|  |  |
| --- | --- |
|  | a-MAZE-ing |
|  |  |
|  | Dmitrii Ponomarev (13LBC)  Computer Science  11/8/21 |

Table of Contents

[Analysis 6](#_Toc100742883)

[Statement 6](#_Toc100742884)

[Background 6](#_Toc100742885)

[Breadth-First-Search 6](#_Toc100742886)

[Dijkstra’s Algorithm 7](#_Toc100742887)

[End-users 7](#_Toc100742888)

[Client 7](#_Toc100742889)

[Research 8](#_Toc100742890)

[Advantages of Interview 8](#_Toc100742891)

[Disadvantages of interview 8](#_Toc100742892)

[Interview 8](#_Toc100742893)

[Advantages of Questionnaire 9](#_Toc100742894)

[Disadvantages of Questionnaire 9](#_Toc100742895)

[Questionnaire/statistics gathered 9](#_Toc100742896)

[Existing System 11](#_Toc100742897)

[Existing system flowchart 11](#_Toc100742898)

[IPSO chart 12](#_Toc100742899)

[Objectives of a proposed system 12](#_Toc100742900)

[Entity Relation Diagram 13](#_Toc100742901)

[Data volumes 14](#_Toc100742902)

[System limitations 14](#_Toc100742903)

[Hardware limitations 14](#_Toc100742904)

[User limitations 14](#_Toc100742905)

[Potential solitions 14](#_Toc100742906)

[Language of Choice 14](#_Toc100742907)

[Libraries of choice 15](#_Toc100742908)

[Chosen solution and justification 15](#_Toc100742909)

[Documented Design 16](#_Toc100742910)

[IPSO chart 16](#_Toc100742911)

[Entity-Attribute Table 17](#_Toc100742912)

[1st Normal Form Table 17](#_Toc100742913)

[2nd Normal Form Table 17](#_Toc100742914)

[3RD Normal Form 18](#_Toc100742915)

[Entity-Attribute model 18](#_Toc100742916)

[2ND Normal form 18](#_Toc100742917)

[3RD Normal Form 19](#_Toc100742918)

[SQL statements 20](#_Toc100742919)

[Data Definition Language 20](#_Toc100742920)

[Data Manipulation Language 21](#_Toc100742921)

[Insert into Maze table 21](#_Toc100742922)

[Insert Into User Table 21](#_Toc100742923)

[Insert into UserMaze Table 21](#_Toc100742924)

[Insert into MazeUploaded Table 21](#_Toc100742925)

[Real SQL statements 21](#_Toc100742926)

[Submit Credentials 21](#_Toc100742927)

[Check Name and Surname Validity 22](#_Toc100742928)

[Updating Password 22](#_Toc100742929)

[Regestering user 22](#_Toc100742930)

[Fetching Mazes 23](#_Toc100742931)

[Inserting Uploaded Mazes 23](#_Toc100742932)

[Uploading Mazes 24](#_Toc100742933)

[Data Dictionary for relational Data Table 25](#_Toc100742934)

[OOp Class Design 26](#_Toc100742935)

[Class Inheritance Diagram 26](#_Toc100742936)

[Game.py 26](#_Toc100742937)

[Generate.py 27](#_Toc100742938)

[Main game options 28](#_Toc100742939)

[Hierarchy Charts 29](#_Toc100742940)

[Game.py 29](#_Toc100742941)

[Generate.py 31](#_Toc100742942)

[MakeGrid.py 32](#_Toc100742943)

[Utils.py 33](#_Toc100742944)

[FLowcharts 34](#_Toc100742945)

[Definitions 34](#_Toc100742946)

[Start of program 34](#_Toc100742947)

[LoginPage subroutine 35](#_Toc100742948)

[Login subroutine 35](#_Toc100742949)

[ForgotPassword Subroutine 36](#_Toc100742950)

[Register Subroutine 37](#_Toc100742951)

[MainMenu subroutine 38](#_Toc100742952)

[Quit Subroutine 39](#_Toc100742953)

[Data Flow Diagram 40](#_Toc100742954)

[Defenitions 40](#_Toc100742955)

[Dos and Don’ts 40](#_Toc100742956)

[Diagrams 41](#_Toc100742957)

[Level 0 41](#_Toc100742958)

[Level 1 41](#_Toc100742959)

[Level 2 42](#_Toc100742960)

[Algorithms 43](#_Toc100742961)

[Binary Tree 43](#_Toc100742962)

[Sidewinder 43](#_Toc100742963)

[Backtracker 44](#_Toc100742964)

[Pims 44](#_Toc100742965)

[Mergesort 45](#_Toc100742966)

[Breadth First Search 45](#_Toc100742967)

[System Security 46](#_Toc100742968)

[hashing passwords 46](#_Toc100742969)

[resetting input boxes 46](#_Toc100742970)

[Input validation 46](#_Toc100742971)

[Default 46](#_Toc100742972)

[name/Surname Validation 46](#_Toc100742973)

[username Validation 47](#_Toc100742974)

[Password Validation 47](#_Toc100742975)

[Interface 48](#_Toc100742976)

[Design process 48](#_Toc100742977)

[Login 48](#_Toc100742978)

[Register 49](#_Toc100742979)

[Main Manu 50](#_Toc100742980)

[Levels 51](#_Toc100742981)

[Inbuilt 52](#_Toc100742982)

[Usermade 52](#_Toc100742983)

[Sandbox 53](#_Toc100742984)

[Upload 54](#_Toc100742985)

[Actual design 54](#_Toc100742986)

[Login 54](#_Toc100742987)

[Register 55](#_Toc100742988)

[Forgot Password 56](#_Toc100742989)

[Main Menu 57](#_Toc100742990)

[Levels 57](#_Toc100742991)

[Workshop 58](#_Toc100742992)

[Sandbox 59](#_Toc100742993)

[Technical Solution 60](#_Toc100742994)

[Game.py 60](#_Toc100742995)

[MakeGrid.py 148](#_Toc100742996)

[Generate.py 155](#_Toc100742997)

[Utils.py 165](#_Toc100742998)

[CreateDB.py 180](#_Toc100742999)

[Testing 188](#_Toc100743000)

[Testing Strategy 188](#_Toc100743001)

[Objective Testing 188](#_Toc100743002)

[screenshots 196](#_Toc100743003)

[Login Page Testing 232](#_Toc100743004)

[Screenshots 234](#_Toc100743005)

[Register Testing 243](#_Toc100743006)

[Screenshots 244](#_Toc100743007)

[Forgot Password Testing 252](#_Toc100743008)

[Screenshots 254](#_Toc100743009)

[Main Menu testing 263](#_Toc100743010)

[Screenshots 264](#_Toc100743011)

[Credits Testing 268](#_Toc100743012)

[Screenshots 268](#_Toc100743013)

[Levels Testing 269](#_Toc100743014)

[Screenshots 270](#_Toc100743015)

[Workshop testing 273](#_Toc100743016)

[screenshots 273](#_Toc100743017)

[Solving Maze Testing 274](#_Toc100743018)

[Sceenshots 275](#_Toc100743019)

[Sandbox testing 279](#_Toc100743020)

[Screenshots 286](#_Toc100743021)

[Evaluation 312](#_Toc100743022)

[Objectives to be met 312](#_Toc100743023)

[Meeting objectives 313](#_Toc100743024)

[Independent Feedback 315](#_Toc100743025)

[Feedback Form 315](#_Toc100743026)

[Analysis of Feedback 317](#_Toc100743027)

[Possible Improvements 317](#_Toc100743028)

# Analysis

## Statement

Mustafa Al Jorani is a college student, doing A-Level Computer Science. The college is known for having a very good computer science department and therefore Mustafa thought he would not struggle with understanding it.

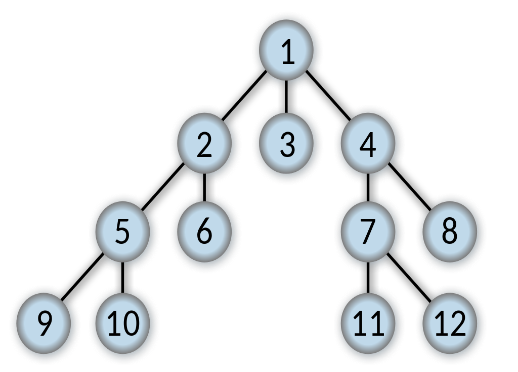
Mustafa likes solving mazes and does that on regular basis when travels to college. He prints mazes on a sheet to solve by hand. Mustafa can solve up to 5 mazes a day if he has spare time. He likes generating mazes himself as well to see how well he can do it and how hard it will turn up. Lately, Mustafa started to run out of ink and cannot print mazes anymore and started to think of software that would help him to generate and solve mazes with other players.

There are many websites allowing him to generate mazes, but not all of them allow to solve or modify them. Mustafa wants a program that would allow him to generate, solve and upload mazes for other users to solve.

## Background

Maze solving algorithms take 2 points on a grid: Start and End. There are many approaches to reach the end. One of them is always turn right, but it’s very inefficient as can lead to a loop in which the player will be stuck forever. Seeing the maze from height allows different algorithms to take place, like Breadth First Search and Dijkstra’s algorithm. Both of them will work exactly the same way if there is no weight in between the nodes\*\*, meaning that distances are the same between neighbour cells. It doesn’t affect an average human, as we don’t follow either approach and rather use intuition or see the path straight away, which sometimes is faster that algorithmic thinking.

#### Breadth-First-Search

It is an algorithm for searching a tree data structure\* for a point that satisfies a given condition e.g. Age should be equal to 10. It starts at the very top of the tree (In computer science known as “The Root”) and explores every branch attached to this node\*\*. Then in moved to the next already explored node, repeating the algorithm. Algorithms explores the tree in a “Flat” way, meaning that it goes across first, then down.

Steps for Breadth-First-Search:

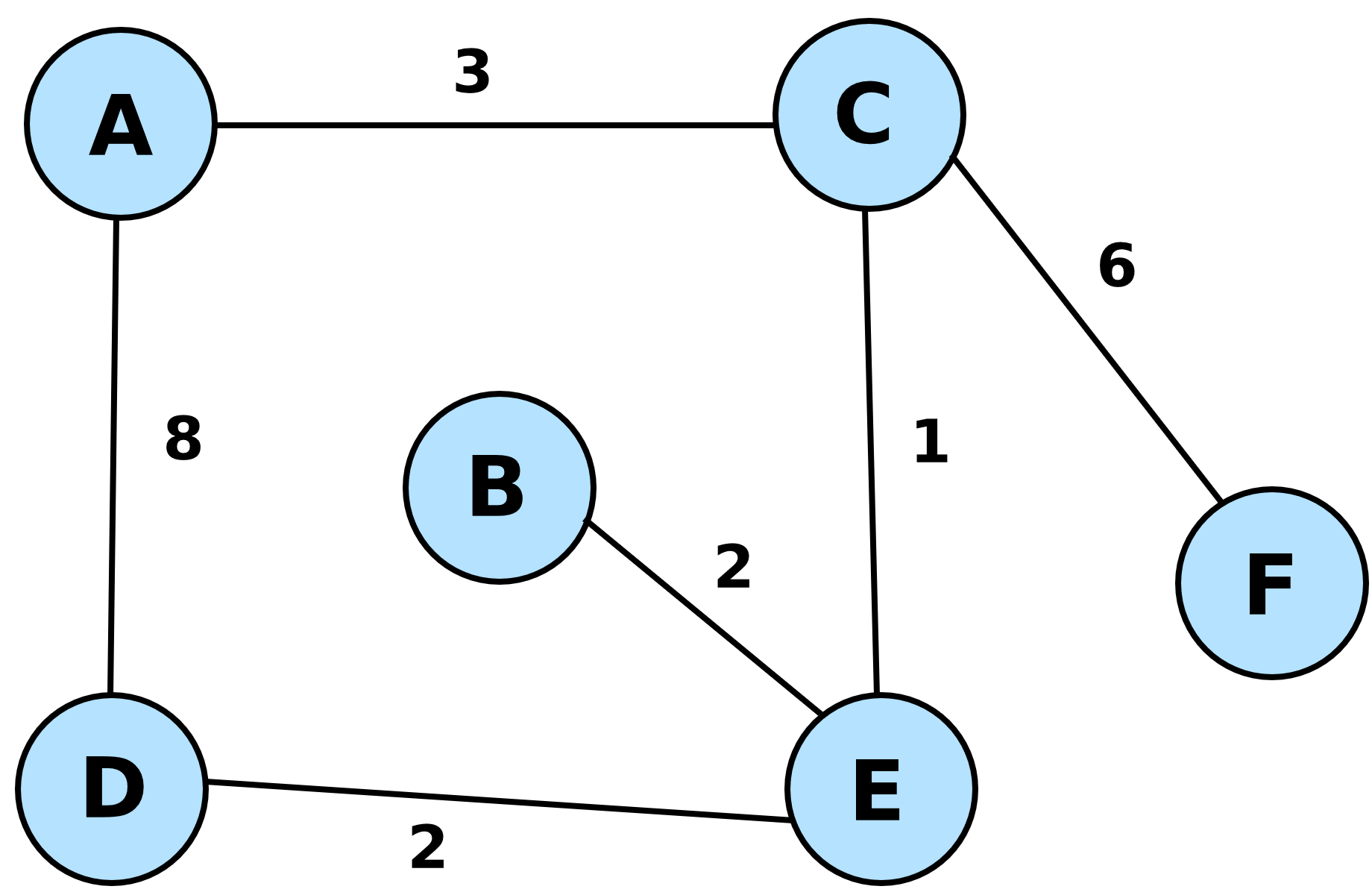
1. Explore the Root node
2. Explore all connected nodes below the root
3. Starting from the left, explore all connected nodes below
4. When all nodes explored, move up to the parent
5. Chose next node on the right
6. Repeat steps 3-6 until all tree is explored

\*Tree – Abstract data type of a tree hierarchy with a root a value and children of a root value as nodes

This image shows the sequence of nodes a Breadth-First-Search algorithm would explore.

\*\*Node – basic unit of a data structure. It contains some data including parent and children nodes if exists.

#### Dijkstra’s Algorithm

It is an algorithm for finding shortest route between 2 nodes in a given graph\*, which can be used to represent road networks for Satnav. Unlike Breadth-First-Search, where the tree is used, Dijkstra’s can explore all nodes without a root needed, as it can start exploring from any node chosen, which will be treater as the root. Biggest difference between those 2 algorithms is that Dijkstra’s finds a node with minimum distance from the source, aka. Weight. This algorithm explores all nodes connected to a parent and choses next node with minimum distance from the root.

Steps for Dijkstra’s algorithm:

1. Explore the root node
2. Check neighbours
3. Append all neighbours to the priority queue\*\*
4. Choose a neighbour from the queue with the least distance away from the parent node
5. Repeat steps 3-4 until priority queue is empty

This image shows the distance between each node. Because the algorithm can start exploring from any point, it doesn’t have a tree structure, but graph

\*Graph – abstract data type consisting of finite connected nodes (vertices) where nodes can be connected directly or via a neighbour.

\*\*Priority Queue – Abstract data structure where all elements have a First in First out structure. Priority is assigned to each element and element with the highest priority is executed first.

## End-users

The main group that will benefit from this application will be students and teachers explaining different shortest path finding algorithms, and any people interested in solving mazes and creating new ones. This application will help users to think outside the box and takes more structured approach to solve mazes.

## Client

My client is Mustafa Al Jorani, who is a computer science student in Beauchamp College in Leicester. As he is tired of solving mazes by printing them of paper, he asked me to develop a program to allow him to solve mazes on his laptop, as it will allow him to play this game anywhere. He is involved in several communities who solve mazes on regular basis and willing to upgrade his skills to compete on low-level tournaments.

The system that I will design will allow him to run the program on the laptop, which would let my client to play and solve mazes anywhere while driving or on a train as well as creating different mazes without needing to search for them online beforehand.

## Research

There are many research methods out there, like Questionnaire, Interview, Observation etc. All of them have as advantages and disadvantages, which can play a huge role in choosing the type of research.

### Advantages of Interview

* Easy correction during speech. It allows my client to give an answer if he changes their mind or to answer more in-depth if he wanted
* Relationship and Body language. During interview, clients tend to relax more and show what they want by non-verbal signals, which would be very helpful for me to fully understand client’s needs and requirements
* Time saving. By asking all questions at once, I can save a lot of once due to the fact that written communication can take a lot more days.

### Disadvantages of interview

* Bias. Client could be unfair to their decision and have an idea in their head, which would drive him and lead to a solution that he wants, rather than for most correct
* Only 1 person. For most in-depth interview I had to interview only my client. To make a wider range of opinions, I would have to interview more that 1 person which would take more time and could more difficult

### Interview

I asked Mustafa Al Jorani about their problem, as he is my client and an end-user that will be using the application on 100%. I decided to gather as much information possible to break down tasks to me and go in-depth to make the system perfect for an end-user.

Q: What do you use right now for solving mazes? Do you use websites or how do you find them?

Mustafa: I use online and cut out mazes from newspapers and magazines on a train station when WIFI isn’t available.

Q: What problems do you experience using the system you use right now?

Mustafa: Because I travel most of the time with laptop, but without internet, I can’t access websites, so I manually print out mazes and solve them using pen/pencil on the way. It becomes inconvenient when I don’t have much free time, as finding and printing out mazes takes some time.

Q: What are the benefits of using current system?

