Table of Contents

[Background 5](#_Toc157376741)

[Problem 5](#_Toc157376742)

[Recognition 5](#_Toc157376743)

[Decomposition 6](#_Toc157376744)

[Client Interview 6](#_Toc157376745)

[Interview Questions 6](#_Toc157376746)

[Answers 7](#_Toc157376747)

[Analysis 7](#_Toc157376748)

[Research 7](#_Toc157376749)

[Existing solutions 7](#_Toc157376750)

[https://www.mazegenerator.net/ 7](#_Toc157376751)

[Parts I can use 8](#_Toc157376752)

[https://keesiemeijer.github.io/maze-generator/ 8](#_Toc157376753)

[Parts I can use 8](#_Toc157376754)

[Maze Generation 8](#_Toc157376755)

[Maze Solving 10](#_Toc157376756)

[Networking 11](#_Toc157376757)

[Requirements 11](#_Toc157376758)

[Modelling 13](#_Toc157376759)

[Implemented Packages 14](#_Toc157376760)

[gRPC Packages 14](#_Toc157376761)

[Json Parsing 14](#_Toc157376762)

[Database Management 14](#_Toc157376763)

[Data Visualization 14](#_Toc157376764)

[Entity-Relationship Diagram 15](#_Toc157376765)

[Data Structures 15](#_Toc157376766)

[Coordinate Class 15](#_Toc157376767)

[Maze Class 16](#_Toc157376768)

[Hierarchy Charts 17](#_Toc157376769)

[Key Algorithms 18](#_Toc157376770)

[Generation Overview 18](#_Toc157376771)

[Solving Overview 18](#_Toc157376772)

[Initializing the maze 18](#_Toc157376773)

[Depth-First Generation 19](#_Toc157376774)

[Recursive Implementation 19](#_Toc157376775)

[Stack implementation 20](#_Toc157376776)

[Wilson’s Generation Algorithm 21](#_Toc157376777)

[Depth First Search Solving Algorithm 21](#_Toc157376779)

[Maze Routing Solving Algorithm 22](#_Toc157376780)

[Security 22](#_Toc157376781)

[Data hiding 23](#_Toc157376782)

[Password Hashing 23](#_Toc157376783)

[Logging in 23](#_Toc157376784)

[The Salt Field 23](#_Toc157376785)

[User Interface 24](#_Toc157376786)

[Program Flow and User Interface 25](#_Toc157376787)

[Project Structure 28](#_Toc157376788)

[Techniques Used 29](#_Toc157376789)

[Network Exception Handling 29](#_Toc157376790)

[Multithreading Exceptions 29](#_Toc157376791)

[Global Variables 29](#_Toc157376792)

[Server Protocols 29](#_Toc157376793)

[Proto Files 29](#_Toc157376794)

[Project File References 30](#_Toc157376795)

[Data structures 30](#_Toc157376796)

[List 30](#_Toc157376797)

[Queue 30](#_Toc157376798)

[Algorithms 30](#_Toc157376799)

[Database Management 30](#_Toc157376800)

[General information 30](#_Toc157376801)

[Table creation 30](#_Toc157376802)

[Triggers 31](#_Toc157376803)

[Maze Generation 31](#_Toc157376804)

[Recursive Backtrack Algorithm 31](#_Toc157376805)

[Growing Tree Algorithm 32](#_Toc157376806)

[Wilson’s Algorithm 32](#_Toc157376807)

[Maze Solving 33](#_Toc157376808)

[Depth-First Search 33](#_Toc157376809)

[Maze-Routing Algorithm 33](#_Toc157376810)

[Breadth-First Search 33](#_Toc157376811)

[Logging in and Registering 34](#_Toc157376812)

[Client-Server Interactions 35](#_Toc157376813)

[Other Algorithms 35](#_Toc157376814)

[Connectivity Status 35](#_Toc157376815)

[Setting the Size of the MazeDisplay Form 35](#_Toc157376816)

[Displaying the maze in the Table Layout Panel 36](#_Toc157376817)

[Requirements Met 36](#_Toc157376818)

[Models Used 41](#_Toc157376819)

[Algorithms Used 42](#_Toc157376820)

[Test Plan 43](#_Toc157376821)

[Screenshots 46](#_Toc157376822)

[Fixes 58](#_Toc157376823)

[My Thoughts 59](#_Toc157376824)

[Login System 59](#_Toc157376825)

[Network Functionality 59](#_Toc157376826)

[Algorithms 59](#_Toc157376827)

[User Interface 60](#_Toc157376828)

[Robustness of System 60](#_Toc157376829)

[Client’s Thoughts 60](#_Toc157376830)

[Description of client’s interaction 60](#_Toc157376831)

[Evaluation of their comments 61](#_Toc157376832)

[Features Suggested 61](#_Toc157376833)

[Discussion of Features 61](#_Toc157376834)

[Shared code 63](#_Toc157376835)

[Protos 63](#_Toc157376836)

[BuildMaze.proto 63](#_Toc157376837)

[CheckIfUserExists.proto 63](#_Toc157376838)

[DeleteMaze.proto 63](#_Toc157376839)

[GetMazes.proto 64](#_Toc157376840)

[GetStats.proto 64](#_Toc157376841)

[greet.proto 65](#_Toc157376842)

[HandleGlobalStats.proto 65](#_Toc157376843)

[HandleUserStats.proto 66](#_Toc157376844)

[LoadMaze.proto 66](#_Toc157376845)

[Login.proto 67](#_Toc157376846)

[Register.proto 67](#_Toc157376847)

[SaveMaze.proto 67](#_Toc157376848)

[SolveMaze.proto 68](#_Toc157376849)

[Client Code 68](#_Toc157376850)

[Classes 68](#_Toc157376851)

[Coordinate.cs 68](#_Toc157376852)

[Globals.cs 69](#_Toc157376853)

[Growing Tree Generation.cs 69](#_Toc157376854)

[Maze.cs 70](#_Toc157376855)

[Recursive Backtrack Generation.cs 71](#_Toc157376856)

[Wilsons Generation.cs 71](#_Toc157376857)

[Program.cs 72](#_Toc157376858)

[Form Code 72](#_Toc157376859)

[MazeDisplay.cs 72](#_Toc157376860)

[MazeLogin.cs 77](#_Toc157376861)

[MazeParameter.cs 78](#_Toc157376862)

[MazeRegister.cs 82](#_Toc157376863)

[Server Code 84](#_Toc157376864)

[Service Definitions 84](#_Toc157376865)

[CheckerIfUserExistsService.cs 84](#_Toc157376866)

[DeleteMazeService 84](#_Toc157376867)

[GetMazesService.cs 85](#_Toc157376868)

[GetStatsService.cs 85](#_Toc157376869)

[GlobalStatHandlerService.cs 87](#_Toc157376870)

[GreeterService.cs 89](#_Toc157376871)

[HandleUserStatsService.cs 89](#_Toc157376872)

[LoadMazeService.cs 90](#_Toc157376873)

[LoginService.cs 91](#_Toc157376874)

[MazeBuilderService.cs 92](#_Toc157376875)

[MazeSolverService.cs 92](#_Toc157376876)

[RegisterService.cs 93](#_Toc157376877)

[SaveMazeService.cs 93](#_Toc157376878)

[Algorithm Classes 94](#_Toc157376879)

[Breadth First Solve.cs 94](#_Toc157376880)

[Coordinate.cs 95](#_Toc157376881)

[DepthFirstSolve.cs 96](#_Toc157376882)

[Growing Tree Generation.cs 96](#_Toc157376883)

[Maze Routing Solve.cs 97](#_Toc157376884)

[Maze.cs 99](#_Toc157376885)

[Program.cs 100](#_Toc157376886)

[Recursive Backtrack Generation.cs 104](#_Toc157376887)

[Solving.cs 105](#_Toc157376888)

[Wilsons Generation.cs 106](#_Toc157376889)

[Data Structures 108](#_Toc157376890)

[List.cs 108](#_Toc157376891)

[Queue.cs 109](#_Toc157376892)

[Additional Files 110](#_Toc157376893)

[Client project file 110](#_Toc157376894)

[Server project file 111](#_Toc157376895)

Analysis

# Background

Currently, there is no lightweight app which allows one to create random mazes of distinctive styles with ease, whilst also providing access to past creations. My client, Greg, develops a puzzle app which allows people to request puzzles from a network and solve them on their device, or request a solution from the network. He has asked me to create an app for users which allows them to request and solve mazes, and a server which can run the generation and solving algorithms, to extend his own puzzle app.

Intrinsically, the apps need to run on computers with access to a network connection with an open port (a public point over which data relevant to the app can be transferred). The hardware requirements of the computers vary depending on their job. The user’s computer should need little processing power as it is just sending and receiving data and allowing basic interaction with the maze. The computer hosting the program which generates and solves mazes may need to have higher-end hardware to facilitate the algorithms and networking.

Henceforth, the app which the user runs will be referred to as the client, and the app generating and solving mazes the server.

# Problem

## Recognition

There are three main problems to solve in the solution:

* Finding a way for a client on one system to interact with the server on another.
* Generating mazes via different algorithms.
* Solving the mazes by the shortest path.

Of these problems, by far the hardest is the first. Once the issue of interaction across a network is solved, adding new generation algorithms, and solving algorithms is simply a case of adding to the pre-existing network structure.

## Decomposition

The networking issue can be split into smaller steps. A first approach at this would be:

1. The server is opened across a port on the network.
2. The client is opened, then looks for and connects to the server.
3. The server assigns the client an ID, or uses an account ID, dependant on my client’s requirements.
4. The client chooses maze settings and creates a JSON/XML file out of them and sends it to the server.
5. The server adds this to a queue.
6. The server generates and perhaps solves the first item in the queue and sends it to the respective client where they can view and interact with it.

Steps 5 and 6 would be repeated while there are requests in the queue.

I will cover maze solving and generation in the Research section of the analysis, as I plan to implement several different algorithms for both.

# Client Interview

## Interview Questions

I will outline the questions I will ask Greg and summarise his answers to each. The aim of the questions is to get to know more about the system and the key requirements he would be looking for in my solution.

1. **How big is your userbase?**  
   This question will inform how robust the network needs to be. I will have to stress test the network if the userbase is large.
2. **Does your system currently accommodate touch screen users?**  
   This question informs whether I need on-screen buttons or whether I can simply use keyboard controls for solving the mazes manually.
3. **How many maze generation algorithms are you looking for in the initial solution at a minimum?**  
   This question establishes a minimum number of algorithms, so I will have to prioritise adding a certain number of algorithms dependant on Greg’s answer.
4. **Is a login system necessary?**  
   This question informs whether a system which attributes a record in a database to a user via a username and password system should be implemented.
5. **Should there be a system which tracks statistics?**  
   This question determines the need for a system which could track some statistics on the server (mazes solved automatically, manually, types of mazes generated, etc.) is necessary to implement or can be left as an additional feature.
6. **What should happen if the user is unable to solve the maze?**  
   This question determines whether the solve option should only be available on a generation request, or whether it should be available post-generation. Either option will make a significant impact on the program which handles solve requests.

## Answers

1. Roughly 150 people.
2. Yes, around 40% of the users are on touch screen devices.
3. 3-4 would be nice, to demonstrate what it can do.
4. The current app stores puzzles locally, so a login system would be appreciated but not absolutely needed.
5. Again, there isn’t something like that in place already, but it would be nice to have.
6. They should have the option to allow the software to solve it.

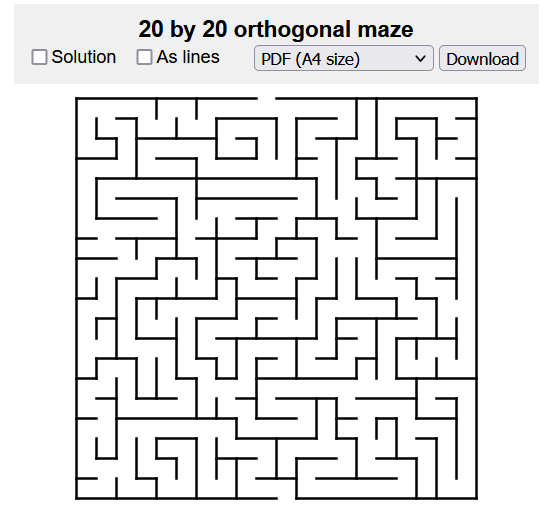
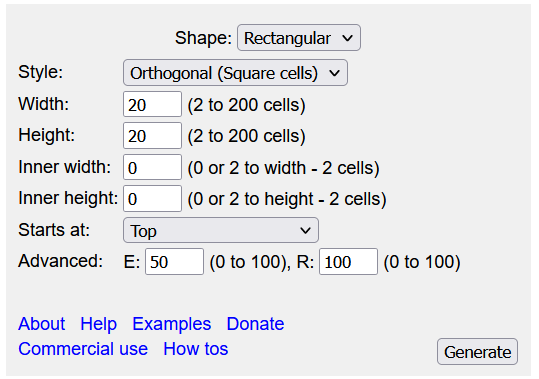
## Analysis

150 people at most would be connected to the one server. Server stress will likely not be an issue unless the algorithms I implement are extremely slow, so I may need to take time complexity into consideration when choosing what to program. There is a considerable proportion of users who may not use a physical keyboard with the app: I will need to add onscreen buttons to accommodate them. If I also implement keyboard controls, I may need to add an onscreen indicator that the keyboard controls are also available. I may be able to put this into a small help menu along with other information on using the app, but this is not a key function of the app so the implementation should stay simple. Greg says that a login system is not a necessity for the solution, so I should only implement this after adding all other key features. A login system, if it is added, would be useful for tracking user-specific stats and server-side saving. Similarly, a stats system should be implemented last alongside the login system. Windows forms has a designer I could use to add graphs, which I could use to pull data from a database into a charted format. Finally, the need for the user to always have the option to solve the maze means the server always needs to be listening for solve requests as well as generation requests, and the code for handling a solve will likely look different.

# Research

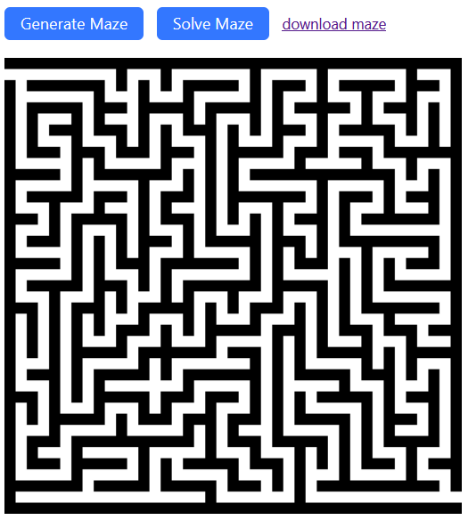
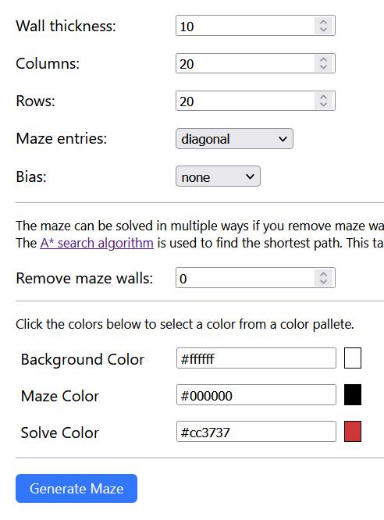
## Existing solutions

### https://www.mazegenerator.net/

  
This website is a free online maze generation platform, available on any web browser. The program can generate mazes in different shapes and sizes, as well as using a structural grid rather than an array-based grid to allow mazes where the cells are not squares. The user interface is simple and uses drop-down menus and text boxes to allow user input for maze parameters. However, there is no clear explanation of the advanced parameters E and R, nor can the user solve the maze on their browser. There is also no way to change the algorithm used.

### Parts I can use

I and my client like the simplicity of the interface and think the use of drop-down boxes to specify qualitative parameters is good. We like the way the maze is displayed as the walls are lines rather than thick cells. The generator also measures the maze height and width in cell number rather than array length, which allows for any size of maze instead of only odd numbers. This is something I will implement in my solution.

https://keesiemeijer.github.io/maze-generator/  


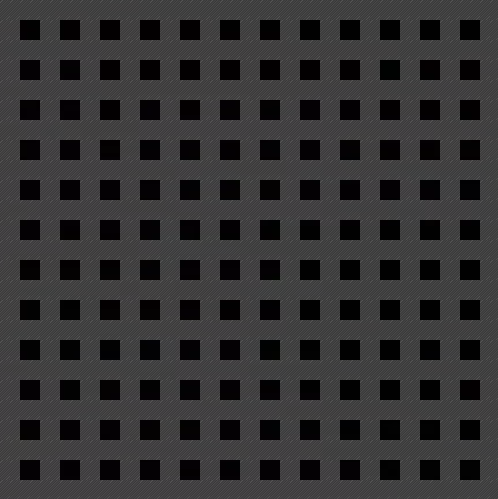
This website is another free online maze generation and solving platform. The parameter selection gives more meaningful names, so you can tell what each mean without a help page. The option for colour is given, but we think that for the initial solution, nonessential graphics are optional. We agreed that the option to remove walls is a straightforward way to generate a labyrinth style maze without a dedicated algorithm. Comparing this with the previous, the maze walls being thick here does not look bad, so we agreed it would be better to not draw walls as lines if it made the program better.

### Parts I can use

I like the option to download mazes, but based on my interview with Greg, I think that this option would be better as an upload to server option, tying into accounts. If I am unable to implement a login in time, then downloading the maze would be a viable alternative. We also think that the use of A\* Pathfinding for solving the mazes is good, so it would be good to include it, or some other shortest path algorithm.

## Maze Generation

A maze on a computer has two ways of being represented. One is to have a grid of cells, which can either be walls or passages. Another is to have a grid of cells which each store information about the state of each of their four walls. In this system I have decided to use the former method, as the algorithms are already complex, so I would like to keep the other aspects simple. A maze-generation algorithm involves first creating a rectangle of walls, then breaking those walls to create a grid of open passages called “cells,” in a shape resembling a waffle.



A typical generation algorithm then has a “constructor” roam the cells, breaking walls between them until a maze is formed. The way the constructor moves around the cells differentiates the algorithms from each other and determines the shape of the maze. For example, certain algorithms will produce mazes with longer/shorter hallways, or more hallways may be horizontal/vertical. I have researched several generation algorithms which I may include in the solution. Note that all these algorithms produce “perfect” mazes: mazes where there is only 1 path between 2 arbitrarily selected points. I should also implement a generator for a “labyrinth” style maze, where 2 points may be connected by multiple paths of varying length.

* Randomized Depth-First Search Algorithm  
  Aside from the binary tree algorithm, this is perhaps the simplest way to generate a maze computationally. Starting from the entrance, the computer selects a random unvisited cell, moves the constructor into it, and destroys the wall between the two cells. If the passage is a dead end – there are no unvisited cells surrounding the constructor’s cell, the constructor backtracks until it finds an unvisited cell. This process is repeated until there are no unvisited cells left in the maze. This algorithm, while simple, tends to have a bias for long passages and little branches.  
  This algorithm can be implemented in two ways:
  + Recursive  
    A new depth-first search is started on each unvisited cell. When a dead end is reached, the depth-first search ceases, and control returns to the previous search.
  + Stack  
    The recursive algorithm, whilst efficient, can cause stack overflow issues on older computers. A stack can instead be used to create larger mazes. The route the constructor takes is stored as cell coordinates and is appended to a stack. When a dead end is reached, the constructor backtracks down the stack, removing the top coordinate each time.
* Kruskal’s Algorithm  
  This algorithm uses a list of all walls and sets each containing one cell to produce a maze with a bias for many branches and short passages. It randomly selects a wall, and if the cells divided by it are in distinct sets, it destroys the wall and joins the two cells, creating one larger set. The process is repeated until only one set remains. The mazes produced by this algorithm tend to be easier to solve.
* Prim’s Algorithm  
  This algorithm would produce mazes identical to those of Kruskal’s algorithm, but Prim’s algorithm effectively introduces weighting to walls, affecting the randomization and creating a stylistic difference. This is done by not using a list of all walls. Instead, only the walls visible to the main chuck of the maze are added to the wall list.
* Wilson’s Algorithm  
  This algorithm creates an unbiased, uniform maze, which tends to be harder to solve. This is done by using a method called a loop-erased random walk. The algorithm starts at the starting cell and starts walking randomly until it reaches the end cell. However, if at any point it runs into its own path, it will backtrack, removing any destruction of walls along the way. Once this initial path has been formed, it picks another random start cell and performs another random walk until it reaches the maze. This is repeated until all cells are visited.

## Maze Solving

A typical maze-solving algorithm works similarly to the generation algorithm, in that it has a “solver” roam the maze and take a path at each crossroad, depending on a set of rules. If the path is a dead end, it will backtrack and take a different route. I have researched some solving/pathfinding algorithms which I could implement.

* Random Mouse Algorithm  
  The most primitive maze-solving algorithm, by far. This algorithm simply proceeds down a passage until it reaches a crossroad, before making a random decision about which passage it should go down. This algorithm is simple to implement and will always find a solution, but can be extremely slow, and this is noticeable on large mazes.
* Trémaux’s Algorithm  
  This is an efficient algorithm, which, while it is guaranteed to solve a maze, may not always find the shortest route. The algorithm works by marking the entrances to the passages in the maze. When a passage is entered, the entrance is marked. The direction to take at a crossroad is decided by the marks of the passage entrances. If none of the new passage entrances are marked, a random one is picked to go down. If any of the new entrances are marked, backtrack down the entrance that was just passed through, unless it is marked twice. Going back through the entrance in this way will mark it again. This rule will always apply at a dead end. Otherwise, pick the entrance with the fewest marks. When the exit is located, the marked passages will form a path back to the entrance, but the random nature of the algorithm means it may not always be the shortest. This algorithm is fast, not too complex, and can solve labyrinth mazes.

* Maze-Routing Algorithm  
  This heuristic algorithm uses Manhattan Distance (The absolute difference between two point’s Cartesian coordinates, henceforth MD), and the idea that on a grid, this distance measurement will change by exactly 1 when moving to a neighbouring cell. This algorithm can find paths between not only the entrance and exit of the maze, but even just any two points, with little information. It can also detect whether the maze is unsolvable. However, it may not find the shortest path. The algorithm works by first deciding if there is a productive path, based on comparing the current MD with the MD of a path. If the path is productive, take it. Otherwise, the algorithm will turn either left or right at a given crossroad, by determining which direction would best fit onto an imaginary line drawn between the start and exit cells. This process is repeated until the current and exit coordinates are the same.

* A\* Pathfinding Algorithm  
  A\* is a complex algorithm for finding the shortest path between the entrance and exit. Put simply, it creates a tree, with branches representing paths through the maze, and extends each branch by 1 cell until the exit is reached. The algorithm uses complex maths to determine which path to extend on each loop, using MD and a cost system to select nodes in a priority queue. This algorithm is complicated in design but will always find the shortest path, even in a labyrinth maze.

## Networking

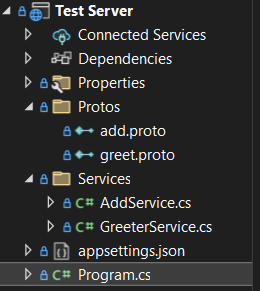
There are three ways of creating a server in a way that works in my solution. One is using a server object, and a console app which references this object, listening for requests to use its functions across an open network port. Another is to use sockets to handle requests rather than a server object, which compresses the server into a single project but may be more complex. I could also use gRPC’s C# implementation, which operates across 2 separate projects (a client and server) using shared protocols defined in proto3-based files. This would retain the complexity inherent to a network environment whilst not introducing deprecated features into the project.

# Requirements

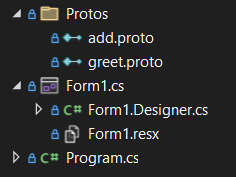
1. The Windows Forms client app which the user will interact with.
   1. The ability for the user to customize their request for a maze with several parameters.
      1. The ability to change the maze generation algorithm used to make the maze.
         1. An implementation of an algorithm to make perfect mazes with a bias for long corridors and low branching.
         2. An implementation of an algorithm to make perfect mazes with a bias for short corridors and high branching.
         3. An implementation of an algorithm to make unbiased perfect mazes with a uniform distribution of branches and corridor lengths (a uniform spanning grid).
         4. (Optional) Add more algorithms by which the user can select and generate mazes.
      2. The ability for the user to select the width and height of the maze they generate, measured in cells.
         1. A width parameter which changes the horizontal cell size of the maze.
         2. A height parameter which changes the vertical cell size of the maze.
      3. The option to remove a user-specified number of walls in the maze, allowing for a performant solution of generating uniquely styled labyrinth mazes.
   2. The ability for the user to allow the server to solve the maze after it has been generated.
      1. An implementation of a simple algorithm for performant solves.
      2. An implementation of the Maze-Routing algorithm allowing for shorter-path solves.
      3. An implementation of any shortest path algorithm, such as A\* or the First-Breadth Search algorithm allowing the user to find the shortest possible path in a labyrinth maze.
   3. The ability for the user to request a maze to be generated by the server with their selected parameters, and have it be returned to their client on completion.
   4. A button on the client available post-generation allowing the user to request a solve of their maze.
   5. A screen on the client which displays the current maze after receiving it from the server.
   6. The ability for the user to navigate the displayed maze pre algorithmic solve and attempt to solve it themselves.
      1. Navigation controls mapped to buttons on the keyboard.
      2. Buttons on the client interface which allow navigation.
   7. The ability for a user to save their generated maze.
      1. If the login system is completed, this should save the maze to the server database. Otherwise, this feature will be a local download.
   8. (Optional) A section of the client which uses the Windows Forms Graphs library to display a user-specified graph of information stored in the server database.
      1. This aspect should allow the user to see personal stats, as well as global stats if the login system is completed.
2. The Console App server which will handle the backend operations of the system, such as maze generation, solving, and database handling.
   1. A database structure storing statistics and login information, as well as storing saved mazes.
   2. A console interface which logs requests.
   3. Services to facilitate logging in.
   4. Services to facilitate registering a new user.
   5. Services to facilitate database management.
      1. Saving/loading mazes on request.
      2. Organize global and user times.
   6. Services to facilitate getting data from the server database.
      1. Fetching user stats.
      2. Fetching global stats.
      3. Fetching saved mazes.

