**Final Project Analysis, Design and ICD - Sudoku Application**

**Rev v2.0**

**Abel Tabor, Thomas Edwards, Brian Jefferson**

**CMSC 495**

**Section 7384**

**Group 5**

**5/8/2022**

**Professor Dao**

**Revision History**

***Project Analysis***

|  |  |  |
| --- | --- | --- |
| **Name** | **Date** | **Description** |
| Abel Tabor | 4/9/2022 | Started and completed Input Data, Output Data, Data processing steps, Context Diagram, Subsystems Diagrams and explanations, Subsystem and linked requirement id chart and Possible Enchantments |
| Brian Jefferson | 4/9/2022 | Added Risks and Risk Management |
| Abel Tabor | 5/4/2022 | Revised Subsystem Diagram and Subsystems Requirements to better reflect final project |
| Thomas Edwards | 5/8/2022 | Updated Cover page (Rev v2.0), minor editing, and finalization. |

***Project Design***

|  |  |  |
| --- | --- | --- |
| **Name** | **Date** | **Description** |
| Abel Tabor | 4/08/2022 | Started Document, Added Sudoku Generator |
| Brian Jefferson | 4/08/2022 | Added starting information for Startup/ Shutdown |
| Abel Tabor | 4/10/2022 | Added GameStats, MenuGUI, GameGUI, LeaderGUI. Started Database |
| Brian Jefferson | 4/11/2022 | Adding the additional scenarios based on program |
| Thomas Edwards | 4/11/2022 | Set up GitHub for software control |
| Thomas Edwards | 4/12/2022 | Comments to error handling. Added invalid input limitation to Risk mitigation. |
| Thomas Edwards | 4/14/2022 | Started outlining Project Test Plan & ICD in accordance with Project Plan & Project Requirements (Week 2). |
| Abel Tabor | 5/4/2022 | Revised pseudo-code to better reflect final project |
| Thomas Edwards | 5/8/2022 | Added descriptions, Pre/Post-Conditions. Minor editing for readability and finalization. |

***ICD***

|  |  |  |
| --- | --- | --- |
| **Name** | **Date** | **Description** |
| Abel Tabor | 4/17/2022 | Outlined and all the ICD |
| Abel Tabor | 5/4/2022 | Updated ICD to better reflect final project |
| Thomas Edwards | 5/8/2022 | Minor editing for readability and finalization. |

**Project Analysis**

Our group worked on this document to analyze our project of developing a desktop application that can let end users play a randomized game of sudoku and save and display previous users scores on a leaderboard.

**Input Data & Sources:**

|  |  |
| --- | --- |
| **Input Data** | **Source** |
| Player Menu Choice | End user selects from GUI between two options, “Start a new game” or “Leaderboards” |
| Player Name | End user input, prompted at the end of the puzzle. |
| Player Time/ Player Score | End user application instance. Occurs when user starts a puzzle and ends once user finish said puzzle |
| Entering puzzle variables | End user is prompted what number to input for missing aspects of the puzzle. Input is limited through a separate GUI pop-up with numpad prompt (1-9). |
| Entering full puzzle | End user enters in complete puzzle from the GUI |

**Output Data & Destination:**

|  |  |
| --- | --- |
| **Outputs** | **Destinations** |
| Game Time / Game Score / Username | Displayed to the user through leaderboard GUI when prompted. |
| Game Time | Displayed while the user plays the game |
| Game Score | Displayed while the user plays the game |
| Sudoku Puzzle | Displays incomplete sudoku puzzle on main game UI. |
| Puzzle State | Displays if entered puzzle is correct or incorrect. |