Mustafa: It teaches me a structured approach: I learn to do things step by step and know where to find what. Now mazes are easier to access and with practice it became easier to solve them.

Q: What features would you like to be implemented?

Mustafa: I think that being able to generate infinite number of mazes is a key, as finding different websites to do that becomes harder and harder. Variety and difficulty level of mazes is important as well, as sometimes I like to forget about everything and do mazes for hours, but sometimes I like to solve several in short time.

Q: Do you track your time taken? If not, do you want a software that analyses your statistics?

Mustafa: Unfortunately, I don’t track my time, movement or anything like that, however I would like to have that kind of feature. It could help me to analyse some data and become more efficient. But I don’t want my private data and statistics to be used anywhere else.

Q: What kind of mazes do you prefer? Perfect with only 1 route possible or braid where there are many solutions?

Mustafa: I like them both, as sometimes braid mazes are harder than perfect ones due to many crossroads and loops that you can get lost in, whereas in perfect ones, you can look for the root for ages, but get into a cul-de-sac and start from the very beginning.

Q: How would you like the movement to be implemented? By arrows, dragging or something else?

Mustafa: Both of them would be preferable, as depending on a maze, the strategy to solve it will vary as well.

### Advantages of Questionnaire

* Cheap to set-up. I used a platform called SurveyMonkey to get responses for free
* Variety of opinions. Since I send my link to almost every person in year 13 (158/224 students), I got a lot of responses, which estimated mean answer, meaning that more people wanted that particular feature.
* Fast results. Platform analysed answers for me and gave me statistics result instantly, which is useful, as I didn’t have to analyse it, which could lead to a potential mistakes and wrong conclusions

### Disadvantages of Questionnaire

* Lack of personality. Because most of the questions had maximum of 4 answers, respondent couldn’t go beyond set answer, therefore could not represent their true response and thoughts
* Erroneous data. A lot of people answering the survey, did it for fun without any intention to help, therefore some accidents occurred, where instead of a proper answer, entire Bee movie script was pasted.
* Dishonest answer. A lot of people answered questions based on what know and if they don’t, they either answer randomly or what they think is true, therefore showing outliers.

### Questionnaire/statistics gathered

To get different opinions from different perspectives, I made a questionnaire on SurveyMonkey, sending it to most of Year 13, Year 12 and a couple of Year 11s. I combined mass questionnaire with interview to get opinion of a mass and adapt a product to more than 1 person. It will make my opinion more diverse, therefore most of the clients will be satisfied with an application.

Some of the questions included:

* ***What year are you in currently?***

|  |  |
| --- | --- |
| Year 13 | 75% |
| Year 12 | 12.50% |
| Year 11 and below | 12.50% |
| Above Year 13 | 0% |

* ***Do you do computer science/ are you interested in computing?***

|  |  |
| --- | --- |
| Yes | 75% |
| No | 25% |
| I don’t mind | 0% |

* ***Where are short path algorithms used?***

|  |  |
| --- | --- |
| Google Maps | 50% |
| Maze Solving | 32% |
| I don’t know | 18% |

* ***How many short path algorithms are there?***

|  |  |
| --- | --- |
| 0-2 | 12% |
| 3-5 | 20% |
| 6-9 | 4% |
| 10+ | 44% |
| I don’t know | 20% |

* ***Are you interested in solving puzzles and mases?***

|  |  |
| --- | --- |
| Yes | 53.85% |
| No | 15.38% |
| I don’t mind | 30.77% |

### Existing System

There are a lot of websites and applications on the web that can solve mazes using different shortest path finding algorithms. They are based on different programming languages, including Python, C#, java and JavaScript. Most of them import images and export already solved maze without solution or showing any steps in between. These websites analyse the maze and create a tree of connected nodes at each crossroad and use Breath-First-Search algorithm to go through the maze and provide quick solution. None of the existing systems have an option to create your own maze and try out various algorithms to see the difference between them.

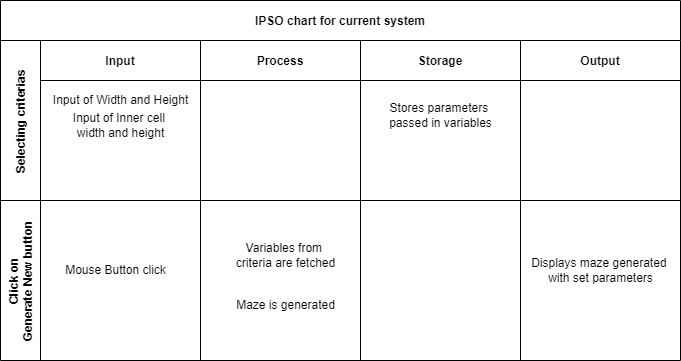
Most of the current system follow this process:

Nothing else can be done with it. In rare occasions, several algorithms are implemented and steps are shown.

#### Existing system flowchart



#### IPSO chart



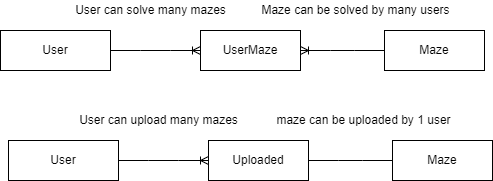
Current system doesn’t have any login system and user cannot see their progress. A lot of processes are missing due to current system is being a black box, which I don’t know the components of.

### Objectives of a proposed system

I have learned a lot from research and observations of a current system by using the software myself and interviewing clients and end-users. I was able to observe features that worked well and needed to be improved.

1. The system has to be intuitive
   1. Buttons have to change colour when cursor is over them
   2. Input boxes should display different colour when not selected
   3. All buttons should be named in intuitive way and say what their function is
2. To proceed to the next level, previous must be solved
   1. When the level is solved, text is displayed to show that the level was solved
      1. Text contains number of steps made by the user
      2. Text contains congratulation message
   2. Path that the user took is displayed with
   3. If in-built levels, if 1st level is solved, 2nd must be unlocked
   4. If last level is solved, nothing is done, as no more in-built levels exist
3. Several maze creating algorithms have to be implemented
   1. At least 3 maze creating algorithms to be implemented
   2. Random algorithm is chosen to create a maze in a Sandbox
4. New player will be able to register
   1. Username is checked against database, if it’s a copy, message is displayed
   2. Passwords are validated to increase security
   3. If any of passwords or username are not valid, message is shown
      1. If password 1 is incorrect, message like Password 1 is incorrect shown
      2. If Password 2 is incorrect, message like Password 2 is incorrect shown
      3. If Username is incorrect, message like Username is not valid shown
   4. Additional questions asked to allow password restoring
   5. If any of the security questions didn’t pass validation, messages are shown
5. User must be able to log in
   1. Check credentials before logging in
      1. If credentials are incorrect, error message is shown
      2. If username doesn’t exist, error message is shown
6. Passwords must be stored secure
   1. Encryption or hashing algorithm used
7. Users must be able to upload mazes
   1. Maze is checked for validity
      1. Solve the maze before uploading it
      2. If path exists, message of success is shown
   2. Maze is checked if it was uploaded earlier on, otherwise error shown
      1. Base128 is checked with table Maze and if record is fetched, no upload is done
8. Users will be able to solve mazes uploaded by users in a workshop
   1. User can select any level in a workshop
9. User will be able to solve levels with their devices
   1. User will be able to use arrow keys to solve levels
   2. User will be able to use WASD keys to solve levels
   3. User will be able to drag mouse to solve levels
10. Each user who forgot their password will be able to restore it anytime
    1. User will be asked for their security question to answer
    2. Passwords are validated and error messages shown if not valid
    3. Records are updated and password changes allowing user to log in by using new password
11. User is able to return to a previous menu screen at any time
    1. By pressing “Return”, the user comes back to a previous screen
    2. By pressing “Quit”, the user quits the program

### Entity Relation Diagram



User is able to upload 1 unique maze, which makes it 1-to-many connection, as they can upload many different mazes. Also, user can solve many mazes, including the same one several times as well as 1 maze can be solved by many different users, making it many-to-many connection. Many-to-Many relationship can be broken down with a use of a UserMaze table that will be containing unique keys of User and Maze tables.

### Data volumes

|  |  |
| --- | --- |
| Data Object | Volume of data |
| New accounts | 10 per month |
| Mazes solved | 10 per day per user |
| Mazes Uploaded | 12 per day |

This data volumes table gives an estimate of how much data my program would need to be able to handle. As it is a niche audience targeting, numbers of new accounts are very small, however number of mazes solved would be high due to most of users practicing solving mazes often.

## System limitations

### Hardware limitations

Because of system in every computer is different and some machines are very slow, I have decided to minimise FPS of the game to 60 to make sure that most of the machines would be able to run it without having a bottleneck by GPU or CPU.

Due to many of my algorithms use stack or another data structure that requires additional memory, I limit width and height of the maze to 40 by 40. Because of blockwise representation, it will become 81x81 cells, therefore could take up 6,561 memory locations, where each location consists of a class with parameters, leading to at least X10 memory usage increase. This limitation allows users to run the game without freezes and as fast as possible with as little memory used as possible.

### User limitations

My client is a typical computer science student at college. He knows how to operate a computer and understands basics with as little introduction as possible, however most of my clients could have limited compute knowledge, therefore requiring very simple user-friendly interface, which could be easier to understand.

I have created big buttons and large enough input boxes to see and chick on. They are visible and all have hover and click effect to differentiate between active and disables buttons. Interactive interface should look interactive, therefore have visible effects when mouse is going over it, which helps user to differentiate between a button and a text.

## Potential solitions

### Language of Choice

Considering language, I should determine several factors:

1. How well I know the language
2. How many inbuilt libraries are there
3. How fast the language is (C++ is fast, Haskell is slow)
4. How capable the language is for OOP (C# doesn’t have in-built OOP)

Summing all the factors, the language of my choice will be Python as I know it well, it’s fast, efficient, has a lot of inbuilt libraries as well as well-developed OOP, which I am planning on using.

### Libraries of choice

It is important to choose current GUI library, as most of the time user on the screen which displays picture. Therefore, GUI has to be neat and easy to understand. It also has to be robust as many functions would be re-used throughout the program with same parameters.

There are many libraries that would let me to built and design graphical user interface. Some of them would be listed in a table below.

|  |  |  |
| --- | --- | --- |
| **Library** | **Pros** | **Cons** |
| tkinter | Easy to use  Doesn’t have to be installed as in-built | Looks antient and old. Could be hard to display maze, as not buttons nor labels |
| Pygame | Allows full customisation  Doesn’t have to be installed as in-built | Complex to refresh as whole screen has to refreshed |
| Kivy | More of an mobile app GUI, than for PC  Customisation is high | Has to be installed separately  More optimised solution has to be done as phone power < computer power |
| PyQt5 | Coding flexibility  Uses Networks and database creation inbuilt  Various UI components (Buttons, radio buttons, scroll and loading bar) | Lack of documentation and support  Requires a lot of time to learn and understand  Has to be installed |

## Chosen solution and justification

Considering options, I have and time left, I chose to do Python + Pygame as it allows to me use customisable Buttons and lets use functionality of cursor separate from affecting anything else. It lets me use colourful constraints and allows to build unique functions for user input verification, which would be nearly impossible to do in Tkinter.

Some of the libraries I will be using will be:

Hashlib: Allows efficient and fast hashing encoding

SQLite3: Allows easy Database and table access. Inbuilt in python, therefore doesn’t need to be installed, so more robust library, than Pandas, which offers more precise DDL and DML, but needs to be installed.

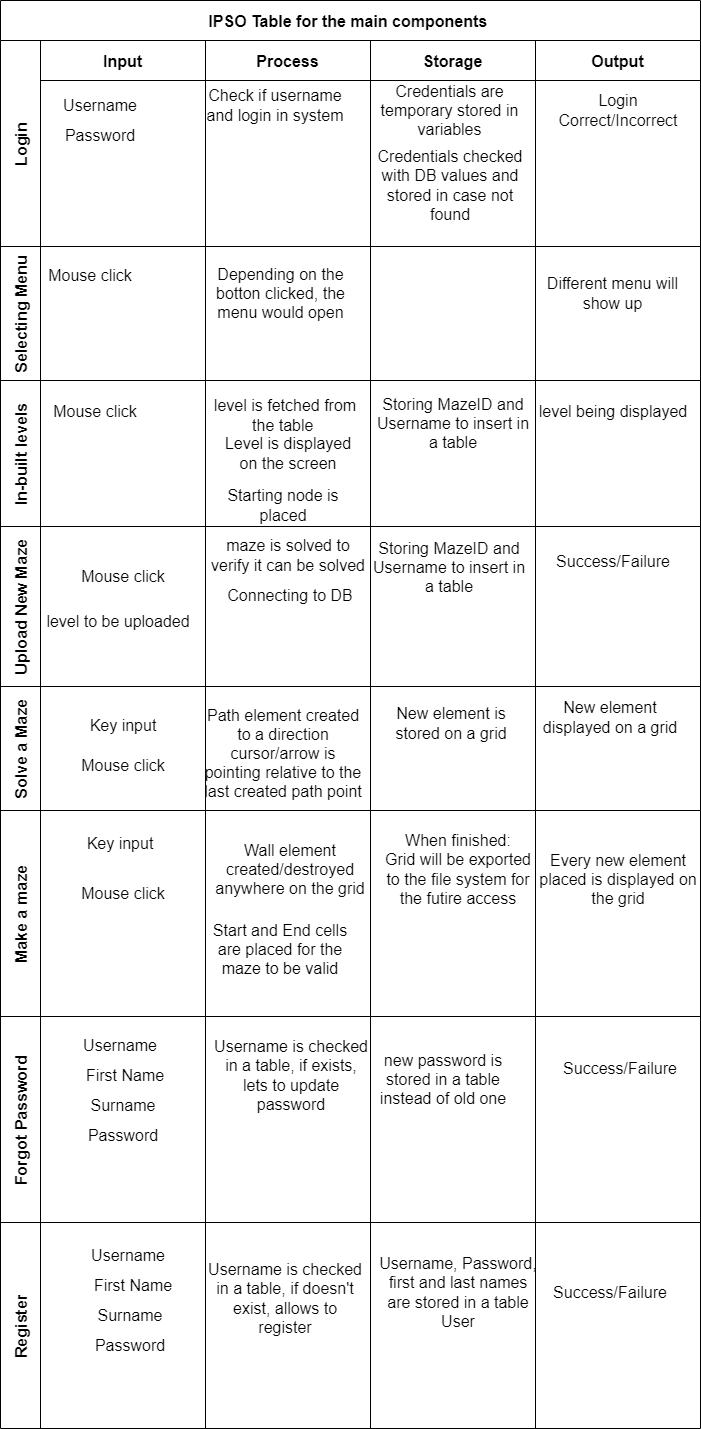
Random: Allows to generate random numbers and easy shuffle of the lists, which come handy in randomising maze generation.

Pygame: Allows to create visual effects and GUI using fully customisable properties.

OS: To disable Pygame Hello Message

# Documented Design

## IPSO chart



## Entity-Attribute Table

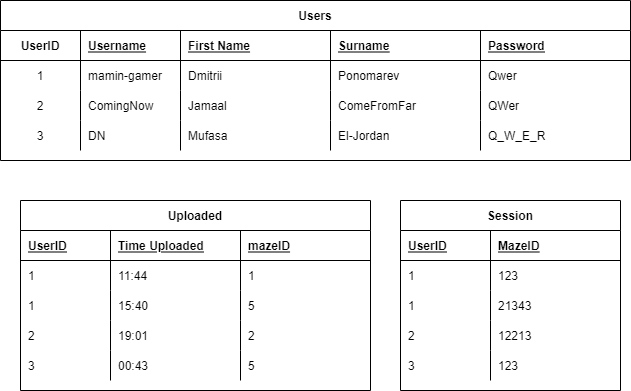
### 1st Normal Form Table

Table Continued…



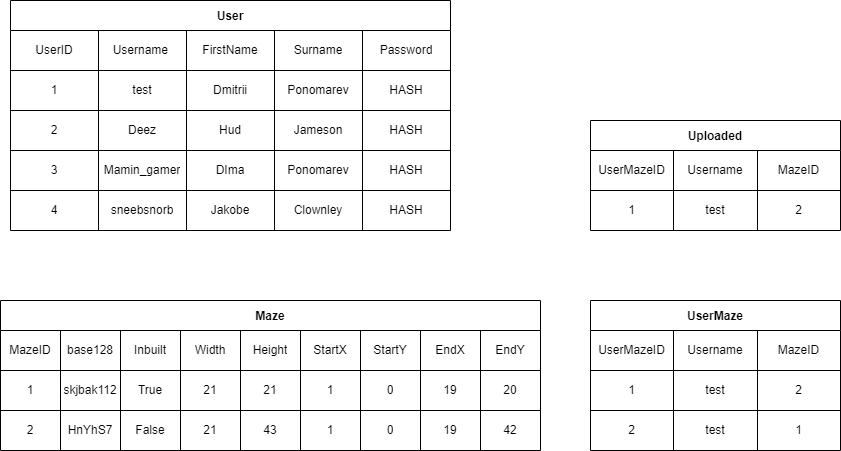
In first normal form, the table is very long and each record doesn’t correspond fully to the key, like Username has no dependencies on MazeID

### 2nd Normal Form Table



In second normal form there are more tables which allow to separate dependencies and let us see what value correspond to what table. There are still partial dependencies, which can be fixed in 3rd normal form

### 3RD Normal Form



Big table has many occurrences of same user and therefore has to be broken down using junction tables Uploaded and UserMaze, which takes primary keys from Mazes (Isn’t shown) and Users. Each table has a Unique key and filly broken down which can be classified as a 3rd normal form with a phrase “Key, the whole key and nothing but the key”.