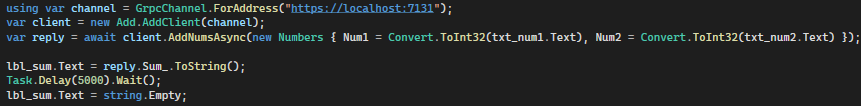
# Modelling

As a prototype of a potential solution, I have created a small client-server network via gRPC to see how the project may be structured.

The structure of the overall solution is split into 2 projects when using gRPC. The server contains a Protos file, containing files which document the format in which data will be transferred for a certain service. These are referenced in an item group in the project file so the objects defined can be generated for use in services. It also contains a Services folder, which contains programs defining the logic of how that request data will be handled and sent back to the respective client. These services are mounted to the server in the main Program.cs file.

I have created a simple Windows Forms test client to test if the test server will be able to contact a graphical client.

The client also contains a Protos file with copies of the server protocols. These are also referenced in an item group like in the server. The client contains no definition of the services however, since it will only send data to the server so the server app can perform the service. This keeps the client lightweight.



This is a call to the server, requesting its “AddNums” service. We establish a connection to the server across a predefined port (here, 7131), and create a clientside request. We pass the server the “Numbers” request object defined in the protocol and await its reply. We then display the data in the reply object to the client’s form.

The solution could use a similar principle, albeit with much more complexity, with many more services and calls.

Design

# Implemented Packages

## gRPC Packages

gRPC is built across 4 packages:

* Grpc.AspNetCore is for the server.
* Google.Protobuf is for the client.
* Grpc.Net.Client is for the client.
* Grpc.Tools is for the client.

I have chosen to build the network using gRPC. This significantly changes the style of the solution. The system will be implemented via 2 separate projects: a client and server. A protos folder containing protocol files will be shared between these projects so the client and server know which services are available, and what the proper request/response objects for these calls are. The server will additionally have a folder defining classes for each of its services, with each protocol file requiring its own class. These classes are mapped to the server when it is run.

## Json Parsing

I will implement Newtonsoft.Json in my project to facilitate the transfer of objects across the network. In gRPC, data is wrapped in a collection defined by a protocol file to be sent across the network. However, these protocol files only support simple data structures, so to share Maze-derivative objects with the server I will have to parse them into a Json format first. I can use the JsonConvert class from this package to convert these. However, I will have to implement rigorous error handling, particularly for deserializing Json objects, as the object may have null components if inappropriate access modifiers are used for properties.

In addition, Json parsing in this way should allow me to make the client even more lightweight: both the client and server will use the Maze, Coordinate, and all child classes, but only the server will need to call their functions, which are purely to build the maze. Therefore, I should not need to provide the client with anything but the class’ properties to build the Json object.

## Database Management

I plan to use the System.Data.SQLite package to manage the databases used in the system. The database file itself can be stored on the server only; if the user needs data from a table, it can be accessed in a controlled way via a network request from their client. Additionally, this package allows the use of a tag system when creating an SQL command, helping prevent SQL injection attacks when using parameterized SQL with user input, such as storing login information.

## Data Visualization

Since the System.Windows.Forms.DataVisualization chart system is deprecated in the version of C# I am using, I shall implement this package[[1]](#footnote-1) to restore the feature so I can display global stats to the user.

# Entity-Relationship Diagram

|  |
| --- |
|  |

This is the structure I have chosen for my data management. When the user creates an account, a record in the User table will be created for them, with a unique UserID. See the security section for comments on password storage and the Salt field. The trigger CreateUserStats will be set up so that a record in the UserStats table will be initialized for that specific user, by referencing their UserID. When the user saves a maze to the server, a new record in the Mazes table is created, with a new MazeID to allow mazes with the same name. It will also reference the UserID of the creator of that maze, allowing only the user to see their own mazes. The GlobalStats table does not reference any other tables as it has no key: It serves as a permanent data store for statistics across the userbase.

# Data Structures

## Coordinate Class

|  |
| --- |
| Coordinate |
| -xPos : int -yPos : int  +XPos : int  +YPos : int  -visited : bool  +Visited : bool |
| +Coordinate(int : xPos, int : yPos) +GetCartesianCoordinates(Maze : maze) : Tuple<int, int>  +GetManhattanDistance(endPoint : Coordinate) : float |

The Coordinate class will be used whenever coordinates are necessary within the program. Most algorithms will utilise a stack of coordinates, adding new coordinates to the stack and popping the top value to backtrack.

The Coordinate class is useful since it allows more flexibility than just using tuples or similar data structures to store coordinates. A tuple can only store the two integers for example, but the coordinate class allows the code to get information from the object, which may simplify the more complex solving algorithms.

The private properties xPos and yPos will store the positions of the cell in a 2D array – this means that the Y coordinate will start at the top of the 2d grid and increase as you move down. I am therefore adding the GetCartesianCoordinates method to obtain traditional coordinates to a grid cell, which some solving algorithms will require. It needs to take an argument which contains the current maze in order to compute this. The public properties XPos and YPos will act as read-only accessors to their private counterparts using the get{} keyword in C#. This will allow coordinates to be initialised at creation, but not be accidentally changed afterward. The visited property is useful for some generation algorithms which need to remember if a certain cell has been visited. The method GetManhattanDistance will return a floating-point value equal to the Manhattan Distance between the coordinates of the Coordinate object and Coordinate argument, which will be useful in more complicated algorithms such as Maze-Routing or A\* pathfinding.

## Maze Class

|  |
| --- |
| *Maze* |
| -mazeActualWidth : int  +MazeActualWidth : int -mazeActualHeight : int +MazeActualHeight : int  -mazeCellWidth : int  +MazeCellWidth : int  -mazeCellHeight : int  +MazeCellHeight : int #mazeWalls : bool[,]  #mazeCoordinates : Coordinate[,] |
| {abstract} +initMaze() : void {abstract} +buildMaze() : void  {virtual} #cellVisited(cellPos : Coordinate) : bool |

The abstract class Maze will be the parent class of all maze types added and should streamline adding new maze types to the program. It provides essential properties for mazes and forces its subclasses to implement buildMaze.

The first 8 properties will hold the integers representing width and height, measured in array dimensions and number of cells, privately with read-only public access. The 2D array mazeWalls will hold the structure of the maze with booleans representing whether a cell contains a wall. I am using booleans since using char or string would make the maze files bigger and may affect network speed. Also, the maze will be displayed in a Windows Forms GUI so using letters would not add any extra utility to the program. I may also implement the 2D array mazeCoordinates as an array of Coordinate objects, as some algorithms may need the ability to always track the position of cells. The abstract method initMaze will be used to initialise the mazeWalls array with the pre-built maze structure, whilst buildMaze will implement the generation algorithm. The method cellVisited will query if a cell has been visited - with most algorithms this means if it still has 4 walls, but I am making it virtual in case any algorithms need their own definition.

## Hierarchy Charts

|  |
| --- |
|  |
| This is how the abstract class Maze would be implemented. Since the mazes I implement will inherit from Maze, they are forced to provide function bodies for the 4 abstract functions, and therefore all mazes must provide standardized output. Almost every algorithm in the entire solution implements Coordinate in some way, which I have shown here. Each class that defines an object which will be sent over the network must |
|  |
| Similarly, each SolvingAlgorithm is forced to implement the SolveMaze function by the abstract class SolvingAlgorithm. |

# Key Algorithms

## Generation Overview

The Maze abstract class means that, whilst the generation methods will be different, they will all have the same output: a 2D array of Booleans which represent the maze. Hence, each algorithm will be at its simplest level some way of manipulating the Coordinate system and the MazeWalls 2D array structure to create a maze. This standardised form is good since it should allow new algorithms to be easily implemented and added.

## Solving Overview

The abstract class Solving requires each solving algorithm to return 1 standardised output: a List of Coordinates which represent the solution to the maze. Therefore, in much the same way as generation, each solving algorithm will at its core be some way of interpreting and manipulating the MazeWalls structure and Visited Boolean of indices of MazeCoordinates respectively, in order to produce this List. The standard form should allow me to write less code to visualize the solution on the user interface.

## Initializing the maze

Whilst some algorithms may start with a grid full of walls, most maze generation algorithms will start with a waffle grid. This is the pseudocode for generating this waffle.

|  |
| --- |
|  |

First, the actual dimensions of the maze are calculated from the user input of cell dimensions. The mazeStructure array is then initialized with these sizes. Then, each cell is iterated through and is defined as either a wall or passage, with False representing passages and vice versa.

## Depth-First Generation

### Recursive Implementation

This algorithm takes the waffle and carves a perfect maze into the array. It uses recursion to backtrack through the passages it carved if a dead end is reached via the system call stack.

|  |
| --- |
|  |

A random cell in the waffle is passed to the subroutine as an argument. This cell is marked as visited. Then a list of all unvisited cells neighbouring the current cell is obtained. The pseudocode checks if the cell north, east, south, and west is visited. If it is not visited, it adds it to the list. I am adding 2 instead of 1 to the coordinates as I need to check the next cell, not the wall separating them. Next, a while loop is entered if there are unvisited neighbour cells. The algorithm picks a random cell from the list and destroys the wall between the current cell and the randomly selected one. The subroutine is then called recursively on the randomly selected cell. When a dead end is reached, the size of the list will equal 0, so the while loop will exit and control will return to the previous iteration of buildMaze, effectively using the call stack to backtrack until an unvisited cell is found.

### Stack implementation

The recursive algorithm is short and powerful but may exceed the maximum limit on recursion depth if the maze is too large. In the worst-case scenario, the algorithm will be called on every cell in the maze. If I run into issues with this, I may use an alternate implementation with an explicit stack of coordinates and a constructor which roams the maze, which has a less strict size limitation.

|  |
| --- |
|  |

A random cell is passed to the subroutine. The constructor is set to that cell and the constructor’s position is added to the stack. The number of unvisited cells in the maze is then calculated. Then, while there are still unvisited cells in the maze, the constructor checks if there are unvisited cells around it. If there are, then it picks a random one and travels to it, destroying the separating wall, adding its new position to the stack, and reducing the number of unvisited cells in the process. If, however, the constructor has reached a dead end, it backtracks down the stack until it finds an unvisited cell to enter. Eventually the constructor will have visited every cell and carved out a perfect maze in the process.

## Wilson’s Generation Algorithm

|  |
| --- |
|  |

This is an implementation of Wilson’s algorithm. Representing the logic as a flowchart is much neater than in pseudocode due to the large amount of data handling per step.

## Depth First Search Solving Algorithm

This is an easy to implement solving algorithm which find a solution, but it may not be the shortest one in a labyrinth-style maze. It is almost identical to the random mouse algorithm but uses the visited feature in the Coordinate class to make sure the solver never goes back down an already explored passage.

|  |
| --- |
|  |

Here, we start at the entrance. We get all the unvisited neighbour cells of the solver. If there are some, we go to the first one and mark it as visited, adding it to the solution in the process. By visiting the first one repeatedly, we essentially hug the left wall of the maze until an exit is found. However, this algorithm will not get stuck as we cannot loop due to marking cells as visited; if we find no unvisited neighbour cells, we backtrack down the solution stack. When the solver’s position is on the exit, we can return the solution stack, which will be a list of coordinates making up the final path the solver took to reach the exit.

## Maze Routing Solving Algorithm

With consideration of the Coordinate class’ implementation, I can implement this algorithm in a slightly different way to that described in the Analysis. I can have the algorithm never fail to find an improvement on its Manhattan Distance by implementing backtracking. The algorithm will have three states: Moving, Backtracking, and Trying Paths. Every update, the number of unvisited neighbour cells will be checked. If there is 0, backtrack: this behaviour replaces choosing based on which path best fits a line, as the algorithm would not normally have stack-based backtracking built in – it would just turn around. If there is exactly 1, it must be on a corridor, so continue moving down that corridor. If there are more than 1, it will try paths – in sequence, it moves down each path until it comes to a second crossroad, where it measures the Manhattan Distance between its current location and the maze exit. By doing this for each path, it can ascertain which path is likely the most productive, and so it will choose that one to proceed down. This will produce a good, but not perfect, solution.

# Security

Since I am implementing logins on a client-server network, I will need to implement some form of password obfuscation to prevent data theft; whilst no important data is stored on this database, the risk of a user choosing a password that is the same as their password on another platform is present and may turn the platform into a medium of attack for a malicious user. I plan to implement a password hashing algorithm, so the server never stores the passwords in plaintext.

## Data hiding

I plan to implement the PasswordChar feature into my user interface, so when the user is entering their password, it will be hidden by the standard dot character.

## Password Hashing

The PBKDF2 algorithm is built into C# and allows me to easily hash a password without the use of an external package. When the user has typed their password and requested a registration from the server, their password will be hashed and converted to hex/base64 so it can be sent as a string. In this way, the plaintext password will never be exposed from their local machine, and only ever the hash will be sent across the network and stored in the database, so it should be unintelligible to a malicious interceptor.

## Logging in

When logging in, the password the user entered on the login screen should be hashed on their local system and sent to the server. The hashes can then be compared using a fixed time comparison, which means that the state of the central system cannot be determined by the rejection time of a password (this could otherwise be used to guess how the password was hashed more easily). If the hashes are the same, the login should be correct, so the user is allowed access. Otherwise, access should be denied with a message that does not indicate what credentials were wrong.

## The Salt Field

The term “Salt” is used to describe a randomised string of characters appended to a password before it is hashed. This allows the same password to be hashed differently each time, increasing security somewhat. When logging in, if we hash the login password with the same salt as the original, we can get the same hash. I plan to store the hash in the login database alongside the salt to do this, but it is possible to extract the salt from the password if you know its length and encoding – I will not be doing this for the sake of more readable code.

# User Interface

|  |
| --- |
|  |

This is a first look at a potential user interface. I have added boxes, allowing the user to specify the parameters of the maze to request it. The box on the right holds a Table Layout Panel. I can use the CellPaint event of the panel to paint specific cells on the grid. However, I may have to have a dynamically sized form rather than a strictly sized one as pictured, since certain widths and heights cause small pixel errors which build up across cells, creating a large blank row and column across the bottom and right of the display. Additionally, high cell counts can make the cells too small. Both of these issues would be fixed by dynamic sizing.

Technical Documentation

# Program Flow and User Interface

|  |  |
| --- | --- |
|  | The client opens with a login interface. The user can enter a username and password. The password will be hidden by the dot character.  The user can also register a new account with the server by using the Register link. |
|  | Clicking the register link will open a new form which allows the user to enter new login credentials. There are several checks on the username and password to confirm they are suitable before the user will be allowed to register (See “Logging In and Registering” for details). |
|  | After registering an account, the register form will automatically close and the login form will refocus. Pressing Login will initiate the verification process (See “Logging In and Registering” for details). |
|  | This is the main window of the system. The window has several features. The selection boxes and number selectors on the left allow the user to select the parameters of the maze. Filling in all boxes whilst being connected to the server will unlock the Request Maze button which will send the information to the server where a maze will be generated (See “Maze Generation” for more detail.) Alternatively, by pressing Get Mazes, the selection box below the button will be populated with mazes the user has saved to the server. The user can load or delete the maze from the server with the relevant buttons. The panel on the right hosts dynamically generated charts showing local and global statistics for best times and mazes generated. |
|  | Here is the main form populated with data. The Request Maze button is now unlocked since each parameter has been filled in. Clicking the button will send the parameters to the server and a maze will be passed back to the client, being displayed in a new form. |
|  | Here is the Display form. The user can use WASD controls or the onscreen buttons to move the blue square through the maze. If they do this, a timer will start and be displayed, stopping when they reach the end of the maze. The Close button will then be unlocked, allowing access to the parameter form. The user can also use the selection box to select the algorithm they would like to request a server solve with. Pressing request solve will send the maze to the server which will send a solution back to the client before displaying the solution. The user also has the option to name their maze for a server save, or save it locally, which will open a File Explorer window where they can choose where to save the formatted image. |
|  | Here is the display form after requesting a solve using the Maze Routing algorithm. The solution is displayed with a purple line. The user can no longer attempt a manual solve. The close button has been enabled. Closing the window will refocus the parameter window and allow the user to begin the process again. |
|  | In the background, the server has been open the entire time. It logs all requests made to it in the console window. The client has exception handling on all server requests, since if the server closes unpredictably, there will be an unavoidable error which needs to be appropriately handled. (See “Network Exception Handling” for more details). |

# Project Structure

|  |  |
| --- | --- |
| Client | Server |
|  |  |
| The client’s code is mainly held within its form codebehind files. The Maze and Maze-derivative files it has contain only properties and [JsonConstructor] tagged constructors since they only need to build objects from the property structure to be read and displayed. This keeps the client lightweight. The client has a copy of every protocol file so it can send and receive the appropriate objects across the network. The client also makes use of a static class Globals (See “Global Variables” for more details.) | The server has many features in its extensive solution. It contains a folder of every protocol file so it can communicate correctly with clients. Each of these protocols have a dedicated service script in the Services folder, defining the management of the data in these requests, and what is sent back to the client. It also contains both the launchSettings.json and the appSettings.json files. These define how the server behaves (such as what port it operates across, whether it logs messages, what network protocol it uses, etc). It also contains the full definitions of all Maze classes and has the Solver and Solver-derivative classes, unlike the client. The server does not make use of global variables. |

# Techniques Used

## Network Exception Handling





Here is an example of how I have managed unpredictable server errors. When we make a network request, we can attach a deadline to it by providing a time: I have provided the current UCT time + 3 seconds for all deadlines, essentially forcing an exception after 3 seconds of server inactivity. Since we are turning an unpredictable server error into a predictable and specific error, we can catch that specific error by checking for the DeadlineExceeded status code, which is thrown when the deadline for a server request is exceeded. In the catch block, I have a procedure which is called whenever there is a server error that outputs an appropriate error message to a label on the form, as well as locking up buttons that interact with the server until a connection is re-established.

## Multithreading Exceptions

The code in my forms has many async and multithreaded methods. In a multithreaded form, threads cannot access objects from other threads without invoking a subroutine that interacts with it by proxy. However, the thread does not know when an object has been disposed, such as when a form closes, so an unpredictable error is thrown when trying to invoke access to a disposed object. Since this error causes no issue if it is ignored, we can stop a client crash if we specifically catch the ObjectDisposedException without handling it.

## Global Variables

See page 69 for the Globals.cs code.

Since excessive use of global variables makes code less robust, I have limited my usage to constant variables only. Furthermore, all global variables are prefixed with “g\_” to differentiate them from local variables. The only 2 non-constant globals are the username and userID, since these cannot be defined in the code as the system has many users. These are updated to the correct values at login and only read from afterwards. The version string is used to update the name of each form with the current client version. It is purely aesthetic but could have use in managing client updates. The cellWidth and cellHeight variables store the pixel width and height of cells in the maze, so the dynamically sized display form can autosize correctly. The keysize and iterations variables are used in hashing passwords (see “Logging In and Registering” for more details).

# Server Protocols

## Proto Files

See page 63 for the protocol files.

Protocol files end in the extension .proto and have their own language and syntax to define a service which operates across the network and the related request and response objects the service uses. The files have a set structure: first, the syntax is set. All my protocols are written in the proto3 syntax, so the top line of every file will set this. Next, the namespace the protocol will be implemented in is set. My project uses the Server namespace and Client namespace, so all my protocols are passed the Server namespace. Next, the service is defined. The service can contain many operations, represented by the rpc keyword. The operation is made up of the method name, the request object name, and the reply object name. Once all the operations have been defined in the service, the request and reply objects must be defined with what data they carry. This is done using the message keyword followed by the object name. Within the braces each variable is defined using a simple data type and an identifier. The number assigned to them is the order in which they must be passed to the collection when creating a request. Some of my protocols do not need to return a reply message to the client since they happen discreetly, such as incrementing stat values. In this case, you can import the google/protobuf/empty.proto, which when set as the reply object in the rpc definition, allows you to create a void network service. Once these protocols are defined, they must be integrated into the project to allow the service code to be written (See “Project File References” for more details).

## Project File References

See pages 110-111 for the client and server project files.

Since protocol files generate obfuscated code to facilitate the low-level transfer of data across the internet, they need to be referenced in the project file, so the compiler knows to create these files when the project is built. This is done by including the path to the file in a protobuf element within an itemgroup in the project file.

# Data structures

## List

See page 108 for this implementation. My list is a generic implementation that manipulates arrays to store data.

## Queue

See page 109 for my implementation. My Queue is implemented as a circular static data structure.

# Algorithms

## Database Management

### General information

Across all SQL commands, I have implemented the same techniques.

* I have used the using keyword on all SQLite objects, so they are properly disposed after use, preventing memory leaks.
* I have used the @ symbol before all SQL commands to make them into multiline verbatim string literals. This allows me to put each component of the SQL command onto a new line, vastly increasing readability.
* I have used the tag system to manually add parameters rather than interpolating them in. This helps prevent SQL injection attacks.

### Table creation

See pages 100-103 for the table creation scripts.

I have used the pragma command to turn foreign key constraints on. This means that SQL commands that would cause the primary and foreign keys across 2 tables to become mismatched are ignored.

The GlobalStats table records 3 pieces of information for each time: an integer time in milliseconds, a string display time, and the username of the user who set it. The latter 2 are used in displaying the best times on the client. The millisecond time is used in a serverside algorithm which orders the times, which would be made harder if the times were formatted strings instead.

### Triggers

See page 102 for the trigger body.

My database uses 1 trigger: CreateStatsRecord. The trigger creates a record in the stats table when a user registers a new account and fills it with default data. It also sets the userID of the record to be the userID of the new user. This is necessary since the stats record is not otherwise automatically created when the user registers, so many of the stat related services would break.

The code below this is similar: we need exactly 1 global stat record, so the SQL checks if there are any records in the GlobalStats table, and only adds a record if there is one. This record is filled with default data when it is initialized.

## Maze Generation

See page 104 for the standard implementation of InitMaze.

The InitMaze algorithm is called first to prepare the array of walls for the generation algorithm. The first thing it does is properly initialize the MazeWalls and MazeCoordinates arrays. This is because to allow the user to enter odd numbered widths and heights, the parameters entered are the widths and heights in cells. To convert this to the actual array dimensions, we put both into the same formula: 2n+1, where n is the cell dimension. The + 1 represents the first wall. The 2n then represents that for every cell requested, we add a passage and a wall.

The arrays are then looped through. Open passages should only be on coordinates where both the X and Y position are odd, so the if statement uses the mod operator to check this. All other cells are initialized as walls. A Coordinate is also created for each cell and added to the MazeCoordinates array. In this way, the waffle shape required for generation is created.

### Recursive Backtrack Algorithm

See page 104 for this algorithm.

The recursive backtrack algorithm uses the call stack as a method of backtracking through the maze: the top coordinate on the call stack is the current position, and if there is no unvisited neighbour cell to move to, the top function call on the stack finishes, so it is essentially being popped off the stack and the current coordinate is the previous cell.

The algorithm first sets the current cell to be visited. It then initialises a list of all unvisited neighbour cells via another function. The GetUnvisitedNeighbours function checks for each of north, south, east, and west, first if accessing the cell would throw an index out of bounds error, and second, if it is visited. If it is accessible and unvisited, it is added to the list. If the list is empty, the constructor has reached a dead end, so the function does not enter the loop and the subroutine ends here. If there are cells, the while loop begins. If there are cells that are not visited, the constructor picks a random cell from that list of cells. It then destroys the wall between it and the target cell by averaging their X and Y coordinates. It removes the target cell from the list and calls the BuildMaze function on the target cell. In this way, every cell in the maze is visited before the call stack is empty.

### Growing Tree Algorithm

See pages 96-97 for this algorithm.

This algorithm is unique since it does not make use of the Visited flag. It instead manages 2 lists of coordinates to decide where to go.

