**Zen Sudoku**



# **Executive Summary**

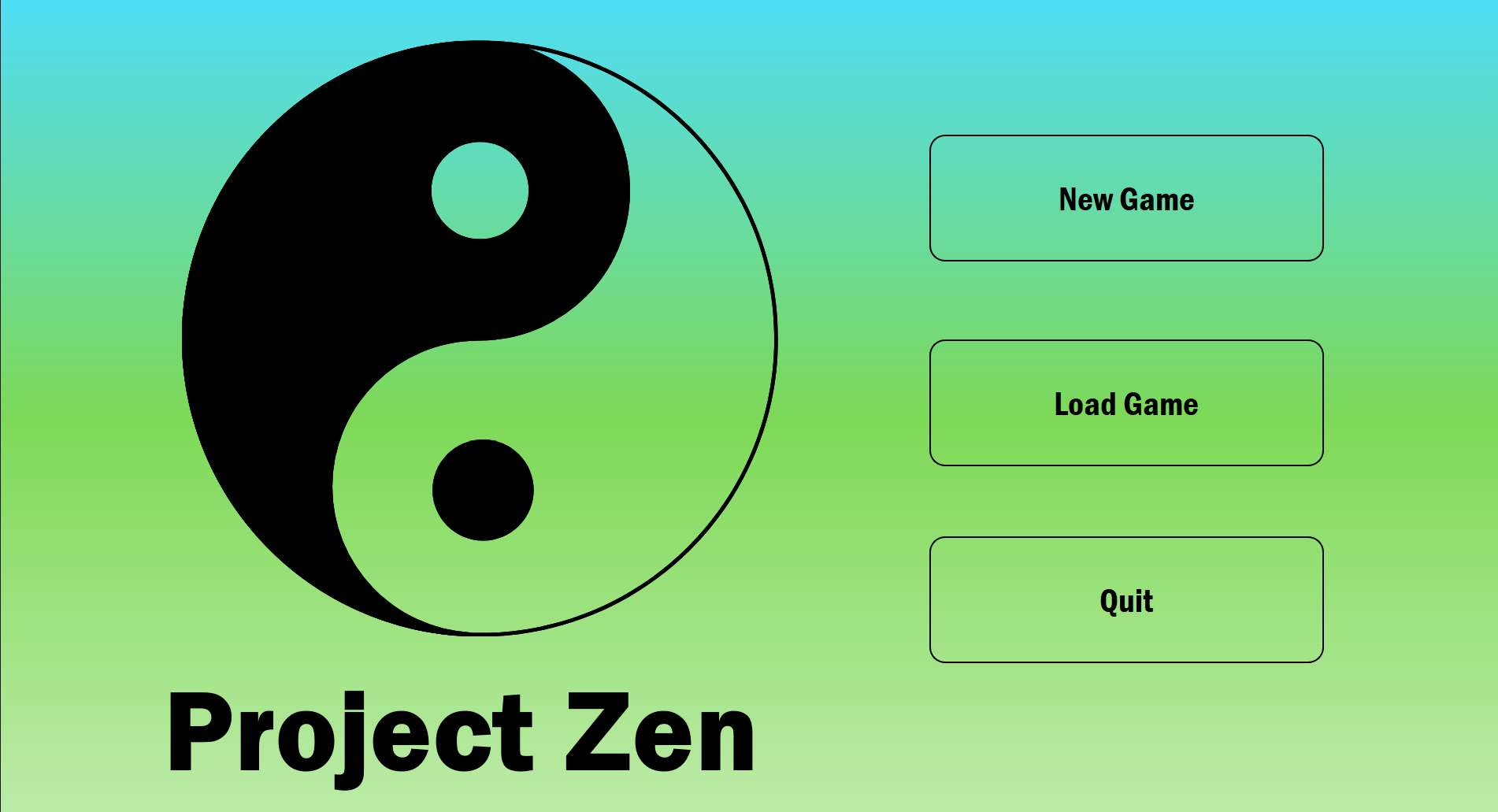
Project Zen seeks to be a calming, zen-like production of Sudoku where users may relax and enjoy the puzzle in a very original form. The target audience will encompass everyone who enjoys Sudoku, not being limited to anyone in particular.