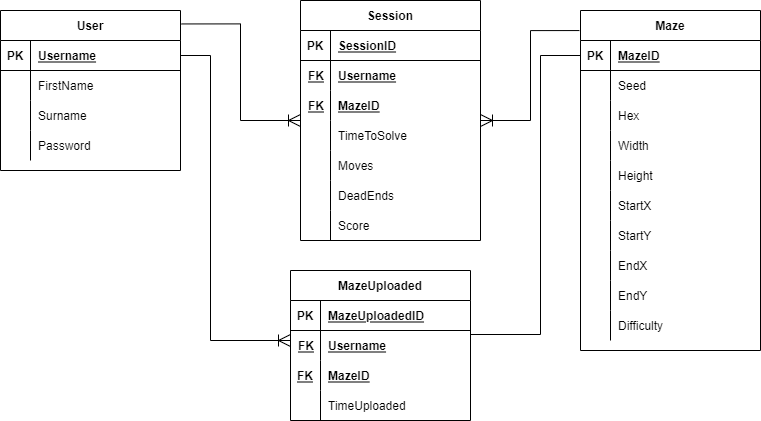
### Entity-Attribute model

#### 2ND Normal form



By representing the table as a table of attributes, rather than an actual example, tables become shorter and flatter, which is easier to understand for humans to understand. It also clearly shows the connection between the table and sets restriction for data flow.

#### 3RD Normal Form



This Entity-Attribute diagram is in 2nd normal form as contains composite keys for Session and MazeUploaded, however, there might be an error occurring as one user can solve one maze several times, leading to the same foreign keys to be importing, causing there to be 2 same entities which might be difficult to deal with.

By removing UserID and changing Primary key of User table to Username, it will eliminate the need to make check of user by the UserID, instead, make unique username and simplify as search as storage in a table.

In this case, I made a breakdown of both Session and MazeUploaded tables to both having primary keys. That made tables having no non-key dependencies, which makes a database be normalised to a 3rd Normal Form.

## SQL statements

### Data Definition Language

Using Data Definition Language, I can see how to create the tables in a Database. It will help me to see what key is related to what table, which will then help me to set up tables and fetch values from them.

CREATE TABLE IF NOT EXISTS *User*

Username text PRIMARY KEY,

FirstName text NOT NULL,

Surname text NOT NULL,

Password text NOT NULL

CRATE IF NOT EXISTS *Maze*

MazeID integer PRIMARY KEY,

Hex text NOT NULL,

Inbuilt Bool NOT NULL,

Width integer NOT NULL,

Height integer NOT NULL,

StartX integer NOT NULL,

StartY integer NOT NULL,

EndX integer NOT NULL,

EndY integer NOT NULL,

CREATE IF NOT EXIXTS *UserMaze*

UserMazeID integer PRIMARY KEY,

Username text NOT NULL,

MazeID text NOT NULL,

FOREIGN KEY (Username) REFERENCES User (Username)

FOREIGN KEY (MazeID) REFERENCES Maze (MazeID)

CREATE IF NOT EXISTS *MazeUploaded*

MazeUploadedID integer PRIMARY KEY,

Username text NOT NULL,

MazeID text NOT NULL,

FOREIGN KEY (Username) REFERENCES User (Username)

FOREIGN KEY (MazeID) REFERENCES Maze (MazeID)

FOREIGN KEY (Username) REFERENCES User (Username)

FOREIGN KEY (Hex) REFERENCES Maze (Hex)

\*Each table name is highlighted in red and italic\*

It is important to have those commands in a separate file to create DB on a new computer and be able to load the game straight away. Without those commands being called, program is less robust and the user will need to create tables on their own, which is difficult for some users who aren’t doing computer science and don’t know how to deal with SQL statements

### Data Manipulation Language

“?” represents values that have to be inserted in a table.

#### Insert into Maze table

INSERT INTO Maze(Base128, Inbuilt, Width, Height, StartX, StartY, EndX, EndY)

VALUES (?,?,?,?,?,?,?,?)

MazeID doesn’t need to be inserted, as it autoincrements

#### Insert Into User Table

INSERT INTO User(Username, FirstName, Surname, Password)

VALUES (?,?,?,?)

Username needs to be inserted manually, as a username is unique and is a primary Key

#### Insert into UserMaze Table

INSERT INTO UserMaze (MazeID, Username)

VALUES(?,?)

Both username and Maze ID are primary keys of different tables which will be combined into one junction table

UserMazeID doesn’t need to be inserted, as it autoincrements

#### Insert into MazeUploaded Table

MazeID\_fetched = Maze.Lastrowid() #fetches the last

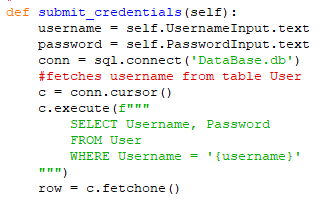
INSERT INTO Uploaded(Username, MazeID,)

Values (?,?)

Before inserting into Uploaded, I inserted to Maze the new maze, without knowing its ID. Another way of doing it would be to select ID from the table where Base128 (Unique parameter) is equal to the one we just inserted

### Real SQL statements

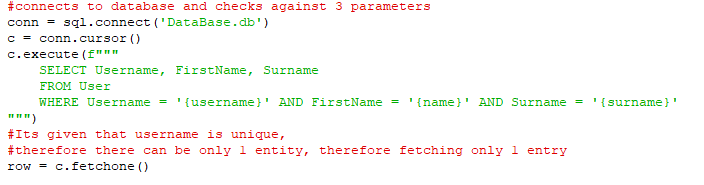
#### Submit Credentials

Fetches Username and Password from table User with the same username.

It’s given that Username is a unique parameter, therefore no need to fetch all values.

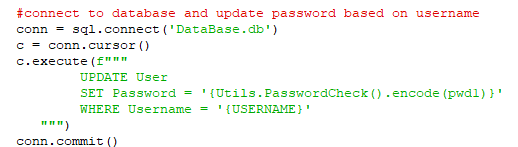
If row is empty, no username exists, therefore asking user to register. If row exists, checking and validating password

### Check Name and Surname Validity



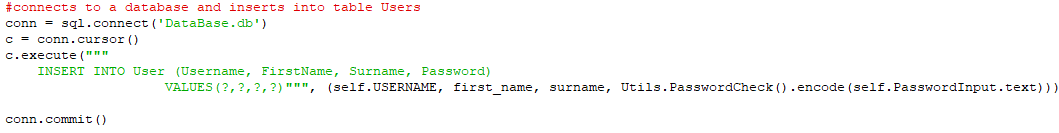
Fetches Username, First name and Surname from the table User where Username, FirstName and Surname and checks if they are equal to username, name and surname respectively. Given that Username is unique, therefore there would be 1 output at most. If there is something in a row, it means that all criteria match and lets user to proceed to reset the password.

### Updating Password



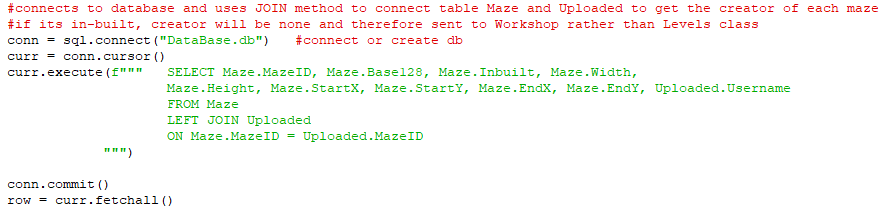
After password was verified, I Update the record of User by setting password to a hashed version of whatever password the user inputs. I use WHERE statement to get unique attribute, which field Password will be replaced with new one.

### Regestering user



Inserts Username, first\_name, surname, hashed password into table User when registering the user. By using ? I don’t have to user f-string in Python, which would make this string look very long and difficult to read.

### Fetching Mazes

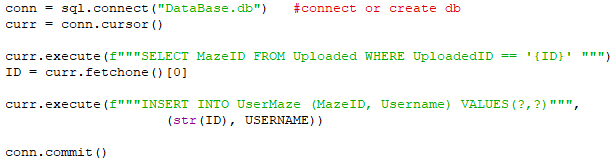


Connects to a database and selects 2 tables by using LEFT JOIN, which connects 2 tables together, which allows to use Uploaded table connecting user-made levels.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| MazeID | Base128 | Inbuilt | **…** | Username |
| 1 | aHGVuy | 1 |  | None |
| 2 | ^Tjgmtuh7&& | 0 |  | Mamin\_gamer |
| 3 | 7HFrtg%^6uI | 0 |  | Test |
| 4 | kjY^uyvf%554$G | 1 |  | None |

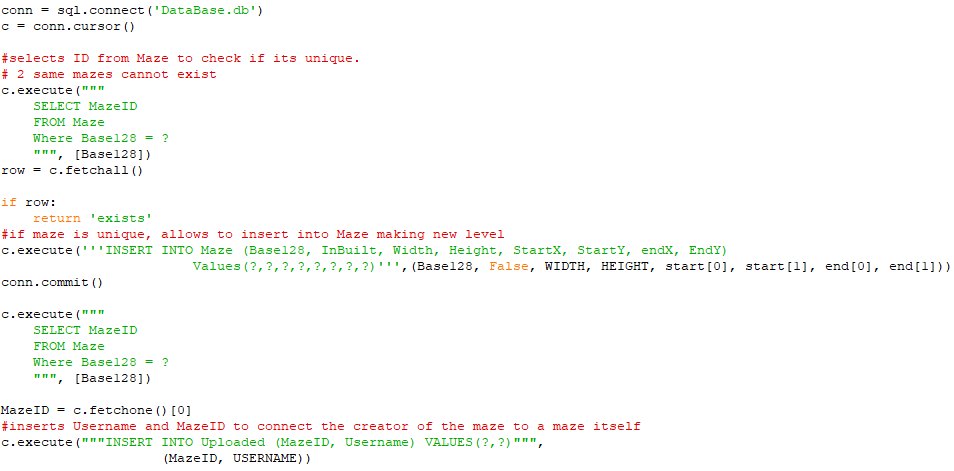
It allows to fetch all entries from table Maze and build levels from it.

### Inserting Uploaded Mazes



Firstly, we fetch MazeID from Uploaded table where we just uploaded a new entry consisting of MazeID and UsernameID. UploadedID is known as it is a function made in Workshop class where UploadedID are stored in levels, therefore no need to fetch it. After fetching MazeID, which is unique as a Primary key, I INSERT it along with USERNAME, which is a global variable into UserMaze table which keeps the record of mazes solved by hand.

### Uploading Mazes



There are 4 steps to upload a maze:

1. Check if maze already in a database
2. If not exists, INSERT new entry into table Maze
3. Fetch new MazeID from Maze
4. Insert new MazeID with USERNAME into Uploaded

## Data Dictionary for relational Data Table

A data dictionary will let me see the data that will be going to a database and validate it. It is important to see what data goes in to identify primary and secondary keys and make sure that no other data will be required/misinterpreted

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | User | | |
| Primary Key | Username | | |
| Data Items | Data type | Validation | Example |
| **Username** | String | NotNull | Mamin\_gamer |
| FirstName | String | NotNull | Mustafa |
| Surname | String | NotNull | Al Jorani |
| Password | String | NotNull | HASH encoding |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Maze | | |
| Primary Key | MazeID | | |
| Data Items | Data type | Validation | Example |
| **MazeID** | Integer | NotNull | 1 |
| Base128 | String | NotNull | 85AaG1 |
| Inbuilt | Boolean | NotNull | True |
| Width | Integer | NotNull | 20 |
| Height | Integer | NotNull | 20 |
| StartX | Integer | >=0 | 7 |
| StartY | Integer | >=0 | 1 |
| EndX | Integer | >=0 | 4 |
| EndY | Integer | >=0 | 8 |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | UserMaze | | |
| Primary Key | **UserMazeID** | | |
| Foreign Key | Username | | |
| Foreign key | MazeID | | |
| Data Items | Data type | Validation | Example |
| **UserMaze** | Integer | NotNull | 52 |
| Username | String | NotNull | Mamin\_gamer |
| MazeID | String | NotNull | Maze001 |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Uploaded | | |
| Primary Key | **MazeUploadedID** | | |
| Foreign Key | MazeID | | |
| Foreign Key | Username | | |
| Data Items | Data type | Validation | Example |
| UploadedID | Integer | NotNull | 2 |
| MazeID | Integer | NotNull | 100 |
| Username | String | NotNull | Mamin\_gamer |

## OOp Class Design

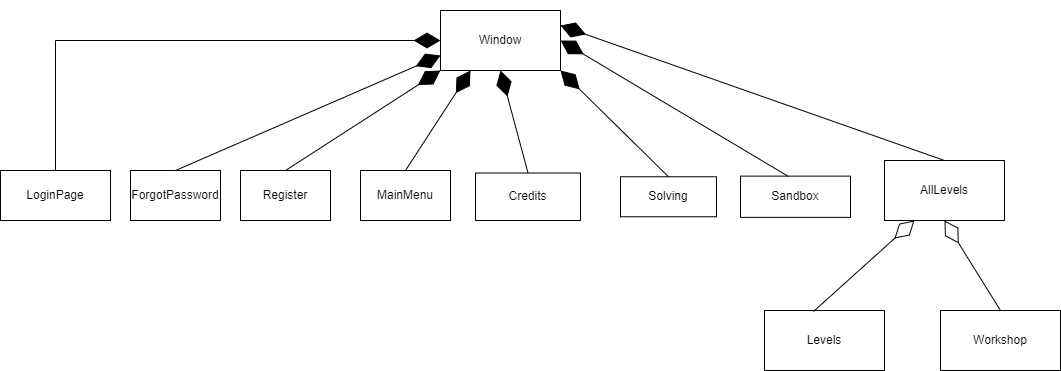
### Class Inheritance Diagram

By using class inheritance, I am able to use methods and parameters of parent Classes, like WIN\_WIDTH from Window is being used all the way in Levels and Workshop.

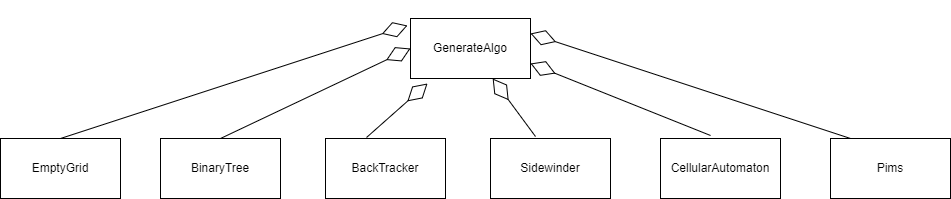
It is useful to be able to inherit functions as it makes the code robust, as an example class AllLevels contains procedure add\_buttons, that is being called in both levels and Worshop, which helps me not to write the same code with minimal changes in both.