Firstly, the 2 lists are initialized and a random cell in the maze is selected to be the starting point. It is added to the list of “active cells”. An active cell is one which is on the border between visited and unvisited cells. Now the algorithm has 1 active cell, it will not end until there are no active cells, which will only happen when there are no unvisited cells left in the maze. The constructor cell is randomly picked from the list of active cells. Then, a list of that cells unvisited neighbours is fetched: here, this is a list of neighbouring cells not in the list of active or visited cells. If there are neighbours, a random one is selected as target. The wall between the constructor and target is broken by removing the wall at the average of their x and y positions. The target cell is now an active cell, so it is added to the list. If there are no neighbours, the cell must no longer be active, so it is removed from the active cell list and added to the visited list. Now, the process repeats, picking a new active cell and moving to its random neighbour, until all active cells have been exhausted.

### Wilson’s Algorithm

See pages 106-108 for this algorithm.

This algorithm uses randomized loop-erased walks (henceforth RLEW) to produce a uniform spanning grid – the maze will be completely unbiased in the length and number of its corridors.

To account for the fact that the constructor breaks no walls until the end of the RLEW, the definition for unvisited in this algorithm is a cell with all 4 of its walls intact. The Visited property is still used, but it now marks cells that are currently a part of the maze and assists in removing loops during the RLEW.

During InitMaze, as well as generating the waffle, a list of all coordinates in the maze is initialized.

The first RLEW is slightly different to the rest, as it always walks from the start Coordinate to the end Coordinate. A list of Coordinates, Path, is initialised and the constructor, starting at the start Coordinate, is added to the list. Then, while the constructor is not at the end Coordinate, it randomly moves around the maze, adding each cell it moves into to the Path and marking it as Visited. If it moves into a Visited cell, it has looped, so it begins backtracking, unVisiting all cells it backtracks through until it reaches the Visited target cell. This erases the loop. When the path eventually reaches the end Coordinate, the Path list is iterated through, destroying all walls in the path. Then, each coordinate now in the maze is removed from the list of all cells in the maze. This is because this list will be used to pick random starting points for subsequent RLEWs.

After the initial RLEW, RLEWs are completed until there are no cells left in the cellsInMaze list. These RLEWs are different to the initial one, as they do not have a specific endpoint. They randomly walk until they reach a Visited coordinate, which must be a point in the maze. The path of this walk is then added to the maze in the same way as the initial RLEW.

## Maze Solving

### Depth-First Search

Depth first solve uses a stack of Coordinates to navigate through the maze. The algorithm is very fast but does not generate the best solution in a labyrinth maze, since it’s path can be summed up as ‘hugging the left wall’.

The solver is placed at the start Coordinate and is added to the solution stack. Its Coordinate is set to be visited. Then, it gets a list of its unvisited Neighbours by checking the Visited flag of its surrounding coordinates and if they are passages. If there are unvisited passages in the list, take the first one by moving the solver to it, adding the new position to the stack, and setting Visited to true. If there are no possible directions, the solver has reached a dead end, so it backtracks by popping the top element of the stack until there are unvisited neighbour cells. This process is repeated until the solver has reached the end Coordinate.

### Maze-Routing Algorithm

See pages 97-99 for my implementation.

My implementation of this algorithm is a heuristic algorithm that finds a good solution to a labyrinth maze. It uses a stack and the GetManhattanDistance function for Coordinates to decide which path will be the best to take.

The algorithm starts with a Coordinate solver and a solution stack. The solver is added to the solution stack and the cell is marked as visited. Then, while the solver is not at the exit, the GetUnvisitedNeighbours function is called. Here, it returns a tuple of the Coordinates of unvisited neighbours and a char representing the cardinal direction they are from the solver. If there are no unvisited neighbour cells, the solver backtracks through the maze until it lands on a cell with unvisited neighbours. If there is only one path, the solver moves to it, adds it to the solution, and marks it as visited. If there are multiple paths, the TryPaths function is called and passed the list of potential neighbour cells. In this function, each potential neighbour’s path is tried in sequence until another fork in the path is reached. At each fork, the Manhattan distance is found between there and the exit, and the bestPath tuple is updated if the distance is lower than the best path currently found. Then, once each path has been tried, the path direction of the most productive path found is returned. The main solver then moves down this path. This whole process is repeated until the maze is solved.

### Breadth-First Search

See 94-95 for my implementation of this algorithm.

My implementation is a method of finding the best path through the maze. It tries many paths in sequence via a queue structure called ActiveCells. When the algorithm reaches a fork, all new unvisited neighbour cells are added to this queue and extended in sequence. This creates a list of many paths, and the shortest one is returned to the client upon completion of the algorithm.

First, a tuple of 2 Coordinates is defined, with the first item being the current cell and the second being the “parent” cell: this is the cell that the current cell was reached from. The algorithm then gets a list of the unvisited neighbour cells to the current cell – as well as unvisited being defined by the Visited flag of a cell, here the algorithm counts any cell in the Path list as visited, so if the algorithm tries to extend one path into another, it knows that path must be shorter since another path reached the same cell in less time, so it can terminate that path. Each of these neighbour cells are added to the queue of active cells and to the path list. This means that if no neighbour cells are found, no active cells are added for that cell, so its path terminates there. Note that active cells and paths are added as tuples with a reference to their parent cells, since once the algorithm is completed, another is called to construct the solution from the path list.

|  |  |
| --- | --- |
| SOURCE: https://en.wikipedia.org/wiki/Breadth-first\_search | This is essentially what the algorithm has done so far: it has mapped out each cell with a reference to the cell which precedes it.  Now, the GetSolution algorithm runs. It finds the path element where the cell is the exit cell. From there, it adds Coordinates to the solution by backtracking through the joined map of the graph it has created, until the entrance is reached. |

## Logging in and Registering

See pages 91, 93, 77-78, 82-84 for the implementation of these features.

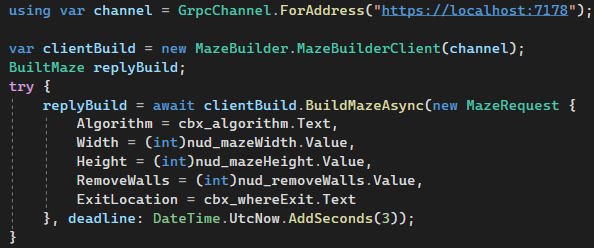
When the user presses the Register button on the registration form, their credentials are not immediately sent to the server. First, their credentials are checked:

* The server is queried as to whether the username is taken.
* The password length is checked.
* The password is checked for special characters using Regex. It must contain at least 1
* The password is checked if it matched the Confirm Password box.

The credentials are processed only if these checks are passed. The first step to processing the credentials is hashing the password. We call the HashPassword subroutine [[2]](#footnote-2)to do this. The salt is randomly generated with a length defined by the keySize global constant. The hash is then produced by the PBKDF2 algorithm with a number of iterations defined by the iterations global constant. This hash is then passed to the server as a hexadecimal string, as well as the hexadecimal conversion of the salt. The hex conversion is necessary as the hash and salt are both byte arrays, which are not supported types in the protocol files. The server creates a new record for the new user and stores their username, password, and salt under a new userID.

When the user attempts a login, the username and password entered are sent to the server’s LoginHandler service. The server uses SQL to find the stored password and salt hashes associated with the entered username. It then hashes the password it received using the same salt and algorithm parameters. It compares the hashes using the FixedTimeEquals method, so the state of the server cannot be guessed via the length of time the comparison takes. If the hashes are the same, the user is granted access and the clientside globals username and userID are updated. If the credentials are incorrect, the message “Username or Password incorrect!” is displayed. The message is ambiguous so the user cannot guess usernames and passwords and get information from it.

## Client-Server Interactions



Here is an example of a client call to the server. There are 3 steps to making a server call, and these are the same regardless of which service the client is requesting. Firstly, a communication channel is established with the server, using the address specified in the server’s launchSettings.Json file. The using keyword is used so the channel is properly disposed after use, to avoid memory leaks. Next, using the channel, the service to request is specified. Finally, the client calls the function of the service, passing in the request object with the relevant parameters. The await keyword is used here, and the method encompassing this code is an async method. This is because the time the server will take to receive, process, and transfer the data is uncertain, and we don’t want the client interface to hang while it is waiting. The async and await keywords allow the user interface to be interactive whilst the server manages data.

## Other Algorithms

These other algorithms are to do with managing the form interface.

### Connectivity Status

See page 78 for this algorithm.

This algorithm updates the label that notifies the user whether they are connected to the server. It has a polling rate of 10 seconds to not overload the server with requests. It uses multithreading to not lock up the main form, since it calls a while true loop. To do multithreading, an async lambda function containing the code we want to be ran is passed to the work item queue of the thread pool. In this code is a network call that tries to ping the server. If it manages to ping the server, another lambda is invoked which updates the label to tell the user they are connected. It also updates the Request Maze button so the user can send mazes. If the server fails to respond by the deadline, the HandleServerError method is called. A second catch block is required due to the unpredictable nature of the ObjectDisposedException (See “Multithreading Exceptions” for more detail). The loop then sleeps for 10 seconds before repeating.

### Setting the Size of the MazeDisplay Form

See page 73 for this algorithm.

The private function SetDisplaySize is called on the load event of the MazeDisplay form. This function handles the dynamic sizing of the said form to ensure all the buttons are visible regardless of maze size.

The width of the form is set first. This uses a ternary operator to decide whether to size the form based on the maze size or the button panel size. It calculates the pixel width of the maze plus the pixel size of the margin between the table layout panel and form and checks if this size is greater than the size of the button panel. If it is, the width of the form is set to the size of the maze and margin. Otherwise, it is set to the size of the button panel.

Next, the height of the form is set to the pixel height of the button panel plus the maze height and margin. The panel containing the maze is then sized appropriately, and placed at a point so the left, bottom and top margins are correctly sized.

### Displaying the maze in the Table Layout Panel

See page 73 for this algorithm.

Since the maze is drawn inside a table layout panel, it is effectively a grid. I have used the CellPaint event of the table layout panel to colour this grid in appropriately based off the maze object, the user’s navigator position, and the solution if applicable. There is no explicit loop necessary, since this is done by the CellPaint event: the event is called for each cell in the table layout panel, with the TableLayoutCellPaintEventArgs parameter “e” being each cell in sequence. I can therefore check if a certain condition is true, and if it is, use the FillRectangle subroutine of the cell “e” to paint that cell the appropriate colour.

If the cell contains the player, paint it blue. If the cell is the entrance, paint it red. If the cell is the exit, paint it green. If the cell is a wall, paint it black. If there is a solution, and the cell is part of the solution, paint it purple. If none of these are true, the cell must be a passage, so paint it white.

# Requirements Met

|  |  |  |
| --- | --- | --- |
| Req. Number | Requirement | Evidence and comments |
| 1.1 | The ability for the user to customize their request for a maze with several parameters. | The MazeParameter form has buttons for 5 customization options. |
| 1.1.1 | The ability to change the maze generation algorithm used to make the maze. | See 1.1: The algorithm DropDownList allows the maze type to be selected. |
| 1.1.1.1 | An implementation of an algorithm to make perfect mazes with a bias for long corridors and low branching. | See pages 104-105: The RecursiveBacktrackGeneration class is an implementation of the recursive backtrack generation algorithm, which has the relevant biases. |
| 1.1.1.2 | An implementation of an algorithm to make perfect mazes with a bias for short corridors and high branching. | See 96-97: The GrowingTreeGeneration class is an implementation of the growing tree generation algorithm, which has the relevant biases. |
| 1.1.1.3 | An implementation of an algorithm to make unbiased perfect mazes with a uniform distribution of branches and corridor lengths (a uniform spanning grid). | See 106-108: The WilsonsGeneration class is an implementation of Wilson’s algorithm, which produces a uniform spanning grid as required. |
| 1.1.2 | The ability for the user to select the width and height of the maze they generate, measured in cells. | See 1.1: The NumericUpDown boxes allow width and height selection.  See the InitMaze algorithm on pages 96-97, 104-105, 106-108: The numbers are cell heights and are translated into actual heights to allow odd numbers of cells. |
| 1.1.2.1 | A width parameter which changes the horizontal cell size of the maze. | See 1.1: The NumericUpDown box labelled Width allows the width in cells to be changed from between 2 to 80 inclusive. |
| 1.1.2.2 | A height parameter which changes the vertical cell size of the maze. | See 1.1: The NumericUpDown box labelled Height allows the height in cells to be changed from between 2 to 40 inclusive. |
| 1.1.3 | The option to remove a user-specified number of walls in the maze, allowing for a performant solution of generating uniquely styled labyrinth mazes. | See 1.1: The NumericUpDown box labelled Remove Walls allows the user to specify the relevant number.  See the RemoveWalls algorithm on pages 96-97, 104-105, 106-108: The specified number of walls are removed by this algorithm during the creation process. |
| 1.2 | The ability for the user to allow the server to solve the maze after it has been generated. | The user can select a solving algorithm from the DropDownList and press the request solve button on the MazeDisplay form. |
| 1.2.1 | An implementation of a simple algorithm for performant solves. | See page 96: The DepthFirstSolve class is an implementation of the Depth first solving algorithm which fits the necessities of this requirement.  See 1.2: The user can select the Depth First option to solve with the relevant algorithm. |
| 1.2.2 | An implementation of the Maze-Routing algorithm allowing for shorter-path solves. | See pages 97-99: The MazeRoutingSolve class is my implementation of the maze routing algorithm which fits the necessities of this requirement.  See 1.2: The user can select the Maze Routing list option to solve their maze with said algorithm. |
| 1.2.3 | An implementation of any shortest path algorithm, such as A\* or the First-Breadth Search algorithm allowing the user to find the shortest possible path in a labyrinth maze | See pages 94-95: The BreadthFirstSolve class is an implementation of the first-breadth solving algorithm, which is an implementation of a best-path algorithm. This fits the necessities of the requirement.  See 1.2: The user can select Breadth First from the list to use the relevant algorithm. |
| 1.3 | The ability for the user to request a maze to be generated by the server with their selected parameters, and have it be returned to their client on completion. | See 1.1: The Request Maze button on the MazeParameter form allows the user to send their parameters to the server, wrapped into a request object. The MazeDisplay form opens with their maze once it is received from the server.  See page 79, btn\_RequestMaze\_Click(), RequestMaze(), and ChangeForm(): This is the code that performs the stated functionality. |
| 1.4 | A button on the client available post-generation allowing the user to request a solve of their maze. | See 1.2: The Request Solve button on the MazeDisplay form provides this functionality.  See page 73, btn\_RequestSolve\_Click(): The underlying code that requests a solve from the server on pressing the button. |
| 1.5 | A screen on the client which displays the current maze after receiving it from the server. | See 1.2: The MazeDisplay form implements this screen as a dynamically sized TableLayoutPanel. |
| 1.6 | The ability for the user to navigate the displayed maze pre algorithmic solve and attempt to solve it themselves. | See pages 74-75: The code that handles movement around the maze by allowing the user to manipulate a coordinate assigned to their “navigator”. |
| 1.6.1 | Navigation controls mapped to buttons on the keyboard. | See page 76: The code that handles keyboard input. |
| 1.6.2 | Buttons on the client interface which allow navigation. | See 1.2: The 4 buttons with text set to the 4 cardinal directions via Unicode characters are set up to translate the user’s “navigator” call on click.  See page 75: The code that performs this functionality. |
| 1.7 | The ability for a user to save their generated maze. | See 1.2: The server and local save buttons implement the 2 save methods. |
| 1.7.1 | If the login system is completed, this should save the maze to the server database. Otherwise, this feature will be a local download. | The login system is completed (See tests 1-7; pages 91, 93, 77-78, 82-84), so the server save functionality is provided by clicking the relevant button (See pages 76-77). I have also implemented the local download feature for completeness (See page 76). |
| 1.8 | A section of the client which uses the Windows Forms Graphs library to display a user-specified graph of information stored in the server database. | See “Program Flow and User Interface” Row 5: The graph is initialised dynamically on request in the panel shown in the empty panel in 1.1.  See pages 80-82: The code that handles dynamic graph generation.  See page 82: The code that handles dynamic generation of a text box to show time leaderboards. |
| 1.8.1 | This aspect should allow the user to see personal stats, as well as global stats if the login system is completed | See 1.1: The “Global?” checkbox allows the user to switch between requesting personal and global stats on request, thanks to the login system.  See pages 80-81: The code that implements the checkbox Boolean. |
| 2 | The Console App server which will handle the backend operations of the system, such as maze generation, solving, and database handling. | A screenshot of the ASP Core .NET gRPC server backend. See subobjectives for details of its implementations. Note that the screenshot does display the server logging activity, but requirement 2.2 provides further proof. |
| 2.1 | A database structure storing statistics and login information, as well as storing saved mazes. | See pages 100-103: The SQL code that creates tables for this purpose. |
| 2.2 | A console interface which logs requests | See tests 2,7,8,10: The server logs all requests that it processes. |
| 2.3 | Services to facilitate logging in. | See page 91: The code that implements this feature. |
| 2.4 | Services to facilitate registering a new user. | See page 93: The code that implements this feature. |
| 2.5 | Services to facilitate database management. | See subobjectives for specific examples of database management, otherwise browse paes 84-94 for gRPC services which interact with the database. |
| 2.5.1 | Saving/loading mazes on request. | See pages 90-91, 93-94: Services which define these features. |
| 2.5.2 | Organize global and user times. | See pages 87-90: Services which define these features. |
| 2.6 | Services to facilitate getting data from the server database. | See subobjectives for specific examples of database management, otherwise browse pages 84-94 for gRPC services which interact with the database. |
| 2.6.1 | Fetching user stats. | See pages 8-87 for the service that implements this. |
| 2.6.2 | Fetching global stats. | See pages 85-87 for the service that implements this. |
| 2.6.3 | Fetching saved mazes. | See pages 90-91 for the service that implements this. |

# Models Used

|  |  |
| --- | --- |
| Complex data model in database | I have used 3 interlinked tables, which fits this criterion. See pages 100-103 or “Entity Relationship Diagram” in the analysis for the database model. |
| Hash tables, lists, stacks, queues, graphs, trees, or structures of equivalent standard | I have used lists, stacks, and a queue modelled by lists throughout the program. See page 108 for my implementation of a list, and page 96 for its use. See pages 109-110 for my implementation of a queue, and page 94 for its use. |
| Files(s) organised for direct access | I have split each of my classes into a separate file, and further organised these by sorting .proto files and server service definitions into their own subfolders for organised and direct access. |
| Complex user-defined use of object-orientated programming (OOP) model, eg classes, inheritance, composition, polymorphism, interfaces | I have used inheritance via the abstract class system, which is also analogous to interfaces. Since a Maze also contains a grid of Cells, there are examples of composition in each generation algorithm. |
| Complex client-server model | The client and server transfer varied data and parse JSON to facilitate the client and server interactions. |
| Multi-dimensional arrays | Each maze contains 2D arrays of Booleans and Coordinates. |

# Algorithms Used

|  |  |
| --- | --- |
| Cross-table parameterised SQL | Saving and loading mazes, registering and signing in, etc all make use of parameterised user input as well as the UID foreign key. |
| Aggregate SQL functions | See pages 84 and 103 for SQL queries that use COUNT(\*). |
| Graph/Tree traversal | A maze is a representation of a graph, so the solving algorithms I have implemented are all graph traversal algorithms. |
| List / Stack operations | Each algorithm utilises lists and stacks, and their respective methods for handling data in them. |
| Hashing | The login system makes use of hashing for passwords. |
| Recursive algorithms | See pages 104-105 for a recursive maze generation algorithm. |
| Complex user-defined algorithms (eg optimisation, minimisation, scheduling, pattern matching) or equivalent difficulty | Maze-solving falls under this level of difficulty. |
| Dynamic generation of objects based on complex user-defined use of OOP model | Maze and coordinate objects are dynamically generated at the user’s request. |
| Server-side scripting using request and response objects and server-side extensions for a complex client-server model | See pages 63-68, 84-94 for a group of server-side scripts. Each protocol file defines request and response objects, which are used throughout the program to service the client-server model. |

Testing

# Test Plan

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Input | Expected Output | Actual Output | Comments |
| 1 | Attempt to register whilst not connected to the server. | UN: greeg  PW: abcdef!  CF: abcdef! | Error label should provide a relevant error message after 3 seconds. | See Fig.1 | Works as expected. |
| 2 | Attempt to register an account with a server connection | UN: greeg  PW: abcdef!  CF: abcdef! | Label should inform user of success, close the form and create records in the Login and UserStats tables. | See Fig.2 | Works as expected. All subsequent tests will be run with this account in the database. |
| 3 | Attempt to register a new account with a taken username | UN: greeg  PW: abcdef!  CF: abcdef!  Database not reset from test 2. | Label should inform user that the username is taken. | See Fig.3 | Works as expected. |
| 4 | Attempt to register an account with invalid password fields | PW: abcde  Then  PW: abcdefgh  Then  PW: abcdef!  CF: xyzxyz | Label should inform the user their password is too short, then that it has no special characters, then that their passwords do not match. | See Fig.4 | Works as expected. |
| 5 | Attempt to log in when not connected to the server. | UN: greeg  PW: abcdef! | Label should inform the user that they are not connected after 3 seconds. | See Fig.5 | Works as expected. |
| 6 | Attempt to log in with incorrect credentials. | UN: greeg  PW: xyzxyz | Label should give a message indicating that the credentials are incorrect. | See Fig.6 | Works as expected. |
| 7 | Attempt to login with correct credentials | UN: greeg  PW: abcdef! | Access is granted: the login form hides and the MazeParameter form opens. | See Fig.7 | Works as expected; the login must have worked since it is logged in the server and the parameter form greets the user in the form title. |
| 8 | Attempt to generate a recursive backtrack maze with odd numbered dimensions. | Algorithm: Recursive Backtrack  Width: 25  Height: 25  Exit: Centre  RemoveWalls: 0 | The maze should generate with the given parameters. The server should log the activity. | See Fig.8 | Works as expected. |
| 9 | Attempt to solve the maze with depth-first search. | Algorithm: Depth First search | The maze should show a solution with a purple line. | See Fig.9 | Works as expected. |
| 10 | Generate a maze with the growing tree algorithm with different width and height dimensions and many removed walls. | Algorithm: Growing Tree  Width: 25  Height: 40  Exit: Border  RemoveWalls: 40 | The maze should generate with the given parameters. The server should log the activity. | See Fig.10 | Works as expected. |
| 11 | Attempt to solve the maze with Breadth-First search. | Algorithm: Breadth First search | The maze should display the best solution to the user. | See Fig.11 | Works as expected. |
| 12 | Save the maze locally. | n/a | The client should display a file explorer window and allow the user to save an image of their maze. | See Fig.12 | Works as expected – a JPEG file was saved to the downloads folder, which you can see opened in the second screenshot. |
| 13 | Attempt to save the maze to the server with no name. | [no input] | The client should display an error message stating that the name field is required. | See Fig.13 | Works as expected. |
| 14 | Attempt to correctly server save the maze. | Name: test | The client should display a message indicating that the maze has been saved. The server should log the activity. | See Fig.14 | Works as expected. |
| 15 | Attempt to generate a Wilson’s maze with minimum sizes. | Algorithm: Wilson’s  Width: 2  Height: 2  Exit: Border  RemoveWalls: 0 | The maze should generate with the given parameters. | See Fig.15 | Works as expected. The dynamic form sizing allows the MazeDisplay form to still show all buttons, even with a small maze. |
| 16 | Attempt to solve the maze with the Maze-routing algorithm | Algorithm: Maze-Routing | The maze should show a solution with a purple line. | See Fig.16 | Works as expected. |
| 17 | Attempt to load the maze saved as “test” from test 14. | Press Get Mazes. Then select the “test” maze and press Load Maze. | The maze should load from the server. | See Fig.17 | Works as expected – the maze is the same as the one shown in test 14. |
| 18 | Attempt to delete the test maze from test 14 | Press Get Mazes and select the “test” maze. Press Delete Maze and then Get Mazes again. | The client should show that it found 0 mazes, indicating that the maze has been deleted. | See Fig.18 | Works as expected. |
| 19 | Generate a small Wilson’s maze and use the keyboard controls to solve it. | WASD controls.  Algorithm: Wilson’s  Width: 5  Height: 5  Exit: Border  RemoveWalls: 0 | A timer should be displayed, which stops when the user reaches the exit. | See Fig.19 | Works as expected.  This test will be repeated 6 times to record data for statistic testing. |
| 20 | Request local times from server | Stat type: Best Times  Global?: False | The times generated from test 19 should be displayed in order. | See Fig.20 | Works as expected. |
| 21 | Request local mazes generated from server | Stat type: Mazes Generated  Global?: False | The pie chart should display W’s:RB:GT in a 7:1:1 ratio. | See Fig.21 | Works as expected. The 6:1:1 ratio is from the 1 recursive backtrack maze generated in test 8, the one growing tree maze generated in test 10, and the 7 Wilson’s mazes generated across tests 15 and 19. |
| 22 | Create another account and generate 1 recursive backtrack maze and solve it manually. Then request each stat type. | UN: greeg2  PW: abcdef!  CF: abcdef!  Algorithm: Recursive backtrack  Width: 5  Height: 5  Exit: Border  RemoveWalls: 0  WASD controls.  Stat type: Best Times  Global?: False Stat type: Best Times  Global?: True Stat type: Mazes Generated  Global?: False Stat type: Mazes Generated  Global?: True | The local best times should have only 1 time, whilst the global best times should display the times set in the “greeg” account. The local mazes generated should be 1:0:0 in favour of recursive backtrack, while the global should now show W’s:RB:GT in a 7:2:1 ratio. | See Fig.22 | Initially was throwing an index out of range error on the server code that fetches global time stats (See pages 85-87). This was due to a missing field in the SELECT section of the SQL. After this was fixed (See “Fixes”), it worked as expected.  This test proves the local and global stat services are handled properly: the screenshots prove that the local stats only show what the user generated/set, whilst the global stats show what both the “greeg” and “greeg2” users both did. |

# Screenshots

|  |  |
| --- | --- |
| Figure # | Screenshots |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |
| 20 |  |
| 21 |  |
| 22 |  |

# Fixes

|  |  |  |
| --- | --- | --- |
| Test Number | Code with error | Fixed Code |
| 22 |  |  |

Evaluation

# My Thoughts

## Login System

I feel that I have implemented a good login system for the scope of my project. The login system as it is provides a flexible way to link data to a specific user via their userID, which I have already used extensively throughout the project. I think the security is strong enough for the complexity of the system – if the system were to be expanded into a wider puzzle system such as my client’s, a more robust password storage system might be advantageous. This could be implemented by adding 2 Factor Authentication via APIs, or using a hash table so password hashes are not aligned with their respective user. I think that my implementation of registering an account is fast and easy, which is appropriate for my solution as it is only operating over a LAN network. However, if it were to be expanded to operate over a website for example, it might be desirable to add more features to registering, such as an email field, using a mail server to send confirmation emails, or a security question for password resets. Finally, the option for the user to delete their account could be implemented easily. The SQL “ON DELETE CASCADE” could be used in database creation commands so when a user’s account is deleted, perhaps with confirmation/security requirements implemented with the methods discussed, the relevant fields in other tables are also deleted, removing redundant data.