It will feature a soothing, encouraging environment to help users enter a flow state whilst solving their puzzles. Standard game logic, puzzle generation, and checksums will be integrated into the program upon completion. Future prospects and endeavors will include new puzzle types, difficulty control, additional sounds and music, a time trial mode, and progress tracking via a metacurrency in game.

# **Scenario | Program Usage Description**

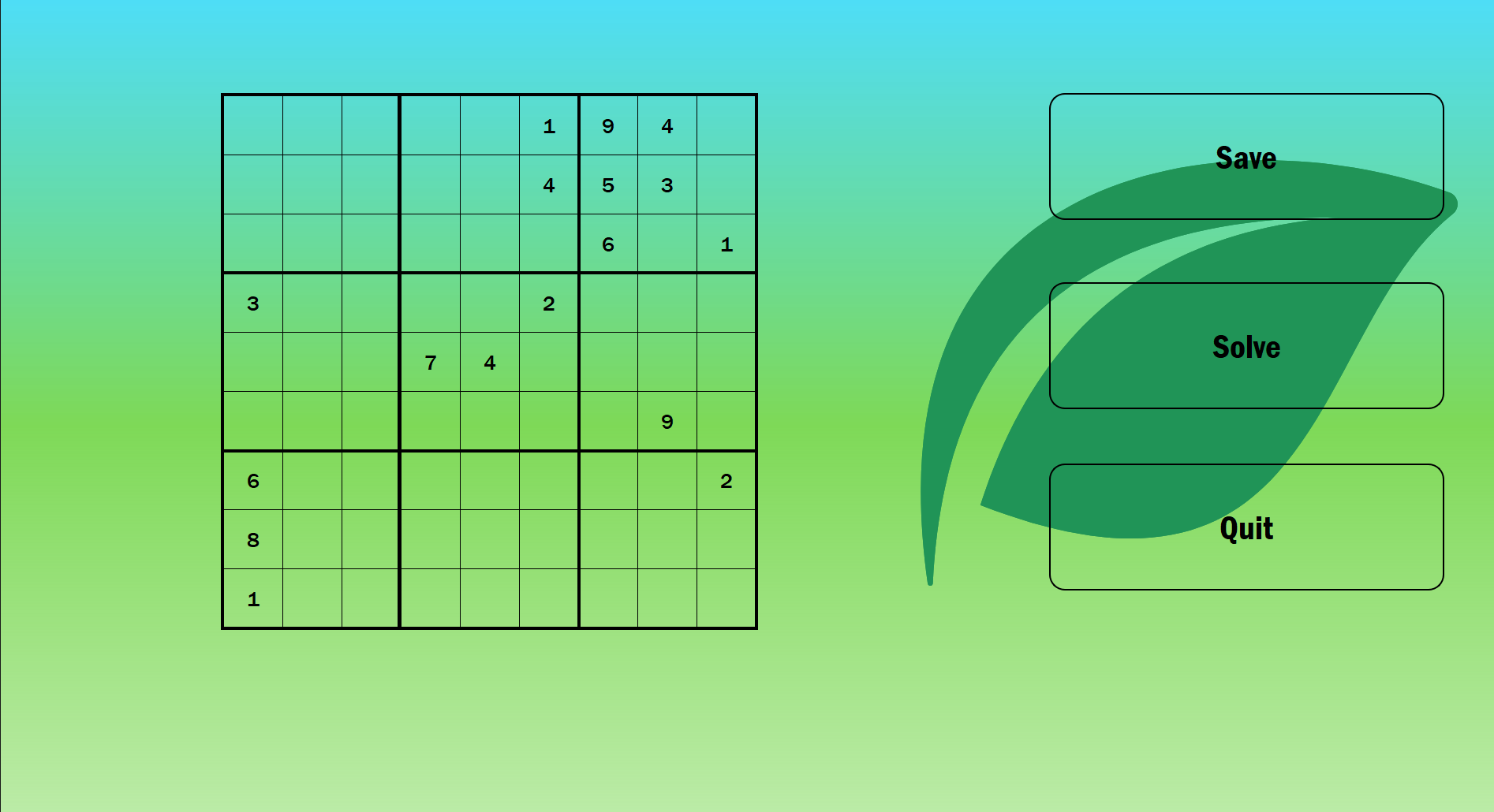
**Title:**

The title screen is meant to greet the player and either allow them to continue their game or create a new one. It is not meant to be simplistic and easy to follow.