By showing aggregation and composition, I can show relationship between 2 classes. As LoginPage, Register etc. require parameters in a Window class to run, they are a part of window

#### Game.py

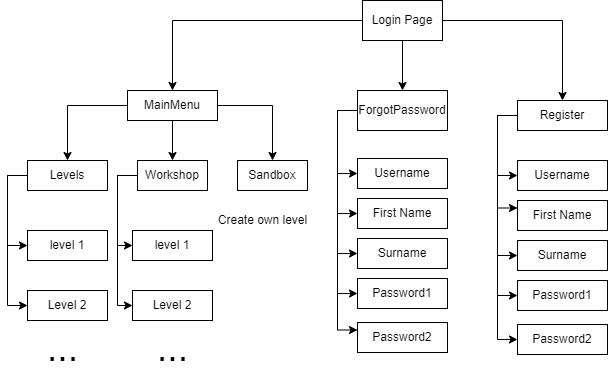


#### Generate.py



EmptyGrid, BinaryTree etc can co-exist without GenerateAlgo Class, therefore aggregation is shown, suggesting that GenerateAlgo has all those classes

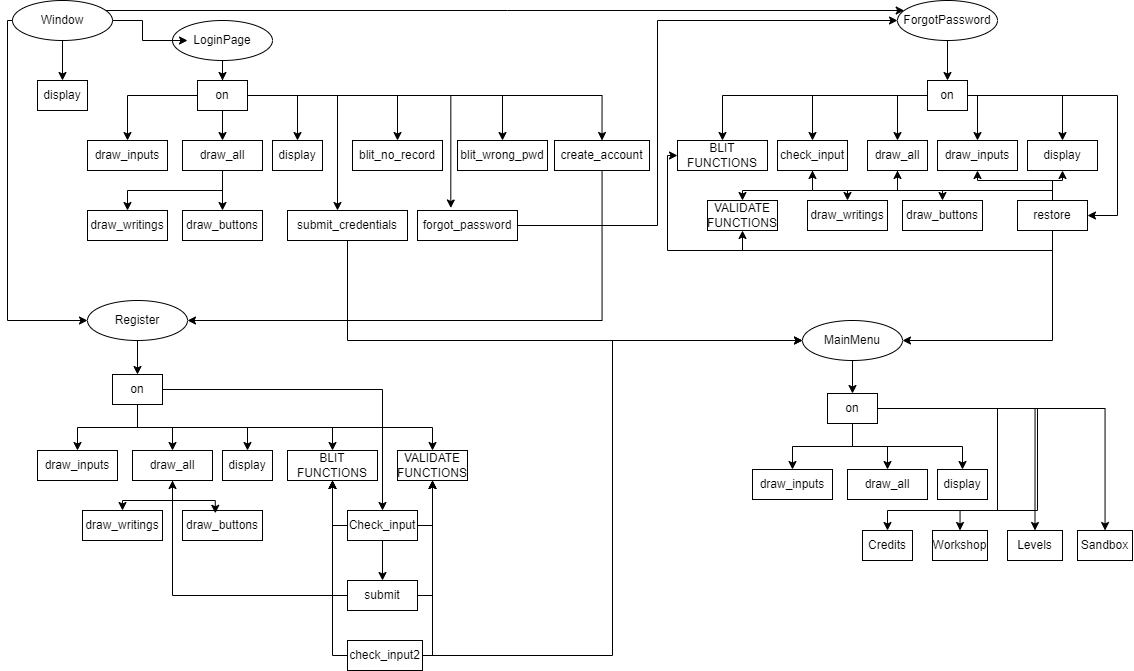
### Main game options

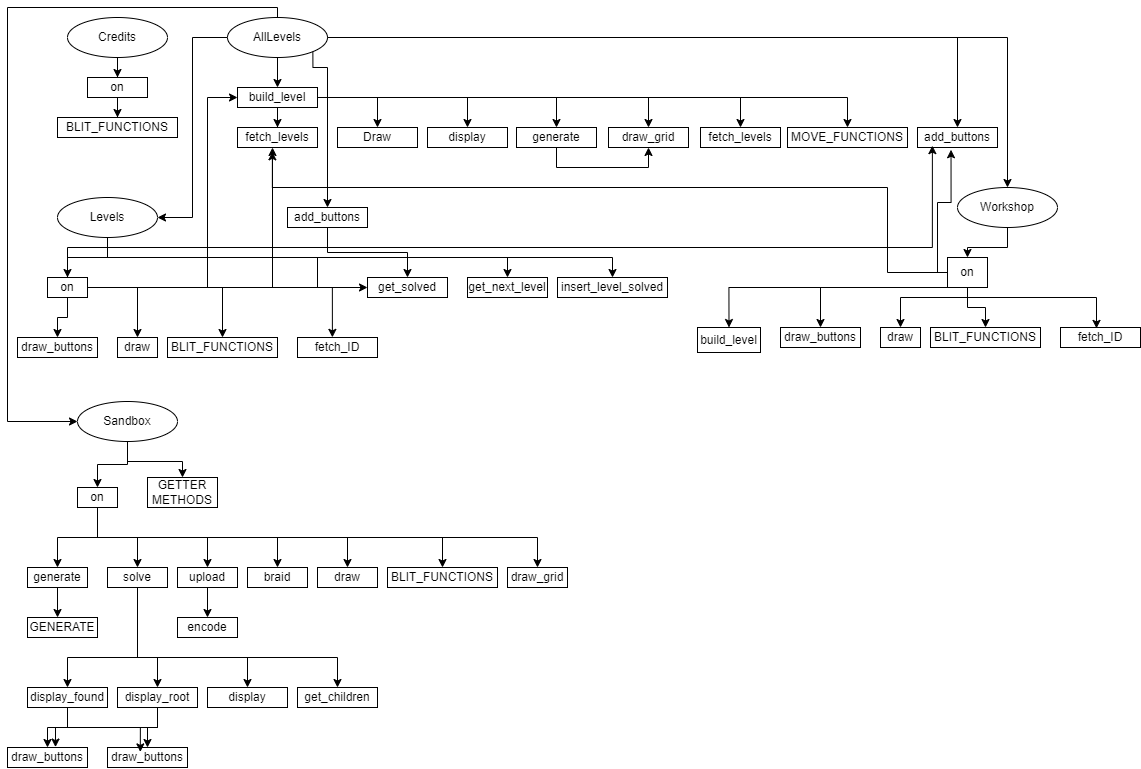


Top-down design of a main function lets me see what is called after what. It’s different from a flowchart, as doesn’t show processes, but rather options the user has and what can be selected from each menu.

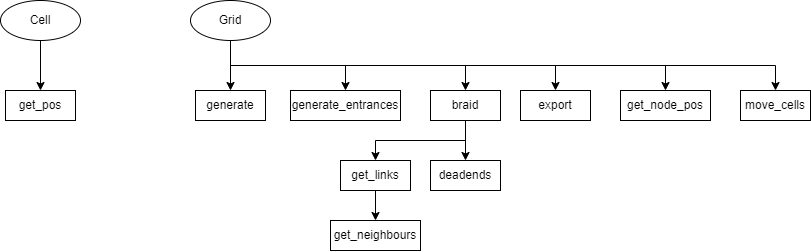
## Hierarchy Charts

### Game.py

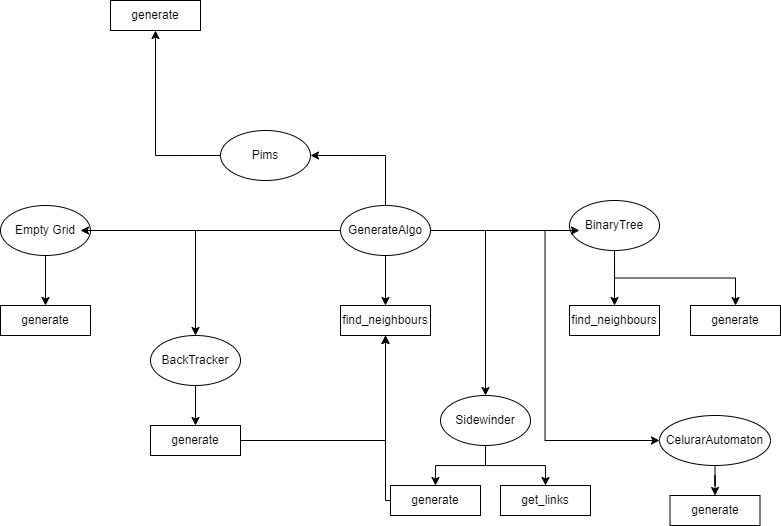




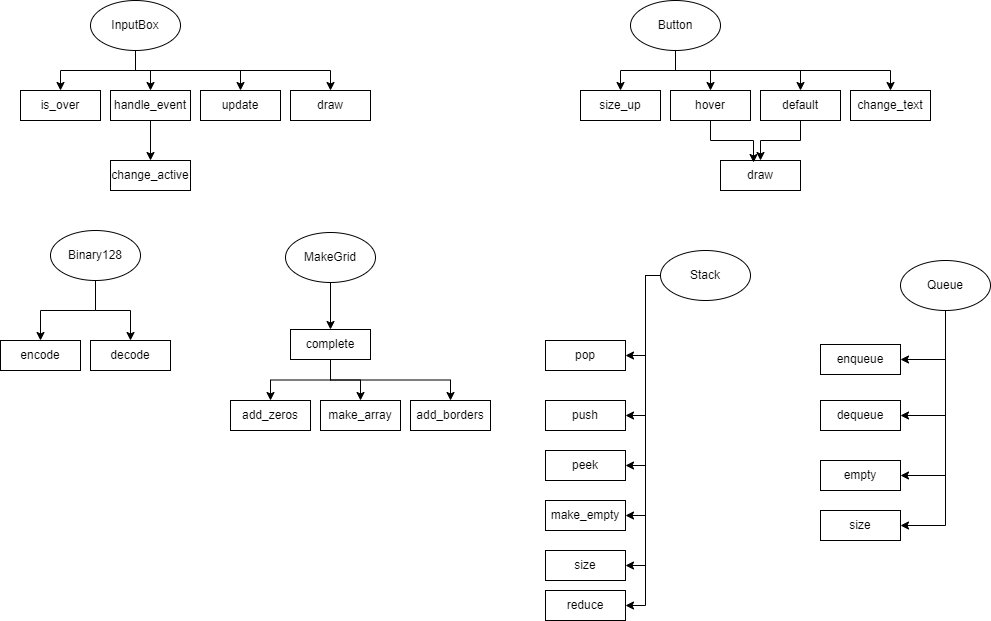
### Generate.py



### MakeGrid.py



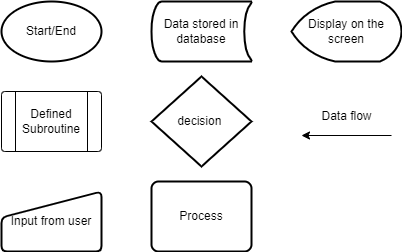
### Utils.py



## FLowcharts

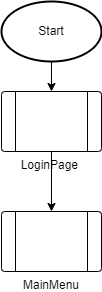
Flowcharts help to understand the basic workflow of the processes and shows representation of functions in visual form. It is easier for people to look at arrows rather than if and else statements. Flowchart shows sequence of steps as boxes of different shapes and the flow as arrows.

### Definitions



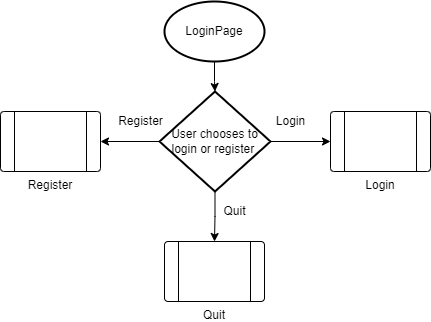
By defining each block, it will be easier to reference and understand flowcharts.

### Start of program



This is the very start of the program. Subroutine “LoginPage” will be called and will have a choice of a Registering if the user is not registered or Login in if the user already exists.

### LoginPage subroutine

“LoginPage” subroutine is used to ask user to decide between 3 buttons that do different actions;

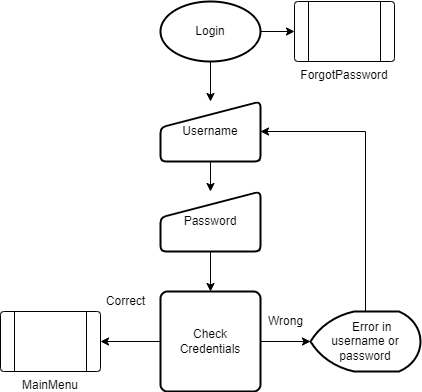
Login button will take users to a “Login” subroutine, which will ask users for login credentials

Register button will taker users to a “Register” subroutine, which will ask users for Username, Password x2, Name and Surname

Quit button will take users to “Quit” subroutine, which closes window and stops the program

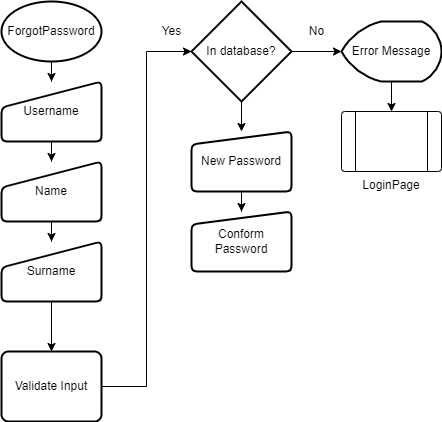
It is essential to have an option to Register or Login as new users wouldn’t be able to use the application without being already in the system, which restricts number of users that could potentially use the system.

### Login subroutine

Login subroutine asks for user login information if known. It is important to have a branch-out available to “ForgotPassword” as user can forget their credential and go straight there.

If the credentials will not be found in a User table, error will be displayed, which will make user either type credentials again or click on “Forgot Password” button that will take user to “ForgotPassword” subroutine

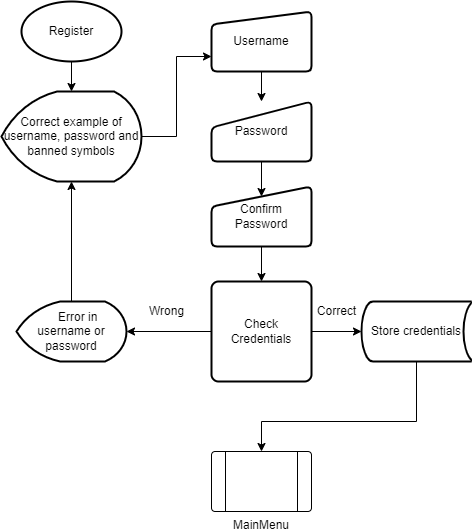
### ForgotPassword Subroutine



“ForgotPassword” subroutine is called when user clicks on the button “Forgot Password”, which means that the user forgot their credentials and wants to either restore it or to create a new account.

It is important to ask for as many details from the user as possible such as Name and Surname so that the program can verify the person and allow them to restore the password. Otherwise people can get access to other people’s accounts.

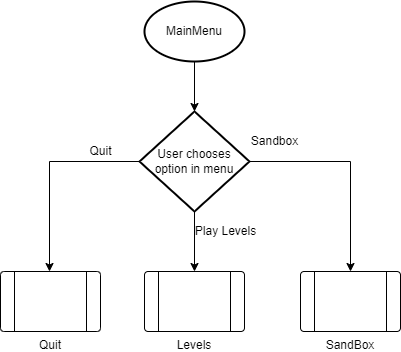
### Register Subroutine



“Register” subroutine allows user to input their credentials for future use of the system. Initially, the side note of correct username and password displayed in order to minimise user mistakes. Input will be checked with the use of Regex to make it easier and faster.

Regex will allow the subroutine not to have any loops and therefore next to no delay in processing. Before storing credentials, in the process we check is same username already exists in the User table not to have duplicates as username is a Primary Key and must be unique.

### MainMenu subroutine



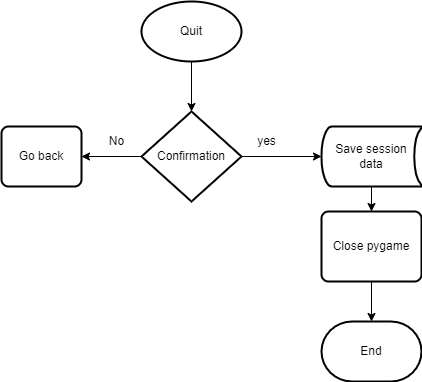
“MainMenu” subroutine is called after successful Login/registration of a user. Main menu contains 3 buttons, where each button calls different functions:

“Quit” will close window and stop program running

“Levels” will open option to choose from: Level and Workshop (Space where users upload their mazes). Levels are pre-set stage where seed applies to generate same layout every time.

“SandBox” will open a grid for the user to experiment it. You will be able to copy any maze form workshop (Space where users upload their mazes).

### Quit Subroutine



“Quit” subroutine allows user to close the game and save the progress they were on. It is important to save the session data as it lets users to continue using the application next time. During quitting the game, entire session in saved in a table using Data Manipulation Language SQL.

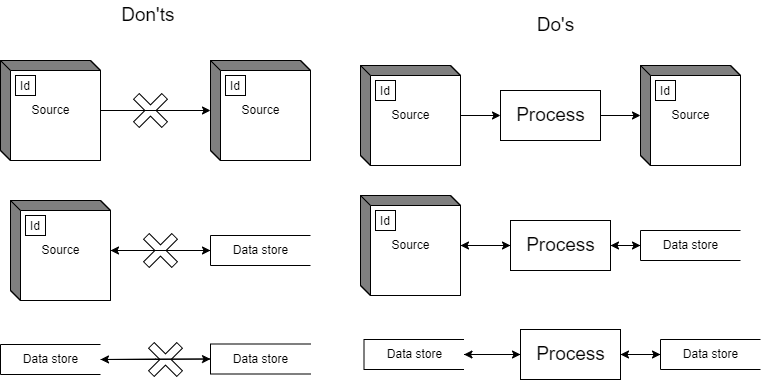
## Data Flow Diagram

Data flow diagrams represent graphically how data is flowing between subroutines and database. Visual representation is very useful for a communication between user and system designer. Data flow Diagrams show very basic flow at level 1 and then with each stage adds more details to it, making it easier to understand in a complex system.

### Defenitions

Before making Data Flow Diagram, user needs to know what each block does and how to use it correctly. However, there are some rules for DFD, which have to be completed in order for diagram to be complete and sufficient.