**Data Processing Steps:**

**Launched Application**

1. User is given three options: Start a new game, see leaderboards or exit.

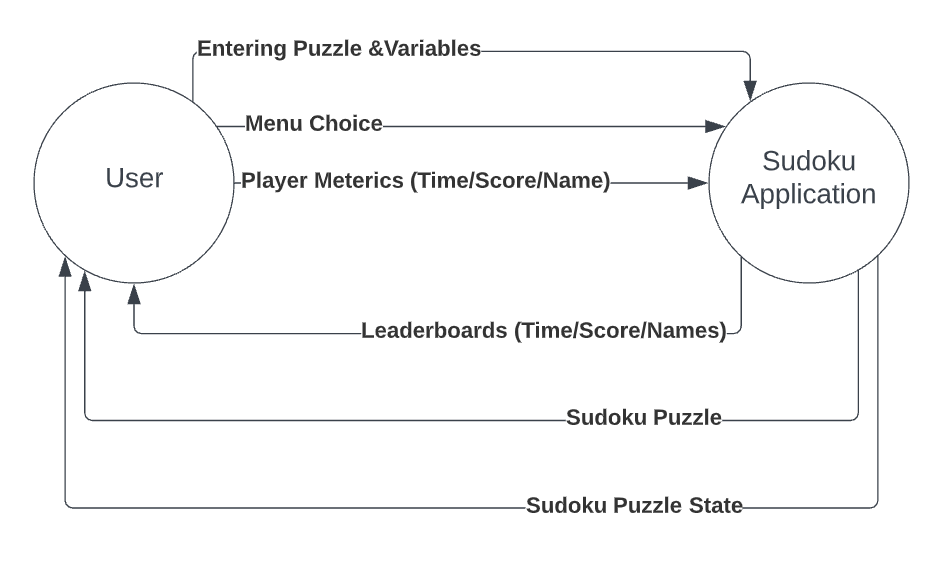
**New game**

1. Generate full sudoku board
   1. Using the generated board, select which variables to hide from the end user.
   2. Randomly select variables to remove and use a sudoku solver method to ensure that puzzle can still be solved
2. Display altered board on the main GUI
   1. Signals the start of the game and Game Time should being to count up from here
   2. User can fill in the board by selecting empty variables and selecting what number to place
3. User enters solved puzzle
   1. User enters solved puzzle by clicking on a button on the GUI, the application then takes in board state of user and compares it to the full board
      1. Every time user attempts to enter a puzzle with mistakes, score decreases
   2. Game displays if entered puzzle is correct or incorrect
4. Entered user board matches with full sudoku board
   1. User is prompted for username for current session. Once a valid username has been entered the username, game score and game time are added to the database alongside a unique session ID.
   2. User is taken back to the start application screen

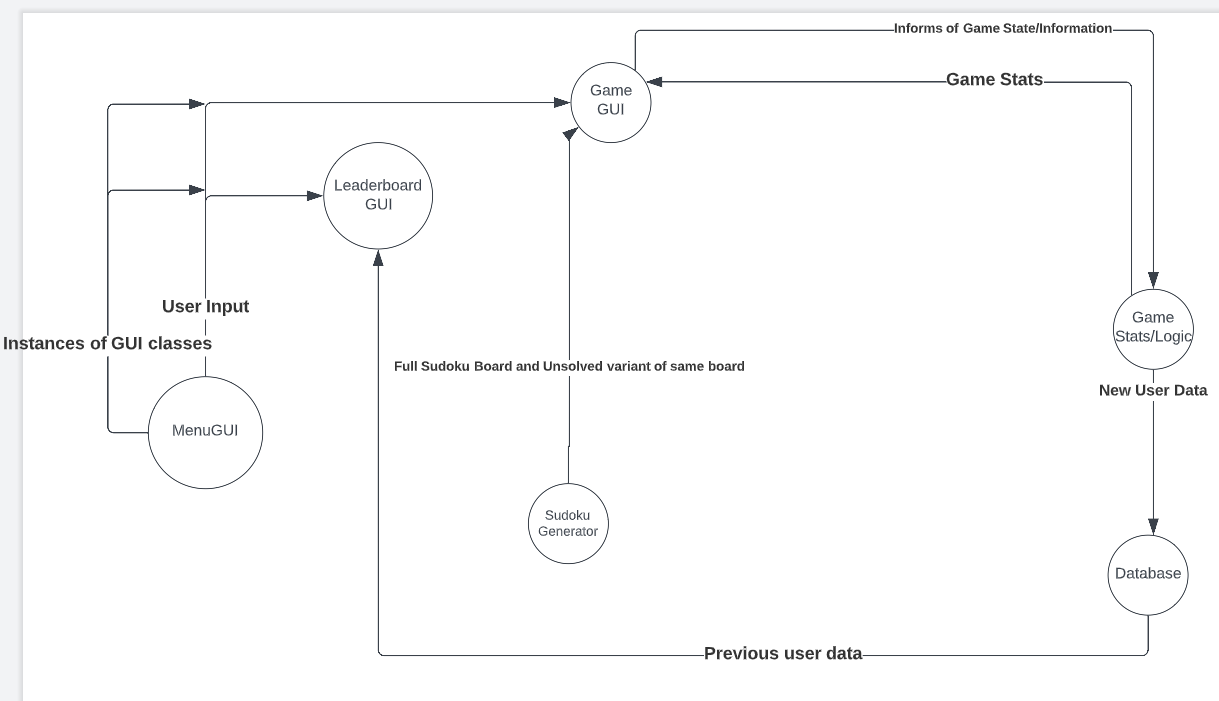
**Leaderboards**

1. Information is taken from the database and displayed in order of score
2. User prompted to go back to start application screen