## Network Functionality

I think the network implementation with gRPC was the strongest choice for my network. The protocol file system allows new services to be defined without interfering with the current network operation, and the ability to control which services are mapped to the server allows controlled testing of new services. Additionally, I think the speed is suitable for the maximum 150-person load. Considering that most of the time only a fraction will be using the software, and that the services tend to complete quickly (See test 10, figure 10: building a maze took 26ms), even if many requests were sent through at once, the server could handle it. The three second deadline on each server call I have implemented allows ample time for even the slowest architecture to support the server. gRPC does support webservers, so adding fully online functionality to the server rather than LAN operation is possible with some changes, but the deadline on server requests might have to be altered to allow for the extra latency caused by the change. However, the REST system is much more commonly used for exposing online services, so it could also be an option to port the system to this instead.

## Algorithms

I believe I have chosen a suitable variety of algorithms for my system. The generation algorithms provide a variety of complexity and looks to fully demonstrate the capabilities of the system: recursive backtrack creates simple mazes with long corridors and sparse branches, growing tree is the opposite, generating many branches of short corridors, and Wilson’s algorithm generates a uniform graph. These algorithms not only provide visual variety, but also technical as well: Their different generation algorithms allows the code to demonstrate how adding a new algorithm to the main generation service can be done in a different sequence of steps depending on the algorithm (See page 92). The solving algorithms are similar in their variety: depth-first search is simple and fast, the maze-routing algorithm is a good middle-ground algorithm, and breadth-first search is slower but finds the best path. This algorithmic variety helps showcase the capabilities of my OOP model. Adding new algorithms is simple, as they can be tested in a separate environment, then added to the server thanks to the abstract class standardizing its output. Adding new algorithms to the server is as easy as adding new cases for them in some switch statements.

## User Interface

I think the UI for the system is ok, but somewhat cluttered. I have made a compact UI with the designer, and by providing hard limits to input fields in control properties, I have eliminated a lot of hard-to-read error handling code. The event hander system provides a readable interface for handling input, which should make adding new features related to interaction with the form easier: I have already implemented some visual features into the form, such as the MazeParameter form greeting the user, or the dynamic server connection label on the same form. However, I don’t think the forms designer was meant to handle a graphical project like this; early on into the project I ran into many issues with subpixel alignment causing the maze to draw incorrectly onto the table, which is why the dynamically sized display form is required. Additionally, double buffering the form, whilst it does stop the table layout panel from flickering on redrawing the maze, does cause some input lag on movement in larger mazes and causes flickering on other components, such as placeholder text in a text box. I think that if I were to approach the project again, I would consider writing it as a Windows Presentation Foundation script to allow for smoother and more robust transitions between forms. Additionally, the XAML code and Model-View-ViewModel structure system would allow for even more organized code and finer control over the properties and rendering of maze graphics. Perhaps even a game engine such as Unity or Godot could be a route to consider for the project, considering the powerful features they bring, in both their libraries and rendering capabilities.

## Robustness of System

I think the overall system is resistant to fatal runtime errors, despite all the potential vectors for them to appear. I have caught multithreading exceptions and used deadlines on all server requests, with specific catch blocks for that status code. This way, I only catch specific errors, so if any other errors arise that are more serious, they can be noticed and fixed faster. Additionally, by adding an error label to each form, it is easy to add more catch blocks to each error handler and display relevant information to the user regarding that error. Since there are several multithreaded components of my system, it does again point towards the use of a different engine for a future version of this system, where multithreading is more directly supported, to eliminate the need for catching multithreading errors.

# Client’s Thoughts

After finishing and testing the system, I provided Greg with it to get his impressions on the project, and what potential features could be added in the future.

## Description of client’s interaction

Greg initially started the client and server and opened the register form via the link label to register a new account. Greg liked the concise account creation process and noted that the initial omission of email confirmation was nice and suitable for his userbase size, as the email could be added later in an account settings page. He then logged in and opened the parameter form. He noted that the initial look of the form was somewhat overwhelming with the number of interactive elements and could be reorganized to better suit his target users. He proceeded to request a 25x25 border-exit maze with Wilson’s algorithm and requested it from the server. He noticed that the parameter form stayed open and suggested that it should hide or have more of its options lock to simplify the visuals for the user. He then proceeded to solve the maze, testing both the on-screen buttons and keyboard controls in the process. Greg liked the presentation of the buttons and the ease of switching between the 2 control methods but mentioned that the label should specifically state the controls are WASD. Once he finished the maze, I directed him to save the maze to test out the load functionality of the system. He first saved a png image of the maze using the local save feature and liked that it used the system file manager rather than a preset path. He then saved the maze to the server and closed the form. He then tried to load the maze using the central buttons on the form. Whilst he thought it was a self-explanatory button layout, he commented that the form should fetch the mazes automatically rather than with a dedicated button. He requested a solve of the loaded maze with the breadth-first solve and was impressed with the responsiveness despite the network system. He mentioned that an additional feature could be to allow for different path colours. He then closed the form and deleted the maze. Finally, he tested out the statistics section of the parameter form. He first requested the local mazes generated and liked the 3D pie chart presentation. He then requested the global best times. He suggested that I could add a button to request the maze a certain leaderboard user solved.

## Evaluation of their comments

I thought Greg’s interaction with the system went well and represented how a user would approach using the system. He suggested many features, which I will discuss below.

### Features Suggested

1. Adding additional account credentials in a settings page.
2. Reorganization of the parameter form.
3. Simplify the visual presentation of the client by only having 1 form visible at a time.
4. Clear labelling of controls.
5. Automatic fetching of mazes from the server.
6. Different path colours.
7. Request a leaderboard user’s maze.

Some of these features are trivial to add but allow for further features to be added in their wake. Some are harder to add or would result in less clean code.

### Discussion of Features

1. The settings page itself would not be hard to add. It can be another form accessible from the parameter form, which on loading, pulls the user’s data and displays it, with buttons and input fields to allow changing it. The more difficult aspect of this feature is inputting data like emails or changing passwords. Both would require some form of confirmation: the email confirmation wouldn’t be too hard to send, as it could just be done via a mail server, but getting the system to recognise that the user has interacted with the confirmation email is difficult with the current setup. Changing passwords is easier in comparison; it just requires a security question or some similar system at account registration. Alternatively, it could implement an API to access a 3rd party two factor authentication method, such as Authy.
2. Reorganizing the forms is a much easier task. The designer has many organization controls, like panels, split panels, and even the option to obscure entire panels of controls into separate tabs. Rearranging the already present controls into a new format with these would not produce any significant programming challenges, but may introduce some UI design challenges, since the form designer can be difficult with complex forms. Manipulating control margins or docking/anchoring to panels can introduce many issues when done with more dense elements like tables.
3. This is just an extra line of code, telling the parameter form to hide until the display form is closed. This is a good feature to add, as interacting with the parameter form while the display form is open, while limited as much as possible, could introduce unexpected behaviour.
4. This feature at its simplest is just changing the text on the label. However, this feature could be combined with feature 1, implementing customisable keyboard controls in the settings menu and dynamically updating the label to reflect the current control layout. This could be done by having a static class with static KeyCode properties for the 4 directions. These could be updated with the relevant keycodes from the settings menu and passed to the input handler function to check for the relevant keys. Each keycode could be matched with an alphanumeric character in an array so the label text could be changed dynamically.
5. This feature requires moving the request server call to another event handler. I think the best would be a focus related hander. When the form is focused, fetch the mazes into the drop down. This is better than something like the on mouse enter event handler for the drop-down list, as it is more reliable – if the user interacts with the drop-down list while it is fetching the mazes, it could introduce unexpected behaviour.
6. Similarly to feature 4, this would use a property in a static class – this time a Colour, which would be passed to the CellPaint event of the table layout panel where the maze is drawn.
7. This is a difficult requirement. A button array would have to be precisely generated dynamically next to the leaderboard users. When the buttons are pressed, they fetch the maze associated with that user. This means the global stats table would also have to store the mazeID of the top 10 times as a foreign key.

Appendix

# Shared code

## Protos

These proto files appear in both the client and server solutions, so I have only attacked them once for brevity.

### BuildMaze.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service MazeBuilder {  rpc BuildMaze (MazeRequest) returns (BuiltMaze);  }    message MazeRequest {  string algorithm = 1;  int64 width = 2;  int64 height = 3;  int64 removeWalls = 4;  string exitLocation = 5;  }    message BuiltMaze {  string maze = 1;  } |

### CheckIfUserExists.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service CheckerIfUserExists {    rpc CheckUser (Query) returns (Exists);  }    message Query {    string username = 1;  }    message Exists {    bool userExists = 1;  } |

### DeleteMaze.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service DeleterMazes {    rpc DeleteMaze (DeleteRequest) returns (SuccessAcknowledge);  }    message DeleteRequest {    int32 mazeID = 1;    int32 userID = 2;  }    message SuccessAcknowledge {    bool success = 1;  } |

### GetMazes.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service GetterMazes {    rpc GetMazes (Request) returns (MazesList);  }    message Request {    int32 userID = 1;  }    message MazesList {    string mazes = 1;  } |

### GetStats.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service StatsGetter {    rpc GetGlobalTimes (GetGlobalTimesRequest) returns (GlobalTimes);    rpc GetUserTimes (GetUserTimesRequest) returns (UserTimes);    rpc GetGlobalMazesGenerated (GetGlobalMazesGeneratedRequest) returns (GlobalMazesGenerated);    rpc GetUserMazesGenerated (GetUserMazesGeneratedRequest) returns (UserMazesGenerated);  }  message GetGlobalTimesRequest {    }  message GlobalTimes {  string time1DisplayTime = 1;  string time1Username = 2;  string time2DisplayTime = 3;  string time2Username = 4;  string time3DisplayTime = 5;  string time3Username = 6;  string time4DisplayTime = 7;  string time4Username = 8;  string time5DisplayTime = 9;  string time5Username = 10;  string time6DisplayTime = 11;  string time6Username = 12;  string time7DisplayTime = 13;  string time7Username = 14;  string time8DisplayTime = 15;  string time8Username = 16;  string time9DisplayTime = 17;  string time9Username = 18;  string time10DisplayTime = 19;  string time10Username = 20;  }    message GetUserTimesRequest {  int32 userID = 1;  }  message UserTimes {  string time1DisplayTime = 1;  string time2DisplayTime = 2;  string time3DisplayTime = 3;  string time4DisplayTime = 4;  string time5DisplayTime = 5;  string time6DisplayTime = 6;  string time7DisplayTime = 7;  string time8DisplayTime = 8;  string time9DisplayTime = 9;  string time10DisplayTime = 10;  }    message GetGlobalMazesGeneratedRequest {    }  message GlobalMazesGenerated {  int32 recursiveBacktrackMazesGenerated = 1;  int32 growingTreeMazesGenerated = 2;  int32 wilsonsMazesGenerated = 3;  }    message GetUserMazesGeneratedRequest {  int32 userID = 1;  }  message UserMazesGenerated {  int32 recursiveBacktrackMazesGenerated = 1;  int32 growingTreeMazesGenerated = 2;  int32 wilsonsMazesGenerated = 3;  } |

### greet.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Client";    package greet;    service Greeter {    rpc SayHello (HelloRequest) returns (HelloReply);  }    message HelloRequest {    string name = 1;  }    message HelloReply {    string message = 1;  } |

### HandleGlobalStats.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    import "google/protobuf/empty.proto";    package greet;    service GlobalStatHandler {    rpc IncrementMaze (MazeType) returns (google.protobuf.Empty);    rpc UploadTime (Time) returns (google.protobuf.Empty);  }    message MazeType {  string mazeType = 1;  }    message Time {  string username = 1;  string time = 2;  int32 timeMilliseconds = 3;  } |

### HandleUserStats.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    import "google/protobuf/empty.proto";    package greet;    service UserStatHandler {    rpc UserIncrementMaze (UserMazeType) returns (google.protobuf.Empty);    rpc UserUploadTime (UserTime) returns (google.protobuf.Empty);  }    message UserMazeType {  string mazeType = 1;  int32 userID = 2;  }    message UserTime {  string time = 1;  int32 timeMilliseconds = 2;  int32 userID = 3;  } |

### LoadMaze.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service LoaderMazes {    rpc LoadMaze (LoadRequest) returns (MazeToLoad);  }    message LoadRequest {    int32 userID = 1;    int32 mazeID = 2;  }    message MazeToLoad {    string maze = 1;    string mazeGenAlg = 2;  } |

### Login.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service LoginHandler {    rpc Login (Credentials) returns (Access);  }    message Credentials {    string username = 1;    string password = 2;  }    message Access {    bool loggedIn = 1;    int32 userID = 2;  } |

### Register.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service Registerer {    rpc Register (Account) returns (Acknowledgement);  }    message Account {    string username = 1;    string password = 2;    string salt = 3;  }    message Acknowledgement {    bool success = 1;  } |

### SaveMaze.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service Saver {    rpc SaveMaze (SaveRequest) returns (SuccessAck);  }    message SaveRequest {    string mazeName = 1;    string mazeType = 2;    string mazeJson = 3;    int32 userID = 4;  }    message SuccessAck {    bool success = 1;  } |

### SolveMaze.proto

|  |
| --- |
| syntax = "proto3";    option csharp\_namespace = "Server";    package greet;    service MazeSolver {  rpc SolveMaze (SolveRequest) returns (Path);  }    message SolveRequest {  string maze = 1;  string algorithm = 2;  string mazeGenerationAlgorithm = 3;  }    message Path {  string path = 1;  } |

# Client Code

## Classes

These are the skeletonised classes from the server that the client uses for building JSON mazes back into objects.

### Coordinate.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Client\_Mazes  {      internal class Coordinate      {          [JsonConstructor]          public Coordinate()          {            }          public Coordinate(int xPos, int yPos) {              this.xPos = xPos;              this.yPos = yPos;              visited = false;          }          public Coordinate(Tuple<int, int> pos) {              xPos = pos.Item1;              yPos = pos.Item2;              visited = false;          }            #region Properties          private int xPos;          public int Xpos {              get { return xPos; }              set { xPos = value; }          }            private int yPos;          public int Ypos {              get { return yPos; }              set { yPos = value; }          }            private bool visited;          public bool Visited {              get { return visited; }              set { visited = value; }          }          #endregion            #region Methods          public bool Equals(Coordinate target) {              return xPos == target.xPos && yPos == target.yPos;          }          #endregion      }  } |

### Globals.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace Client  {      internal static class Globals      {          public static string? g\_username = null;          public static int? g\_userID = null;            public const string g\_version = "v1.5";            public const int g\_cellWidth = 10;          public const int g\_cellHeight = 10;            public const int g\_keySize = 64;          public const int g\_iterations = 350000;      }  } |

### Growing Tree Generation.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Client\_Mazes  {      internal class GrowingTreeGeneration : Maze      {          List<Coordinate> cellsInMaze = new();            [JsonConstructor]          public GrowingTreeGeneration() {            }          public GrowingTreeGeneration(int cellWidth, int cellHeight) {              MazeCellWidth = cellWidth;              MazeCellHeight = cellHeight;              rgen = new();          }      }  } |

### Maze.cs

|  |
| --- |
| namespace Client\_Mazes  {      internal abstract class Maze {          #region Properties          private int mazeActualWidth;          public int MazeActualWidth {              get { return mazeActualWidth; }              set { mazeActualWidth = value; }          }            private int mazeActualHeight;          public int MazeActualHeight {              get { return mazeActualHeight; }              set { mazeActualHeight = value; }          }            private int mazeCellWidth;          public int MazeCellWidth {              get { return mazeCellWidth; }              set { mazeCellWidth = value; }          }            private int mazeCellHeight;          public int MazeCellHeight {              get { return mazeCellHeight; }              set { mazeCellHeight = value; }          }            private bool[,]? mazeWalls;          public bool[,]? MazeWalls {              get { return mazeWalls; }              set { mazeWalls = value; }          }            private Coordinate[,]? mazeCoordinates;          public Coordinate[,]? MazeCoordinates {              get { return mazeCoordinates; }              set { mazeCoordinates = value; }          }            private Coordinate? mazeEntranceCoordinate;          public Coordinate? MazeEntranceCoordinate {              get { return mazeEntranceCoordinate; }              set { mazeEntranceCoordinate = value; }          }            private Coordinate? mazeExitCoordinate;          public Coordinate? MazeExitCoordinate {              get { return mazeExitCoordinate; }              set { mazeExitCoordinate = value;}          }            protected Random rgen = new();          #endregion            #region Methods            public virtual void ResetVisited() {              foreach (Coordinate v in mazeCoordinates) {                  v.Visited = false;              }          }          #endregion      }  } |

### Recursive Backtrack Generation.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Client\_Mazes  {      internal class RecursiveBacktrackGeneration : Maze      {          [JsonConstructor]          public RecursiveBacktrackGeneration()          {            }            public RecursiveBacktrackGeneration(int cellWidth, int cellHeight){              MazeCellWidth = cellWidth;              MazeCellHeight = cellHeight;              rgen = new();          }      }  } |

### Wilsons Generation.cs

|  |
| --- |
| using Newtonsoft.Json;  namespace Client\_Mazes  {      internal class WilsonsGeneration : Maze      {          List<Coordinate> cellsInMaze = new();          bool exitAtBorder;            [JsonConstructor]          public WilsonsGeneration() {            }          public WilsonsGeneration(int cellWidth, int cellHeight, bool exitAtBorder) {              MazeCellWidth = cellWidth;              MazeCellHeight = cellHeight;              rgen = new();              this.exitAtBorder = exitAtBorder;          }      }  } |

### Program.cs

The main entry point for the Forms client.

|  |
| --- |
| //The client will operate over port 7178    namespace Client  {      internal static class Program      {          [STAThread]          static void Main()          {              ApplicationConfiguration.Initialize();              Application.Run(new frm\_mazeLogin());          }      }  } |