### Dos and Don’ts



Data flow diagrams allow us to see how data is travelling in between the user and storage. It is necessary to have DFD to make sure the flow is complete and no rules are broken. There must be a process between transferring data to somewhere e.g., Validating user input before submitting to DB

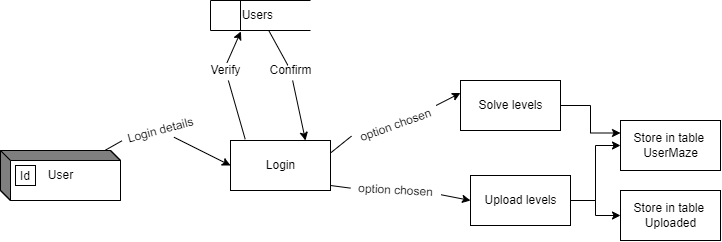
### Diagrams

#### Level 0



Level 0 diagram has only 1 source and 1 process that exchange data

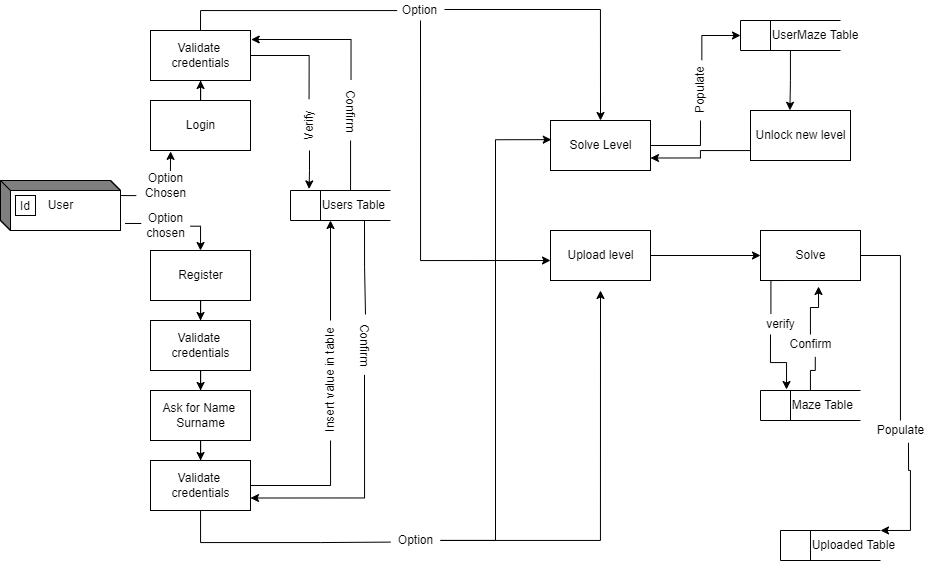
#### Level 1



Level 1 is made more in-depth to show the process of Solve/Upload maze. As each process is a black box, we can go in-depth as long as we can write structures pseudocode from the function knowing for sure what it does.

At level 1, its very hard to say what each function does, therefore we need to explore more down

#### Level 2



Level 2 diagram shows most of the processes and their chains. It’s made for people to understand the flow of data and better functionality of the program. I didn’t specify in Do’s and Don’ts, but you can make data flow from Process to Process without anything in the middle. It’s called chain of flow. I use it to show register and login processes more in-depth, but not too much, as each of those processes can contain more sub-processes from another subroutines and files.

## Algorithms

My program allows a random algorithm to be used to create a unique maze. Algorithms are very hard to design, therefore most of them are adapted from this website <https://pragprog.com/titles/jbmaze/mazes-for-programmers/>. All of the code is in Ruby language, so I took time and translated the code into Pseudocode, only after than Python.

It helped me to structure my programming and let see the steps in the code I have to take.

### Binary Tree

Generates a single path looking like branches because it can choose only 1 neighbour form east and south of the node. It doesn’t use any memory, meaning that there is no limit to the size of Maze you can create. It can build the entire maze by looking at each cell independently. This is the most straightforward and fastest algorithm possible.

Grid = 2Darray (Cell, wall) #creates a 2D array of a Cell of walls

FOR row in HEIGHT

FOR column in WIDTH

Grid[row][column] = empty #empties the cell in a grid by making it empty grid

Random\_neighbour = random((0,1), (1,0),) #picking a random set of values to get a new random neighbour cell

Grid[Row+random\_neighbour][column + random\_neighbour] = empty #making a neighbour empty to connect them

Return (grid)

### Sidewinder

Binary Tree- like maze generator, but considers whole row at a time, rather than one single cell. While Binary tree has 2 passages throughout 2 connected borders, Sidewinder has only 1.

1. For each cell randomly decide whether to carve a passage leading East
2. If the passage is carved add the cell to the current run set
3. If the passage is not carved, randomly pick one cell from the route set, carve a passage leading North and empty the current run set

Grid = 2Darray (Cell, wall) #creates a 2D array of a Cell of walls

SKEW = 0.5 #more skew, more vertical the maze becomes, 0.5 to keep it mixed of both

FOR col in WIDTH

Grid[1][col] = empty #first row is always empty, because you cant carve north

For row in HEIGHT

Run = []

For col in WIDTH

Grid[row][col] = empty

Run.append(Grid[row][col])

Carve = random() > SKEW #picks a random number within 0 and 1 and if less than

IF Carve and col < WIDTH # SKEW, then it carves east cell, otherwise north

Grid[row][col+1] = empty

ELSE

Cell = Random(Run)

Cell[row+1][col] = empty

### Backtracker

Only 1 path exists that is most likely to go through entire maze. It is very slow to generate and takes a lot of memory as requires as stack-like structure is used

Grid = 2Darray (Cell, wall) #creates a 2D array of a Cell of walls

Row = random(1, HEIGHT)

Col = random(1, WIDTH)

Track = STACK()

Track.push((row, col)) #makes use of stack

Grid[row][column] = Empty

WHILE Track

Row, col = track.peek()

Neighbours = find\_neighbours(row, col) #finds random neighbours

If neighbours.length = 0 #if no neighbours found (deadend) -> returns to previous

Track.pop() #node

Else

R, C = neighbours[0]

Grid[R][C] = empty

Track.push((R,C))

### Pims

Prim's algorithm creates a tree by getting the adjacent cells and finding the best one to travel to next. To Generate mazes using Prim's, we will instead take a random cell to travel to the next one. Running faster, it still requires storage proportional to the size of the Maze.

Grid = 2Darray (Cell, wall) #creates a 2D array of a Cell of walls

row = Random(1, HEIGHT) #Picks random node to start and sets it to empty

col = Random(1, WIDTH)

Grid[row][col] = empty

Neighbours = random\_neighbour(Cell) #finds neighbours connected to a starting cell

Visited = 1

WHILE visited < WIDTH\*HEIGHT #goes through every single cell -> slow

Rn = random(neighbours).index() #gets a Random Node index from neighbours list

Row, col = grid[RN].get\_pos()

Neighbours = neighbours.pop(Rn) #removed Random Node index from neighbours

Y, X = first\_random\_neighbour().get\_pos() #gets random neighbour and makes it empty

Grid[Y][X] = empty

Neighbours.append(Unvisited(Cell)) #appends to list neighbours all unvisited cells being #neighbours of selected cell

### Mergesort

Sorts an array with the use of recursion by splitting each element in array into 2 even chunks until there are as many arrays of single elements in there as there are elements. It uses more O(n) memory, where n is number of elements and average run-time is O(nlog(n)), which is significantly faster than Bubble sort.

I used merge sort queue in Dijkstra’s algorithm. I needed to fetch the node with minimum fCost, which would be done most effective by using Merge Sort

MergeSort(arr, left, right):

if left > right

return

mid = (left+right)/2

mergeSort(arr, left, mid) #splits into arrays for left and right

mergeSort(arr, mid+1, right) #using recursion

merge(arr, left, mid, right)

### Breadth First Search

Algorithm for searching in a tree data structure. It allows to search graphs as well, which can have loops inside. Extra memory is used and can be slow, as there is no end node visible, therefore searches in every direction

BFS(root):

Q = Queue() #use of queue for efficiency and simplifying task

Root = explored

Q.enqueue(root)

While Q.is\_empty = FALSE

V = dequeue()

If V = end\_node:

Return V

For edges v to w in root.neighbours() #discoveres neighbours and adds them to

If w not explored #explored queue

W = explored

Q.enqueue(W)

## System Security

### hashing passwords

System security is one of the most important things coming to storing passwords. Therefore, I’m implementing SHA hashing algorithm by hashlib library, that will hash the password for me to make it the most secure.

Class Password check is responsible for hashing and comparing hashed string to a string fetched from a table User Password.

class PasswordCheck:

#uses hashlob library to encode password and return hex value of it

def encode(self, text):

return hashlib.sha1(text.encode()).hexdigest()

#compares encoded string to hashed value taken from database

def check(self, text, pwd):

if self.encode(text) == pwd:

return True

return False

### resetting input boxes

Each time user inputs something into an input box like username of password, it must be erased, so when the user returns, field is empty and there is no password or text being displayed. It prevents from other users seeing the password’s length, which is another step to protect from password leaking and prevent from brute force attack.

self.UsernameInput.text = ''

self.UsernameInput.display\_text = '' #resetting input fields to display nothing in it

self.PasswordInput.text = ''

self.PasswordInput.display\_text = ''

## Input validation

Input validation is very important in my game as it restricts user from using SQL injections or makes password more secure. Easiest way to validate an input is Regex. It is a expression such as sequence of characters that specifies a search pattern in text.

I use validation for all inputs as Pygame has no inbuilt input boxes and therefore a validation for them. Because I input characters from Unicode, there can be characters as SHIFT or ESC (Special Purpose), which will add length to input box, but will not be displayed.

### Default

I set me default regex to be [A-Za-z0-9@$!%\*#?&-\_ ] it has all upper and lower case letters including digits. The rest are special characters, which also can be inputted into login or password.

### name/Surname Validation

Function is set to check this regex ^[a-zA-Z]{2,30}$ against string inserted. It starts by checking lowercase and uppercase letter. ^ means the start of a string, $ means the end of the string. {2,30} states, that the name should be between 2 and 30 characters

### username Validation

Function to validate the username is important as username can contain special symbols, but not allowed by set specified. I narrowed down special characters use to – and \_ , because I think they are the most used ones.

Validation: ^[a-zA-Z0-9\_-]{2,30}$ . Username can have upper and lowercase letters as well as numbers and dash with underscore. It also must be within 2 and 30 characters, as 1 letter username will be taken too quickly, and it’s not secure enough.

### Password Validation

Password is a little more complex to validate as it must meet some criteria’s, such as: At least 1 Upper case, At least 1 lower case, At least 1 digit, At least 1 special characters chosen from these: @$!%\*#?&-\_

Validation: ^(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*#\_?&-])[A-Za-z0-9@$!%\*\_#?&-]{8,30}$

^ # Start of a string

(?= # Positive Lookahead - Matches a group after the main expression without including it in the result.

. # Matches any character except line breaker

\* # Quantifier – marches 0 or more preceding tokens

[ # Character Set – match any character inside the set

A-Z # matches all uppercase letters

]

)

(?=

.

\*

\d # matches any digits from 0 to 9 including

)

(?=

.

\*

[

@ # matches symbol “ @ ”

$ # matches symbol “ $ ”

! # matches symbol “ ! “

% # matches symbol “ % ”

\* # matches symbol “ \* ”

# # matches symbol “ # ”

- # matches symbol “ - ”

\_ #matches symbol “ \_ ”

? # matches symbol “ ? ”

& # matches symbol “ & ”

]

)

[

A-Z # matches any uppercase letters

a-z # matches any lowercase letters

0-9 # matches any digits

@

$

!

%

\*

#

-

\_

?

&

]

{8,30} # Quantifier – states that length of the string should be between 8 and 30

$ # End of string

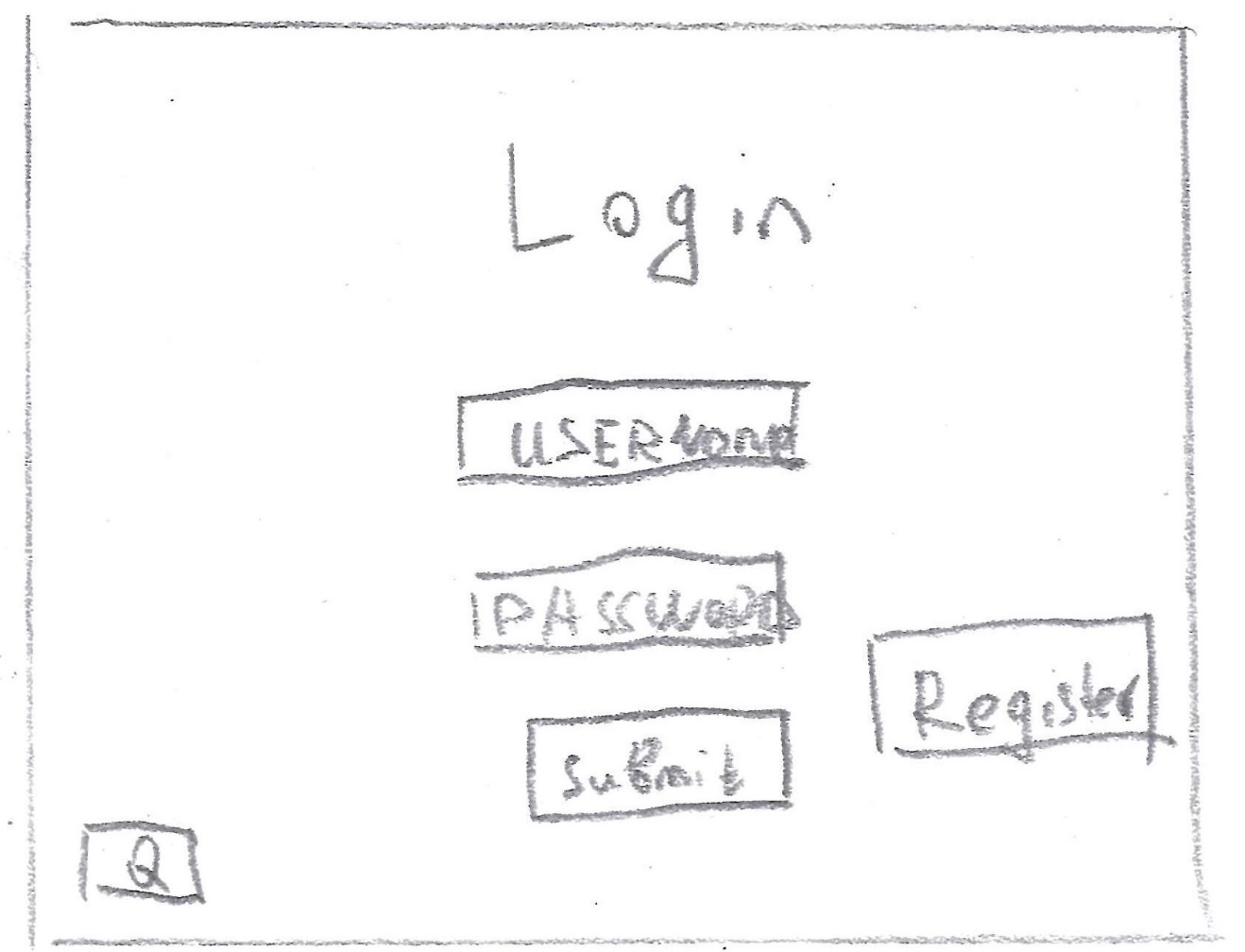
## Interface

By designing interface beforehand, I can structure my approach. Those drawings are my first design of interface and a lot will change in a future. They are my first generation and will let me see the pros and cons of each design.

### Design process

#### Login

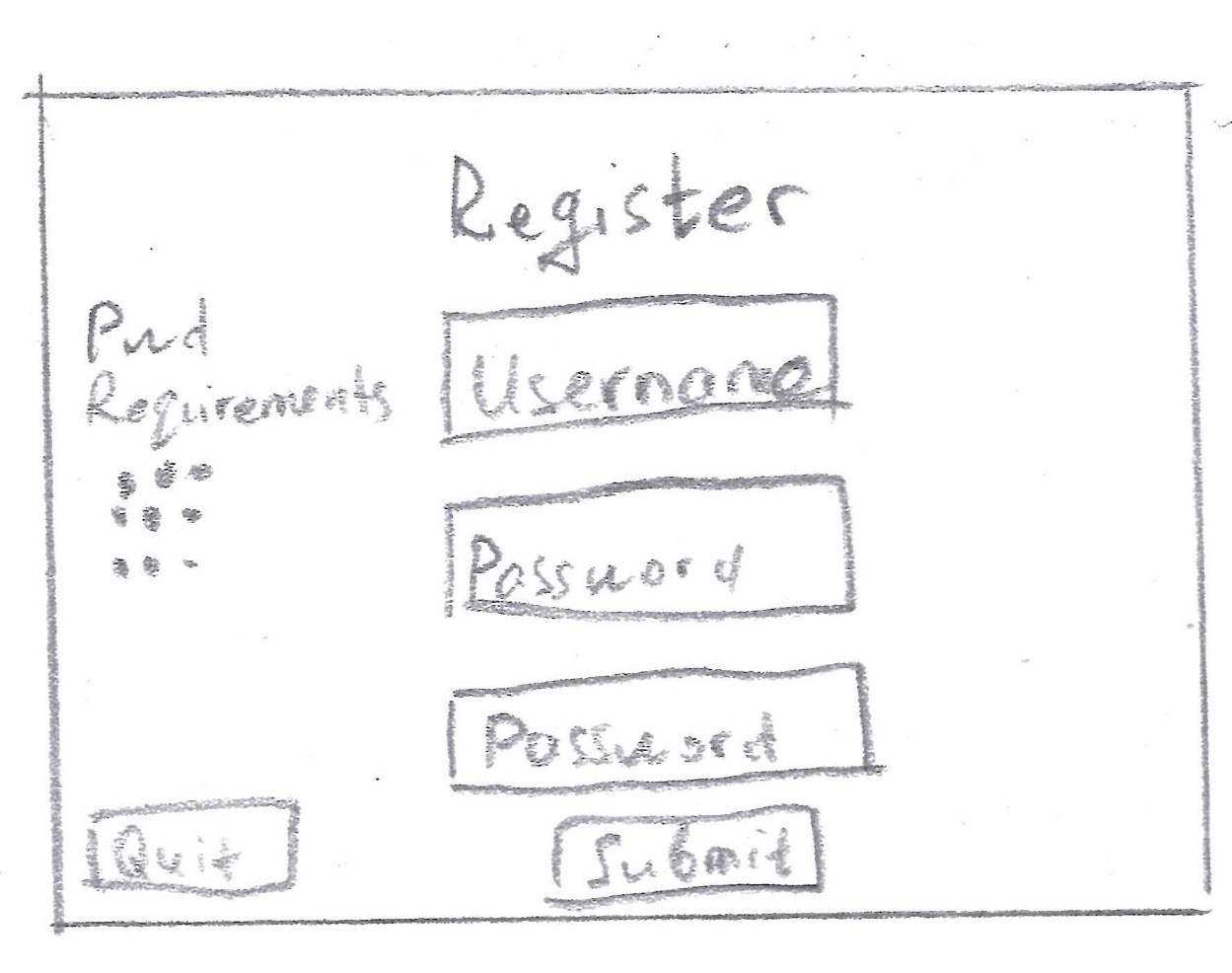
Design for login should include several input boxes to let user sign in and move to the next stage.



