**INTERNATIONAL UNIVERSITY – NATIONAL UNIVERSITY**

**HO CHI MINH CITY**

**TRƯỜNG ĐẠI HỌC QUỐC TẾ - ĐẠI HỌC QUỐC GIA**

**TP. HỒ CHÍ MINH**



**Algorithms & Data Structures**

**Instructor:** T. T. Tín

**Student:** Ngô Tùng Chương

**=== ENTERTAINMENT SECTION ===**

|  |
| --- |
| **SUDOKU** |
| 1. **Overview**   This Sudoku consists of **three primary** components:   * **Menu:** Includes options to pick difficulty (This also functioning as a restart button), solve the board, and a timer.  |  | | --- | | **Pic 1.0**: Menu slide |  * **Number picker:** Prevent player to input weird character beside 1 – 9 number into tile.     **Pic 1.1**: Right side of the board is Number picker   * **Gameplay logic:** Fill the all the tiles on board. With each row, column, and grid, not duplication of number exists.     **Pic 1.2**: Sample how a correctly filled board should be.    **Pic 1.3:** Rule |
| 1. **Data structures**   **2.1/ Arrays**   * **Usage**: Use to **store** the number to generate Puzzle, and solution for that Puzzle * **Purpose**: Allow to **access tile randomly**, which is effective when it comes to **compare** player input to solution generated beforehand. * **Advantages**:   + **Constant-time access** to coordinates (tile)  + **Access**: O(1)  + **Update:** O(n), where n is rows, columns, grids    **Pic 2.0**: Constant-time access allows us to quick access random tile to check, and compare it value. |
| **2.2/ Random**   * **Usage**: Generate random empty tile on board * **Purpose**: Ensure each set generate is unique * **Advantages**: Time Complexity 0(1)     **Pic 2.1**: Random pick tile on board to generate a level. Each is difference from each other. |
| 1. **Algorithms**   This is a started of total 4 mini-project so in this one to simplify I only use **brute-force** from head to tail.  **3.1/ Generate Puzzle**   * **Description:**   +Generate a solution board  + Randomly empty a few tiles from solution board  + Display board   * **Implementation:**   + Solution board is filled in a loop pattern  (1 to 9, then 1 to 9). Each tile generates will be **recursive** checked if from that number generated on can be solved or not.  + Just select randomly [1-9] [1-9] tile to remove number, simple but still bring refresh experience each set.  + The number of blank tiles indicate what difficult game is set.    **Pic 2.2**: Difficulty mechanic |
| **3.2/ Check input**   * **Description:**   + Player input a number (1-9)  + If it a wrong move -> highlight text  + else nothing happens. Number display on input tile  **Implementation**:  + Generate a solution beforehand  + Compare inputArray.value to solutionArray.value  **Ex:** inputArray[1][1] == 8  solutionArray[1][1] == 7   * Indicate this is a wrong move with highlight text colour   **3.3/ Victory condition**   * **Description:**   + All tiles are populated  + Each row, column, and grid no detecting duplicate value   * **Implementation:**   + With each input , run check input above  + If input is correctly, check if board contain any blank or incorrect tile.  + All conditions satisfied, win. GG😊    **Pic 2.3:** Show how checkWinCondition() work |
| 1. **Conclusion**   This **Sudoku** game features a user interface with a menu (difficulty selection, solve option, timer), a number picker (restricting input to digits 1–9), and core gameplay logic that enforces Sudoku constraints by **validating uniqueness across rows, columns,** and **3×3 sub grids**. The board generation leverages a **brute-force backtracking algorithm**, which sequentially fills the grid using **recursive depth-first search** to ensure a valid complete solution. Difficulty is adjusted by randomly removing a specified number of tiles from the solved board to create the puzzle. |

|  |
| --- |
| **MINESWEEPER** |
| 1. **Overview**   This game main feature is the **undo** button.  Other options are list as **Quality of Life (QoL):**  + **Flag**: to set up an flag icon where you doubt exist bomb.  + **Bomb** **count**: Indicate how many bombs are on the field.    **Pic 1.4:** Main playground    **Pic 1.5**: Rule |
| 1. **Data Structure**   **2.1/ Arrays**   * **Usage**: Use to **store** the minefield, the safe field; * **Purpose**: Allow to **access tile randomly**, which is effective when it comes to **compare** player input to minefield Array. * **Advantages**:   + **Constant-time access** to coordinates (tile)  + **Access**: O(1)  + **Update:** O(n), where n is rows, columns  **2.2/ Random**   * **Usage**: Generate random mine on field * **Purpose**: Ensure each set generate is unique * **Advantages**: Time Complexity 0(1)     **Pic 2.4:** The benefit of random() when it comes to generate level  **2.3/ Stack**   * **Usage**: Use to **store** the history (Secret behind undo button) * **Purpose**: Can store multi steps player have do (last in). And each time, undo it can pop (first out) * **Advantages**:   + **Constant-time access** to coordinates (tile)  + **Access**: O(1)  + **Update:** O(n), where n is rows, columns |
| 1. **Algorithms** |
| 1. **Conclusion** |