# **Scenario | Program Usage Description**

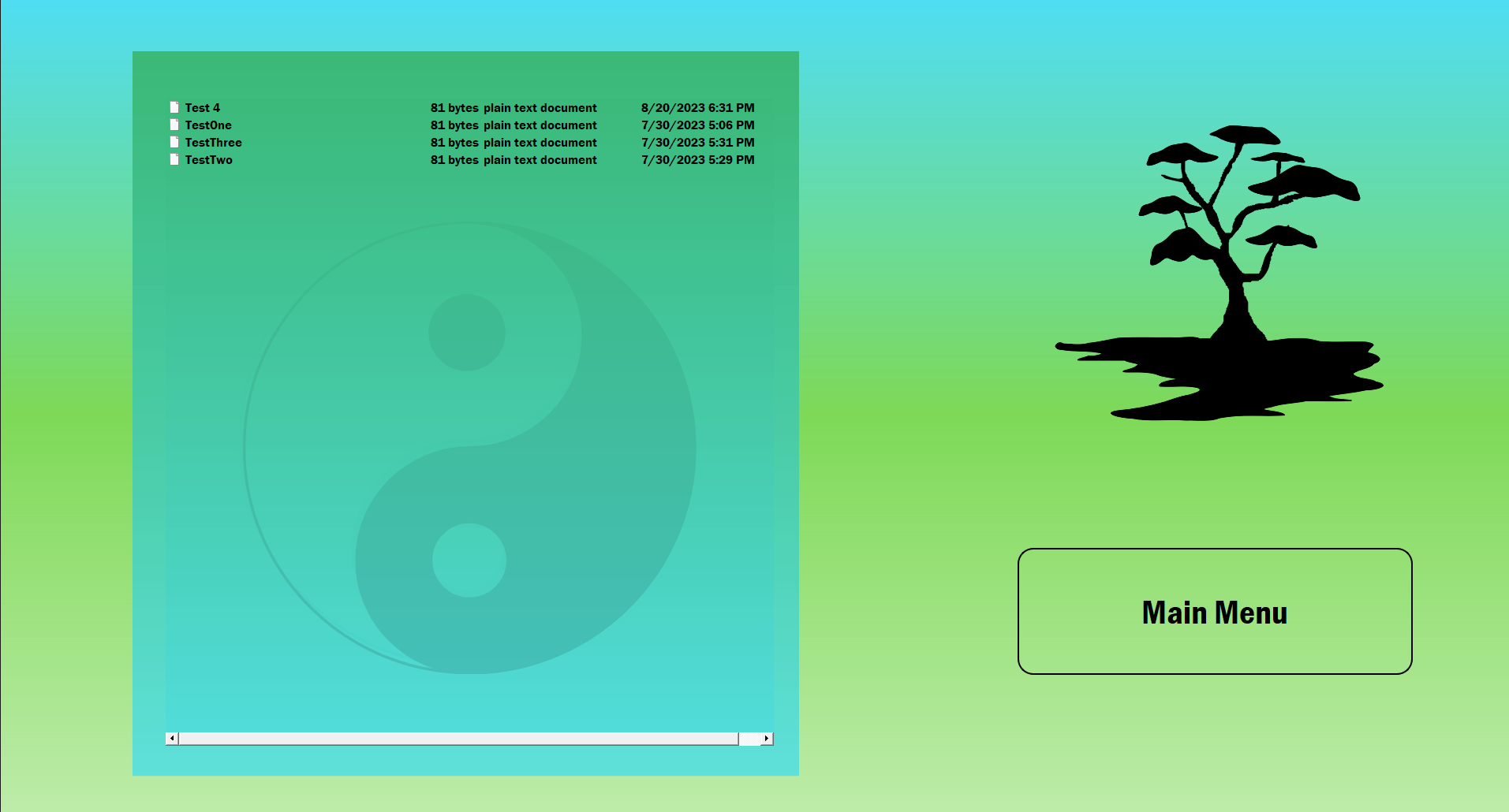
**Game:**

This is the mainstay of the program. It is used to play the provided puzzle and may also solve/quit the game via push buttons. The majority of the user's experience will be on this screen. It will interact with the user by informing them if their inputs are good/bad and if they successfully win the game. Users can save their progress, solve it, or quit the game from this screen. 

# **Scenario | Program Usage Description**

**Load:**

This screen allows the user to load a game via (Double Click) or return to the main menu.



# **Scenario | Program Usage Description**

**End Game or Victory Screen:**

This screen allows the user to start a new game, load an old one, or quit the application. The victory screen can only be access upon the successful completion or solving of a game.



# 

# 

# **Project Architecture**

# 

# 

# 

## **Source Code Structure**

The source code structure is relatively compact due to the nature of QT 6.5. All of the source code is under the ZenSudoku directory, while the supporting resource files are located in MyResources and SaveGame directories.

The additional two directories are build implementations provided by QT. They are required to functionally build the program from QT Creator Enterprise to obtain proper CMake files.

| **Code Directory** | |
| --- | --- |
| **Directory** | **Usage** |
| ZenSudoku | Source code for cpp files, CMakeList, and UI models |
| MyResources | QRC (Resource file) for all UI elements of UI Mainwindow |
| SaveGame | File containing test files for saves, additional saved games |
