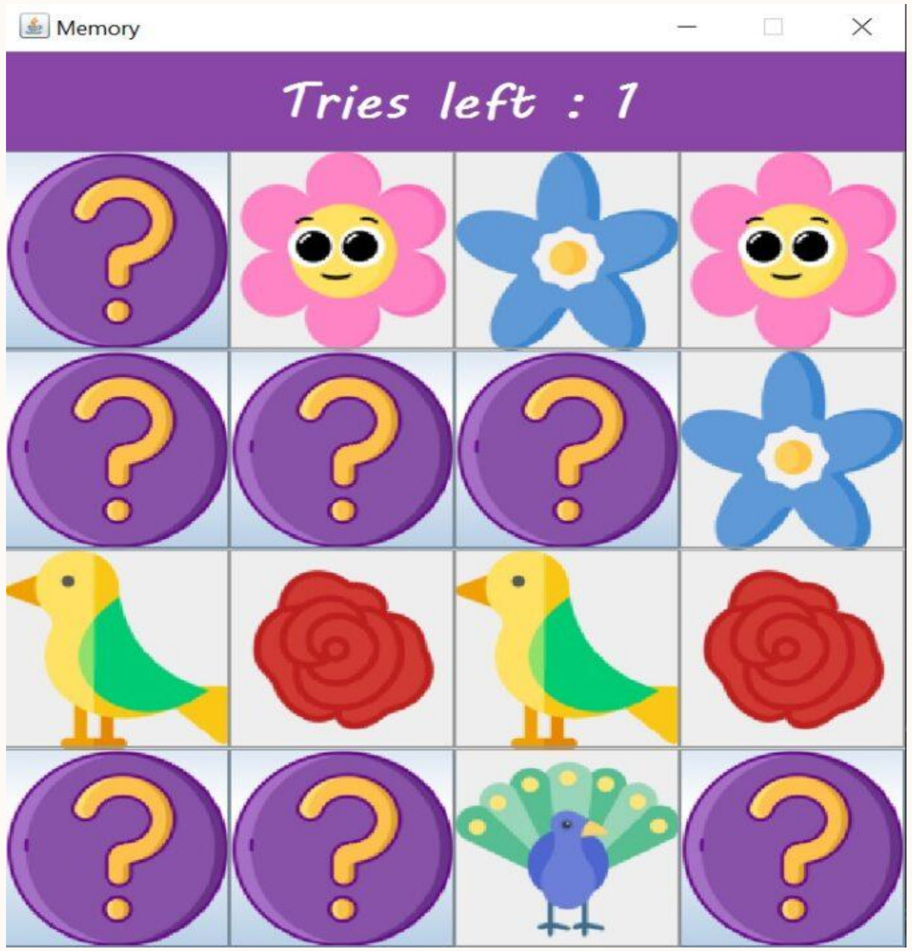
Column	Data Type	Description
score_id	INT	Primary key, auto-incrementing
game_id	INT	Foreign key (references game_id)
player_id	INT	Foreign key (references player_id)
score	INT	Player's score in the game

BASE CODE

Output

Output:





```
import javax.imageio.ImageIO;
Import javax.imageio.stream.ImageInputStream;
import javax.swing.*;import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;
import java.util.stream.Collectors;
import javax.swing.Timer;
class Game {
        public static class Controller {
            final JFrame window;
            Model model;
            View view;
           public Controller(Model model) {
this.window = new JFrame("Memory");
this.window.setDefaultCloseOperation
(WindowConstants.EXIT_ON_CLOSE);
this.window.setResizable(false);
        this.reset(model);
}
```

```
 public void reset(Model model) {                this.model
=model;
 this.view = new View(model);
this.window.setVisible(false);
this.window.setContentPane(view);
this.window.pack();
this.window.setLocationRelativeTo(null);          for (JButton
button : this.model.getButtons()) {
button.addActionListener(newButtonActionListener(this));
      }
 Utilities.timer(200, (ignored) ->
this.window.setVisible(true)); // Show the window after
200ms       }
 public JFrame getWindow() {
          return this.window;
}       public Model getModel()
{          return this.model;       }
  public View getView() {
      return this.view;       }    }
 public static class Model {
  // Constants for the game
  static final String[] AVAILABLE_IMAGES = new
String[]{"0.png", "1.png", "2.png", "3.png", "4.png",
"5.png", "6.png", "7.png", "8.png"};
 static final Integer MAX_REGISTERED_SCORES =
10;
 final ArrayList<Float> scores;
  final ArrayList<JButton> buttons;
   final int columns; // Number of columns
   int tries; // Number of tries left
   boolean gameStarted;
// Is the game started
      public Model(int columns) {
 this.columns = columns; // Number of columns
this.buttons = new ArrayList<>(); // List of buttons in the
game
  this.scores = new ArrayList<>(); //
 this.tries =  10; // Number of tries initially
this.gameStarted = false; // Game is not started initially
 int numberOfImage = columns * columns; // Number
of images
 Vector<Integer> v = new Vector<>(); // Vector to store
the images
 for (int i = 0; i < numberOfImage -numberOfImage % 2;
i++) { // Add the images twice              v.add(i %
(numberOfImage / 2));          }
 if (numberOfImage % 2 != 0)
v.add(AVAILABLE_IMAGES.length - 1); // Add the last
image if the number of images is odd           // Add the
images as a button to the game          for (int i = 0; i <
numberOfImage; i++) {              int rand = (int)
(Math.random() * v.size()); // Randomly select an image
String reference =
AVAILABLE_IMAGES[v.elementAt(rand)]; // Get the
image              this.buttons.add(new
MemoryButton(reference)); // Add the image as a button
v.removeElementAt(rand); // Remove the image from the
vector          }      }
public int getColumns() {
  return columns;
   }
   public ArrayList<JButton> getButtons() {
     return buttons;
    }      // Get the number of tries left
   public int getTries() {
   return tries;
  }      // Decrement the tries count by calling this method
    public void decrementTries() {
       this.tries--;
   }      // return if the game has started
    public boolean isGameStarted() {
    return this.gameStarted;
    }      // start the game
  public void startGame() {
    this.gameStarted = true;      }   }
```

CONCLUSION

The Memory Game, is made entirely out of Java. It has a Graphical User Interface (GUI) with all of the functions that will enhance the user experience.

This **Memory Game In Java** main purpose is to provide a leisure time activity as well as for enhancing memory.

THANK YOU

Achanta Bhavya Sree
RA2211003010316

Parvathy V Nair
RA2211003010295

Anyesha Biswas
RA2211003010298