# NORTH SOUTH UNIVERSITY

## Java Project Report

**Project Name: Memory Matching Game**

**Course Initial: CSE215**

**Course Name: Java**

**Faculty: Dr. Mohammad Shifat-E-Rabbi [MSRb]**

**Section: 07**

**Group Members:**

1. **Fardin Sarker – 2411298042**
2. **Sohail Kabir – 2413483042**
3. **Tanjim Farah - 2411267642**

# Project Report: Memory Card Matching Game

## Project Overview

The **Memory Card Matching Game** is an interactive application developed to test and enhance users' memory skills. Initially, an analog console-based program (`test.java`) was developed, which implemented the core game logic. Later, we expanded and improved the project by integrating JavaFX to create a graphical user interface (GUI). This report outlines the thought process, challenges faced, and solutions adopted in transitioning from the console-based version to the GUI-based application (`MemoryGameApp.java`).

## Initial Development: Analog Program (`test.java`)

In the initial console-based version, the focus was on establishing the game's foundational mechanics:

1. **Deck Initialization**: A random selection of cards was used to generate pairs, ensuring variability for each game session.
2. **Shuffling and Hiding Cards**: Cards were shuffled and represented as `*` to keep them hidden from the player.
3. **User Interaction**: Players selected two cards by entering their positions. If the cards matched, they remained revealed; otherwise, they reverted to hidden.
4. **Error Handling**: Mechanisms were added to manage invalid inputs (e.g., non-integer values or positions out of range).

### Challenges and Solutions

- **Card Matching Logic**: Ensuring matched cards stayed revealed without affecting unmatched cards required careful validation.
- **Delay Mechanism**: Simulating a delay for revealing cards before hiding unmatched pairs used `Thread.sleep`, but this approach lacked user feedback and needed refinement in the GUI.

# Transition to GUI: JavaFX Application (`MemoryGameApp.java`)

The GUI version introduced a more interactive and visually appealing game experience. The program uses JavaFX components like `GridPane` for layout, `Button` for cards, and `PauseTransition` for delays.

## Key Features Implemented

1. **Card Deck Initialization**: Similar to the analog version, the deck was initialized with pairs of random cards. This logic was reused and adapted for GUI elements.
2. **Button Grid for Cards**: A `GridPane` was used to arrange card buttons in a 2x4 grid, allowing intuitive interaction.
3. **Card Flip and Matching**: Clicking a button revealed the card's value. If two cards matched, they remained revealed; otherwise, they reverted after a delay using `PauseTransition`.
4. **End-of-Game Popup**: An alert displayed a congratulatory message when all cards were matched.

## Challenges Faced and Solutions

### 1. Positioning Buttons in the Grid

- Initially, aligning buttons in a clean grid layout was challenging. Using JavaFX's `GridPane` with proper spacing (`Hgap` and `Vgap`) resolved this.
- *Internet Resource*: Documentation on `GridPane` and spacing properties.

### 2. Font Adjustment for Cards

- Displaying hidden (`*`) and revealed cards required dynamic font resizing. The `Font` class was used to modify font sizes based on card states.
- *Internet Resource*: Tutorials on JavaFX `Font` and dynamic styling.

### 3. Delay for Card Reversion

- The analog version used `Thread.sleep`, which was unsuitable for GUI. Instead, `PauseTransition` was used to introduce a non-blocking delay, allowing the interface to remain responsive.
- *Internet Resource*: Articles on `PauseTransition` and JavaFX animation utilities.

### 4. Card Selection Logic

- Ensuring that already revealed cards couldn't be reselected caused unexpected behavior during development. Logic was added to check if a card was already revealed before processing the selection.
- *Testing and Debugging*: Multiple iterations of code refinement resolved these issues.

## Conclusion

The transition from a console-based application to a GUI-based interactive game required adapting core logic and solving new challenges introduced by the graphical interface. By incorporating JavaFX, we enhanced the game's usability and aesthetics, making it more engaging for users. The project allowed us to learn and implement advanced Java concepts such as event handling, GUI layout management, and animation utilities. The challenges faced during development and the solutions adopted have deepened our understanding of JavaFX programming and software development practices.