| build-ZenSudoku-Desktop\_Qt\_6\_5\_0\_MinGW\_64\_bit-Debug | Holds QT 6.5 version of build implementations for current project. Both are required to properly run project. |
| build-ZenSudoku-Imported\_Kit\_temporary-Debug | Build implementation that is imported from a previous version of QT. |
| *Yellow highlights indicate source code. Green highlights are source files.* | |

# 

# **Executables**

### ZenSudoku.exe - This will be the main executable for the entire program. All elements and files are contained within the application and do not require additional executables.

ZenSudoku.exe currently does not run due to not having been put through production mode in QT 6.5 and is missing required .dll files to execute properly.

# **Code Architecture**

The code architecture is broken down into four main components: UI interface details, Main Window, Sudoku Class, and Saved Games.

The mainwindow.ui holds roughly 1000 lines of code that cannot be edited by normal means in QT, but through the designer exclusively. This holds all the stylesheets, parameters, and orientations of all the UI objects being used by the MainWindow class.

The MainWindow class holds the majority of object logic, signals and connections, and data flow control between itself and the Sudoku class. It is the most significant in that it is used as an intermediary to communicate with the UI objects and the Sudoku class to get a functional game going. Generally, the objects signal the class that a change has been made, pushing the data to the MainWindow class and then executing said protocols dependent of the received data.

