Group 2 Final Project Documentation

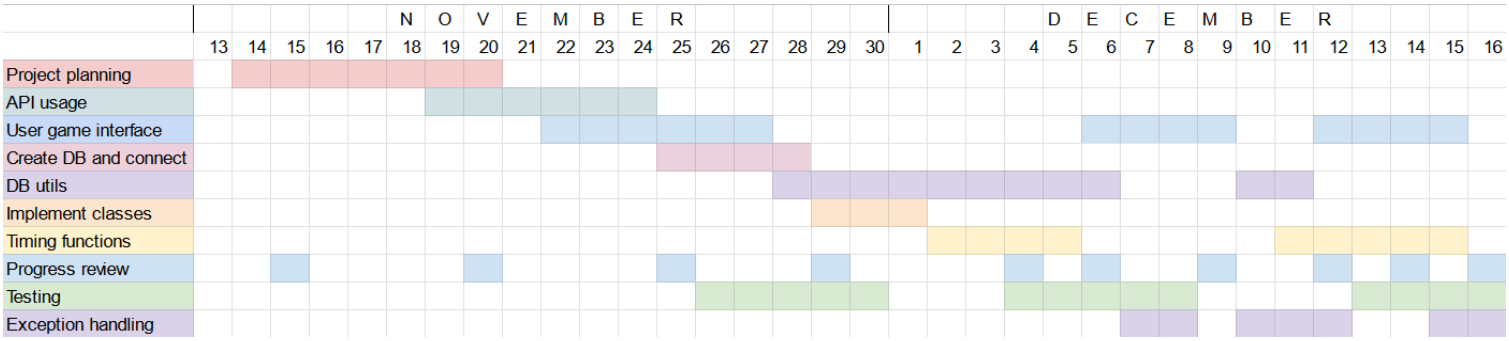
Team Members: Emma, Helena, Karimah, Kay, Gwen (partial)

INTRODUCTION:

We have built an app allowing users to play the game Sudoku called SudokuMania. This app will ask the user for the desired difficulty and use an existing sudoku API to generate sudokus for the user to play. The user can then input numbers into the board to play the sudoku. The app will check if the user's moves/plays are valid and the user has the option to reset the board at any point. If the user wishes to pause/exit the program, the app will store their current sudoku in a database that they can retrieve to continue later. The database will further store each game played with timestamps for completion and a potential scoring system based on difficulty and time spent. Users can access past games/ scores from the database.

Project Roadmap

A roadmap of the work to be completed and when is outlined in the image below. Thus presenting a somewhat even distribution of tasks over the completion period with regular progress reviews.



BACKGROUND:

Sudoku is a logic game in which a player is presented with a 9-by-9 grid, an example image is given here:

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

A sudoku game aims to complete the grid with numbers 1 to 9. The rules are that every row, every column and every 3-by-3 sub-grid must contain each number only once.

Due to the nature of the game, our project is aimed at users with a keen interest in problem-solving and similar puzzles. In particular, if a user wishes to test their skills and timing, our program's ability to time and score the user's completed boards is perfect!

SPECIFICATIONS AND DESIGN:

Initial design

The original idea for this project came from a combined love of Sudoku puzzles, after an initial search we found a suitable API for retrieving game boards and we decided to create a game which allows the user to insert values into the Sudoku until the board is complete. Our key requirements for this app are broken down into functional and non-functional requirements.

Functional Requirements

- Menu to choose options between starting a new game, continuing a game, seeing high scores or quitting.
- When starting a new game, the system asks the user for their choice of difficulty and generates a board of that difficulty.
- The user can add numbers to the sudoku board. They cannot change numbers that are already given but can change their inputted numbers.
- After completing the sudoku, the system checks that the solution is correct (that there are unique numbers in each row, column and box).
- The system times how long it takes for the user to complete the sudoku.
- The user has the option to restart the sudoku which would remove all their answers and leave the starting numbers.
- If there is a mistake with the sudoku in the final check the system alerts the user that the sudoku has a mistake.
- Throughout the game, the user has the option to pause the timer. Then they have the choice to either save and exit the game, restart the sudoku or continue.
- If the user chooses to continue and if there is a saved game currently in the database, they can continue playing that game and their time starts again.
- The user can request to see their fastest times for each difficulty.

