**Project Name:** Memory Card Game

**Github Link:** https://github.com/projectsforstudents2022/Memory\_Card\_Game.git

**Why was this project created?**

The Memory Game Project is all about showing off your JavaScript prowess. You'll create a fully functioning browser-based card matching game. However, this is not just an ordinary memory game. It's a feature-rich, well-designed memory game.

**What problem is it solving?**

There are sixteen "cards" on the game board, which are placed in a grid. Eight separate pairings of cards, each with a unique emblem on one side, make up the deck. The symbols on the cards are placed face-down on the grid in a random order. The game's rules are relatively straightforward: Find the matching cards by turning over two concealed cards at a time.

**Entire explanation of project**

* **PROPOSED APPROACH**

We have a programme that only ever draws a tile grid before stopping. We want our programme to display a variety of images starting with face-down tiles, followed by clicked tiles and, if everything goes well for the player, a win screen.

Moving all of our drawing code to the Processing JS draw function is the next logical step. While the application is running, the computer will continue to execute draw(), causing the tiles to continue to be drawn based on whether they are facing up or down. Some of those tiles are now facing up. The player must click on a tile in order to flip it. We can define a mouseClicked function in Processing JS scripts to react to clicking.

When our application notices that the player has clicked someplace, we want to use mouseX and mouseY to detect if they have clicked on a tile. Let's begin by giving Tile an isUnderMouse method that returns true if a given x and y are inside the area of a tile.

We should only return true if the supplied x is between this.x and this.x + this.size, and if the given y is between this.y and this.y + this.size, because the way we've designed the tiles corresponds to the tile's x and y being in the upper left corner. Now that we have that technique, we can check in mouseClicked to see if each tile is beneath the mouseX and mouseY using a for loop.

* **RESULT**



**CONCLUSION**

It turns out that a fairly straightforward approach is the best way to play the memory game. But the research and evidence offered here required a lot of work. Is there a simpler way to demonstrate the findings? We anticipate that the techniques we utilized here will be applicable elsewhere, even if the outcomes of our work are mostly of recreational value. We want to emphasize once more the crucial significance that experimentation and automated symbolic computations have played in this effort.