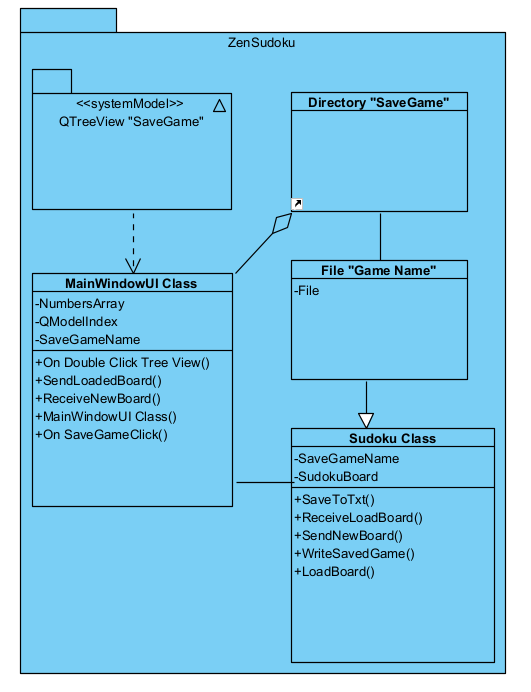
The Sudoku class is responsible for holding in real time the data being used for the UI objects residing in the MainWindow class. It is responsible for generating new game data, validating user inputs, saving game data to files, and sending signals related to the game state (Such as a victory).

The final component of the architecture is the database related to saving and loading games from the application. The Sudoku class saves games through a 2D array written to a text file, representing the Sudoku game board produced in the main window. It is saved to the “SaveGame” folder located under the common file build path. From there, it is retrieved by using a QTreeView model and rewritten back into a 2D array from the file.

## **Data Storage and Retrieval**

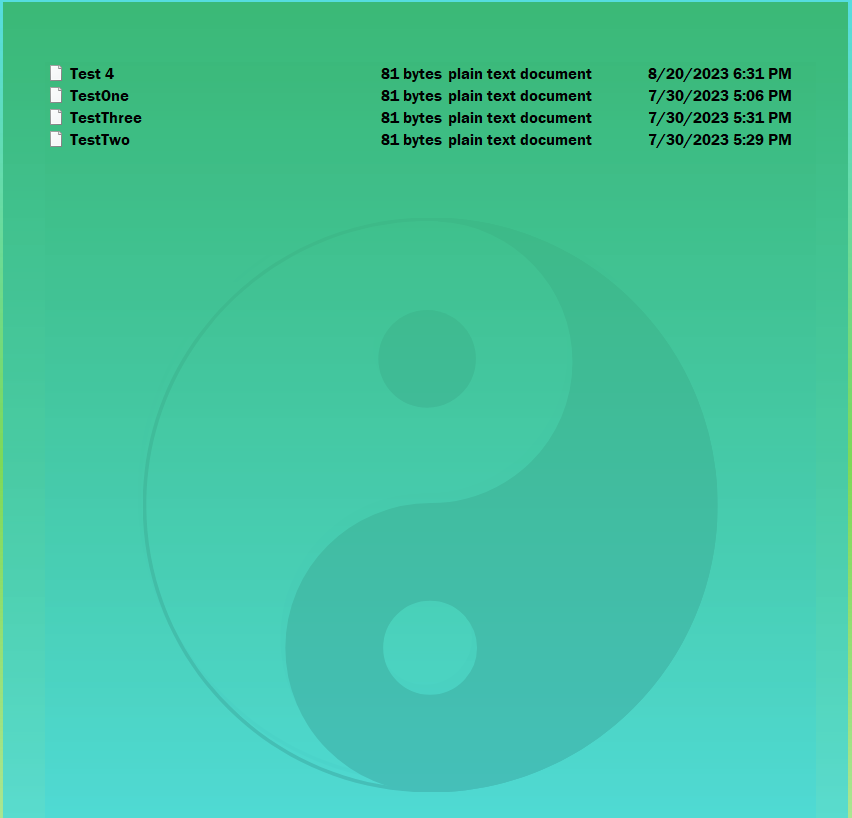
Data for the project during runtime is stored in the Sudoku class as the SudokuBoard item. When a save game is initiated it creates or writes to the file directory in “SaveGame” file.

When a game is loaded, the file selected is read from as a Qstring, then converted into a Qchar and then converted into an integer, placed in a 2D array. It is then sent to the Sudoku Class for modeling/preparation to be sent to the main game class board. QModelIndex is used to calculate the File path for the selected loaded files. Below shows the involved methods for the database.