Non-functional Requirements

Performance and scalability: The program is designed with a small scale in mind but has the availability to create as many sudoku boards as the user wishes due to the nature of API usage and DB storage. Due to the program only working with one board at a time and saving when not in use, adjusting to higher workloads would simply increase the size of the DB used and not affect the app's overall performance.

Portability and compatibility: The version runs on a Python environment with a connection to a MySQL database, therefore the user must have access to a local version of each software to run this app, details on connecting and setting up the database for use are available in the project README.md.

Reliability, maintainability, and availability: Exception handling is implemented to reduce critical failures during app use. The program has gone through thorough testing so any issues liable to arise are unlikely to require much downtime in comparison to user availability time.

Security: The app does not require storage of personal details aside from the user name, which is stored locally in a user's database after a local connection is established.

Localization: As this is a localised app only, the user is expected to run with local software as discussed above.

Usability: The app has high usability due to its variety of messages output to users informing of potential actions and requirements.

Files and Functions

This project includes a README.md file with key information about the project and instructions on how a player can run the program.

The source file directory is organised such that all project files are contained within the 'src' folder, all database files are within the 'db' folder, an example diagram is given here:

```
src
├── db
|   ├── sql_code.sql
|   ├── config.py
|   ├── connect.py
|   └── utils.py
├── unit_tests
|   ├── dbConnectionTests.py
|   ├── SudokuClassTests.py
|   ├── userTests.py
|   └── timingTests.py
├── main.py
├── sudoku_board.py
├── time_decorator.py
└── user.py
```

Database files

Within the db folder, there are files:

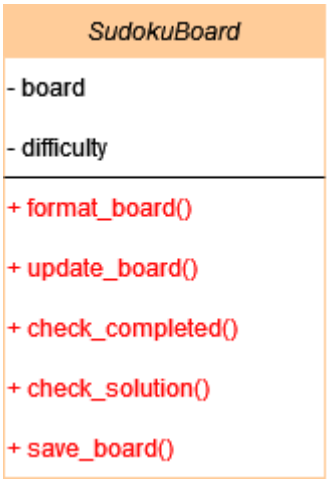
- sql_code.sql - which stores the relevant SQL code for creating the db - user must run this in their MySQL to create the DB (see more information in the project README.md). This file also contains insert statements for populating the database with data for a player "Jane" who has previously completed sudoku board, these are purely to populate the DB for testing the scoreboard functionality of the program.
- config.py - which holds the values relevant to SQL configuration, user must update their host, user and password information in this file before attempting to run the system
- connect.py - which calls on the values in the previous file and contains the function _connect_to_db() responsible for establishing the connection to the database using the values
- utils.py - which contains the functions with the SQL queries:
    - save_player() takes a player name as an argument and saves it to the player table in the database
    - get_player_id() which takes a player's name and returns their ID which is automatically incremented when the player's name is saved to the DB.
    - save_board_to_db() takes a dictionary of values as an argument and inserts it into the boards' table in the DB (note each row of a sudoku board is stored as a string of characters)
    - get_unfinished_board() retrieves the most recent unfinished board from the db for the relevant player currently using the program
    - get_score_info() will run the query to get the difficulty and time for all completed boards by the given user

Main file

The main.py file has the core functionality for the program and contains functions:

- play_game() takes board, player_id and previous_time as arguments. The argument board is an object of the class SudokuBoard. The argument player_id is the id of the current user and is an integer. The argument previous_time is an integer and is the stored time in seconds for the unfinished board in the database if the player is continuing their game, else if it is a new game then this is set to 0.
- sudoku_game_loop() takes the board as an argument. This is the function that allows the user to play the game. The function is timed by the @recordtime decorator. The user is asked for their input for the number they want to add to the board and which cell to add to by the function get_user_move(), they also have the option to pause the timer by exiting the function, this stops the timer which was started by the decorator.
- print_menu_options() prints the options from the user menu as described at the beginning of this section
- get_choice() asks the user to choose a menu option. It is also able to handle when the user doesn't input a valid option by calling ValueError.
- main() when called allows the user to play the whole game by offering the user the four menu options that allow them to start a new game, continue a game, view high scores and exit the game. This is implemented within a while loop which allows the user to continue making choices from the list of options until they choose to exit.