## Form Code

### MazeDisplay.cs

|  |
| --- |
| using Client\_Mazes;  using Grpc.Core;  using Grpc.Net.Client;  using Newtonsoft.Json;  using Server;  using System.Diagnostics;    namespace Client  {      public partial class frm\_mazeDisplay : Form      {          private readonly Maze maze;          private Coordinate player;          private List<Coordinate> solution;          private string mazeType;            private bool solved = false;          private bool startedManualSolve = false;          private Stopwatch sw = new();            //forces form to fully render before displaying, removing flickering.          protected override CreateParams CreateParams {              get {                  CreateParams @params = base.CreateParams;                  @params.ExStyle |= 0x2000000;                  return @params;              }          }            public frm\_mazeDisplay(string mazeToDisplay, string mazeType) {              InitializeComponent();                this.mazeType = mazeType;                switch (mazeType) {                  case "Recursive Backtrack":                      maze = JsonConvert.DeserializeObject<RecursiveBacktrackGeneration>(mazeToDisplay);                      break;                  case "Wilson's":                      maze = JsonConvert.DeserializeObject<WilsonsGeneration>(mazeToDisplay);                      break;                  case "Growing Tree":                      maze = JsonConvert.DeserializeObject<GrowingTreeGeneration>(mazeToDisplay);                      break;              }                player = new Coordinate(maze.MazeEntranceCoordinate.Xpos, maze.MazeEntranceCoordinate.Ypos);                btn\_requestSolve.Enabled = false;              cbx\_solveType.DropDownStyle = ComboBoxStyle.DropDownList;          }            private async void btn\_requestSolve\_Click(object sender, EventArgs e) {              player = null;              string mazeToSolve = JsonConvert.SerializeObject(maze);              using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new MazeSolver.MazeSolverClient(channel);              var reply = await client.SolveMazeAsync(new SolveRequest {                  Maze = mazeToSolve,                  Algorithm = cbx\_solveType.Text,                  MazeGenerationAlgorithm = mazeType              });                HandleSolveRender(reply);          }            private void HandleSolveRender(Server.Path reply) {              solution = JsonConvert.DeserializeObject<List<Coordinate>>(reply.Path\_);                solved = true;                btn\_close.Enabled = true;              btn\_requestSolve.Enabled = false;              btn\_left.Enabled = false;              btn\_right.Enabled = false;              btn\_up.Enabled = false;              btn\_down.Enabled = false;                sw = null;                tlp\_MazeDisplay.Refresh();          }            private void SetDisplaySize() {              Width = (45 + Globals.g\_cellWidth \* maze.MazeActualWidth > 510) ? 45 + Globals.g\_cellWidth \* maze.MazeActualWidth : 510;              Height = 145 + Globals.g\_cellHeight \* maze.MazeActualHeight;              pnl\_mazeContainer.Width = Globals.g\_cellWidth \* maze.MazeActualWidth + 5;              pnl\_mazeContainer.Height = Globals.g\_cellHeight \* maze.MazeActualHeight + 5;              pnl\_mazeContainer.Location = new Point(10, 80);          }            private void tlp\_MazeDisplay\_CellPaint(object sender, TableLayoutCellPaintEventArgs e) {              if (player != null && player.Equals(new Coordinate(e.Column, e.Row))) //Draw player.                  e.Graphics.FillRectangle(Brushes.Blue, e.CellBounds); //Blue                else if (maze.MazeEntranceCoordinate.Equals(new Coordinate(e.Column, e.Row))) //Draw entrance.                  e.Graphics.FillRectangle(Brushes.Red, e.CellBounds); //Red                else if (maze.MazeExitCoordinate.Equals(new Coordinate(e.Column, e.Row))) //Draw exit.                  e.Graphics.FillRectangle(Brushes.LawnGreen, e.CellBounds); //LawnGreen                else if (maze.MazeWalls[e.Row, e.Column]) //Draw wall.                  e.Graphics.FillRectangle(Brushes.Black, e.CellBounds); //Black                else if (solution != null) { //Draw solution.                  foreach (Coordinate c in solution) {                      if (c.Xpos == e.Column && c.Ypos == e.Row)                          e.Graphics.FillRectangle(Brushes.Purple, e.CellBounds); //Purple                  }              }                else //Draw path.                  e.Graphics.FillRectangle(Brushes.White, e.CellBounds); //White          }            private void frm\_mazeDisplay\_Load(object sender, EventArgs e) {              SetDisplaySize();                Text = $"MazeClient {Globals.g\_version}";                tlp\_MazeDisplay.ColumnStyles.Clear();              tlp\_MazeDisplay.RowStyles.Clear();                tlp\_MazeDisplay.RowCount = maze.MazeActualHeight;              tlp\_MazeDisplay.ColumnCount = maze.MazeActualWidth;                for (int i = 0; i < maze.MazeActualHeight; i++)                  tlp\_MazeDisplay.RowStyles.Add(new RowStyle(SizeType.Absolute, Globals.g\_cellHeight));              for (int i = 0; i < maze.MazeActualWidth; i++)                  tlp\_MazeDisplay.ColumnStyles.Add(new ColumnStyle(SizeType.Absolute, Globals.g\_cellWidth));            }          private void btn\_close\_Click(object sender, EventArgs e) {              Close();          }            private async void CheckSolved() {              if (!startedManualSolve) {                  startedManualSolve = true;                  HandleTimer();              }                if (!player.Equals(maze.MazeExitCoordinate)) return;                solved = true;              lbl\_solved.ForeColor = Color.Green;              lbl\_solved.Text = "Solved!";              btn\_close.Enabled = true;              btn\_requestSolve.Enabled = false;              btn\_left.Enabled = false;              btn\_right.Enabled = false;              btn\_up.Enabled = false;              btn\_down.Enabled = false;                using var channel = GrpcChannel.ForAddress("https://localhost:7178");                var clientGlobal = new GlobalStatHandler.GlobalStatHandlerClient(channel);              try {                  var replyGlobal = await clientGlobal.UploadTimeAsync(new Time {                      TimeMilliseconds = (int)sw.ElapsedMilliseconds,                      Time\_ = sw.Elapsed.ToString(),                      Username = Globals.g\_username                  });              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) { }                var clientUser = new UserStatHandler.UserStatHandlerClient(channel);              try {                  var replyUser = await clientUser.UserUploadTimeAsync(new UserTime {                      TimeMilliseconds = (int)sw.ElapsedMilliseconds,                      Time = sw.Elapsed.ToString(),                      UserID = (int)Globals.g\_userID                  });              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) { }          }            private void HandleTimer() {              sw.Start();                  ThreadPool.QueueUserWorkItem((state) => {                  try {                      while (!solved)                          Invoke(() => {                              if (sw != null)                                  lbl\_timer.Text = sw.Elapsed.ToString();                              else                                  lbl\_timer.Text = string.Empty;                          });                  }                  catch { }              });          }            private bool IsWall(Coordinate player, string direction) {              try {                  return direction switch {                      "Up" => !maze.MazeWalls[player.Ypos - 1, player.Xpos],                      "Down" => !maze.MazeWalls[player.Ypos + 1, player.Xpos],                      "Left" => !maze.MazeWalls[player.Ypos, player.Xpos - 1],                      "Right" => !maze.MazeWalls[player.Ypos, player.Xpos + 1],                      \_ => true                  };              }              catch { return false; }          }              private void btn\_left\_Click(object sender, EventArgs e) {              if (solved) return;                if (IsWall(player, "Left"))                  player = new Coordinate(player.Xpos - 1, player.Ypos);              tlp\_MazeDisplay.Refresh();              CheckSolved();          }            private void btn\_right\_Click(object sender, EventArgs e) {              if (solved) return;                if (IsWall(player, "Right"))                  player = new Coordinate(player.Xpos + 1, player.Ypos);              tlp\_MazeDisplay.Refresh();              CheckSolved();          }            private void btn\_up\_Click(object sender, EventArgs e) {              if (solved) return;                if (IsWall(player, "Up"))                  player = new Coordinate(player.Xpos, player.Ypos - 1);              tlp\_MazeDisplay.Refresh();              CheckSolved();          }            private void btn\_down\_Click(object sender, EventArgs e) {              if (solved) return;                if (IsWall(player, "Down"))                  player = new Coordinate(player.Xpos, player.Ypos + 1);              tlp\_MazeDisplay.Refresh();              CheckSolved();          }            #region WASD input listeners          private void frm\_mazeDisplay\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_left\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_right\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_up\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_down\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_requestSolve\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void cbx\_solveType\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_serverSave\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            private void btn\_localSave\_KeyDown(object sender, KeyEventArgs e) {              if (solved) return;              HandleKeyDown(e);          }            #endregion            private void HandleKeyDown(KeyEventArgs e) {              switch (e.KeyCode) {                  case Keys.W:                      if (IsWall(player, "Up"))                          player = new Coordinate(player.Xpos, player.Ypos - 1);                      break;                    case Keys.S:                      if (IsWall(player, "Down"))                          player = new Coordinate(player.Xpos, player.Ypos + 1);                      break;                    case Keys.A:                      if (IsWall(player, "Left"))                          player = new Coordinate(player.Xpos - 1, player.Ypos);                      break;                    case Keys.D:                      if (IsWall(player, "Right"))                          player = new Coordinate(player.Xpos + 1, player.Ypos);                      break;                    default:                      break;              }              if (new Keys[] { Keys.W, Keys.A, Keys.S, Keys.D }.Contains(e.KeyCode)) {                  tlp\_MazeDisplay.Refresh();                  CheckSolved();              }          }            private void cbx\_solveType\_SelectedIndexChanged(object sender, EventArgs e) {              if (cbx\_solveType.Text != string.Empty) {                  btn\_requestSolve.Enabled = true;              }              else {                  btn\_requestSolve.Enabled = false;              }          }            private void btn\_localSave\_Click(object sender, EventArgs e) {              Coordinate tempPlayer = player;              player = null;              tlp\_MazeDisplay.Refresh();                int width = tlp\_MazeDisplay.Size.Width;              int height = tlp\_MazeDisplay.Size.Height;                Bitmap mazeToSave = new(width, height);              tlp\_MazeDisplay.DrawToBitmap(mazeToSave, new Rectangle(0, 0, width, height));                SaveFileDialog sf = new();              sf.Filter = "JPEG Image (.jpeg)|\*.jpeg|Png Image (.png)|\*.png";              sf.ShowDialog();              var path = sf.FileName;                mazeToSave.Save(path);                player = tempPlayer;              tlp\_MazeDisplay.Refresh();          }            private async void btn\_serverSave\_Click(object sender, EventArgs e) {                if (txb\_mazeName.Text == string.Empty) {                  lbl\_error.ForeColor = Color.Red;                  lbl\_error.Text = "Maze needs a name!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(1000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return;              }                this.maze.ResetVisited();                string maze = JsonConvert.SerializeObject(this.maze);              using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new Saver.SaverClient(channel);              try {                  var reply = await client.SaveMazeAsync(new SaveRequest {                      MazeName = txb\_mazeName.Text,                      MazeType = mazeType,                      MazeJson = maze,                      UserID = (int)Globals.g\_userID                  },                  deadline: DateTime.UtcNow.AddSeconds(3));                    if (reply.Success) {                      lbl\_error.ForeColor = Color.Green;                      lbl\_error.Text = "Success!";                      ThreadPool.QueueUserWorkItem((state) => {                          Thread.Sleep(1000);                          Invoke(() => lbl\_error.Text = string.Empty);                      });                  }                  else {                      lbl\_error.ForeColor = Color.Red;                      lbl\_error.Text = "Error!";                      ThreadPool.QueueUserWorkItem((state) => {                          Thread.Sleep(1000);                          Invoke(() => lbl\_error.Text = string.Empty);                      });                  }              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_error.ForeColor = Color.Red;                  lbl\_error.Text = "Error!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(1000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });              }              catch (InvalidOperationException) { }          }      }  } |

### MazeLogin.cs

|  |
| --- |
| using Grpc.Core;  using Grpc.Net.Client;  using Server;    namespace Client  {      public partial class frm\_mazeLogin : Form      {          public frm\_mazeLogin() {              InitializeComponent();          }            private void frm\_mazeLogin\_Load(object sender, EventArgs e) {              Text = $"MazeClient {Globals.g\_version}";              lbl\_error.ForeColor = Color.Red;          }            private void llb\_register\_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e) {              Form register = new frm\_mazeRegister();              register.ShowDialog();          }            private async void btn\_login\_Click(object sender, EventArgs e) {              using var channel = GrpcChannel.ForAddress("https://localhost:7178");                var clientGreet = new Greeter.GreeterClient(channel);              try {                  var replyGreet = await clientGreet.SayHelloAsync(new HelloRequest                  { Name = Environment.MachineName },                      deadline: DateTime.UtcNow.AddSeconds(3));              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_error.Text = "Cannot connect to\nserver!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return;              }                var clientLogin = new LoginHandler.LoginHandlerClient(channel);              var replyLogin = await clientLogin.LoginAsync(new Credentials {                  Username = txb\_username.Text,                  Password = txb\_password.Text              });                if (replyLogin.LoggedIn) {                  Globals.g\_userID = replyLogin.UserID;                  Globals.g\_username = txb\_username.Text;                  Form mazeParams = new frm\_mazeParams();                  Hide();                  mazeParams.Closed += (s, args) => Close();                  mazeParams.Show();              }              else {                  lbl\_error.Text = "Username or Password\nincorrect!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });              }          }      }  } |

### MazeParameter.cs

|  |
| --- |
| using Client\_Mazes;  using Grpc.Core;  using Grpc.Net.Client;  using Newtonsoft.Json;  using Server;  using System.Windows.Forms.DataVisualization.Charting;    namespace Client  {      public partial class frm\_mazeParams : Form      {          private CancellationTokenSource cts = new CancellationTokenSource();          private bool connected = false;          private bool algorithmSelected = false;          private bool exitSelected = false;            public frm\_mazeParams() {              InitializeComponent();          }            private void MazeClient\_Load(object sender, EventArgs e) {              btn\_requestMaze.Enabled = false;                cbx\_algorithm.DropDownStyle = ComboBoxStyle.DropDownList;              cbx\_whereExit.DropDownStyle = ComboBoxStyle.DropDownList;              cbx\_statType.DropDownStyle = ComboBoxStyle.DropDownList;              cbx\_loadedMazes.DropDownStyle = ComboBoxStyle.DropDownList;                Text = $"MazeClient {Globals.g\_version} : Welcome {Globals.g\_username}";                ThreadPool.QueueUserWorkItem(async (state) => {                  while (true) {                        using var channel = GrpcChannel.ForAddress("https://localhost:7178");                      var client = new Greeter.GreeterClient(channel);                      try {                          var reply = await client.SayHelloAsync(new HelloRequest {                              Name = Environment.MachineName                          },                              deadline: DateTime.UtcNow.AddSeconds(3));                          Invoke(() => {                              lbl\_connectionError.Text = "Connected to server!";                              lbl\_connectionError.ForeColor = Color.Green;                              connected = true;                              HandleAllowSend();                          });                        }                      catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                          Invoke(() => {                              HandleServerError();                          });                      }                      catch (ObjectDisposedException) { }                        Thread.Sleep(10000);                  }              }, cts.Token);          }            private void HandleServerError() {              lbl\_connectionError.Text = "Not connected to server!";              lbl\_connectionError.ForeColor = Color.Red;              connected = false;              HandleAllowSend();          }            private void HandleAllowSend() {              if (connected &&                  algorithmSelected &&                  exitSelected) {                  btn\_requestMaze.Enabled = true;              }              else {                  btn\_requestMaze.Enabled = false;              }          }            private async void btn\_requestMaze\_Click(object sender, EventArgs e) {              btn\_requestMaze.Enabled = false;              string mazeToDisplay = await RequestMaze();              if (mazeToDisplay != string.Empty)                  ChangeForm(mazeToDisplay, cbx\_algorithm.Text);          }            private async Task<string> RequestMaze() {              using var channel = GrpcChannel.ForAddress("https://localhost:7178");                var clientBuild = new MazeBuilder.MazeBuilderClient(channel);              BuiltMaze replyBuild;              try {                  replyBuild = await clientBuild.BuildMazeAsync(new MazeRequest {                      Algorithm = cbx\_algorithm.Text,                      Width = (int)nud\_mazeWidth.Value,                      Height = (int)nud\_mazeHeight.Value,                      RemoveWalls = (int)nud\_removeWalls.Value,                      ExitLocation = cbx\_whereExit.Text                  }, deadline: DateTime.UtcNow.AddSeconds(3));              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_connectionError.Text = "Not connected to server!";                  lbl\_connectionError.ForeColor = Color.Red;                  HandleServerError();                  return string.Empty;              }                var clientStatsGlobal = new GlobalStatHandler.GlobalStatHandlerClient(channel);              try {                  var replyStatsGlobal = await clientStatsGlobal.IncrementMazeAsync(new MazeType {                      MazeType\_ = cbx\_algorithm.Text                  }, deadline: DateTime.UtcNow.AddSeconds(3));              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) { }                var clientStatsUser = new UserStatHandler.UserStatHandlerClient(channel);              try {                  var replyStatsUser = await clientStatsUser.UserIncrementMazeAsync(new UserMazeType {                      MazeType = cbx\_algorithm.Text,                      UserID = (int)Globals.g\_userID                  },                      deadline: DateTime.UtcNow.AddSeconds(3));              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) { }                return replyBuild.Maze;          }            private void ChangeForm(string maze, string algorithm) {              Form mazeDisplay = new frm\_mazeDisplay(maze, algorithm);              mazeDisplay.FormClosed += (s, args) => HandleAllowSend();              mazeDisplay.ShowDialog();              mazeDisplay.Focus();          }            private void frm\_mazeParams\_FormClosing(object sender, FormClosingEventArgs e) {              cts.Cancel();              cts.Dispose();          }            private async void btn\_getMazes\_Click(object sender, EventArgs e) {              cbx\_loadedMazes.Items.Clear();                using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new GetterMazes.GetterMazesClient(channel);              try {                  var reply = await client.GetMazesAsync(new Request {                      UserID = (int)Globals.g\_userID                  }, deadline: DateTime.UtcNow.AddSeconds(3));                  var mazes = JsonConvert.DeserializeObject<List<(int mazeID,string mazeName)>>(reply.Mazes);                  foreach (var maze in mazes) {                      cbx\_loadedMazes.Items.Add($"{maze.mazeID}: {maze.mazeName}");                  }                  lbl\_loadError.Text = $"Found {mazes.Count} mazes!";              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_loadError.Text = "Error fetching\nmazes!";              }          }            private async void btn\_loadMaze\_Click(object sender, EventArgs e) {              if (cbx\_loadedMazes.Text == string.Empty) return;                using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new LoaderMazes.LoaderMazesClient(channel);              try {                  var reply = await client.LoadMazeAsync(new LoadRequest {                      UserID = (int)Globals.g\_userID,                      MazeID = Convert.ToInt32(cbx\_loadedMazes.Text.Split(':')[0])                  },                      deadline: DateTime.UtcNow.AddSeconds(3));                    ChangeForm(reply.Maze, reply.MazeGenAlg);                }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_loadError.Text = "Error loading\nmazes!";              }          }            private async void btn\_deleteMaze\_Click(object sender, EventArgs e) {              if (cbx\_loadedMazes.Text == string.Empty) return;                using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new DeleterMazes.DeleterMazesClient(channel);              try {                  var reply = await client.DeleteMazeAsync(new DeleteRequest {                      UserID = (int)Globals.g\_userID,                      MazeID = Convert.ToInt32(cbx\_loadedMazes.Text.Split(':')[0])                  },                      deadline: DateTime.UtcNow.AddSeconds(3));                  if (reply.Success) {                      lbl\_loadError.Text = "Deleted maze\nsuccessfully!";                      cbx\_loadedMazes.Items.RemoveAt(cbx\_loadedMazes.SelectedIndex);                      cbx\_loadedMazes.SelectedIndex = -1;                  }                  else {                      lbl\_loadError.Text = "Error Deleting\nmaze!";                  }                }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_loadError.Text = "Error Deleting\nmaze!";              }          }            private async void btn\_displayStats\_Click(object sender, EventArgs e) {              if (cbx\_statType.Text == string.Empty) return;                using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new StatsGetter.StatsGetterClient(channel);                if (cbx\_statType.Text == "Mazes Generated" && chbx\_globalStats.Checked) {                  try {                      var reply = await client.GetGlobalMazesGeneratedAsync(                          new GetGlobalMazesGeneratedRequest { },                          deadline: DateTime.UtcNow.AddSeconds(3));                        Chart chrt\_mazesGenerated = HandleMazesGeneratedStatsView();                      Series series = new("Maze Types Generated");                      series.ChartType = SeriesChartType.Pie;                        string[] segmentNames = {                          "Recursive Backtrack",                          "Growing tree",                          "Wilson's"                      };                      double[] segmentValues = {                          reply.RecursiveBacktrackMazesGenerated,                          reply.GrowingTreeMazesGenerated,                          reply.WilsonsMazesGenerated                      };                      series.Points.DataBindXY(segmentNames, segmentValues);                        chrt\_mazesGenerated.Series.Add(series);                  }                  catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                      HandleServerError();                  }              }              else if (cbx\_statType.Text == "Mazes Generated" && !chbx\_globalStats.Checked) {                  try {                      var reply = await client.GetUserMazesGeneratedAsync(new GetUserMazesGeneratedRequest {                          UserID = (int)Globals.g\_userID                      },                          deadline: DateTime.UtcNow.AddSeconds(3));                        Chart chrt\_mazesGenerated = HandleMazesGeneratedStatsView();                      Series series = new("Maze Types Generated");                      series.ChartType = SeriesChartType.Pie;                        string[] segmentNames = {                          "Recursive Backtrack",                          "Growing Tree",                          "Wilson's"                      };                      double[] segmentValues = {                          reply.RecursiveBacktrackMazesGenerated,                          reply.GrowingTreeMazesGenerated,                          reply.WilsonsMazesGenerated                      };                      series.Points.DataBindXY(segmentNames, segmentValues);                        chrt\_mazesGenerated.Series.Add(series);                  }                  catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                      HandleServerError();                  }              }              else if (cbx\_statType.Text == "Best Times" && chbx\_globalStats.Checked) {                  try {                      var reply = await client.GetGlobalTimesAsync(new GetGlobalTimesRequest { },                          deadline: DateTime.UtcNow.AddSeconds(3));                        RichTextBox rtb\_times = HandleTimeStatsView();                      rtb\_times.Text += "Global Best Times\n";                      rtb\_times.Text += $"1st:  {reply.Time1Username} : {reply.Time1DisplayTime}\n" +                          $"2nd:  {reply.Time2Username} : {reply.Time2DisplayTime}\n" +                          $"3rd:  {reply.Time3Username} : {reply.Time3DisplayTime}\n" +                          $"4th:  {reply.Time4Username} : {reply.Time4DisplayTime}\n" +                          $"5th:  {reply.Time5Username} : {reply.Time5DisplayTime}\n" +                          $"6th:  {reply.Time6Username} : {reply.Time6DisplayTime}\n" +                          $"7th:  {reply.Time7Username} : {reply.Time7DisplayTime}\n" +                          $"8th:  {reply.Time8Username} : {reply.Time8DisplayTime}\n" +                          $"9th:  {reply.Time9Username} : {reply.Time9DisplayTime}\n" +                          $"10th: {reply.Time10Username} : {reply.Time10DisplayTime}";                  }                  catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                      HandleServerError();                  }              }              else if (cbx\_statType.Text == "Best Times" && !chbx\_globalStats.Checked) {                  try {                      var reply = await client.GetUserTimesAsync(new GetUserTimesRequest {                          UserID = (int)Globals.g\_userID                      },                          deadline: DateTime.UtcNow.AddSeconds(3));                        RichTextBox rtb\_times = HandleTimeStatsView();                      rtb\_times.Font = new Font("Calibri", 20, FontStyle.Bold);                      rtb\_times.Text += "Your Best Times\n";                      rtb\_times.Font = DefaultFont;                      rtb\_times.Text += $"1st:  {reply.Time1DisplayTime}\n" +                          $"2nd:  {reply.Time2DisplayTime}\n" +                          $"3rd:  {reply.Time3DisplayTime}\n" +                          $"4th:  {reply.Time4DisplayTime}\n" +                          $"5th:  {reply.Time5DisplayTime}\n" +                          $"6th:  {reply.Time6DisplayTime}\n" +                          $"7th:  {reply.Time7DisplayTime}\n" +                          $"8th:  {reply.Time8DisplayTime}\n" +                          $"9th:  {reply.Time9DisplayTime}\n" +                          $"10th: {reply.Time10DisplayTime}";                  }                  catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                      HandleServerError();                  }              }          }            private RichTextBox HandleTimeStatsView() {              RichTextBox rtb\_times;              pnl\_graph.Controls.Clear();              pnl\_graph.Controls.Add(rtb\_times = new());                rtb\_times.Dock = DockStyle.Fill;                return rtb\_times;          }            private Chart HandleMazesGeneratedStatsView() {              Chart chrt\_generatedStats;              pnl\_graph.Controls.Clear();              pnl\_graph.Controls.Add(chrt\_generatedStats = new());                chrt\_generatedStats.Dock = DockStyle.Fill;              chrt\_generatedStats.ChartAreas.Add(new ChartArea("MazeChartArea"));              chrt\_generatedStats.ChartAreas["MazeChartArea"].Area3DStyle.Enable3D = true;                  return chrt\_generatedStats;          }            private void cbx\_algorithm\_SelectedIndexChanged(object sender, EventArgs e) {              algorithmSelected = true;              HandleAllowSend();          }            private void cbx\_whereExit\_SelectedIndexChanged(object sender, EventArgs e) {              exitSelected = true;              HandleAllowSend();          }      }  } |

### MazeRegister.cs

|  |
| --- |
| using Grpc.Net.Client;  using System.Text.RegularExpressions;  using System.Security.Cryptography;  using System.Text;  using Server;  using Grpc.Core;    namespace Client  {      public partial class frm\_mazeRegister : Form      {          HashAlgorithmName hashAlgorithm = HashAlgorithmName.SHA512;            public frm\_mazeRegister() {              InitializeComponent();          }          private void frm\_mazeRegister\_Load(object sender, EventArgs e) {              Text = $"MazeClient {Globals.g\_version}";              lbl\_error.ForeColor = Color.Red;          }            private async void btn\_register\_Click(object sender, EventArgs e) {              using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var clientGreet = new Greeter.GreeterClient(channel);              try {                  var replyGreet = await clientGreet.SayHelloAsync(new HelloRequest {                      Name = Environment.MachineName                  },                      deadline: DateTime.UtcNow.AddSeconds(3));              }              catch (RpcException ex) when (ex.StatusCode == StatusCode.DeadlineExceeded) {                  lbl\_error.Text = "Cannot connect to server!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return;              }                if (await CheckCrendentials()) {                    var password = HashPasword(txb\_password.Text, out var salt);                    var clientRegister = new Registerer.RegistererClient(channel);                  var replyRegister = await clientRegister.RegisterAsync(new Account {                      Username = txb\_username.Text,                      Password = password,                      Salt = Convert.ToHexString(salt),                  });                    if (replyRegister.Success) {                      lbl\_error.ForeColor = Color.Green;                      lbl\_error.Text = "Registered!";                      Task.Delay(1000).Wait();                      Close();                  }                  else {                      lbl\_error.ForeColor = Color.Red;                      lbl\_error.Text = "Failed to register!";                      Task.Delay(1000).Wait();                      lbl\_error.Text = string.Empty;                  }              }          }              private async Task<bool> CheckCrendentials() {              using var channel = GrpcChannel.ForAddress("https://localhost:7178");              var client = new CheckerIfUserExists.CheckerIfUserExistsClient(channel);              var reply = await client.CheckUserAsync(new Query {                  Username = txb\_username.Text              });                if (reply.UserExists) {                  lbl\_error.Text = "Username already taken!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return false;              }              if (txb\_password.Text.Length < 7) {                  lbl\_error.Text = "Password must be at least\n 7 characters!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return false;              }              if (!Regex.IsMatch(txb\_password.Text, @"[!-**\/**:-@[-`{-~]")) {                  lbl\_error.Text = "Password must contain at\n least 1 special character!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return false;              }              if (txb\_password.Text != txb\_confirm.Text) {                  lbl\_error.Text = "Passwords do not match!";                  ThreadPool.QueueUserWorkItem((state) => {                      Thread.Sleep(3000);                      Invoke(() => lbl\_error.Text = string.Empty);                  });                  return false;              }                return true;          }            // SOURCE: https://code-maze.com/csharp-hashing-salting-passwords-best-practices/          string HashPasword(string password, out byte[] salt) {              salt = RandomNumberGenerator.GetBytes(Globals.g\_keySize);              var hash = Rfc2898DeriveBytes.Pbkdf2(                  Encoding.UTF8.GetBytes(password),                  salt,                  Globals.g\_iterations,                  hashAlgorithm,                  Globals.g\_keySize);              return Convert.ToHexString(hash);          }      }  } |

# Server Code

## Service Definitions

### CheckerIfUserExistsService.cs

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class CheckerIfUserExistsService : CheckerIfUserExists.CheckerIfUserExistsBase      {          public override Task<Exists> CheckUser(Query request, ServerCallContext context) {              using SQLiteConnection conn = new(                  "Data Source=mazeData.db; " +                  "Version=3; " +                  "New=True; " +                  "Compress=True; ");              conn.Open();                using SQLiteCommand cmd = conn.CreateCommand();              cmd.CommandText = "SELECT COUNT(\*) FROM User WHERE Username = @username";              cmd.Parameters.AddWithValue("@username", request.Username);                int rowCount = Convert.ToInt32(cmd.ExecuteScalar());                if (rowCount > 0) {                  return Task.FromResult(new Exists { UserExists = true });              }              else {                  return Task.FromResult(new Exists { UserExists = false });              }          }      }  } |