#### Register

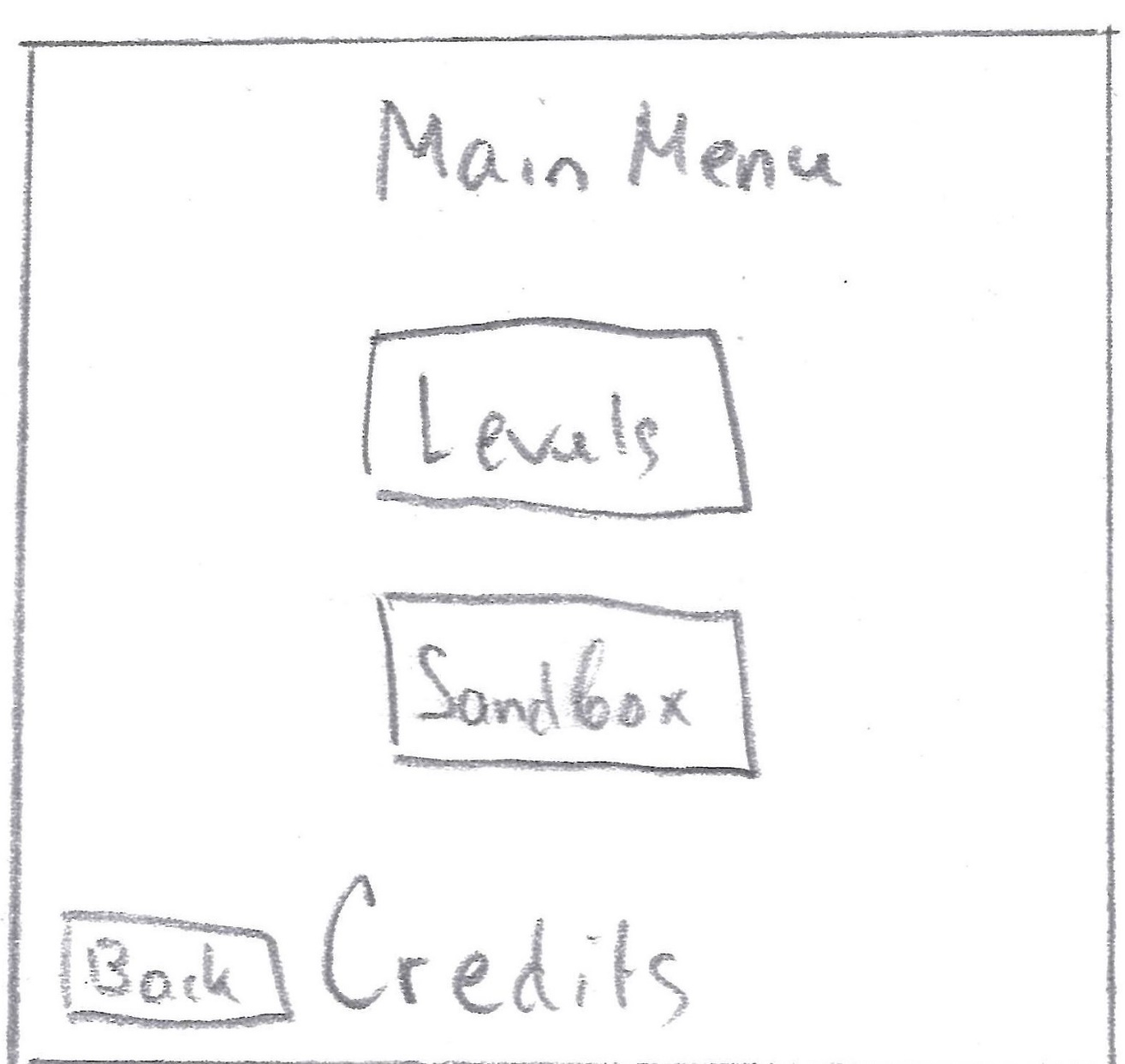
Password Requirements should be:

* At least 1 Uppercase character
* At least 1 lowercase character
* At least 1 digit
* At least 1 special character



#### Main Manu

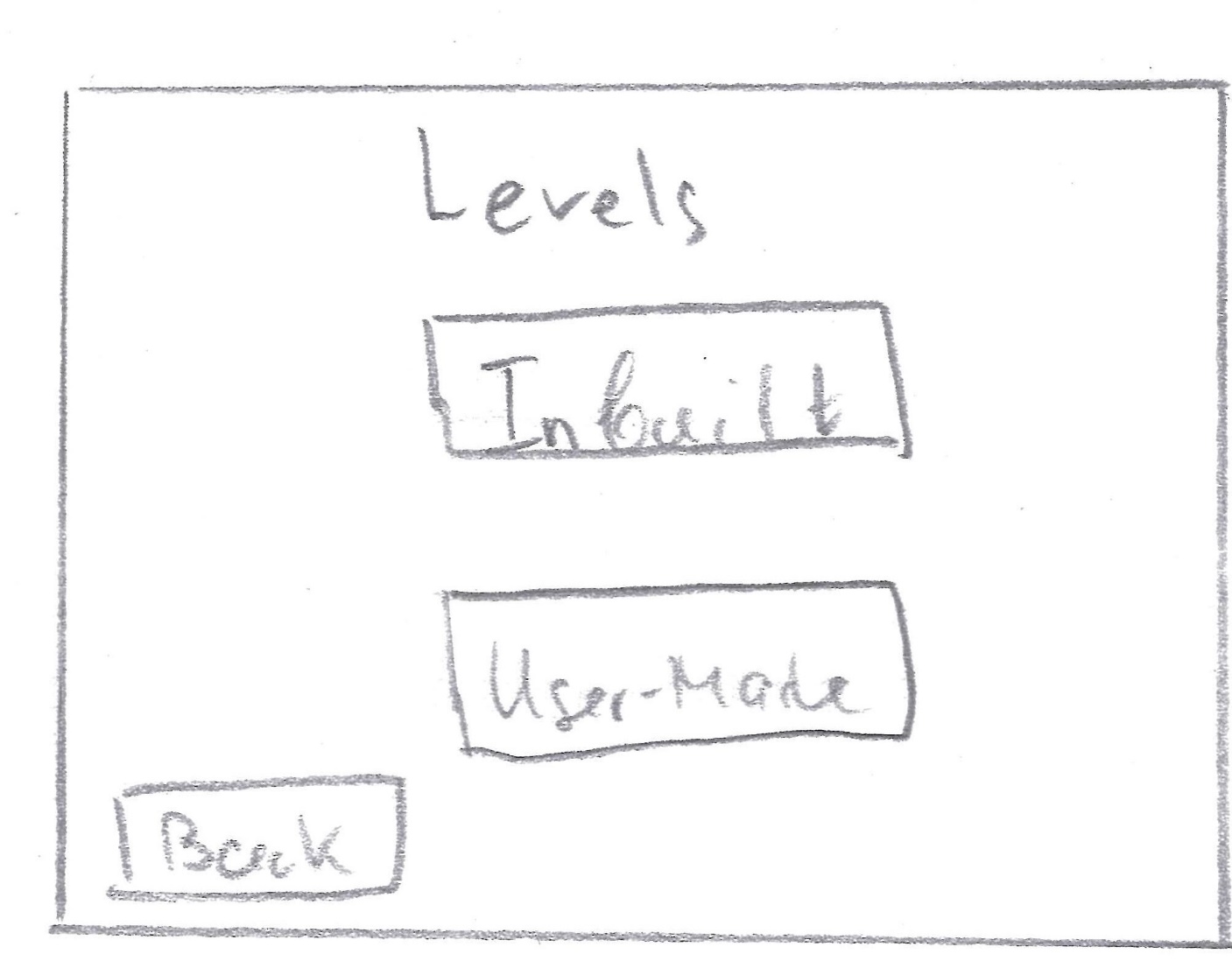
Main menu should include an option of choosing between Levels and Sandbox. In addition, I wanted to include Credits, which is not compulsory, but could be useful to show functionality of text display functionality.



#### Levels

Levels should include an option to choose between inbuilt levels and user-made levels. It is important for user-made mazes to exist to populate database and show all functionality of my program.

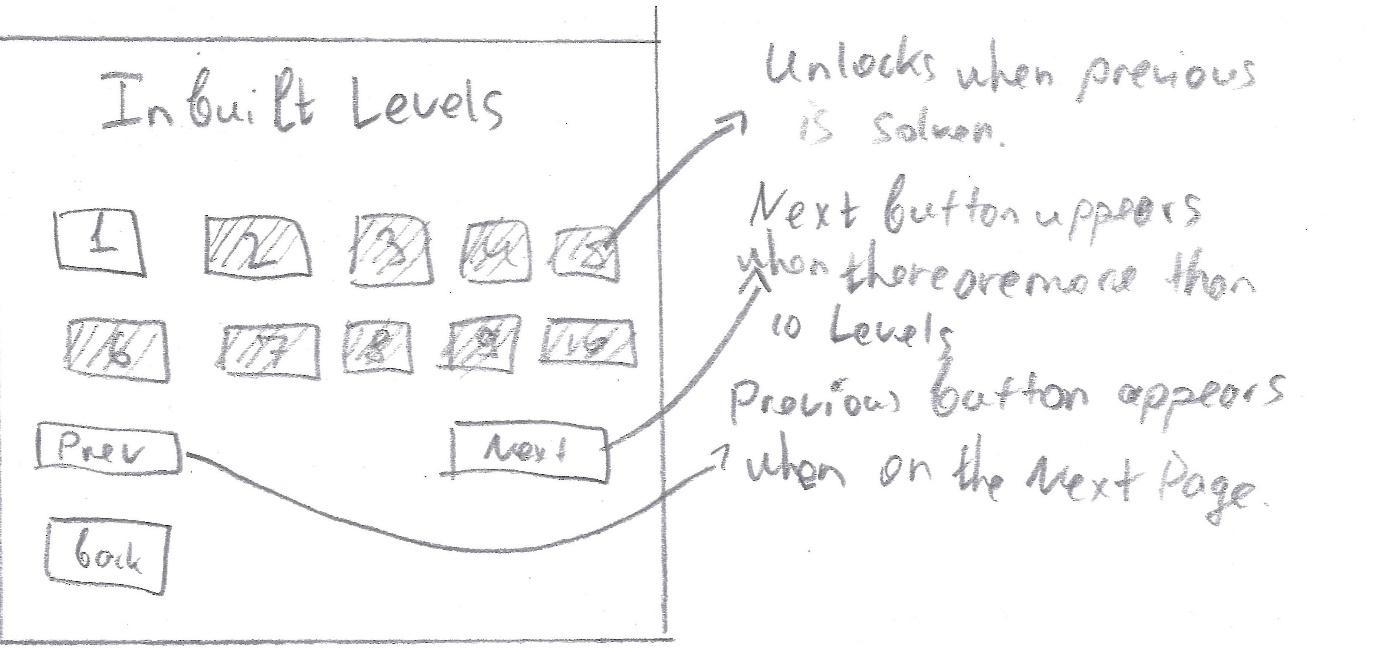
Both of them should be displaying levels in form of buttons to make interface easier



#### Inbuilt

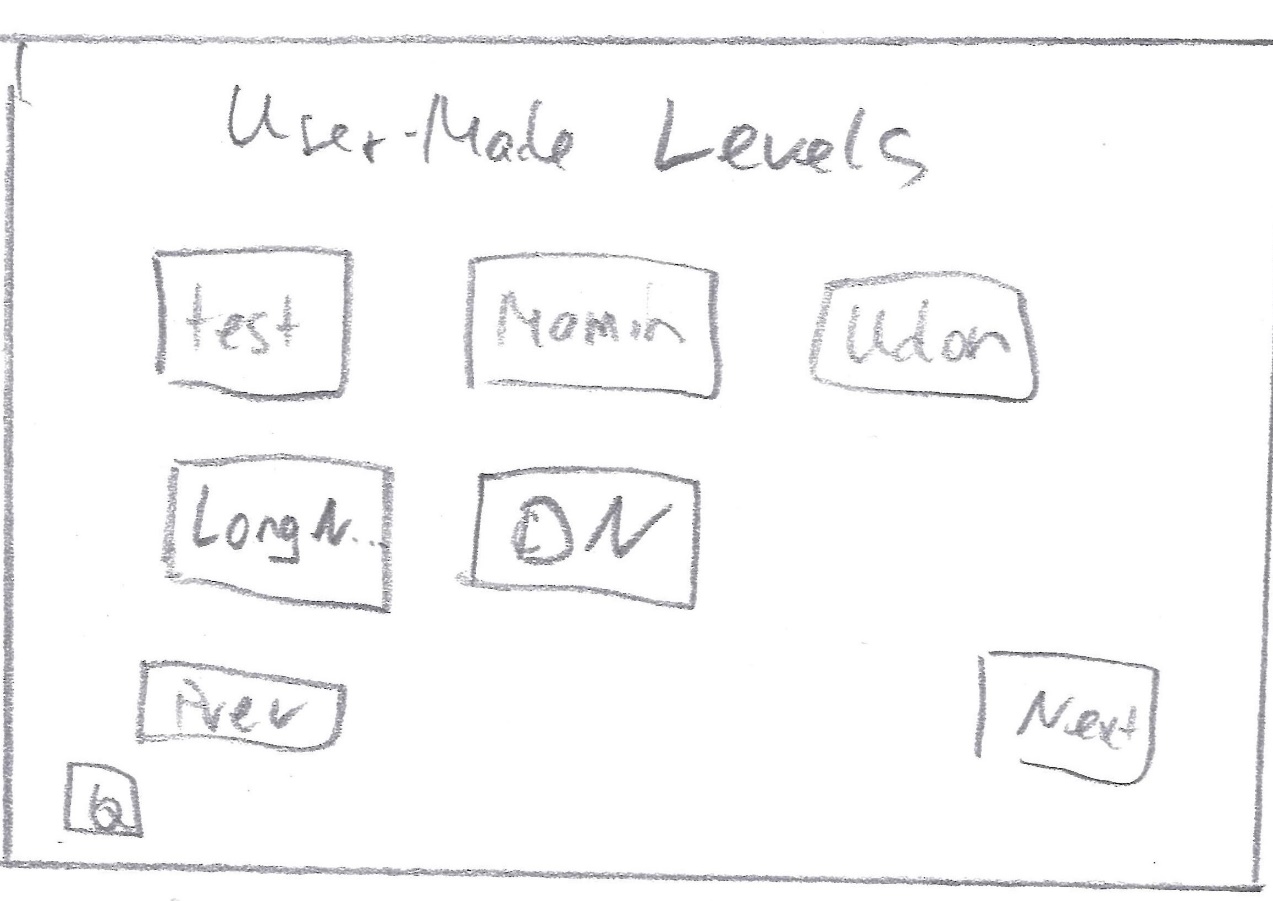
Inbuilt levels should display pre-built levels which would be already in a database table Maze. Without pre-built mazes, program would only consist of user-made mazes and wouldn’t demonstrate its full functionality, which is important for me and my client.

First level should be unlocked by a default as the user who just registered should have an access to in-built levels. Each next level will be unlocked as soon as the user solves previous maze.