## **Views, Stored Procedures and User Defined Functions**

The primary view used in this program is the QTreeView which is modeled off of the directory “SaveGame” located in the build files for ZenSudoku. It uses the system root data model index to locate the files. Loading is executed upon double clicking the file.



**Programming Language (C++, CMake)**

QT Creator 6.5 was used to implement this project with its interactive GUI. The base components of QT are written in C++, CMake, and are styled with CSS stylesheets. Components required to run the program successfully will include a modern version of CMake (3.2 or higher) and QT (6.5 or higher).

QT can be implemented or included as a package in a variety of IDE’s such as VS Code. It can also be used with other languages such as Python. This project exclusively uses CMake and C++ for the core logic and implementation. To run as a developer, ensure all files are downloaded and ensure the correct kits are installed (Compiler) to build the CMake.txt file into the IDE being used.

**Project Classes**

### **UI Elements and Data Flow | MainWindow.cpp**

This class contains all of the UI elements, UI triggers, data flow, user input validation, and sends data to the Sudoku Class. It is the main driver in identifying user interactions and takes that information fed into the Sudoku board (Q Widget Table) and converts/sends it to the Sudoku Class for validation.

It also utilizes models to get/receive data from the system directories upon loading/saving games. From there, it can fetch data in files and feed it to the Sudoku class.

Finally, this class holds all of the connections/signals/slots for the information/interactions between itself, the UI, and the Sudoku Class. This includes signaling for saving/loading games within the main UI and sending them to the Sudoku class.

### **Sudoku Game Class and Validation| SudokuClass.cpp**

This class is used to collect, generate, validate, and write the SudokuBoard, creating a game win condition/loop in the process. It takes user input from the MainWindow class and utilizes it to validate and update the SudokuBoard and the Q Table widget holding the information on the game screen.

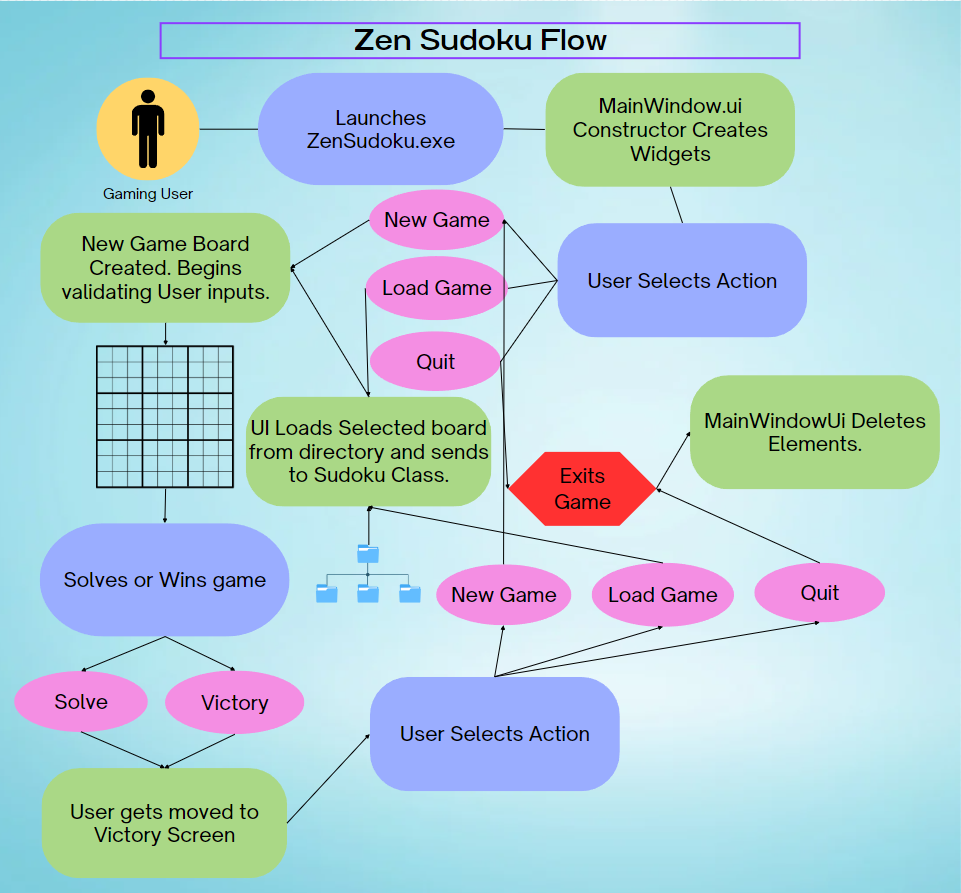
This class also writes text files to the system directory “SaveGame” and obtains a new board state from a loaded file.