Within the file sudoku_board.py, the class SudokuBoard is defined to take a board (as a list of lists) and difficulty as arguments. An object of this class is created for each new board generated from the API or for each board retrieved from the DB.



The SudokuBoard class contains the following functions:

- format_board() which formats the board in an ASCII table
- update_board() which takes additional arguments of row, column and number. This function updates the cell in the row and column specified with the number given
- check_completed() which will validate if the board is filled
- check_solution() which will return True if the board is a valid solution
- save_board() which formats the board and its relevant info in a dictionary and calls on the save_to_boards_table() function within utils to save the board to the DB.

The sudoku_board.py file also contains the following non-class functions:

- get_sudoku_from_api() which takes the user's choice of difficulty as an argument and makes a call to the API to retrieve a board of the desired difficulty
- generate_new_board() which takes users' difficulty as an argument and uses the above function to get from API - creates a new object from the SudokuBoard class and returns the new board
- format_db_board() a function which is used after a board is retrieved from the DB (e.g. if the user chooses to continue an existing board) to convert the tuple of DB return values into a timestamp and an object of the SudokuBoard class

The file user.py contains the functions:

- get_player_name() which asks the user for input and returns the name
- get_difficulty() which asks the user to select a difficulty and checks if it is valid, returns difficulty if valid
- valid_difficulty() checks if difficulty is "easy", "medium" or "hard"

- get_user_move() which asks the user to select a row, column and number respectively and prints an error if invalid choices or returns restart or exit if the user types “restart” or “exit”
- valid_number() returns true if a given number in the integer range 1-9
- pretty_scoreboard() contains a wrapper for formatting the user scoreboard in a more appealing way
- get_user_score() is the function to which the previous wrapper is applied, gets the user time and difficulty from the DB by using get_score_info and outputs in a scoreboard the time, difficulty and score for each board completed by the user. The score is calculated based on a scoring system for the time taken and the difficulty of a board.

The file time_decorator.py contains the following functions:

- calc_time_taken() which takes two timestamps as arguments and returns the time difference in seconds
- record_time() decorator which is used to wrap around the sudoku_game_loop function in the main.py file - it takes a timestamp at the start and end of the function and returns the result of the calc_time_taken function for the two timestamps.
- convert_secs_to_hhmmss() will take a time in seconds as input and convert to the corresponding hh:mm:ss format - for a more user-friendly interface. We’ve also included the inverse function in the comments, this was created but not used in the final product, instead the function CAST to format the time from hh:mm:ss to seconds in MySQL.

Unit test files

The folder unit_tests contains four files containing unit tests for the programme: dbConnectionTests.py, SudokuClassTests.py, timingTests.py and userTests.py.