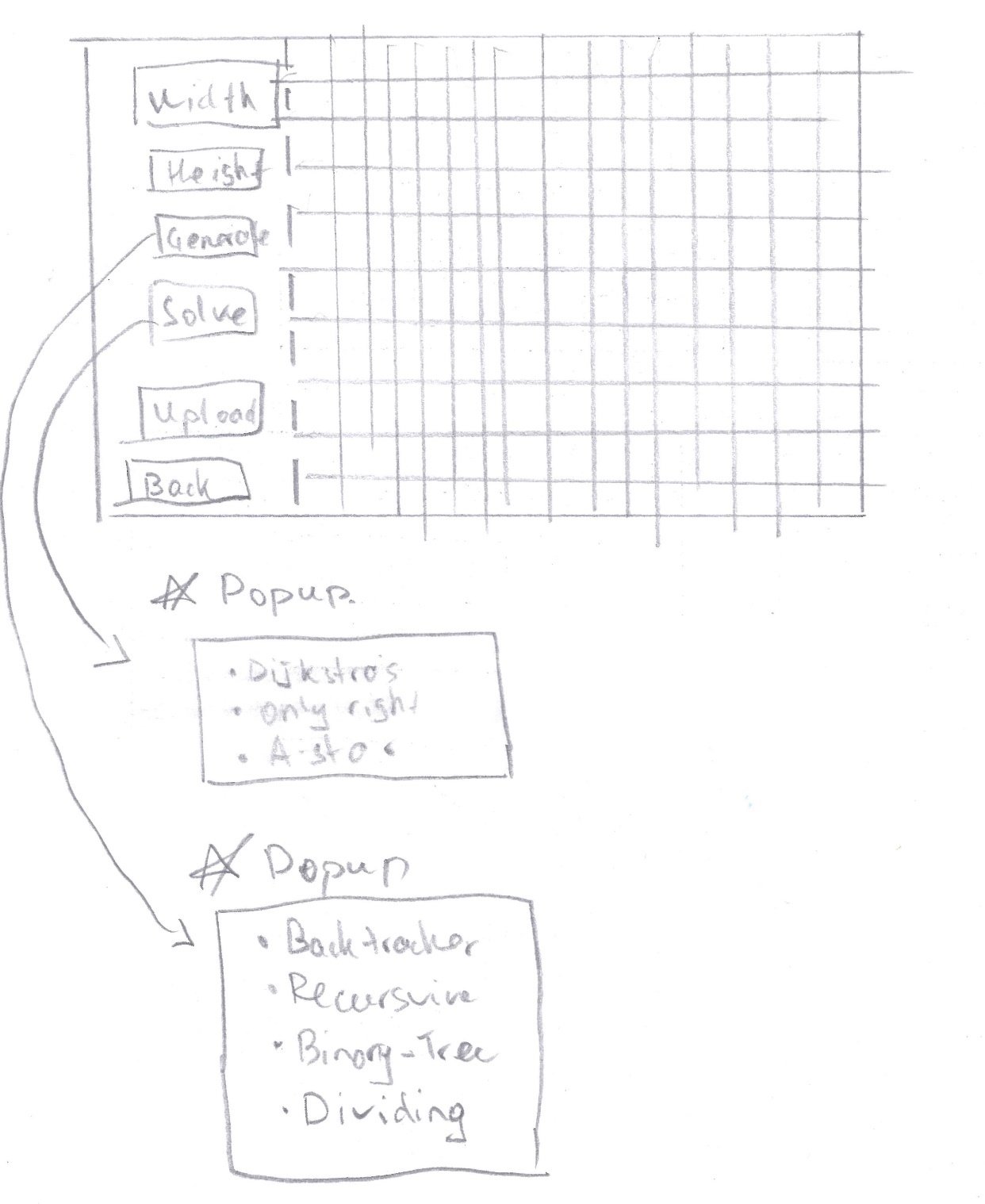
#### Usermade

User-made levels should be populated by levels generated by users only. All components will be having exactly the same functionality as both will inherit from the same parent class having same functions.



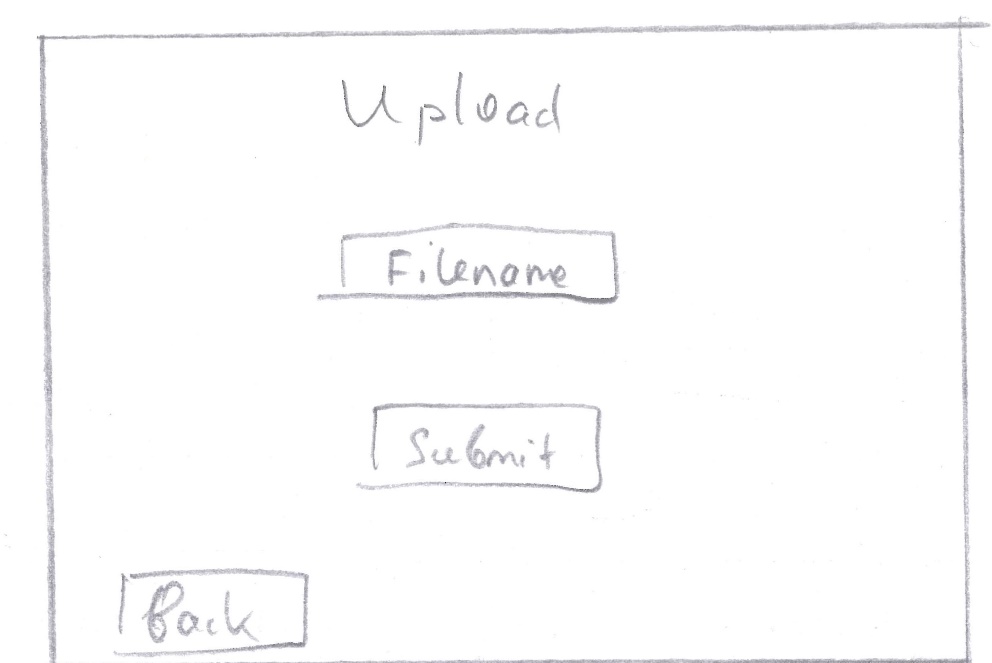
#### Sandbox

Sandbox will be having inputs of width and height to create and grid. To generate grid, user will be expected to select an algorithm to generate grid for him from the pop-up. Same will be implemented for an upload, as in pop-up will be displayed showing options to solve a maze.



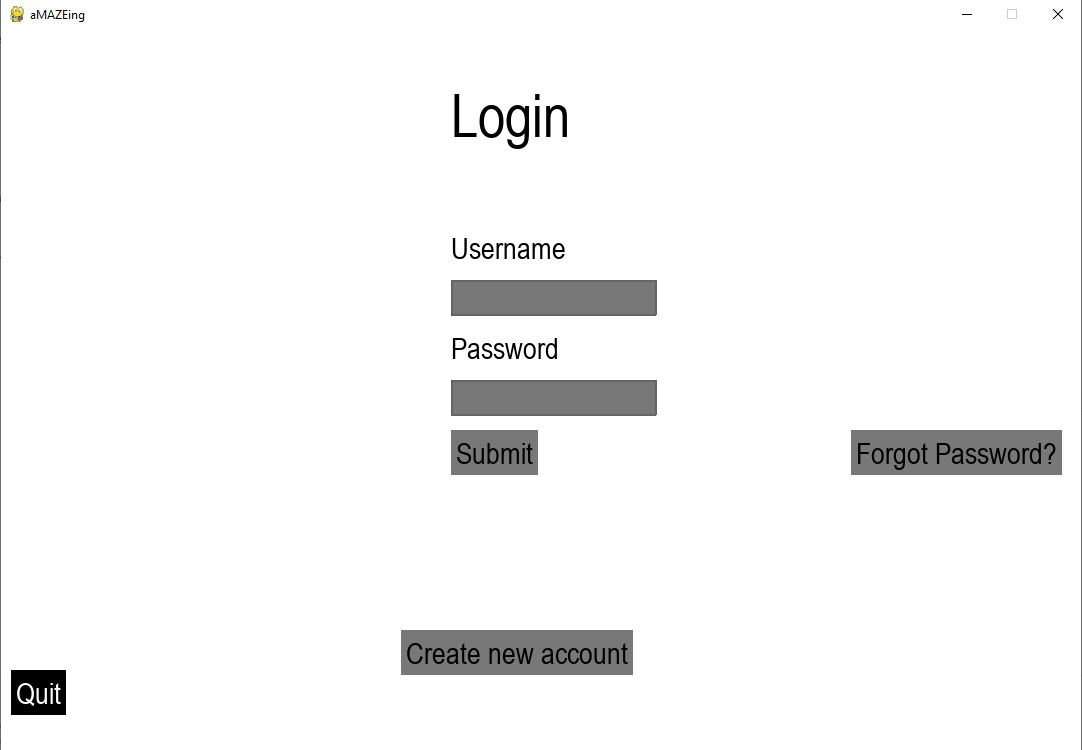
#### Upload

Upload should as user to input the filename the maze should be saved under and a submit button to populate the database. Filename input field should validate the name and don’t allow to input characters any special characters.



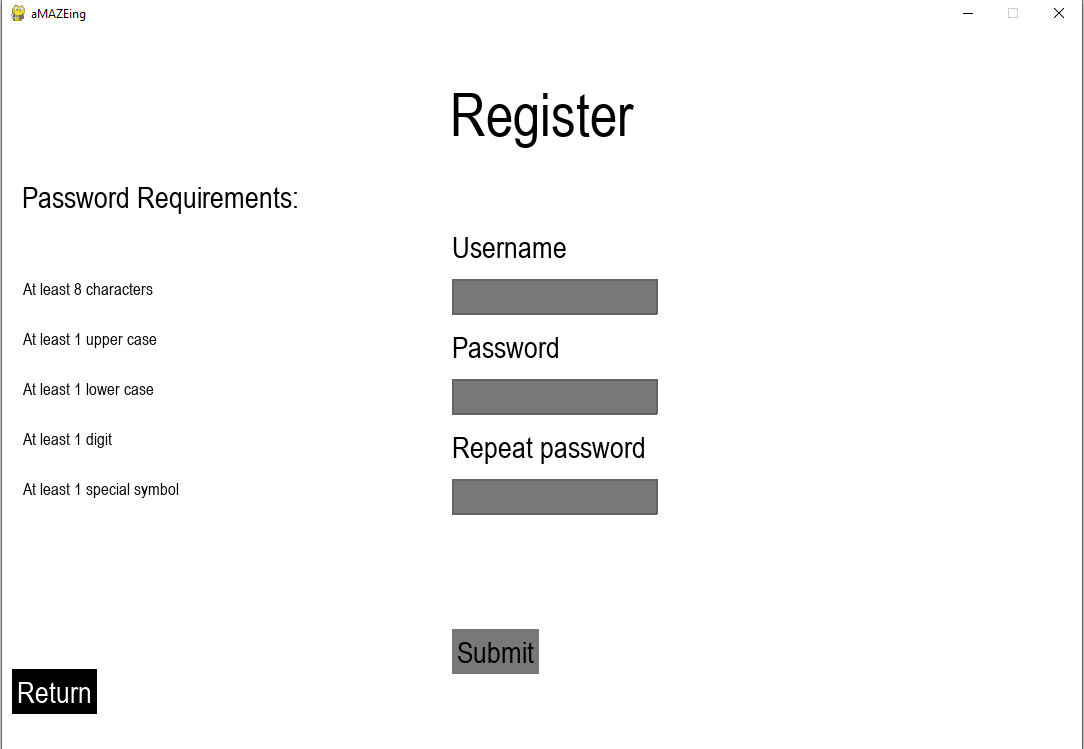
### Actual design

#### Login

Login stayed the same way as in intended, but due pygame logic and complex implementation of text inside input field, I have decided that it would be better to display the text right above the input box to increase readability and make the program easier for users to read and understand

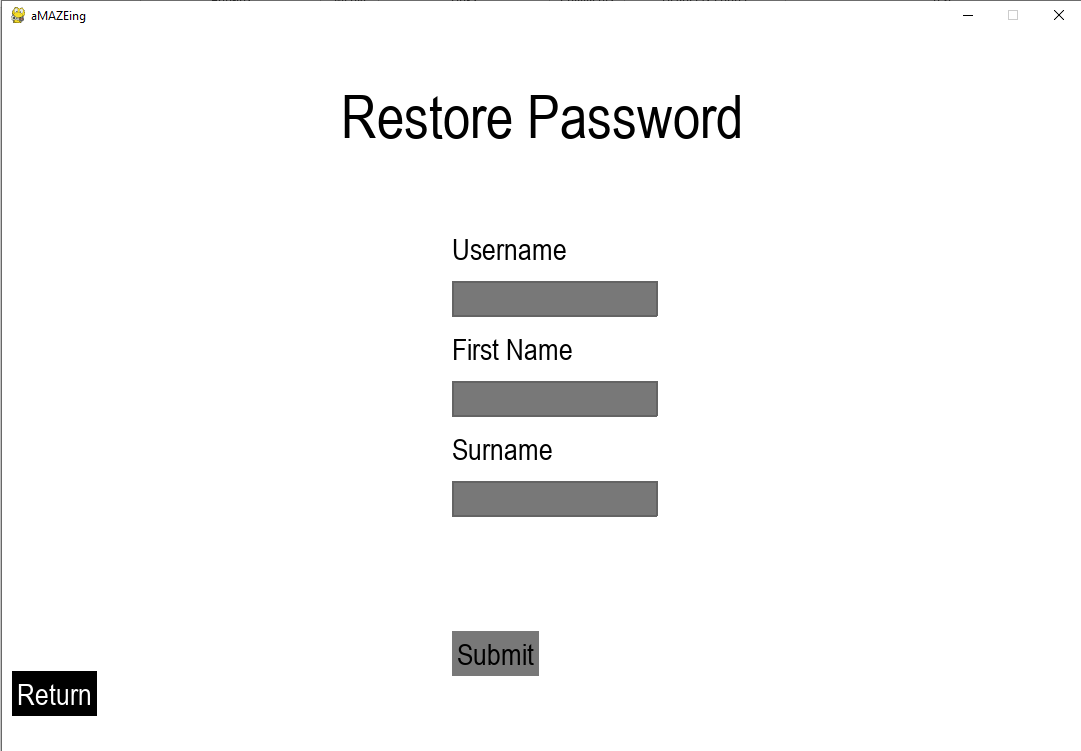
#### Register

Register’s password requirements stayed the same as intended. Same problem as in login occurred and same decision made.

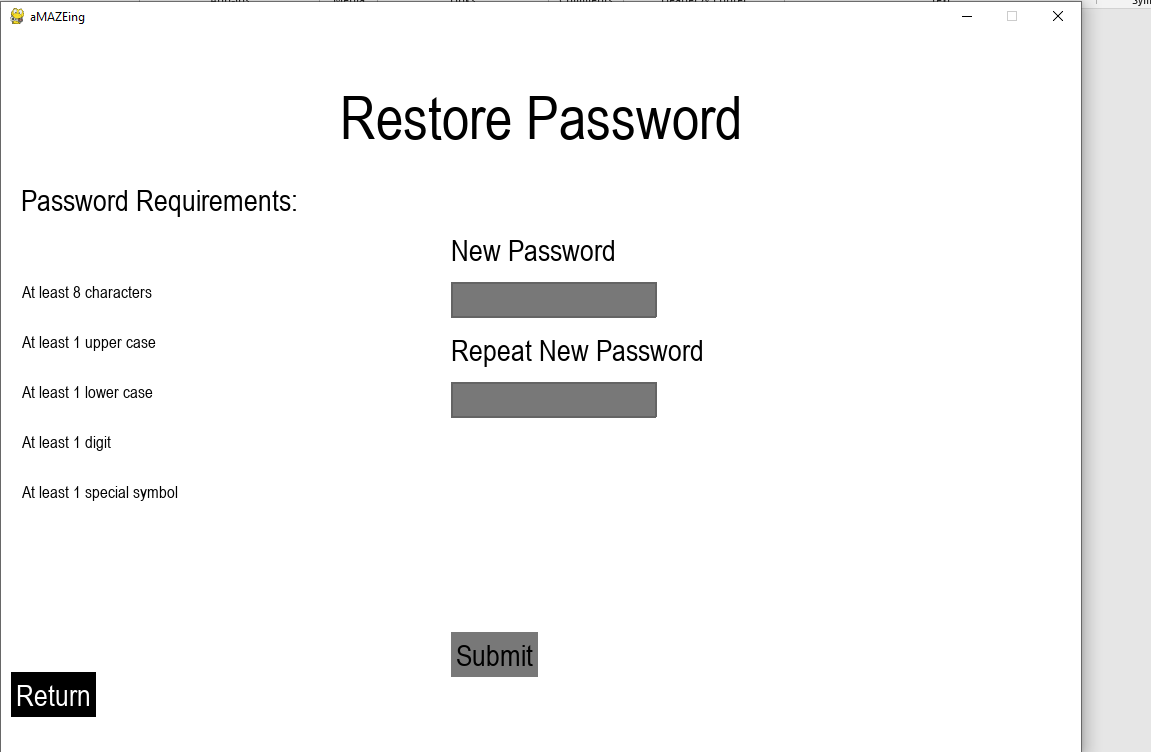


#### Forgot Password

Restore password changed a bit to increase security of users. It would be easier to get access to an account just by knowing their username and instead, I made double verification by username and their first and last name.

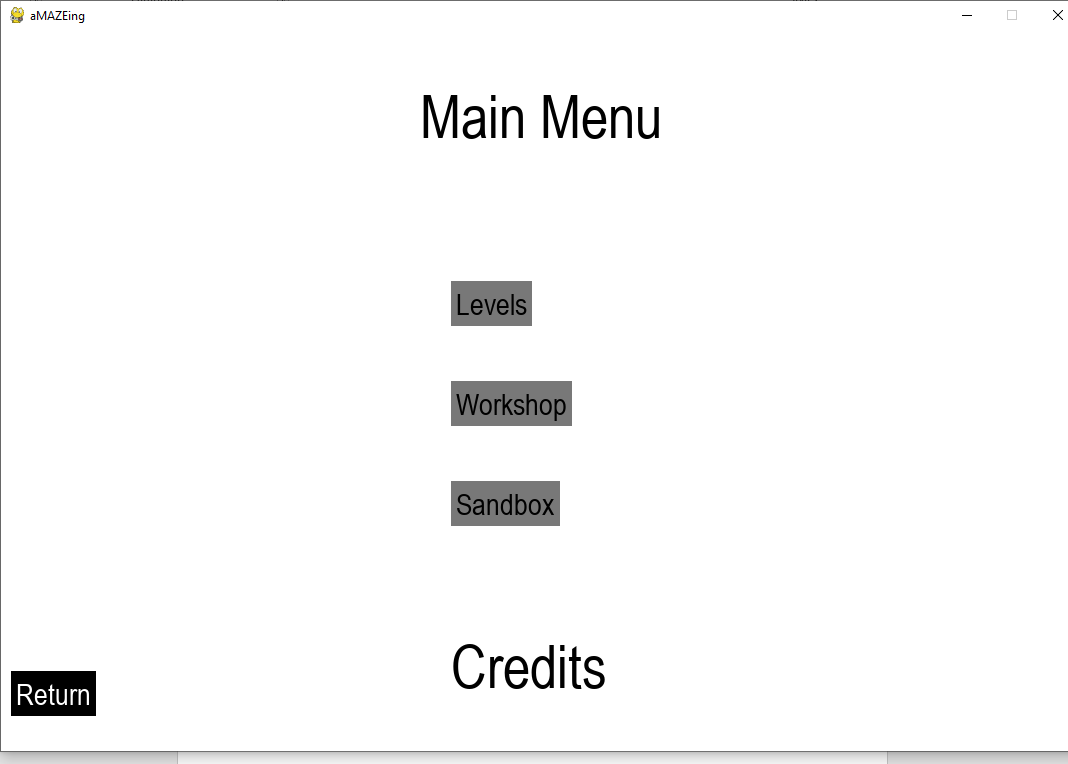


After all parameters matching, next screen would appear asking for double password input to verify both of them validate and match the requirements and only after that procedure password would be reset.