**Context Diagram:**



**Subsystem Diagram:**



1. **Sudoku Generator subsystem** 
   1. This takes an empty board and produces a sudoku board that adheres to the basic sudoku rules (1-9, no repeating numbers vertically, horizontally and within each square quadrant).
2. **Game Stats/Logic** 
   1. The application informs this system when a new puzzle has started and when the user attempts to submit a new puzzle
   2. Updates game score and keeps track of the time passed on the current puzzle
      1. GameGUI displays this information to the user
   3. Once the game state updates to solved, the GameGUI prompts the user for their name and the information is passed this system which is passed to the database
3. **Database** 
   1. The database stores user data
      1. Information Stored
         1. Primary Key: Game ID -> Generated by querying total amount of entries and numbering them
         2. Game Score -> Received by Game Stats/Logic
         3. Game Time -> Received by Game Stats/Logic
         4. Username -> Received by Game Stats/Logic
   2. The leaderboard GUI for the application receives and displays information from the database
   3. Handles connecting to the database and managing resources
4. **MenuGUI**
   1. Allows for player to start a new game or see the leaderboards
   2. Contains the instances of the GUI classes to allow classes to interact with each other
5. **Game GUI** 
   1. Displays necessary information for the user to interface with the puzzle
   2. Provides the user input for the application
6. **Leaderboard GUI** 
   1. Displays necessary information from database to rank past players on their performance

**Subsystem and Requirements**:

**Note:** Windows-based system required to run “SpeedSudoku.exe”

|  |  |
| --- | --- |
| **ID** | **Subsystem** |
| 01 | Sudoku Generator |
| 02 | Game GUI |
| 03 | GameLogic |
| 04 | Database |
| 05 | Leaderboard GUI |
| 06 | Menu GUI |

**Possible Enhancements:**

Rather than every new puzzle score being treated as a new person, a log-in system with usernames and passwords can be used to allow users to review past puzzles.

Implement a difficulty option using the sudoku solver and the number of missing variables of any given puzzle

**Risk & Risk Management -**

Limit the amount of input that the user can input. Certain risks:

**Username**

* This is the most open type of user input acknowledged in the system. It is also inserted into the database so SQL injections are a risk. The input should be sanitized and maybe even limit what usernames can be.

**Submitting the puzzle**

* When users submit the puzzle, the score decreases by a certain amount. Users that spam submit puzzles might be capable of causing some issues if not handled properly.

**Project Design**

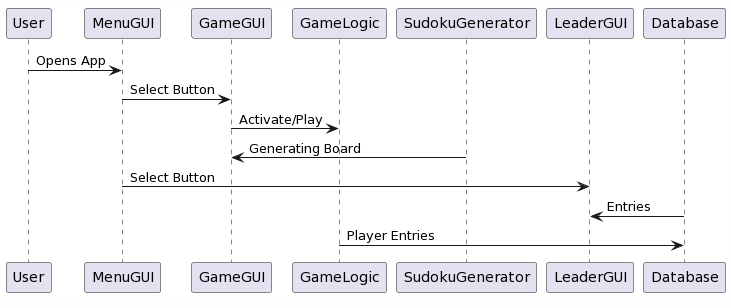
**Event Trace Diagrams:**

**\*Pre-Condition for all scenarios:**

Users must have the “Release v0.2” folder from GitHub repository (<https://github.com/ThomasJE85/CMSC-495-Group-5.git>)> ”Release v0.2”> “SpeedSudoku.exe” (Windows required) > \*download and run\*. \*\*Recommend saving to desktop for easier application launch\*\*

**Startup Scenario Description:**

A normal startup will have the “CMIS 495: A Sudoku Game” GUI displayed on the user’s device.

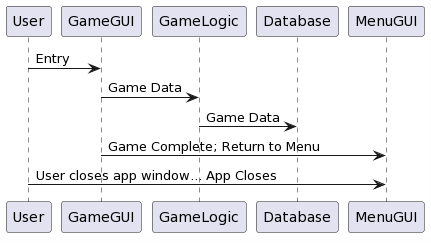
****

**Post-Conditions:**

From application startup, the user can “Start a new game”, view “Leaderboards”, minimize the window, maximize the window, or close the application window.