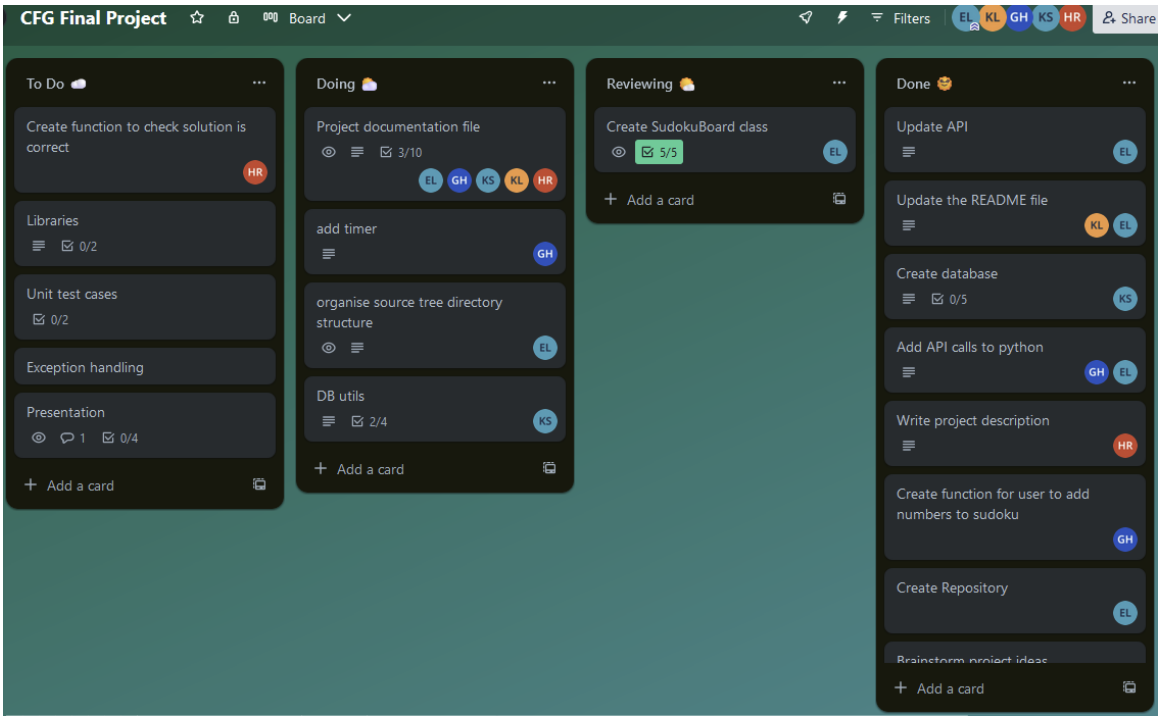
- dbConnectionTests.py: this tests the function test_connection_to_db() using the method is_connected()
- SudokuClassTests.py: this file tests the two methods of the SudokuBoard class, check_completed() and check_solution()
- timingTests.py: this file tests the decorator @recordtime by using the library time and the function mock_game(). This file also tests the functions convert_hhmmss_to_secs() and convert_secs_to_hhmmss()
- userTests.py: this file tests the functions valid_difficulty() and valid_number()

IMPLEMENTATION AND EXECUTION:

Development approach

The team uses a Trello board to plan, document and review tasks, under a self-assign basis, tasks can be added, updated and moved by any team member. The available tasks were regularly reviewed and updated during team meetings when we reassessed what needed completing.

Each team member recorded their contributions and time spent on the project activity log in the shared sheets. The log details time spent, tasks done and team members involved. The team underwent regular meetings to test and review the work, taking time to reevaluate all code functionality and purpose.



Due to varied schedules and personal circumstances, having regular meetings and a balanced workload was challenging. As the project progressed we found that certain members often had to have tasks delegated to them as and when available as opposed to discussing and self-assigning tasks. Due to this, we adopted more of a scrum framework for the vast majority of work with members Emma and Helena interchangeably taking on scrum master roles alongside their original project responsibilities. Often this involved standups, sprint planning and reviews, board administration and regular 1 on 1s to keep all team members on track.

Tools and Libraries

To access the API and make requests we used the library requests, this allowed us to get requests from the sudoku API for sudokus of varying difficulties.

When creating the timer for this app we used the Python library ‘time’ and the function time from this library. The timer function is a decorator that we used on the function sudoku_game_loop. It finds the current time at the start and the end of when the function is called and returns their difference in seconds.

When creating the restart function for this app we used the library copy and the function deepcopy. This was so that we could create a deep copy of the original board before the user inputted any moves and the original board would not update. Then, if the user wanted to restart we could reset to the original board.

Agile Development

We implemented agile elements in our team by using an iterative approach to complete this project, breaking down the main project into smaller chunks and continuously testing as outlined in the project roadmap. We took user feedback at various points as an opportunity to review and realign our requirements if necessary. We regularly reviewed each other's code and aimed to have at least a second review for every pull request created on the project repository. Many of our

reviews resulted in new meetings to discuss and often code refactoring to incorporate new ideas for implementation.