### DeleteMazeService

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class DeleteMazeService : DeleterMazes.DeleterMazesBase      {          public override Task<SuccessAcknowledge> DeleteMaze(DeleteRequest request, ServerCallContext context) {              try {                  using (SQLiteConnection conn = new(                      "Data Source= mazeData.db; " +                      "Version = 3; " +                      "New = True; " +                      "Compress = True; ")) {                      conn.Open();                      using SQLiteCommand cmd = conn.CreateCommand();                        cmd.CommandText = @"DELETE FROM Mazes                                          WHERE @MazeID = MazeID                                          AND @UserID = UID";                      cmd.Parameters.AddWithValue("@MazeID", request.MazeID);                      cmd.Parameters.AddWithValue("@UserID", request.UserID);                      cmd.ExecuteNonQuery();                  }                    return Task.FromResult(new SuccessAcknowledge { Success = true });                }              catch (Exception) {                  return Task.FromResult(new SuccessAcknowledge { Success = false });              }          }      }  } |

### GetMazesService.cs

|  |
| --- |
| using Grpc.Core;  using Newtonsoft.Json;  using System.Data.SQLite;    namespace Server.Services  {      public class GetMazesService : GetterMazes.GetterMazesBase      {          public override Task<MazesList> GetMazes(Request request, ServerCallContext context) {              List<(int mazeID, string mazeName)> mazes = new();                using (SQLiteConnection conn = new(                  "Data Source= mazeData.db; " +                  "Version = 3; " +                  "New = True; " +                  "Compress = True; ")) {                  conn.Open();                    using SQLiteCommand cmd = conn.CreateCommand();                  cmd.CommandText = "SELECT MazeID, MazeName FROM Mazes WHERE @UID = UID";                  cmd.Parameters.AddWithValue("@UID", request.UserID);                  using SQLiteDataReader reader = cmd.ExecuteReader();                  while (reader.Read()) {                      mazes.Add((reader.GetInt32(0), reader.GetString(1)));                  }              }                return Task.FromResult(new MazesList {                  Mazes = JsonConvert.SerializeObject(mazes)              });          }      }  } |

### GetStatsService.cs

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class GetStatsService : StatsGetter.StatsGetterBase      {          public override Task<GlobalMazesGenerated> GetGlobalMazesGenerated(GetGlobalMazesGeneratedRequest request, ServerCallContext context) {              int recursiveBacktrackMazesGenerated = 0;              int growingTreeMazesGenerated = 0;              int wilsonsMazesGenerated = 0;                using SQLiteConnection conn = new SQLiteConnection(                  "Data Source=mazeData.db; " +                  "Version=3; " +                  "New=True; " +                  "Compress=True; ");              conn.Open();              using SQLiteCommand cmd = conn.CreateCommand();              cmd.CommandText = @"SELECT RecursiveBacktrackMazesGenerated,                                         GrowingTreeMazesGenerated,                                         WilsonsMazesGenerated                                  FROM GlobalStats";              using SQLiteDataReader reader = cmd.ExecuteReader();              while (reader.Read()) {                  recursiveBacktrackMazesGenerated = reader.GetInt32(0);                  growingTreeMazesGenerated = reader.GetInt32(1);                  wilsonsMazesGenerated = reader.GetInt32(2);              }              conn.Close();                return Task.FromResult(new GlobalMazesGenerated {                  RecursiveBacktrackMazesGenerated = recursiveBacktrackMazesGenerated,                  GrowingTreeMazesGenerated = growingTreeMazesGenerated,                  WilsonsMazesGenerated = wilsonsMazesGenerated              });          }            public override Task<GlobalTimes> GetGlobalTimes(GetGlobalTimesRequest request, ServerCallContext context) {                string time1DisplayTime = string.Empty;              string time2DisplayTime = string.Empty;              string time3DisplayTime = string.Empty;              string time4DisplayTime = string.Empty;              string time5DisplayTime = string.Empty;              string time6DisplayTime = string.Empty;              string time7DisplayTime = string.Empty;              string time8DisplayTime = string.Empty;              string time9DisplayTime = string.Empty;              string time10DisplayTime = string.Empty;              string time1Username = string.Empty;              string time2Username = string.Empty;              string time3Username = string.Empty;              string time4Username = string.Empty;              string time5Username = string.Empty;              string time6Username = string.Empty;              string time7Username = string.Empty;              string time8Username = string.Empty;              string time9Username = string.Empty;              string time10Username = string.Empty;                using SQLiteConnection conn = new("" +                  "Data Source=mazeData.db; " +                  "Version=3; " +                  "New=True; " +                  "Compress=True; ");              conn.Open();              using SQLiteCommand cmd = conn.CreateCommand();              cmd.CommandText = @"SELECT Time1Display,                                         Time2Display,                                         Time3Display,                                         Time4Display,                                         Time5Display,                                         Time6Display,                                         Time7Display,                                         Time8Display,                                         Time9Display,                                         Time10Display,                                         Time1Name,                                         Time2Name,                                         Time3Name,                                         Time4Name,  Time5Name,                                         Time6Name,                                         Time7Name,                                         Time8Name,                                         Time9Name,                                         Time10Name                                  FROM GlobalStats";              using SQLiteDataReader reader = cmd.ExecuteReader();                while (reader.Read()) {                  time1DisplayTime = reader.GetString(0);                  time2DisplayTime = reader.GetString(1);                  time3DisplayTime = reader.GetString(2);                  time4DisplayTime = reader.GetString(3);                  time5DisplayTime = reader.GetString(4);                  time6DisplayTime = reader.GetString(5);                  time7DisplayTime = reader.GetString(6);                  time8DisplayTime = reader.GetString(7);                  time9DisplayTime = reader.GetString(8);                  time10DisplayTime = reader.GetString(9);                  time1Username = reader.GetString(10);                  time2Username = reader.GetString(11);                  time3Username = reader.GetString(12);                  time4Username = reader.GetString(13);                  time5Username = reader.GetString(14);                  time6Username = reader.GetString(15);                  time7Username = reader.GetString(16);                  time8Username = reader.GetString(17);                  time9Username = reader.GetString(18);                  time10Username = reader.GetString(19);              }                return Task.FromResult(new GlobalTimes {                  Time1DisplayTime = time1DisplayTime,                  Time2DisplayTime = time2DisplayTime,                  Time3DisplayTime = time3DisplayTime,                  Time4DisplayTime = time4DisplayTime,                  Time5DisplayTime = time5DisplayTime,                  Time6DisplayTime = time6DisplayTime,                  Time7DisplayTime = time7DisplayTime,                  Time8DisplayTime = time8DisplayTime,                  Time9DisplayTime = time9DisplayTime,                  Time10DisplayTime = time10DisplayTime,                  Time1Username = time1Username,                  Time2Username = time2Username,                  Time3Username = time3Username,                  Time4Username = time4Username,                  Time5Username = time5Username,                  Time6Username = time6Username,                  Time7Username = time7Username,                  Time8Username = time8Username,                  Time9Username = time9Username,                  Time10Username = time10Username              });          }            public override Task<UserMazesGenerated> GetUserMazesGenerated(GetUserMazesGeneratedRequest request, ServerCallContext context) {              int recursiveBacktrackMazesGenerated = 0;              int growingTreeMazesGenerated = 0;              int wilsonsMazesGenerated = 0;                using SQLiteConnection conn = new(                  "Data Source=mazeData.db; " +                  "Version=3; " +                  "New=True; " +                  "Compress=True; ");              conn.Open();              using SQLiteCommand cmd = conn.CreateCommand();              cmd.CommandText = @$"SELECT RecursiveBacktrackMazesGenerated,                                          GrowingTreeMazesGenerated,                                          WilsonsMazesGenerated                                  FROM UserStats                                  WHERE UID = {request.UserID}";              using SQLiteDataReader reader = cmd.ExecuteReader();              while (reader.Read()) {                  recursiveBacktrackMazesGenerated = reader.GetInt32(0);                  growingTreeMazesGenerated = reader.GetInt32(1);                  wilsonsMazesGenerated = reader.GetInt32(2);              }              conn.Close();                return Task.FromResult(new UserMazesGenerated {                  RecursiveBacktrackMazesGenerated = recursiveBacktrackMazesGenerated,                  GrowingTreeMazesGenerated = growingTreeMazesGenerated,                  WilsonsMazesGenerated = wilsonsMazesGenerated              });          }            public override Task<UserTimes> GetUserTimes(GetUserTimesRequest request, ServerCallContext context) {                string time1DisplayTime = string.Empty;              string time2DisplayTime = string.Empty;              string time3DisplayTime = string.Empty;              string time4DisplayTime = string.Empty;              string time5DisplayTime = string.Empty;              string time6DisplayTime = string.Empty;              string time7DisplayTime = string.Empty;              string time8DisplayTime = string.Empty;              string time9DisplayTime = string.Empty;              string time10DisplayTime = string.Empty;                using SQLiteConnection conn = new("" +                  "Data Source=mazeData.db; " +                  "Version=3; " +                  "New=True; " +                  "Compress=True; ");              conn.Open();              using SQLiteCommand cmd = conn.CreateCommand();              cmd.CommandText = @$"SELECT Time1Display,                                          Time2Display,                                          Time3Display,                                          Time4Display,                                          Time5Display,                                          Time6Display,                                          Time7Display,                                          Time8Display,                                          Time9Display,                                          Time10Display                                  FROM UserStats                                  WHERE UID = {request.UserID}";              using SQLiteDataReader reader = cmd.ExecuteReader();                while (reader.Read()) {                  time1DisplayTime = reader.GetString(0);                  time2DisplayTime = reader.GetString(1);                  time3DisplayTime = reader.GetString(2);                  time4DisplayTime = reader.GetString(3);                  time5DisplayTime = reader.GetString(4);                  time6DisplayTime = reader.GetString(5);                  time7DisplayTime = reader.GetString(6);                  time8DisplayTime = reader.GetString(7);                  time9DisplayTime = reader.GetString(8);                  time10DisplayTime = reader.GetString(9);              }                return Task.FromResult(new UserTimes {                  Time1DisplayTime = time1DisplayTime,                  Time2DisplayTime = time2DisplayTime,                  Time3DisplayTime = time3DisplayTime,                  Time4DisplayTime = time4DisplayTime,                  Time5DisplayTime = time5DisplayTime,                  Time6DisplayTime = time6DisplayTime,                  Time7DisplayTime = time7DisplayTime,                  Time8DisplayTime = time8DisplayTime,                  Time9DisplayTime = time9DisplayTime,                  Time10DisplayTime = time10DisplayTime              });          }      }  } |

### GlobalStatHandlerService.cs

|  |
| --- |
| using Google.Protobuf.WellKnownTypes;  using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class GlobalStatHandlerService : GlobalStatHandler.GlobalStatHandlerBase      {          public override Task<Empty> IncrementMaze(MazeType request, ServerCallContext context) {              using (SQLiteConnection conn = new("" +                  "Data Source= mazeData.db; " +                  "Version = 3; " +                  "New = True; " +                  "Compress = True; ")) {                  conn.Open();                  using SQLiteCommand cmd = conn.CreateCommand();                  switch (request.MazeType\_) {                      case "Recursive Backtrack":                          cmd.CommandText = @"UPDATE GlobalStats                                              SET RecursiveBacktrackMazesGenerated = RecursiveBacktrackMazesGenerated + 1";                          cmd.ExecuteNonQuery();                          break;                      case "Wilson's":                          cmd.CommandText = @"UPDATE GlobalStats                                              SET WilsonsMazesGenerated = WilsonsMazesGenerated + 1";                          cmd.ExecuteNonQuery();                          break;                      case "Growing Tree":                          cmd.CommandText = @"UPDATE GlobalStats                                              SET GrowingTreeMazesGenerated = GrowingTreeMazesGenerated + 1";                          cmd.ExecuteNonQuery();                          break;                  }              }              return Task.FromResult(new Empty());            }            public override Task<Empty> UploadTime(Time request, ServerCallContext context) {              List<int> milliseconds = new();              List<string> displayTimes = new();              List<string> usernames = new();                using (SQLiteConnection conn = new(                  "Data Source= mazeData.db; " +                  "Version = 3; " +                  "New = True; " +                  "Compress = True; ")) {                  conn.Open();                  using SQLiteCommand cmd = conn.CreateCommand();                    cmd.CommandText = @"SELECT Time1Milliseconds,                                             Time2Milliseconds,                                             Time3Milliseconds,                                             Time4Milliseconds,                                             Time5Milliseconds,                                             Time6Milliseconds,                                             Time7Milliseconds,                                             Time8Milliseconds,                                             Time9Milliseconds,                                             Time10Milliseconds                                      FROM GlobalStats";                  using SQLiteDataReader readerMilliseconds = cmd.ExecuteReader();                  while (readerMilliseconds.Read()) {                      for (int i = 0; i < 10; i++) {                          milliseconds.Add(readerMilliseconds.GetInt32(i));                      }                  }                  readerMilliseconds.Close();                    cmd.CommandText = @"SELECT Time1Display,                                             Time2Display,                                             Time3Display,                                             Time4Display,                                             Time5Display,                                             Time6Display,                                             Time7Display,                                             Time8Display,                                             Time9Display,                                             Time10Display                                      FROM GlobalStats";                  using SQLiteDataReader readerDisplay = cmd.ExecuteReader();                  while (readerDisplay.Read()) {                      for (int i = 0; i < 10; i++) {                          displayTimes.Add(readerDisplay.GetString(i));                      }                  }                  readerDisplay.Close();                    cmd.CommandText = @"SELECT Time1Name,                                             Time2Name,                                             Time3Name,                                             Time4Name,                                             Time5Name,                                             Time6Name,                                             Time7Name,                                             Time8Name,                                             Time9Name,                                             Time10Name                                      FROM GlobalStats";                  using SQLiteDataReader readerUsername = cmd.ExecuteReader();                  while (readerUsername.Read()) {                      for (int i = 0; i < 10; i++) {                          usernames.Add(readerUsername.GetString(i));                      }                  }                  readerUsername.Close();                    int place = -1;                  for (int i = 0; i < 10; i++) {                      if (request.TimeMilliseconds < milliseconds[i] || milliseconds[i] == -1) {                          milliseconds.Insert(i, request.TimeMilliseconds);                          place = i;                          if (milliseconds.Count > 10) milliseconds.RemoveAt(milliseconds.Count - 1);                          break;                      }                  }                    if (place == -1) { return Task.FromResult(new Empty()); }                    displayTimes.Insert(place, request.Time\_);                  if (displayTimes.Count > 10) displayTimes.RemoveAt(displayTimes.Count - 1);                    usernames.Insert(place, request.Username);                  if (usernames.Count > 10) usernames.RemoveAt(usernames.Count - 1);                        for (int i = 0; i < 10; i++) {                      cmd.CommandText = $@"UPDATE GlobalStats                                           SET Time{i + 1}Milliseconds = {milliseconds[i]}";                      cmd.ExecuteNonQuery();                      cmd.CommandText = $@"UPDATE GlobalStats                                           SET Time{i + 1}Display = '{displayTimes[i]}'";                      cmd.ExecuteNonQuery();                      cmd.CommandText = $@"UPDATE GlobalStats                                           SET Time{i + 1}Name = '{usernames[i]}'";                      cmd.ExecuteNonQuery();                  }              }              return Task.FromResult(new Empty());          }      }  } |

### GreeterService.cs

|  |
| --- |
| using Grpc.Core;  using Server;    namespace Server.Services  {      public class GreeterService : Greeter.GreeterBase      {          private readonly ILogger<GreeterService> \_logger;          public GreeterService(ILogger<GreeterService> logger)          {              \_logger = logger;          }            public override Task<HelloReply> SayHello(HelloRequest request, ServerCallContext context)          {              return Task.FromResult(new HelloReply              {                  Message = "Hello " + request.Name              });          }      }  } |

### HandleUserStatsService.cs

|  |
| --- |
| using Google.Protobuf.WellKnownTypes;  using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class HandleUserStatsService : UserStatHandler.UserStatHandlerBase      {          public override Task<Empty> UserIncrementMaze(UserMazeType request, ServerCallContext context) {              using (SQLiteConnection conn = new(                  "Data Source= mazeData.db; " +                  "Version = 3; " +                  "New = True; " +                  "Compress = True; ")) {                  conn.Open();                  using SQLiteCommand cmd = conn.CreateCommand();                  switch (request.MazeType) {                      case "Recursive Backtrack":                          cmd.CommandText = $@"UPDATE UserStats                                               SET RecursiveBacktrackMazesGenerated = RecursiveBacktrackMazesGenerated + 1                                               WHERE UID = {request.UserID}";                          cmd.ExecuteNonQuery();                          break;                      case "Wilson's":                          cmd.CommandText = $@"UPDATE UserStats                                               SET WilsonsMazesGenerated = WilsonsMazesGenerated + 1                                               WHERE UID = {request.UserID}";                          cmd.ExecuteNonQuery();                          break;                      case "Growing Tree":                          cmd.CommandText = $@"UPDATE UserStats                                               SET GrowingTreeMazesGenerated = GrowingTreeMazesGenerated + 1                                               WHERE UID = {request.UserID}";                          cmd.ExecuteNonQuery();                          break;                  }              }              return Task.FromResult(new Empty());          }            public override Task<Empty> UserUploadTime(UserTime request, ServerCallContext context) {              List<int> milliseconds = new();              List<string> displayTimes = new();                using (SQLiteConnection conn = new(                  "Data Source= mazeData.db; " +                  "Version = 3; " +                  "New = True; " +                  "Compress = True; ")) {                  conn.Open();                  using SQLiteCommand cmd = conn.CreateCommand();                    cmd.CommandText = @$"SELECT Time1Milliseconds,                                              Time2Milliseconds,                                              Time3Milliseconds,                                              Time4Milliseconds,                                              Time5Milliseconds,                                              Time6Milliseconds,                                              Time7Milliseconds,                                              Time8Milliseconds,                                              Time9Milliseconds,                                              Time10Milliseconds                                      FROM UserStats                                      WHERE UID = {request.UserID}";                  using SQLiteDataReader readerMilliseconds = cmd.ExecuteReader();                  while (readerMilliseconds.Read()) {                      for (int i = 0; i < 10; i++) {                          milliseconds.Add(readerMilliseconds.GetInt32(i));                      }                  }                  readerMilliseconds.Close();                    cmd.CommandText = @$"SELECT Time1Display,                                              Time2Display,                                              Time3Display,                                              Time4Display,                                              Time5Display,                                              Time6Display,                                              Time7Display,                                              Time8Display,                                              Time9Display,                                              Time10Display                                      FROM UserStats                                      WHERE UID = {request.UserID}";                  using SQLiteDataReader readerDisplay = cmd.ExecuteReader();                  while (readerDisplay.Read()) {                      for (int i = 0; i < 10; i++) {                          displayTimes.Add(readerDisplay.GetString(i));                      }                  }                  readerDisplay.Close();                        int place = -1;                  for (int i = 0; i < 10; i++) {                      if (request.TimeMilliseconds < milliseconds[i] || milliseconds[i] == -1) {                          milliseconds.Insert(i, request.TimeMilliseconds);                          place = i;                          if (milliseconds.Count > 10) milliseconds.RemoveAt(milliseconds.Count - 1);                          break;                      }                  }                    if (place == -1) { return Task.FromResult(new Empty()); }                    displayTimes.Insert(place, request.Time);                  if (displayTimes.Count > 10) displayTimes.RemoveAt(displayTimes.Count - 1);                          for (int i = 0; i < 10; i++) {                      cmd.CommandText = $@"UPDATE UserStats                                           SET Time{i + 1}Milliseconds = {milliseconds[i]}                                           WHERE UID = {request.UserID}";                      cmd.ExecuteNonQuery();                      cmd.CommandText = $@"UPDATE UserStats                                           SET Time{i + 1}Display = '{displayTimes[i]}'                                           WHERE UID = {request.UserID}";                      cmd.ExecuteNonQuery();                  }              }              return Task.FromResult(new Empty());          }      }  } |

### LoadMazeService.cs

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class LoadMazeService : LoaderMazes.LoaderMazesBase      {          public override Task<MazeToLoad> LoadMaze(LoadRequest request, ServerCallContext context) {              string maze;              string mazeGenAlg;                using (SQLiteConnection conn = new(                  "Data Source= mazeData.db; " +                  "Version = 3;" +                  " New = True; " +                  "Compress = True; ")) {                  conn.Open();                    using SQLiteCommand cmd = conn.CreateCommand();                  cmd.CommandText = @"SELECT MazeObject, MazeGenAlg                                      FROM Mazes                                      WHERE @MazeID = MazeID                                      AND @UserID = UID";                  cmd.Parameters.AddWithValue("@MazeID", request.MazeID);                  cmd.Parameters.AddWithValue("@UserID", request.UserID);                    using SQLiteDataReader reader = cmd.ExecuteReader();                  if (reader.Read()) {                      maze = reader.GetString(0);                      mazeGenAlg = reader.GetString(1);                  }                  else {                      maze = string.Empty;                      mazeGenAlg = string.Empty;                  }              }                return Task.FromResult(new MazeToLoad {                  Maze = maze, MazeGenAlg = mazeGenAlg              });          }      }  } |

### LoginService.cs

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;  using System.Security.Cryptography;  using System.Text;    namespace Server.Services  {      public class LoginService : LoginHandler.LoginHandlerBase      {          public const int keySize = 64;          public const int iterations = 350000;          HashAlgorithmName hashAlgorithm = HashAlgorithmName.SHA512;          public override Task<Access> Login(Credentials request, ServerCallContext context) {                string password;              string salt;              int userID;                using (SQLiteConnection conn = new(                  "Data Source= mazeData.db; " +                  "Version = 3; " +                  "New = True; " +                  "Compress = True; ")) {                  conn.Open();                    using SQLiteCommand cmd = conn.CreateCommand();                  cmd.CommandText = $@"SELECT UID, Password, Salt                                       FROM User                                       WHERE Username = @username";                  cmd.Parameters.AddWithValue("@username", request.Username);                    using SQLiteDataReader reader = cmd.ExecuteReader();                  if (reader.Read()) {                      userID = reader.GetInt32(0);                      password = reader.GetString(1);                      salt = reader.GetString(2);                  }                  else {                      return Task.FromResult(new Access { LoggedIn = false });                  }              }                return Task.FromResult(new Access {                  LoggedIn = VerifyPassword(request.Password, password, Convert.FromHexString(salt)),                  UserID = userID              });          }            // SOURCE: https://code-maze.com/csharp-hashing-salting-passwords-best-practices/          bool VerifyPassword(string password, string hash, byte[] salt) {              var hashToCompare = Rfc2898DeriveBytes.Pbkdf2(                  password,                  salt,                  iterations,                  hashAlgorithm,                  keySize);              return CryptographicOperations.FixedTimeEquals(hashToCompare, Convert.FromHexString(hash));          }      }  } |

### MazeBuilderService.cs

|  |
| --- |
| using Grpc.Core;  using Newtonsoft.Json;  using System.Text;    namespace Server.Services  {      public class MazeBuilderService : MazeBuilder.MazeBuilderBase      {          public override Task<BuiltMaze> BuildMaze(MazeRequest request, ServerCallContext context) {                Maze maze = null;                switch (request.Algorithm) {                  case "Recursive Backtrack":                      maze = new RecursiveBacktrackGeneration(                          (int)request.Width,                          (int)request.Height);                      maze.InitMaze();                      maze.BuildMaze(maze.MazeCoordinates[1, 1]);                      maze.RemoveWalls((int)request.RemoveWalls);                      maze.CreateEntranceExit(request.ExitLocation == "Border");                      break;                  case "Wilson's":                      maze = new WilsonsGeneration(                          (int)request.Width,                          (int)request.Height,                          request.ExitLocation == "Border");                      //algorithm requires exitlocation initialised before other algorithms                      //so it knows where the exit will be for the intial loop erased walk.                      maze.InitMaze();                      maze.CreateEntranceExit(request.ExitLocation == "Border");                      maze.BuildMaze(maze.MazeCoordinates[1, 1]);                      maze.RemoveWalls((int)request.RemoveWalls);                      break;                  case "Growing Tree":                      maze = new GrowingTreeGeneration(                          (int)request.Width,                          (int)request.Height);                      maze.InitMaze();                      maze.BuildMaze(null); //startCoordinate unnecessary for this algorithm                      maze.RemoveWalls((int)request.RemoveWalls);                      maze.CreateEntranceExit(request.ExitLocation == "Border");                      break;                  }                string jsonMaze = JsonConvert.SerializeObject(maze);                return Task.FromResult(new BuiltMaze { Maze = jsonMaze });          }      }  } |

### MazeSolverService.cs

|  |
| --- |
| using Grpc.Core;  using Newtonsoft.Json;    namespace Server.Services  {      public class MazeSolverService : MazeSolver.MazeSolverBase      {          public override Task<Path> SolveMaze(SolveRequest request, ServerCallContext context)          {              SolvingAlgorithm solver = null;              Maze maze = null;                switch (request.MazeGenerationAlgorithm) {                  case "Recursive Backtrack":                      maze = JsonConvert.DeserializeObject<RecursiveBacktrackGeneration>(request.Maze);                      break;                  case "Wilson's":                      maze = JsonConvert.DeserializeObject<WilsonsGeneration>(request.Maze);                      break;                  case "Growing Tree":                      maze = JsonConvert.DeserializeObject<GrowingTreeGeneration>(request.Maze);                      break;              }                  switch (request.Algorithm) {                  case "Depth First":                      solver = new DepthFirstSolve();                      break;                  case "Maze Routing":                      solver = new MazeRoutingSolve();                      break;                  case "Breadth First":                      solver = new BreadthFirstSolve();                      break;              }                List<Coordinate> path = solver.SolveMaze(maze);                return Task.FromResult(new Path { Path\_ = JsonConvert.SerializeObject(path) });          }      }  } |