### **MainWindowUI CSS | mainwindow.ui**

This is an auto generated class based on what the user changes in the QT designer. It cannot be modified by normal means, only by using the designer itself. It contains all of the stylesheet elements and modifications used in the program during development.

**Program Start and End Flow**

The program is designed to loop gameplay until the user decides that they want to quit the application by hitting any of the “Quit buttons. It begins by creating or loading a new game, then completing the puzzle, to finally go to the victory screen, starting anew.



**Summary**

Currently the game is about 97% complete and has just a few extra steps needed to push out to production/release of the game. It currently is not cross platform and requires a Windows Desktop to run. It handles all game functions nicely and could use some optimization, but is designed to be expanded upon in the future. Additional features can be implemented easily, as well as new and/or updated algorithms could be refactored to make for a better gaming experience. A step by step protocol for deployment can be found here: <https://doc.qt.io/qt-6/windows-deployment.html> for any developers are interested in completing the deployment. Deployment will use the Windows deployment tool to generate the appropriate .dll files to properly execute ZenSudoku.exe.

# **APPENDIX B (BUILD AND RELEASE PROCESS)**

**Development**

Update program features, enhancements, and refactoring through an IDE capable of supporting both CMake and QT 6.5. Once completed, a submission to GitHub for further QA may be possible if the project is set to public. Note that the program is in C++ and CMake, so a new build of the files will need to be configured upon downloading the repository.

**QA**

Testing will be done either locally by the owner or through a branch copy of the main program. Testing for stability, inaccuracy, and correct feature implementation can be done by simply building the program and checking for optimizations.

A key focus will be algorithm optimization and implementation of new/key features.

**UAT (**User acceptance test)

If the project is to be moved to a cross platform (Currently desktop only) environment and submitted for commercial means, UAT may be necessary to test the validity and functionality of the program. For now, it is not necessary

**Staging**

Once new features and enhancements are implemented, a branch staging package will be available to test/download. Users will be able to further test for bugs/improvements.

**Production**

Production may happen at any time. As of this moment Project Zen is a toy application, not meant for redistribution or sale beyond personal means. It will be uploaded/implemented through GitHub for download.

Distantly, there is a possibility to expand the application to the Android platform and be placed on market.

# **APPENDIX C (CLIENT INSTALLATION INSTRUCTIONS)**

1. Download the required files from the client repository
2. Double click on the installer to install the application to your desktop.
3. Launch ZenSudoku.exe

# **APPENDIX D (DEVELOPER SETUP INSTRUCTIONS)**

1. QT installation/files are required to build the project and modify the UI elements, specifically anything related to QObject. Ensure that QT 6.5 or higher is installed through your IDE to allow for proper use of the designer tools. Note that certain elements, such as the media player will not work with older versions of QT (6.2 or lower). It is highly recommended to directly use the QT Creator Enterprise edition to build the game.
2. CMake should also be installed in order to gain/fetch files associated with building the program. You can obtain this through your IDE or through the QT Creator maintenance tool.
3. Set your kits to the most recent compilers for CMake and build the project from the ‘projectname.txt’ Cmake file.. The system will likely automatically detect an imported build that should work for the proper kits. (See QT documentation on kits for CMake) to configure the project.