Implementation Challenges

A challenge faced within the initial stages of building our app was an issue found with our original choice of API. The original API struggled to retrieve specific difficulties due to the API format requiring multiple calls to do this (and ‘medium’ difficulties being generated far more often than ‘easy’ or ‘hard’). To resolve the issue we found an alternative API which generates an independent ‘easy’, ‘medium’ and ‘hard’ board with each call. After discussion with the group, it was decided that the API could be updated and previous API calls amended as required.

We discovered early on that the sudoku’s called from the API had multiple correct solutions, since there was no unique solution we couldn’t check each input against a correct value which was how we had initially planned to make the function check_solution(). Instead, we created a function that checked whether the values in each row, column, square of the sudoku were unique.

We found that writing data to the DB tables in the form of a matrix would prove challenging so we resolved to adjust the data entered to be in the form of strings for each row of the sudoku board. We converted between the two data types before saving a given board and after retrieving a board.

When storing the time in the DB we needed the format hh:mm:ss as opposed to the default format of a value in seconds generated when using the time.time function in Python. To convert between the seconds and the format hh:mm:ss we created the function convert_secs_to_hhmmss(). The time was then added to the DB by using the command CAST to cast the “hh:mm:ss” string to a time format to write to the board's table. However, we found it difficult to implement adding time in seconds onto this format when continuing a game. Originally, we created a function convert_hhmmss_to_seconds() to work in the reverse of the previous function but later found the MySQL function TIME_TO_SEC() would do this for us within the query so our python function convert_hhmmss_to_seconds() was no longer necessary (commented out in time_decorator.py). Converting to seconds resolved our issue of combining times and having the function to convert to hh:mm:ss provides the option to output a more user-friendly timestamp when necessary.

Taking over the work on the timers that Gwen had done was challenging, but since Gwen had kept us updated on her work and included clear comments in the code, understanding what had been done was clear. Initially, the play_game function had the timer wrapper one, this presented a challenge when we wanted to save the game and the time within this function but couldn’t access the time. To solve this we created a smaller function within the play_game function that could be timed. Ultimately this led to a clearer organisation of code.

TESTING AND EVALUATION:

Testing strategy

To test the code, we created unit tests. These tested individual functions to ensure they were working. Examples of functions we created unit tests for are the functions that check whether a sudoku board is complete and whether it is a valid solution, the timing wrapper that times functions, and the functions that check the validity of user input for choice of difficulty and number. We tried to

find edge cases for the unit tests, an example of one would be inputting nothing into the function that tests the validity of the user's input.

Functional and User Testing

Once the game had been completed and could be run all the way through, we gave the game to others to run through and test. The feedback we received from this was that the user options could be clearer at certain points, such as when choosing the difficulty. Other feedback included that the print statements we had included to show the connection and close off the database were unnecessary for the typical user when playing the game. As a result, we added clearer instructions for the user and removed unnecessary print statements.

System Limitations

A limitation in the system is that the restart function can only remove the entries that the user has entered in that session. For example, if they make changes to the board save and exit, and then wish to continue playing and restart they won't be able to remove previously entered numbers. One way we could implement this if we had more time would be to save the sudoku board initially to the database and store it there to be recalled when the user wants to restart.

CONCLUSION

To conclude, our SudokuMania app meets the majority of the requirements specified. Comparing our final project to the design brief, we met the majority of requirements but failed to implement the requirements to only allow users to update new cells of the board.

If we wished to develop this project further we could allow the user to complete any of their previous unfinished games (currently, they can only access the most recent). We might do this by asking the user to give a memorable name to each board, this could be associated with the corresponding board_id and called upon. Additionally, we could have an option to generate a random board of unknown difficulty for the user.

With extra time we would add other functionalities to the program such as the ability to save the boards at the start of the gameplay loop to the database so that the user would be able to fully restart the board. Another feature that could be added would be to call a leaderboard that compares different users' high scores against certain difficulties or to rank a user's high scores by score value as opposed to the current rank by difficulty and time taken.