### RegisterService.cs

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;      namespace Server.Services  {      public class RegisterService : Registerer.RegistererBase      {          public override Task<Acknowledgement> Register(Account request, ServerCallContext context) {              try {                  using (SQLiteConnection conn = new(                      "Data Source= mazeData.db; " +                      "Version = 3; " +                      "New = True; " +                      "Compress = True; ")) {                      conn.Open();                        using SQLiteCommand cmd = conn.CreateCommand();                      cmd.CommandText = @"INSERT INTO User(Username, Password, Salt)                                          VALUES(@Username, @Password, @Salt)";                      cmd.Parameters.AddWithValue("@Username", request.Username);                      cmd.Parameters.AddWithValue("@Password", request.Password);                      cmd.Parameters.AddWithValue("@Salt", request.Salt);                        cmd.ExecuteNonQuery();                  }                    return Task.FromResult(new Acknowledgement { Success = true });              }              catch (Exception) {                  return Task.FromResult(new Acknowledgement { Success = false });              }          }      }  }    // test code: retrieves all usernames and password hashes    //cmd.CommandText = "SELECT \* FROM Login";  //SQLiteDataReader sqliteDataReader = cmd.ExecuteReader();    //while (sqliteDataReader.Read()) {  //    string myReader = $"{sqliteDataReader.GetString(0)}, {sqliteDataReader.GetString(1)}";  //    Console.WriteLine(myReader);  //}  //sqliteDataReader.Close(); |

### SaveMazeService.cs

|  |
| --- |
| using Grpc.Core;  using System.Data.SQLite;    namespace Server.Services  {      public class SaveMazeService : Saver.SaverBase      {          public override Task<SuccessAck> SaveMaze(SaveRequest request, ServerCallContext context) {              try {                  using (SQLiteConnection conn = new(                      "Data Source= mazeData.db; " +                      "Version = 3; " +                      "New = True; " +                      "Compress = True; ")) {                      conn.Open();                      using SQLiteCommand cmd = conn.CreateCommand();                        cmd.CommandText = @"INSERT INTO Mazes(MazeObject, MazeGenAlg, MazeName, UID)                                          VALUES(@MazeObject, @MazeGenAlg, @MazeName, @UID)";                      cmd.Parameters.AddWithValue("@MazeObject", request.MazeJson);                      cmd.Parameters.AddWithValue("@MazeGenAlg", request.MazeType);                      cmd.Parameters.AddWithValue("@MazeName", request.MazeName);                      cmd.Parameters.AddWithValue("@UID", request.UserID);                      cmd.ExecuteNonQuery();                  }                    return Task.FromResult(new SuccessAck { Success = true });                }              catch (Exception) {                  return Task.FromResult(new SuccessAck { Success = false });              }          }      }  } |

## Algorithm Classes

### Breadth First Solve.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Runtime.Serialization.Formatters.Binary;  using System.Text;  using System.Threading.Tasks;    namespace Server {      internal class BreadthFirstSolve : SolvingAlgorithm      {          List<(Coordinate cell, Coordinate parentCell)> paths = new List<(Coordinate cell, Coordinate parentCell)>();          \_Queue<(Coordinate cell, Coordinate parentCell)> activeCells = new \_Queue<(Coordinate cell, Coordinate parentCell)>(100);            public override List<Coordinate> SolveMaze(Maze maze) {                  List<Coordinate> cellsNeighbouringEntrance = GetUnvisitedNeighbours(maze.MazeEntranceCoordinate, maze);              Coordinate startCell = new Coordinate(cellsNeighbouringEntrance[0].Xpos, cellsNeighbouringEntrance[0].Ypos);              paths.Add((maze.MazeCoordinates[startCell.Ypos, startCell.Xpos], maze.MazeEntranceCoordinate));              activeCells.Enqueue((maze.MazeCoordinates[startCell.Ypos, startCell.Xpos], maze.MazeEntranceCoordinate));              maze.MazeCoordinates[startCell.Ypos, startCell.Xpos].Visited = true;                bool finished = false;              while(!finished) {                  (Coordinate cell, Coordinate parentCell) = activeCells.Peek();                  List<Coordinate> neighbourCells = GetUnvisitedNeighbours(cell, maze);                    foreach (Coordinate neighbourCell in neighbourCells) {                      paths.Add((maze.MazeCoordinates[neighbourCell.Ypos, neighbourCell.Xpos], maze.MazeCoordinates[cell.Ypos, cell.Xpos]));                      activeCells.Enqueue((maze.MazeCoordinates[neighbourCell.Ypos, neighbourCell.Xpos], maze.MazeCoordinates[cell.Ypos, cell.Xpos]));                      maze.MazeCoordinates[neighbourCell.Ypos, neighbourCell.Xpos].Visited = true;                      if (neighbourCell.Equals(maze.MazeExitCoordinate)) finished = true;                  }                    activeCells.Dequeue();                }                return GetSolution(paths, maze);          }            private List<Coordinate> GetSolution(List<(Coordinate cell, Coordinate parentCell)> paths, Maze maze) {              List<Coordinate> solution = new();                Coordinate parentCoordinate = null;              foreach (var cell in paths) {                  if (cell.cell.Equals(maze.MazeExitCoordinate)) {                      solution.Add(cell.cell);                      parentCoordinate = cell.parentCell;                      break;                  }              }                while (!parentCoordinate.Equals(maze.MazeEntranceCoordinate)) {                  foreach (var cell in paths) {                      if (cell.cell.Equals(parentCoordinate)) {                          solution.Add(cell.cell);                          parentCoordinate = cell.parentCell;                          break;                      }                  }              }                return solution;          }            private List<Coordinate> GetUnvisitedNeighbours(Coordinate cell, Maze maze) {                List<Coordinate> cells = new();                if (cell.Ypos - 1 >= 0                  && !maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos].Visited                  && !maze.MazeWalls[cell.Ypos - 1, cell.Xpos]                  && !IsInPath(maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos]))                  cells.Add(maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos]);                if (cell.Xpos + 1 < maze.MazeActualWidth                  && !maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1].Visited                  && !maze.MazeWalls[cell.Ypos, cell.Xpos + 1]                  && !IsInPath(maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1]))                  cells.Add(maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1]);                if (cell.Ypos + 1 < maze.MazeActualHeight                  && !maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos].Visited                  && !maze.MazeWalls[cell.Ypos + 1, cell.Xpos]                  && !IsInPath(maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos]))                  cells.Add(maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos]);                if (cell.Xpos - 1 >= 0                  && !maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1].Visited                  && !maze.MazeWalls[cell.Ypos, cell.Xpos - 1]                  && !IsInPath(maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1]))                  cells.Add(maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1]);                return cells;          }            private bool IsInPath(Coordinate cell) {                foreach (var coordinate in paths) {                  if (coordinate.cell.Equals(cell) || coordinate.parentCell.Equals(cell)) return true;              }              return false;          }      }  } |

### Coordinate.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Server  {      internal class Coordinate      {          [JsonConstructor]          public Coordinate() {            }          public Coordinate(int xPos, int yPos)          {              this.xPos = xPos;              this.yPos = yPos;              visited = false;          }          public Coordinate(Tuple<int, int> pos)          {              xPos = pos.Item1;              yPos = pos.Item2;              visited = false;          }            #region Properties          private int xPos;          public int Xpos          {              get { return xPos; }              set { xPos = value; }          }            private int yPos;          public int Ypos          {              get { return yPos; }              set { yPos = value; }          }            private bool visited;          public bool Visited          {              get { return visited; }              set { visited = value; }          }          #endregion            #region Methods          public (int x, int y) GetCartesianCoordinates(Maze maze)          {              return (xPos + 1, maze.MazeActualHeight - yPos + 1);          }          public int GetManhattanDistance(Coordinate targetCoordinate)          {              return Math.Abs(Xpos - targetCoordinate.Xpos) + Math.Abs(Ypos - targetCoordinate.Ypos);          }            public bool Equals(Coordinate target)          {              return xPos == target.xPos && yPos == target.yPos;          }          #endregion      }  } |

### DepthFirstSolve.cs

|  |
| --- |
| namespace Server  {      internal class DepthFirstSolve : SolvingAlgorithm      {          public override List<Coordinate> SolveMaze(Maze maze) {              Coordinate solver = new(maze.MazeEntranceCoordinate.Xpos, maze.MazeEntranceCoordinate.Ypos);              List<Coordinate> path = new();              path.Add(solver);              maze.MazeCoordinates[solver.Ypos, solver.Xpos].Visited = true;                while (!solver.Equals(maze.MazeExitCoordinate)) {                  \_List<Coordinate> unvisitedNeighbours = GetUnvisitedNeighbours(solver, maze);                    if (unvisitedNeighbours.Count > 0) { //if paths exist, take the first one.                      solver = new(unvisitedNeighbours[0].Xpos, unvisitedNeighbours[0].Ypos);                      path.Add(solver);                      maze.MazeCoordinates[solver.Ypos, solver.Xpos].Visited = true;                  }                  else { //otherwise, backtrack.                      solver = path[^2];                      path.RemoveAt(path.Count - 1);                  }              }                return path;          }          private \_List<Coordinate> GetUnvisitedNeighbours(Coordinate cell, Maze maze) {                \_List<Coordinate> cells = new();                if (cell.Ypos - 1 >= 0                  && !maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos].Visited                  && !maze.MazeWalls[cell.Ypos - 1, cell.Xpos])                  cells.Add(maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos]);                if (cell.Xpos + 1 < maze.MazeActualWidth                  && !maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1].Visited                  && !maze.MazeWalls[cell.Ypos, cell.Xpos + 1])                  cells.Add(maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1]);                if (cell.Ypos + 1 < maze.MazeActualHeight                  && !maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos].Visited                  && !maze.MazeWalls[cell.Ypos + 1, cell.Xpos])                  cells.Add(maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos]);                if (cell.Xpos - 1 >= 0                  && !maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1].Visited                  && !maze.MazeWalls[cell.Ypos, cell.Xpos - 1])                  cells.Add(maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1]);                return cells;          }      }  } |

### Growing Tree Generation.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Server  {      internal class GrowingTreeGeneration : Maze      {          List<Coordinate> cellsInMaze = new();            [JsonConstructor]          public GrowingTreeGeneration() {            }          public GrowingTreeGeneration(int cellWidth, int cellHeight) {              MazeCellWidth = cellWidth;              MazeCellHeight = cellHeight;              rgen = new();          }            public override void BuildMaze(Coordinate startCell) {              List<Coordinate> activeCells = new();              List<Coordinate> visitedCells = new();              activeCells.Add(cellsInMaze[rgen.Next(cellsInMaze.Count)]);                while (activeCells.Count > 0) {                  Coordinate cell = activeCells[rgen.Next(activeCells.Count)];                  List<Coordinate> unvisited = GetUnvisitedNeighbours(cell, activeCells, visitedCells);                    if (unvisited.Count > 0) {                      Coordinate targetCell = unvisited[rgen.Next(unvisited.Count)];                      DestroyWall(cell, targetCell);                      activeCells.Add(targetCell);                  }                  else {                      activeCells.Remove(cell);                      visitedCells.Add(cell);                      continue;                  }              }          }          private void DestroyWall(Coordinate cell1, Coordinate cell2) {              int midX = Math.Min(cell1.Xpos, cell2.Xpos) + Math.Abs(cell1.Xpos - cell2.Xpos) / 2;              int midY = Math.Min(cell1.Ypos, cell2.Ypos) + Math.Abs(cell1.Ypos - cell2.Ypos) / 2;              MazeWalls[midY, midX] = false;          }            private List<Coordinate> GetUnvisitedNeighbours(Coordinate cell, List<Coordinate> activeCells, List<Coordinate> visitedCells) {              List<Coordinate> unvisited = new();                if (cell.Ypos - 2 >= 0)//N                  if (!activeCells.Contains(MazeCoordinates[cell.Ypos - 2, cell.Xpos]) &&                      !visitedCells.Contains(MazeCoordinates[cell.Ypos - 2, cell.Xpos]))                      unvisited.Add(MazeCoordinates[cell.Ypos - 2, cell.Xpos]);                if (cell.Xpos + 2 < MazeActualWidth)//E                  if (!activeCells.Contains(MazeCoordinates[cell.Ypos, cell.Xpos + 2]) &&                      !visitedCells.Contains(MazeCoordinates[cell.Ypos, cell.Xpos + 2]))                      unvisited.Add(MazeCoordinates[cell.Ypos, cell.Xpos + 2]);                if (cell.Ypos + 2 < MazeActualHeight)//S                  if (!activeCells.Contains(MazeCoordinates[cell.Ypos + 2, cell.Xpos]) &&                      !visitedCells.Contains(MazeCoordinates[cell.Ypos + 2, cell.Xpos]))                      unvisited.Add(MazeCoordinates[cell.Ypos + 2, cell.Xpos]);                if (cell.Xpos - 2 >= 0)//W                  if (!activeCells.Contains(MazeCoordinates[cell.Ypos, cell.Xpos - 2]) &&                      !visitedCells.Contains(MazeCoordinates[cell.Ypos, cell.Xpos - 2]))                      unvisited.Add(MazeCoordinates[cell.Ypos, cell.Xpos - 2]);                return unvisited;          }              public override void CreateEntranceExit(bool atBorder) {              MazeWalls[1, 0] = false; //entrance              MazeEntranceCoordinate = new Coordinate(0, 1);                if (atBorder) //border exit              {                  Maze​Walls[MazeActualHeight - 2, MazeActualWidth - 1] = false; //exit                  MazeExitCoordinate = new Coordinate(MazeActualWidth - 1, MazeActualHeight - 2);              }              else //central exit              {                  int centerX, centerY;                  centerX = MazeActualWidth / 2;                  centerY = MazeActualHeight / 2;                  MazeWalls[centerY, centerX] = false;                  MazeExitCoordinate = new Coordinate(centerX, centerY);              }                ResetVisited();          }            public override void InitMaze() {              MazeActualHeight = 2 \* MazeCellHeight + 1;              MazeActualWidth = 2 \* MazeCellWidth + 1;                MazeWalls = new bool[MazeActualHeight, MazeActualWidth];              MazeCoordinates = new Coordinate[MazeActualHeight, MazeActualWidth];                for (int y = 0; y < MazeActualHeight; y++) {                  for (int x = 0; x < MazeActualWidth; x++) {                      MazeCoordinates[y, x] = new Coordinate(x, y);                        if (y % 2 != 0 && x % 2 != 0) {                          MazeWalls[y, x] = false;                          cellsInMaze.Add(MazeCoordinates[y, x]);                      }                        else                          MazeWalls[y, x] = true;                  }              }          }            public override void RemoveWalls(int wallsToRemove) {              int wallsRemoved = 0;                while (wallsRemoved < wallsToRemove) {                  int xPos = rgen.Next(1, MazeActualWidth - 1);                  int yPos = rgen.Next(1, MazeActualHeight - 1);                  Coordinate cellToRemove = new(xPos, yPos);                    if (IsWall(cellToRemove)) {                      MazeWalls[yPos, xPos] = false;                      wallsRemoved++;                  }              }          }            private bool IsWall(Coordinate cell) {              if (MazeWalls[cell.Ypos + 1, cell.Xpos] == false                  && MazeWalls[cell.Ypos - 1, cell.Xpos] == false                  && MazeWalls[cell.Ypos, cell.Xpos + 1] == true                  && MazeWalls[cell.Ypos, cell.Xpos - 1] == true) {                  return true;              }              else if (MazeWalls[cell.Ypos + 1, cell.Xpos] == true                  && MazeWalls[cell.Ypos - 1, cell.Xpos] == true                  && MazeWalls[cell.Ypos, cell.Xpos + 1] == false                  && MazeWalls[cell.Ypos, cell.Xpos - 1] == false) {                  return true;              }              else return false;          }      }  } |

### Maze Routing Solve.cs

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Linq;  using System.Text;  using System.Threading.Tasks;    namespace Server  {      internal class MazeRoutingSolve : SolvingAlgorithm      {          public override List<Coordinate> SolveMaze(Maze maze) {              List<Coordinate> solution = new List<Coordinate>();              Coordinate solver = new Coordinate(maze.MazeEntranceCoordinate.Xpos, maze.MazeEntranceCoordinate.Ypos);                int bestMD = solver.GetManhattanDistance(maze.MazeExitCoordinate);              solution.Add(solver);                maze.MazeCoordinates[solver.Ypos, solver.Xpos].Visited = true;                while (!solver.Equals(maze.MazeExitCoordinate)) {                  List<(Coordinate coordinate, char direction)> unvisited = GetUnvisitedNeighbours(maze, solver);                    if (unvisited.Count == 0) {                      do {                          solution.Remove(solution.Last());                          solver = new Coordinate(solution[^1].Xpos, solution[^1].Ypos);                          unvisited = GetUnvisitedNeighbours(maze, solver);                      } while (unvisited.Count == 0);                      continue;                  }                  else if (unvisited.Count == 1) {                      solver = new Coordinate(unvisited[0].coordinate.Xpos, unvisited[0].coordinate.Ypos);                      solution.Add(maze.MazeCoordinates[solver.Ypos, solver.Xpos]);                      maze.MazeCoordinates[solver.Ypos, solver.Xpos].Visited = true;                  }                  else {                      char directionToMove = TryPaths(maze, unvisited);                      Coordinate cellToMoveTo = null;                      foreach (var cell in unvisited) {                          if (cell.direction == directionToMove) {                              cellToMoveTo = cell.coordinate;                              break;                          }                      }                      solver = new Coordinate(cellToMoveTo.Xpos, cellToMoveTo.Ypos);                      solution.Add(maze.MazeCoordinates[solver.Ypos, solver.Xpos]);                      maze.MazeCoordinates[solver.Ypos, solver.Xpos].Visited = true;                  }              }                  return solution;          }            private char TryPaths(Maze maze, List<(Coordinate coordinate, char direction)> paths) {              Coordinate tempSolver;              List<Coordinate> cellsVisited = new List<Coordinate>();              (char direction, int BestMD) bestPath = (' ', 99999999); //effectively infinity for these mazes                foreach (var path in paths) {                  List<(Coordinate coordinate, char direction)> unvisited;                    do {                      tempSolver = new Coordinate(path.coordinate.Xpos, path.coordinate.Ypos);                      unvisited = GetUnvisitedNeighbours(maze, tempSolver);                        if (unvisited.Count == 1) {                          tempSolver = new Coordinate(unvisited[0].coordinate.Xpos, unvisited[0].coordinate.Ypos);                          cellsVisited.Add(maze.MazeCoordinates[tempSolver.Ypos, tempSolver.Xpos]);                          maze.MazeCoordinates[tempSolver.Ypos, tempSolver.Xpos].Visited = true;                      }                      else {                          int MD = tempSolver.GetManhattanDistance(maze.MazeExitCoordinate);                            if (MD < bestPath.BestMD) {                              bestPath = (path.direction, MD);                          }                      }                  } while (unvisited.Count == 1);              }                foreach (var cell in cellsVisited) {                  cell.Visited = false;              }                return bestPath.direction;          }            private List<(Coordinate coordinate, char direction)> GetUnvisitedNeighbours(Maze maze, Coordinate cell) {              List<(Coordinate coordinate, char direction)> cells = new();                if (cell.Ypos - 1 >= 0                  && !maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos].Visited                  && !maze.MazeWalls[cell.Ypos - 1, cell.Xpos])                  cells.Add((maze.MazeCoordinates[cell.Ypos - 1, cell.Xpos], 'N'));                if (cell.Xpos + 1 < maze.MazeActualWidth                  && !maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1].Visited                  && !maze.MazeWalls[cell.Ypos, cell.Xpos + 1])                  cells.Add((maze.MazeCoordinates[cell.Ypos, cell.Xpos + 1], 'E'));                if (cell.Ypos + 1 < maze.MazeActualHeight                  && !maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos].Visited                  && !maze.MazeWalls[cell.Ypos + 1, cell.Xpos])                  cells.Add((maze.MazeCoordinates[cell.Ypos + 1, cell.Xpos], 'S'));                if (cell.Xpos - 1 >= 0                  && !maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1].Visited                  && !maze.MazeWalls[cell.Ypos, cell.Xpos - 1])                  cells.Add((maze.MazeCoordinates[cell.Ypos, cell.Xpos - 1], 'W'));                return cells;          }      }  } |

### Maze.cs

|  |
| --- |
| namespace Server  {      internal abstract class Maze      {          #region Properties          private int mazeActualWidth;          public int MazeActualWidth          {              get { return mazeActualWidth; }              set { mazeActualWidth = value; }          }            private int mazeActualHeight;          public int MazeActualHeight          {              get { return mazeActualHeight; }              set { mazeActualHeight = value; }          }            private int mazeCellWidth;          public int MazeCellWidth          {              get { return mazeCellWidth; }              set { mazeCellWidth = value; }          }            private int mazeCellHeight;          public int MazeCellHeight          {              get { return mazeCellHeight; }              set { mazeCellHeight = value; }          }            private bool[,]? mazeWalls;          public bool[,]? MazeWalls          {              get { return mazeWalls; }              set { mazeWalls = value; }          }            private Coordinate[,]? mazeCoordinates;          public Coordinate[,]? MazeCoordinates          {              get { return mazeCoordinates; }              set { mazeCoordinates = value; }          }            private Coordinate? mazeEntranceCoordinate;          public Coordinate? MazeEntranceCoordinate          {              get { return mazeEntranceCoordinate; }              set { mazeEntranceCoordinate = value; }          }            private Coordinate? mazeExitCoordinate;          public Coordinate? MazeExitCoordinate          {              get { return mazeExitCoordinate; }              set { mazeExitCoordinate = value; }          }            protected Random rgen = new();          #endregion            #region Methods          public abstract void InitMaze();          public abstract void BuildMaze(Coordinate startCell);          public abstract void CreateEntranceExit(bool atBorder);          public abstract void RemoveWalls(int wallsToRemove);          protected virtual bool CellVisited(Coordinate cellPos)          {              return cellPos.Visited;          }          protected virtual void ResetVisited()          {              foreach (Coordinate v in mazeCoordinates)              {                  v.Visited = false;              }          }          #endregion      }  } |

### Program.cs

The main entry point for the server, containing definitions for all database tables and the code which maps available services to the server request pipeline.