**Shut-Down Scenario Description:**

Regardless of where the user is within the application, the option to close the application window is always present. If a game is at-run, the user will be prompted about “unsaved progress”. A normal shutdown scenario will see the application window closed. \*SpeedSudoku.exe relaunch required to run application again\*

****

**Shutdown Post-Conditions:**

Assuming a complete game has been played (user enters name), data is stored in the Database for Leaderboard access. Otherwise, no data is added to the Database. Additionally, this applies to the user closing the application window entirely. \*SpeedSudoku.exe relaunch required to run application again\*

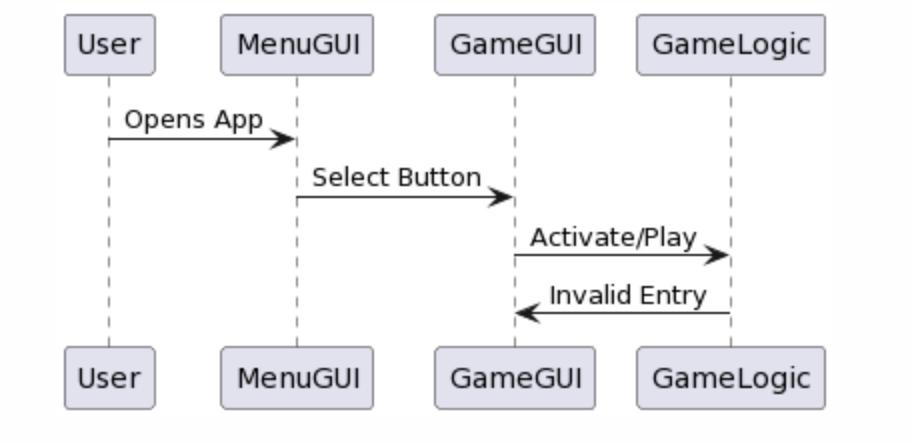
**Error Handling Scenarios Description:**

Possible error handling scenarios that were considered include leaving a game early, the ability to input duplicate/incorrectly complete a game, and the ability for the user to input an invalid username.

* The application window shall always prompt the user about “unsaved data” if they choose to either end a game early (to return to the Menu or Leaderboards) or close the application window entirely.

**Invalid Entry Description:**

* The ability to input an invalid option shall be mitigated by restricting the option to buttons with numbers 1-9 (Game GUI class).
* The ability for the user to either complete a game incorrectly or enter duplicate entries shall be mitigated by the Game Logic.
* The ability for the user to input an incorrect username shall be mitigated by the GameLogic and prompt the user to input a correctly formatted username (only alphanumeric).

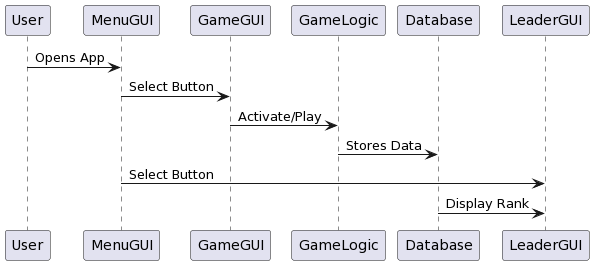
****

**Post-Conditions:**

At any instance of an invalid entry, the user shall be either prompted that an “incorrect” game has been attempted or that their username is not valid and they should retry their input with a correct one.

**Leaderboard Screen Description:**

The Leaderboard can either be accessed from the application menu screen or from within the game screen. On application launch, the Leaderboard will be empty. After a completed game and the user has entered a valid name, the Leaderboard will display game data (Username, Time, and Score) in descending order from the top of the Leaderboard.

****

**Post-Conditions:**

* Accessing the Leaderboard from the main menu will display any existing game data in descending order, ranking all previous users.
* If the Leaderboard is accessed from the game screen, the user will be warned that their data will not save (if a game is active).
* Normal access happens after a game has been completed. The user will be prompted to enter a valid username; which stores completed game data (Score, Time, and Username), and updates the existing Leaderboard in descending order.
* Should the user close the application window, SpeedSudoku.exe relaunch required to run application again.