|  |
| --- |
| using Server.Services;  using System.Data.SQLite;    //initialize database  SQLiteConnection conn = new(      "Data Source= mazeData.db; " +      "Version = 3; " +      "New = True; " +      "Compress = True; ");  conn.Open();  using (SQLiteCommand cmd = conn.CreateCommand()) {      cmd.CommandText = "PRAGMA foreign\_keys = ON;";      cmd.ExecuteNonQuery();        cmd.CommandText = @"CREATE TABLE IF NOT EXISTS User(                              UID INTEGER PRIMARY KEY,                              Username VARCHAR,                              Password VARCHAR,                              Salt VARCHAR                          )";      cmd.ExecuteNonQuery();        cmd.CommandText = @"CREATE TABLE IF NOT EXISTS Mazes(                              MazeID INTEGER PRIMARY KEY,                              MazeObject VARCHAR,                              MazeGenAlg VARCHAR,                              MazeName VARCHAR(10),                              UID INTEGER,                              FOREIGN KEY(UID) REFERENCES User(UID)                          )";      cmd.ExecuteNonQuery();        cmd.CommandText = @"CREATE TABLE IF NOT EXISTS UserStats(                              UID INTEGER PRIMARY KEY,                              Time1Display VARCHAR,                              Time1Milliseconds INTEGER,                              Time2Display VARCHAR,                              Time2Milliseconds INTEGER,                              Time3Display VARCHAR,                              Time3Milliseconds INTEGER,                              Time4Display VARCHAR,                              Time4Milliseconds INTEGER,                              Time5Display VARCHAR,                              Time5Milliseconds INTEGER,                              Time6Display VARCHAR,                              Time6Milliseconds INTEGER,                              Time7Display VARCHAR,                              Time7Milliseconds INTEGER,                              Time8Display VARCHAR,                              Time8Milliseconds INTEGER,                              Time9Display VARCHAR,                              Time9Milliseconds INTEGER,                              Time10Display VARCHAR,                              Time10Milliseconds INTEGER,                              RecursiveBacktrackMazesGenerated INTEGER,                              GrowingTreeMazesGenerated INTEGER,                              WilsonsMazesGenerated INTEGER,                              FOREIGN KEY(UID) REFERENCES User(UID)                          )";      cmd.ExecuteNonQuery();      //ON DELETE CASCADE deletes the relevent record if the user is deleted        cmd.CommandText = @"                      CREATE TABLE IF NOT EXISTS GlobalStats (                          Time1Display VARCHAR,                          Time1Milliseconds INTEGER,                          Time1Name VARCHAR,                            Time2Display VARCHAR,                          Time2Milliseconds INTEGER,                          Time2Name VARCHAR,                            Time3Display VARCHAR,                          Time3Milliseconds INTEGER,                          Time3Name VARCHAR,                            Time4Display VARCHAR,                          Time4Milliseconds INTEGER,                          Time4Name VARCHAR,                            Time5Display VARCHAR,                          Time5Milliseconds INTEGER,                          Time5Name VARCHAR,                            Time6Display VARCHAR,                          Time6Milliseconds INTEGER,                          Time6Name VARCHAR,                            Time7Display VARCHAR,                          Time7Milliseconds INTEGER,                          Time7Name VARCHAR,                            Time8Display VARCHAR,                          Time8Milliseconds INTEGER,                          Time8Name VARCHAR,                            Time9Display VARCHAR,                          Time9Milliseconds INTEGER,                          Time9Name VARCHAR,                            Time10Display VARCHAR,                          Time10Milliseconds INTEGER,                          Time10Name VARCHAR,                            RecursiveBacktrackMazesGenerated INTEGER,                          GrowingTreeMazesGenerated INTEGER,                          WilsonsMazesGenerated INTEGER                      )";      cmd.ExecuteNonQuery();        cmd.CommandText = @"CREATE TRIGGER IF NOT EXISTS CreateStatsRecord                          AFTER INSERT ON User                          BEGIN                              INSERT INTO UserStats                              VALUES (NEW.UID,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      '',                                      -1,                                      0,                                      0,                                      0);                          END;";      cmd.ExecuteNonQuery();        cmd.CommandText = "SELECT COUNT(\*) FROM GlobalStats";      int rowCount = Convert.ToInt32(cmd.ExecuteScalar());      if (rowCount == 0) {          cmd.CommandText = @"INSERT INTO GlobalStats                              VALUES('',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     '',                                     -1,                                     '',                                     0,                                     0,                                     0)";          cmd.ExecuteNonQuery();      }  }    conn.Close();    var builder = WebApplication.CreateBuilder(args);    // Add services to the container.  builder.Services.AddGrpc();    var app = builder.Build();    // Configure the HTTP request pipeline.  app.MapGrpcService<GreeterService>();  app.MapGrpcService<MazeBuilderService>();  app.MapGrpcService<MazeSolverService>();  app.MapGrpcService<RegisterService>();  app.MapGrpcService<CheckerIfUserExistsService>();  app.MapGrpcService<LoginService>();  app.MapGrpcService<SaveMazeService>();  app.MapGrpcService<GetMazesService>();  app.MapGrpcService<LoadMazeService>();  app.MapGrpcService<DeleteMazeService>();  app.MapGrpcService<GlobalStatHandlerService>();  app.MapGrpcService<HandleUserStatsService>();  app.MapGrpcService<GetStatsService>();    app.Run(); |

### Recursive Backtrack Generation.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Server  {      internal class RecursiveBacktrackGeneration : Maze      {          [JsonConstructor]          public RecursiveBacktrackGeneration() {            }          public RecursiveBacktrackGeneration(int cellWidth, int cellHeight)          {              MazeCellWidth = cellWidth;              MazeCellHeight = cellHeight;              rgen = new();          }            public override void InitMaze()          {              MazeActualHeight = 2 \* MazeCellHeight + 1;              MazeActualWidth = 2 \* MazeCellWidth + 1;                MazeWalls = new bool[MazeActualHeight, MazeActualWidth];              MazeCoordinates = new Coordinate[MazeActualHeight, MazeActualWidth];                for (int y = 0; y < MazeActualHeight; y++)              {                  for (int x = 0; x < MazeActualWidth; x++)                  {                      if (y % 2 != 0 && x % 2 != 0)                          MazeWalls[y, x] = false;                      else                          MazeWalls[y, x] = true;                        MazeCoordinates[y, x] = new Coordinate(x, y);                  }              }          }            public override void BuildMaze(Coordinate cell)          {              cell.Visited = true;                List<Coordinate> neighbourCells = GetUnvisitedNeighbours(cell);                while (neighbourCells.Count > 0)              {                  Coordinate targetCell = neighbourCells[rgen.Next(0, neighbourCells.Count)];                  if (targetCell.Visited) return;                  DestroyWall(cell, targetCell);                  neighbourCells.Remove(targetCell);                  BuildMaze(targetCell);              }          }            public override void RemoveWalls(int wallsToRemove)          {              int wallsRemoved = 0;                while (wallsRemoved < wallsToRemove)              {                  int xPos = rgen.Next(1, MazeActualWidth - 1);                  int yPos = rgen.Next(1, MazeActualHeight - 1);                  Coordinate cellToRemove = new(xPos, yPos);                    if (IsWall(cellToRemove))                  {                      MazeWalls[yPos, xPos] = false;                      wallsRemoved++;                  }              }          }            private bool IsWall(Coordinate cell)          {              if (MazeWalls[cell.Ypos + 1, cell.Xpos] == false                  && MazeWalls[cell.Ypos - 1, cell.Xpos] == false                  && MazeWalls[cell.Ypos, cell.Xpos + 1] == true                  && MazeWalls[cell.Ypos, cell.Xpos - 1] == true)              {                  return true;              }              else if (MazeWalls[cell.Ypos + 1, cell.Xpos] == true                  && MazeWalls[cell.Ypos - 1, cell.Xpos] == true                  && MazeWalls[cell.Ypos, cell.Xpos + 1] == false                  && MazeWalls[cell.Ypos, cell.Xpos - 1] == false)              {                  return true;              }              else return false;          }            private List<Coordinate> GetUnvisitedNeighbours(Coordinate cell)          {                List<Coordinate> cells = new();                if (cell.Ypos - 2 >= 0 && !MazeCoordinates[cell.Ypos - 2, cell.Xpos].Visited)                  cells.Add(MazeCoordinates[cell.Ypos - 2, cell.Xpos]);                if (cell.Xpos + 2 < MazeActualWidth && !MazeCoordinates[cell.Ypos, cell.Xpos + 2].Visited)                  cells.Add(MazeCoordinates[cell.Ypos, cell.Xpos + 2]);                if (cell.Ypos + 2 < MazeActualHeight && !MazeCoordinates[cell.Ypos + 2, cell.Xpos].Visited)                  cells.Add(MazeCoordinates[cell.Ypos + 2, cell.Xpos]);                if (cell.Xpos - 2 >= 0 && !MazeCoordinates[cell.Ypos, cell.Xpos - 2].Visited)                  cells.Add(MazeCoordinates[cell.Ypos, cell.Xpos - 2]);                return cells;          }            private void DestroyWall(Coordinate cell1, Coordinate cell2)          {              int midX = Math.Min(cell1.Xpos, cell2.Xpos) + Math.Abs(cell1.Xpos - cell2.Xpos) / 2;              int midY = Math.Min(cell1.Ypos, cell2.Ypos) + Math.Abs(cell1.Ypos - cell2.Ypos) / 2;              MazeWalls[midY, midX] = false;          }            public override void CreateEntranceExit(bool atBorder)          {              MazeWalls[1, 0] = false; //entrance              MazeEntranceCoordinate = new Coordinate(0, 1);                if (atBorder) //border exit              {                  Maze​Walls[MazeActualHeight - 2, MazeActualWidth - 1] = false; //exit                  MazeExitCoordinate = new Coordinate(MazeActualWidth - 1, MazeActualHeight - 2);              }              else //central exit              {                  int centerX, centerY;                  centerX = MazeActualWidth / 2;                  centerY = MazeActualHeight / 2;                  MazeWalls[centerY, centerX] = false;                  MazeExitCoordinate = new Coordinate(centerX, centerY);              }                  ResetVisited();          }      }  } |

### Solving.cs

|  |
| --- |
| namespace Server  {      internal abstract class SolvingAlgorithm      {          public abstract List<Coordinate> SolveMaze(Maze maze);      }  } |

### Wilsons Generation.cs

|  |
| --- |
| using Newtonsoft.Json;    namespace Server  {      internal class WilsonsGeneration : Maze      {          public List<Coordinate> cellsInMaze = new();          public bool exitAtBorder;            [JsonConstructor]          public WilsonsGeneration() {            }          public WilsonsGeneration(int cellWidth, int cellHeight, bool exitAtBorder) {              MazeCellWidth = cellWidth;              MazeCellHeight = cellHeight;              rgen = new();              this.exitAtBorder = exitAtBorder;          }            public override void BuildMaze(Coordinate startCell) {                /\*startCoordinate and endCoordinate should be where the algorithm starts and ends, but since these are in the walls, they are              inaccessible to the algorithm, so we must pick the neighbouring cells.\*/                InitialLoopErasedWalk(MazeCoordinates[1, 1],                  exitAtBorder ? MazeCoordinates[MazeExitCoordinate.Ypos, MazeExitCoordinate.Xpos - 1] : cellsInMaze[(int)(cellsInMaze.Count \* 0.5f)]);              //if the exit is in the centre, we can find the cell by finding the middle element of the cellsInMaze list              while (cellsInMaze.Count > 0) {                  LoopErasedWalk(cellsInMaze[rgen.Next(cellsInMaze.Count)]);              }              ResetVisited();          }            private void InitialLoopErasedWalk(Coordinate startCoordinate, Coordinate endCoordinate) {              List<Coordinate> path = new List<Coordinate>();              Coordinate ctor = new Coordinate(startCoordinate.Xpos, startCoordinate.Ypos);              path.Add(MazeCoordinates[ctor.Ypos, ctor.Xpos]);              MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = true;                while (!ctor.Equals(endCoordinate)) {                  List<Coordinate> unvisitedCells = GetUnvisitedNeighbours(ctor);                  Coordinate targetCell;                  do {                      int index = rgen.Next(unvisitedCells.Count);                      targetCell = unvisitedCells[index];                      if (path.Contains(targetCell)) unvisitedCells.RemoveAt(index);                      if (unvisitedCells.Count == 0) break;                  } while (path.Contains(targetCell));                      if (!path.Contains(targetCell)) {                      // keep walking, we will destroy all walls after the walk.                      // We cannot do this dynamically as we have to backtrack sometimes                      ctor = new Coordinate(targetCell.Xpos, targetCell.Ypos);                      path.Add(MazeCoordinates[ctor.Ypos, ctor.Xpos]);                      MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = true;                  }                  else { //we must have looped, so backtrack until ctor = targetCell and try again                      while (!ctor.Equals(targetCell)) {                          MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = false;                            path.RemoveAt(path.Count - 1);                          ctor = new Coordinate(path.Last().Xpos, path.Last().Ypos);                      }                  }              }                DestroyAllWallsInPath(path);                foreach (Coordinate coord in path) {                  cellsInMaze.Remove(coord);              }                DestroyExitWalls();          }            private void DestroyExitWalls() {              if (exitAtBorder)                  Maze​Walls[MazeActualHeight - 2, MazeActualWidth - 1] = false;              else {                  int centerX = MazeActualWidth / 2;                  int centerY = MazeActualHeight / 2;                  MazeWalls[centerY, centerX] = false;              }          }            private void LoopErasedWalk(Coordinate startCoordinate) { //walk until we hit the maze              List<Coordinate> path = new List<Coordinate>();              Coordinate ctor = new Coordinate(startCoordinate.Xpos, startCoordinate.Ypos);              path.Add(MazeCoordinates[ctor.Ypos, ctor.Xpos]);              MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = true;                while (true) {                  List<Coordinate> unvisitedCells = GetNeighbouringCells(ctor);                  Coordinate targetCell;                  do {                      int index = rgen.Next(unvisitedCells.Count);                      targetCell = unvisitedCells[index];                        if (path.Contains(targetCell)) unvisitedCells.RemoveAt(index);                        else if (targetCell.Visited) { //if we find the maze, which is a visited cell that is not in the path                          ctor = new Coordinate(targetCell.Xpos, targetCell.Ypos);                          path.Add(MazeCoordinates[ctor.Ypos, ctor.Xpos]);                          MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = true;                          DestroyAllWallsInPath(path);                            foreach (Coordinate coord in path) {                              cellsInMaze.Remove(coord);                          }                          return;                      }                        if (unvisitedCells.Count == 0) break;                    } while (path.Contains(targetCell));                      if (!path.Contains(targetCell)) {                      // keep walking, we will destroy all walls after the walk.                      // We cannot do this dynamically as we have to backtrack sometimes                      ctor = new Coordinate(targetCell.Xpos, targetCell.Ypos);                      path.Add(MazeCoordinates[ctor.Ypos, ctor.Xpos]);                      MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = true;                  }                  else { //we must have looped, so backtrack until ctor = targetCell and try again                      while (!ctor.Equals(targetCell)) {                          MazeCoordinates[ctor.Ypos, ctor.Xpos].Visited = false;                            path.RemoveAt(path.Count - 1);                          ctor = new Coordinate(path.Last().Xpos, path.Last().Ypos);                      }                  }              }          }            private void DestroyAllWallsInPath(List<Coordinate> pathToDestroy) {              for (int i = 0; i < pathToDestroy.Count - 1; i++) {                  DestroyWall(pathToDestroy[i], pathToDestroy[i + 1]);              }          }            private void DestroyWall(Coordinate cell1, Coordinate cell2) {              int midX = Math.Min(cell1.Xpos, cell2.Xpos) + Math.Abs(cell1.Xpos - cell2.Xpos) / 2;              int midY = Math.Min(cell1.Ypos, cell2.Ypos) + Math.Abs(cell1.Ypos - cell2.Ypos) / 2;              MazeWalls[midY, midX] = false;          }            private List<Coordinate> GetUnvisitedNeighbours(Coordinate cell) {              List<Coordinate> cells = new List<Coordinate>();              //we can handle adding things to the maze with careful management of the 'visited' property.              //This means we will have to reimplement getUnivisitedNeighbours              //to account for not just being able to go if !Visited.              //We will instead redefine unvisited as having all 4 walls and the visited property being true will              //mean being part of the maze.              //The following checks if all adjacent cells have all 4 walls intact.              if (cell.Ypos - 2 >= 0 &&                  MazeWalls[cell.Ypos - 2 - 1, cell.Xpos] &&                  MazeWalls[cell.Ypos - 2, cell.Xpos + 1] &&                  MazeWalls[cell.Ypos - 2 + 1, cell.Xpos] &&                  MazeWalls[cell.Ypos - 2, cell.Xpos - 1]) // N                  cells.Add(MazeCoordinates[cell.Ypos - 2, cell.Xpos]);              if (cell.Xpos + 2 < MazeActualWidth &&                  MazeWalls[cell.Ypos - 1, cell.Xpos + 2] &&                  MazeWalls[cell.Ypos, cell.Xpos + 2 + 1] &&                  MazeWalls[cell.Ypos + 1, cell.Xpos + 2] &&                  MazeWalls[cell.Ypos, cell.Xpos + 2 - 1]) // E                  cells.Add(MazeCoordinates[cell.Ypos, cell.Xpos + 2]);              if (cell.Ypos + 2 < MazeActualHeight &&                  MazeWalls[cell.Ypos + 2 - 1, cell.Xpos] &&                  MazeWalls[cell.Ypos + 2, cell.Xpos + 1] &&                  MazeWalls[cell.Ypos + 2 + 1, cell.Xpos] &&                  MazeWalls[cell.Ypos + 2, cell.Xpos - 1]) // S                  cells.Add(MazeCoordinates[cell.Ypos + 2, cell.Xpos]);              if (cell.Xpos - 2 >= 0 &&                  MazeWalls[cell.Ypos - 1, cell.Xpos - 2] &&                  MazeWalls[cell.Ypos, cell.Xpos - 2 + 1] &&                  MazeWalls[cell.Ypos + 1, cell.Xpos - 2] &&                  MazeWalls[cell.Ypos, cell.Xpos - 2 - 1]) // W                  cells.Add(MazeCoordinates[cell.Ypos, cell.Xpos - 2]);                return cells;          }          private List<Coordinate> GetNeighbouringCells(Coordinate cell) {              List<Coordinate> cells = new List<Coordinate>();                if (cell.Ypos - 2 >= 0) // N                  cells.Add(MazeCoordinates[cell.Ypos - 2, cell.Xpos]);              if (cell.Xpos + 2 < MazeActualWidth) // E                  cells.Add(MazeCoordinates[cell.Ypos, cell.Xpos + 2]);              if (cell.Ypos + 2 < MazeActualHeight) // S                  cells.Add(MazeCoordinates[cell.Ypos + 2, cell.Xpos]);              if (cell.Xpos - 2 >= 0) // W                  cells.Add(MazeCoordinates[cell.Ypos, cell.Xpos - 2]);                return cells;          }              public override void CreateEntranceExit(bool atBorder) {              MazeWalls[1, 0] = false; //entrance              MazeEntranceCoordinate = new Coordinate(0, 1);                if (atBorder) //border exit              {                   //exit                  MazeExitCoordinate = new Coordinate(MazeActualWidth - 1, MazeActualHeight - 2);              }              else //central exit              {                  int centerX, centerY;                  centerX = MazeActualWidth / 2;                  centerY = MazeActualHeight / 2;                    MazeExitCoordinate = new Coordinate(centerX, centerY);              }          }            public override void InitMaze() {              MazeActualHeight = 2 \* MazeCellHeight + 1;              MazeActualWidth = 2 \* MazeCellWidth + 1;                MazeWalls = new bool[MazeActualHeight, MazeActualWidth];              MazeCoordinates = new Coordinate[MazeActualHeight, MazeActualWidth];                for (int y = 0; y < MazeActualHeight; y++) {                  for (int x = 0; x < MazeActualWidth; x++) {                      MazeCoordinates[y, x] = new Coordinate(x, y);                        if (y % 2 != 0 && x % 2 != 0) {                          MazeWalls[y, x] = false;                          cellsInMaze.Add(MazeCoordinates[y, x]);                      }                        else                          MazeWalls[y, x] = true;                  }              }          }            public override void RemoveWalls(int wallsToRemove) {              int wallsRemoved = 0;                while (wallsRemoved < wallsToRemove) {                  int xPos = rgen.Next(1, MazeActualWidth - 1);                  int yPos = rgen.Next(1, MazeActualHeight - 1);                  Coordinate cellToRemove = new(xPos, yPos);                    if (IsWall(cellToRemove)) {                      MazeWalls[yPos, xPos] = false;                      wallsRemoved++;                  }              }          }          private bool IsWall(Coordinate cell) {              if (MazeWalls[cell.Ypos + 1, cell.Xpos] == false                  && MazeWalls[cell.Ypos - 1, cell.Xpos] == false                  && MazeWalls[cell.Ypos, cell.Xpos + 1] == true                  && MazeWalls[cell.Ypos, cell.Xpos - 1] == true) {                  return true;              }              else if (MazeWalls[cell.Ypos + 1, cell.Xpos] == true                  && MazeWalls[cell.Ypos - 1, cell.Xpos] == true                  && MazeWalls[cell.Ypos, cell.Xpos + 1] == false                  && MazeWalls[cell.Ypos, cell.Xpos - 1] == false) {                  return true;              }              else return false;          }      }  } |

## Data Structures

### List.cs

|  |
| --- |
| namespace Server  {      public class \_List<T>      {          private T[] items;          private int count;          public int Count          {              get { return count; }          }          public T this[int index]          {              get {                  if (index < 0 || index >= count) {                      throw new IndexOutOfRangeException("Index is out of range.");                  }                  return items[index];              }              set {                  if (index < 0 || index >= count) {                      throw new IndexOutOfRangeException("Index is out of range.");                  }                  items[index] = value;              }          }            public \_List() {              items = new T[4];              count = 0;          }            public void Add(T item) {              if (count == items.Length) {                  Array.Resize(ref items, items.Length + 1);              }              items[count++] = item;          }            public void RemoveAt(int index) {              if (index < 0 || index >= count) {                  throw new IndexOutOfRangeException("Index is out of range.");              }                for (int i = index; i < count - 1; i++) {                  items[i] = items[i + 1];              }                items[count - 1] = default(T);              count--;          }          public bool Remove(T item) {              int index = IndexOf(item);              if (index != -1) {                  RemoveAt(index);                  return true;              }              return false;          }            private int IndexOf(T item) {              for (int i = 0; i < count; i++) {                  if (Equals(items[i], item)) {                      return i;                  }              }              return -1;          }        }  } |

### Queue.cs

|  |
| --- |
| namespace Server  {      public class \_Queue<T>      {          private T[] items;          private int front;          private int rear;          private int count;            public \_Queue(int capacity) {              items = new T[capacity];              front = 0;              rear = -1;              count = 0;          }            public void Enqueue(T item) {              if (count == items.Length) {                  throw new InvalidOperationException("Queue is full.");              }              rear = (rear + 1) % items.Length;              items[rear] = item;              count++;          }            public T Dequeue() {              if (count == 0) {                  throw new InvalidOperationException("Queue is empty.");              }              T item = items[front];              front = (front + 1) % items.Length;              count--;              return item;          }            public int Count          {              get { return count; }          }              public T Peek() {              if (count == 0) {                  throw new InvalidOperationException("Queue is empty.");              }              return items[front];          }              public bool IsEmpty() {              return count == 0;          }              public bool IsFull() {              return count == items.Length;          }      }  } |

# Additional Files

## Client project file

|  |
| --- |
| <Project Sdk="Microsoft.NET.Sdk">      <PropertyGroup>  <OutputType>WinExe</OutputType>  <TargetFramework>net6.0-windows</TargetFramework>  <Nullable>enable</Nullable>  <UseWindowsForms>true</UseWindowsForms>  <ImplicitUsings>enable</ImplicitUsings>    </PropertyGroup>      <ItemGroup>  <None Include="..\.editorconfig" Link=".editorconfig" />    </ItemGroup>      <ItemGroup>  <PackageReference Include="Google.Protobuf" Version="3.25.1" />  <PackageReference Include="Grpc.Net.Client" Version="2.59.0" />  <PackageReference Include="Grpc.Tools" Version="2.60.0">    <PrivateAssets>all</PrivateAssets>    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>  </PackageReference>  <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />  <PackageReference Include="WinForms.DataVisualization" Version="1.9.1" />    </ItemGroup>      <ItemGroup>  <Protobuf Include="Protos\greet.proto" GrpcServices="Client" />  <Protobuf Include="Protos\BuildMaze.proto" GrpcServices="Client" />  <Protobuf Include="Protos\SolveMaze.proto" GrpcServices="Client" />  <Protobuf Include="Protos\Register.proto" GrpcServices="Client" />  <Protobuf Include="Protos\CheckIfUserExists.proto" GrpcServices="Client" />  <Protobuf Include="Protos\Login.proto" GrpcServices="Client" />  <Protobuf Include="Protos\SaveMaze.proto" GrpcServices="Client" />  <Protobuf Include="Protos\GetMazes.proto" GrpcServices="Client" />  <Protobuf Include="Protos\LoadMaze.proto" GrpcServices="Client" />  <Protobuf Include="Protos\DeleteMaze.proto" GrpcServices="Client" />  <Protobuf Include="Protos\HandleGlobalStats.proto" GrpcServices="Client" />  <Protobuf Include="Protos\HandleUserStats.proto" GrpcServices="Client" />  <Protobuf Include="Protos\GetStats.proto" GrpcServices="Client" />    </ItemGroup>    </Project> |

## Server project file

|  |
| --- |
| <Project Sdk="Microsoft.NET.Sdk.Web">      <PropertyGroup>      <TargetFramework>net6.0</TargetFramework>      <Nullable>enable</Nullable>      <ImplicitUsings>enable</ImplicitUsings>    </PropertyGroup>      <ItemGroup>      <Protobuf Include="Protos\greet.proto" GrpcServices="Server" />  <Protobuf Include="Protos\BuildMaze.proto" GrpcServices="Server" />  <Protobuf Include="Protos\SolveMaze.proto" GrpcServices="Server" />  <Protobuf Include="Protos\Register.proto" GrpcServices="Server" />  <Protobuf Include="Protos\CheckIfUserExists.proto" GrpcServices="Server" />  <Protobuf Include="Protos\Login.proto" GrpcServices="Server" />  <Protobuf Include="Protos\SaveMaze.proto" GrpcServices="Server" />  <Protobuf Include="Protos\GetMazes.proto" GrpcServices="Server" />  <Protobuf Include="Protos\LoadMaze.proto" GrpcServices="Server" />  <Protobuf Include="Protos\DeleteMaze.proto" GrpcServices="Server" />  <Protobuf Include="Protos\HandleGlobalStats.proto" GrpcServices="Server" />  <Protobuf Include="Protos\HandleUserStats.proto" GrpcServices="Server" />  <Protobuf Include="Protos\GetStats.proto" GrpcServices="Server" />    </ItemGroup>      <ItemGroup>      <PackageReference Include="Grpc.AspNetCore" Version="2.59.0" />      <PackageReference Include="Newtonsoft.Json" Version="13.0.3" />      <PackageReference Include="System.Data.SQLite" Version="1.0.118" />    </ItemGroup>    </Project> |

1. https://github.com/kirsan31/winforms-datavisualization [↑](#footnote-ref-1)
2. Sourced from https://code-maze.com/csharp-hashing-salting-passwords-best-practices/ [↑](#footnote-ref-2)