**Class Design**

1. Sudoku Generator

Class SudokuGenerator {

Private final int boardSize = 9;

Private int[][] fullBoard = new int[boardSize][boardSize] ;

Private int[][] gameBoard = new int[boardSize][boardSize];

Public SudokuGenerator() {

fill();

remove();

}

Public int[][] getFullBoard() {

Return this.fullBoard;

}

Public int[][] getGameBoard() {

Return this.gameBoard;

}

Private boolean colCheck (int col, int number) {

Checks if given number is unique within given col

}

Private boolean rowCheck (int row, int number) {

Checks if given number is unique within given row

}

Private boolean squareCheck (int row, int col, int number) {

Checks if given number is unique within 3x3 square determined by given row and col

}

private void fill() {

Generate random number and check using colCheck, rowCheck, and squareCheck and add to fullBoard if all true

}

Private void remove() {

Removes values from randomly selected grid points. Calls Sudoku Solver, if true writes new board to gameBoard

}

1. Game GameLogic

Class GameLogic {

Private int sec, min, hour, score;

Private String name;

Database db;

Public GameLogic {

This.sec = 0;

This.min = 0;

This.hour = 0;

This.score = 10000;

}

Public int[] getTime() {

Returns time in an array

}

Public string getTimeString() {

Returns the time as a string

}

Public void incrTime() {

Increments the time by a second, used in conjunction with javax.swing.timer

}

Public void updateScore() {

Updates the score

}

Public void enterDB() {

Uses the database class to enter this objects fields to the database

}

}

1. MenuGUI

Class MenuGUI extends JFrame {

Private MenuGUI;

Private GameGUI;

Private LeaderGUI;

Private JPanel jpanel;

Private Jbuttons jbuttons;

Public MenuGUI {

Constructs and displays the menu gui

}

Public getGUI() {

Getters for the created gui classes to allow classes to interact w/eachother

}

}

1. GameGUI

Class GameGUI extends JFrame {

Private Jpanel jpanel;

Private JButtons jbuttons;

Private GameLogic

Public GameGUI {

Constructs and displays the game gui

}

}

1. Database

Class Database {

Public boolean enterDatabase(String name, String time, int score) {

Create connection to database and enters in given information. If given information was entered without any problems, return true else return false then close connection to database

}

Public String[] getData() {

Creates connection to database and returns all entries. Closes connection to database

}

}

1. LeaderGUI

Class LeaderGUI extends JFrame {

Private Jpanel jpanel;

Private JButtons jbuttons;

Private database db;

Public LeaderGUI() {

Constructs and displays the game gui and database connection

}

}

**Previous Risks identified**:

* Invalid entries on game UI
* Invalid name entries for Leaderboard

**Risk Mitigation:**

* Utilize buttons numbered 1-9 to restrict invalid entries
* Restricted valid characters for name entry to alphanumeric values.

**Username submission**:

* Utilize regex expression to sanitize user input
* When submitting username to database, utilize parameterized queries

**Puzzle submission:**

* Limit reduction of score below 0

**Interface Control Documentation**

**Data between MenuGUI and GameGUI:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *From* | *To* | *Name* | *Type* | *Value Range* | *Default Value* |
| MenuGUI | GameGUI | MenuGUI | MenuGUI | Null or created object | Created object |
| MenuGUI | GameGUI | GameGUI | GameGUI | Null or created object | Created object |
| MenuGUI | GameGUI | LeaderGUI | LeaderGUI | Null or created object | Created object |

**Data between MenuGUI and LeaderGUI**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *From* | *To* | *Name* | *Type* | *Value Range* | *Default Value* |
| MenuGUI | LeaderGUI | MenuGUI | MenuGUI | Null or created object | Created object |
| MenuGUI | LeaderGUI | GameGUI | GameGUI | Null or created object | Created object |
| MenuGUI | LeaderGUI | LeaderGUI | LeaderGUI | Null or created object | Created object |

**Data between Database and LeaderGUI**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *From* | *To* | *Name* | *Type* | *Value Range* | *Default Value* |
| Database | LeaderGUI | userData | ArrayList<ArrayList<String>> | null - filled in values | null |

**Data between SudokuGenerator and GameGUI**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *From* | *To* | *Name* | *Type* | *Value Range* | *Default Value* |
| SudokuGenerator | GameGUI | board | int[][] | 0-9 | 0 |
| SudokuGenerator | GameGUI | fullBoard | int[] | 0-9 | 0 |

**Data between GameGUI and GameLogic**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *From* | *To* | *Name* | *Type* | *Value Range* | *Default Value* |
| GameLogic | GameGUI | score | int | 0-10,000 | 10,000 |
| GameLogic | GameGUI | timeString | String | a-z | null |
| GameGUI | GameLogic | name | String | a-z | null |

**Data between GameLogic and Database**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *From* | *To* | *Name* | *Type* | *Value Range* | *Default Value* |
| GameLogic | Database | score | int | 0-10,000 | 10,000 |
| GameLogic | Database | time | int[] | 0-60 | 0, 0, 0 |
| GameLogic | Database | name | String | a-z | null |
| Database | GameLogic | validEntry | boolean | true - false | null |