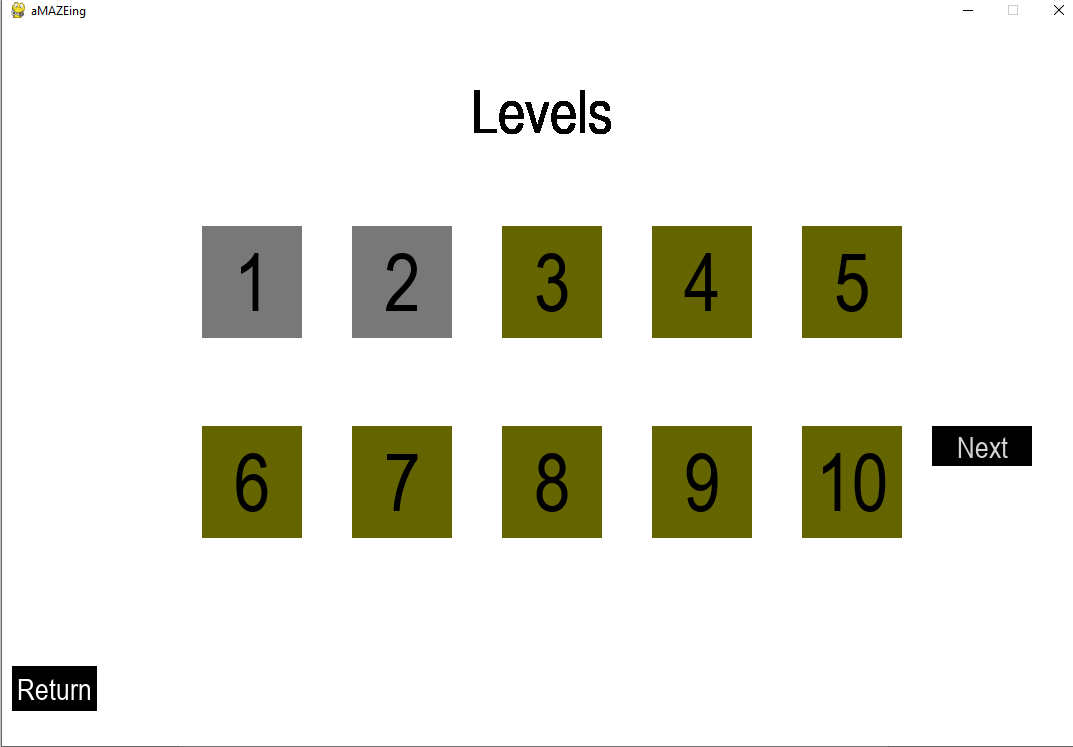
#### Main Menu

I have decided to get rid of “Levels” option which then would just make another window asking user to choose between In-built and User-made levels. It would be faster for users to reach what they wanted by just choosing straight from the menu.



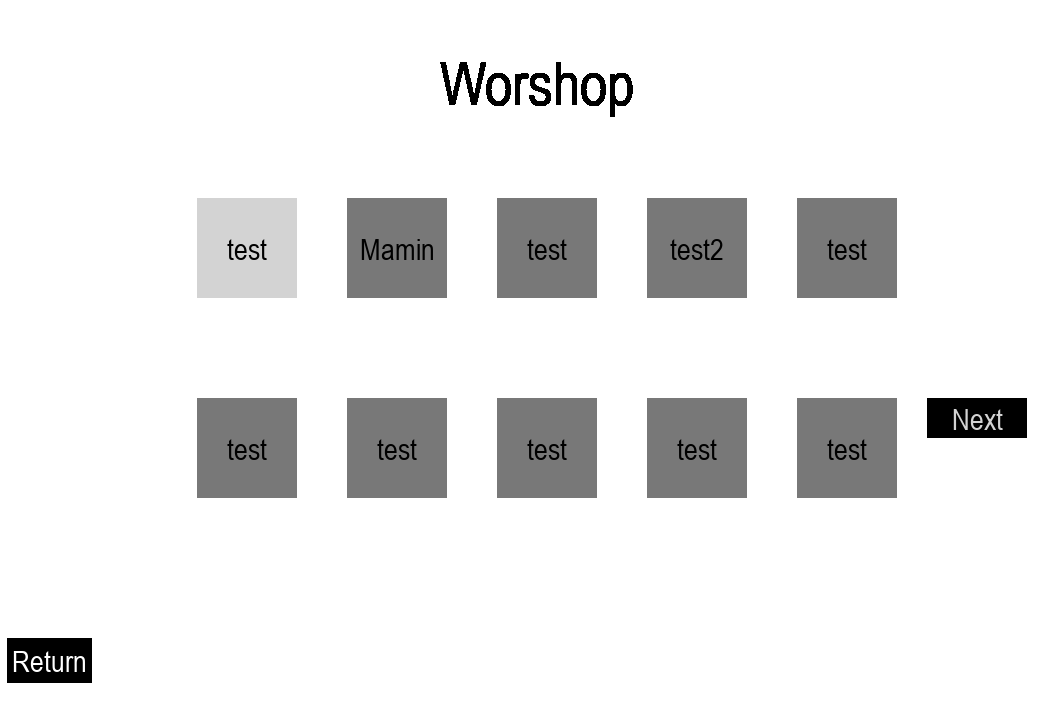
#### Levels

Levels stayed the same as intended. Green buttons are unclickable and button colour doesn’t change when hovered, whereas Grey buttons show unlocked levels, which are clickable and change colour to LightGrey when the cursor is over the button.



#### Workshop

Nothing changed from plan. All buttons are grey to show that they all are unlocked. Users can choose any levels from any page.



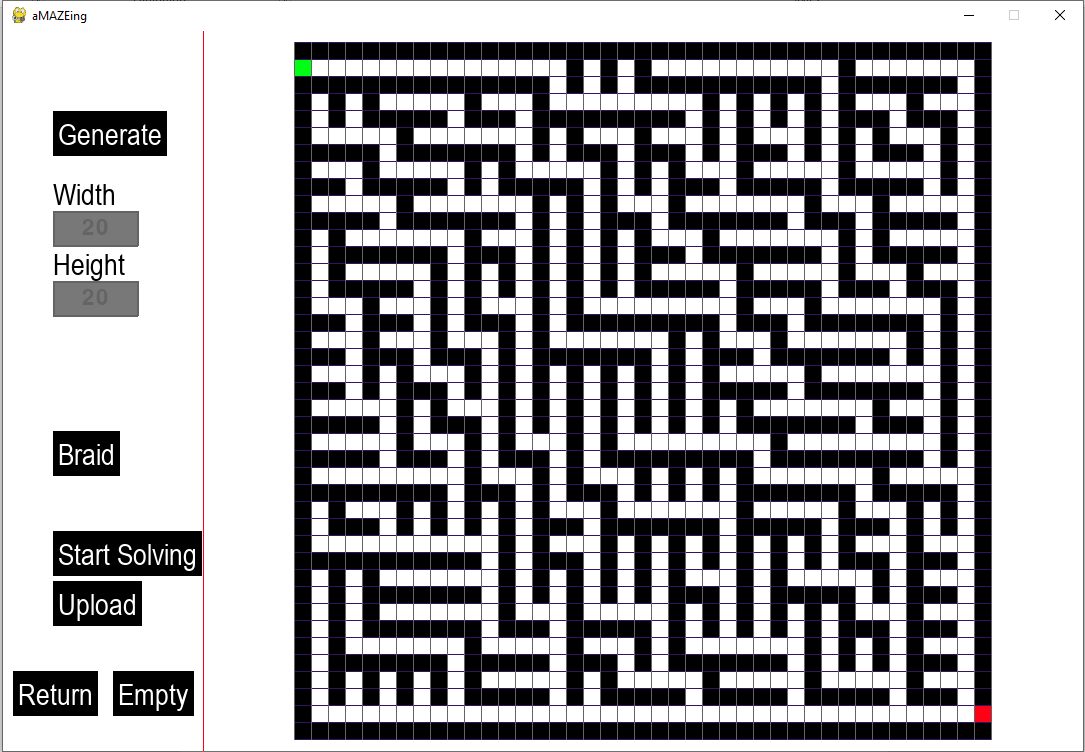
#### Sandbox

Sandbox will now only display grid when user pressed generate and inputs width with height in the limits.

Empty button was created as well to allow user to create empty grid and move nodes by dragging the cursor. It allows user to create their own maze in any shape or form without using any of the algorithms.

Pop-ups were removed due to their complexity and pygame display function that needed to refresh the whole screen and would cause rapid updates which are very bad for user to experience.





# Technical Solution

## Game.py

#imports to set up environment

try:

    import os

    os.environ['PYGAME\_HIDE\_SUPPORT\_PROMPT'] = "hide" #hides hello message from pygame

    import pygame

    pygame.font.init()

except:

    print('No pygame installed')

#import all files of the game and test for all files presence

try:

    import CreateDB

    import MakeGrid

    import Generate

    import Utils

    from Utils import Button

except:

    print('not all files downloaded')

from random import choice

import sqlite3 as sql

#Fonts

Font = pygame.font.SysFont('arial', 60)

SubFont = pygame.font.SysFont('arial', 30)

SmallFont = pygame.font.SysFont('arial', 17)

#Colours

White = (255,255,255)

Black = (0,0,0)

Grey = (120, 120, 120)

LightGrey = (211, 211, 211)

Red = (255,0,20)

Green = (0,255,20)

Blue = (20,0,255)

Cyan = (112, 150, 225)

Yellow = (230, 225, 105)

LightBlue = (18, 231, 255)

Purple = (140, 10, 200)

#Main class to display window

class Window:

    FPS = 60

    def \_\_init\_\_(self, width = 1080, height = 720, caption = 'aMAZEing'):

        self.WIN\_WIDTH = width

        self.WIN\_HEIGHT = height

        self.WIN = pygame.display.set\_mode((self.WIN\_WIDTH, self.WIN\_HEIGHT))

        pygame.display.set\_caption(caption)

    #refreshes window

    def display(self):

        pygame.display.update()

        pygame.display.flip()

#initial class for login page

class LoginPage(Window):

    start\_x = 450

    #main loop that sets values and has main loop

    def on(self):

        self.LoginText = Font.render('Login', True, Black)

        self.UsernameText = SubFont.render('Username', True, Black)

        self.PasswordText = SubFont.render('Password', True, Black)

        #inputs for the login

        self.UsernameInput = Utils.InputBox(self.start\_x, 250, 200, 30, validate = lambda x: Utils.check\_username(x))

        self.PasswordInput = Utils.InputBox(self.start\_x, 350, 200, 30, password=True, validate = lambda x: Utils.check\_password(x))

        self.input\_boxes = [self.UsernameInput, self.PasswordInput]

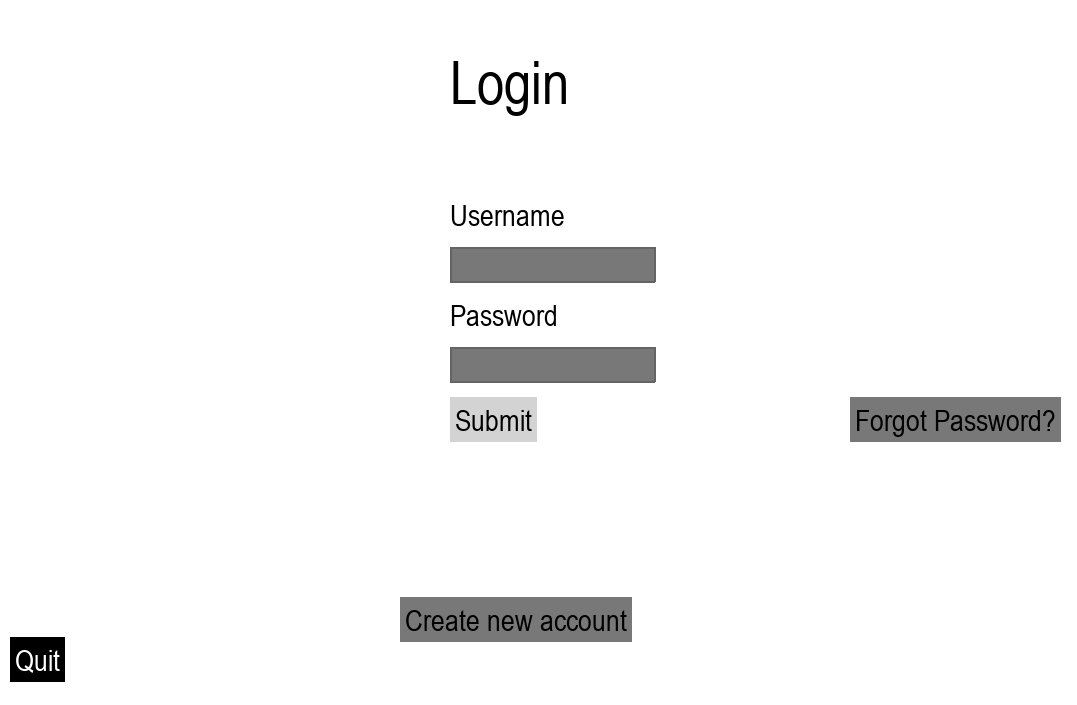
        #buttons for the menu

        quitbtn = Button(10, self.WIN\_HEIGHT-80, 'Quit', SubFont, Black, Grey, White, command= 'return', margin = 10)

        submitbtn = Button(450, 400, 'Submit', SubFont, Grey, LightGrey, Black, command = lambda: self.submit\_credentials\_wrapper(), margin=10)

        forgotpwd = Button(850, 400, 'Forgot Password?', SubFont, Grey, LightGrey, Black, command = lambda: self.forgot\_password\_wrapper(), margin=10)

        createacc = Button(400, 600, 'Create new account', SubFont, Grey, LightGrey, Black, command = lambda: self.create\_account(), margin=10)



submitbtn

createacc

forgotpwd

quitbtn

PasswordInput

UsernameInput

LoginText

PasswordText

UsernameText

self.buttons = [quitbtn, submitbtn, forgotpwd, createacc]

        self.draw\_all()

        #main loop

        run = True

        while run:

            self.display()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    run = False

                    return False

                pos = pygame.mouse.get\_pos()

                self.draw\_all()

                self.draw\_inputs()



                #return can be different depending on what record outputs

                try:

                    if resume == 'wrong pwd':

                        self.blit\_wrong\_pwd()

                    if resume == 'no record':

                        self.blit\_no\_record()

                except:

 pass

                #updates input boxes

                for box in self.input\_boxes:

                    box.handle\_event(event)

                #updates buttons

                for btn in self.buttons:

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        #registers mouse left click

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    #completes the command and must have a return

                                    resume = btn.command()

                                    #if returned parameter is not False

                                    if resume != False:

                                        #if resume is none, no update is done

                                        if resume == None:

                                            continue

                                        #if resume is True, it causes update of the screen

                                        if resume == True:

                                            self.draw\_buttons()

                                            self.draw\_all()

                                    else:

                                        #if resume is false, program stops running

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

        return False

    #procedure to display text on screen

    def draw\_writings(self):

        self.WIN.blit(self.LoginText, (self.start\_x, 50))

        self.WIN.blit(self.UsernameText, (self.start\_x, 200))

        self.WIN.blit(self.PasswordText, (self.start\_x, 300))

    #procedure to draw all buttons

    def draw\_buttons(self):

        for btn in self.buttons:

            btn.draw(self.WIN, btn.button\_colour)

    #procedure to draw all inputs

    def draw\_inputs(self):

        for box in self.input\_boxes:

            box.update()

            box.draw(self.WIN)

    #procedure that combines drawing text and buttons

    def draw\_all(self):

        self.WIN.fill(White)

        self.draw\_writings()

        self.draw\_buttons()

    #wrapper for function that allows return to be possible

    def create\_account(self):

        return Register().on()

    #wrapper for function for people who forgot password that allows return to be possible

    def forgot\_password\_wrapper(self):

        return ForgotPassword().on()

    #wrapper for funciton to submit credentials

    def submit\_credentials\_wrapper(self):

        return self.submit\_credentials()

    #function that sunbmits credentials

    def submit\_credentials(self):

        username = self.UsernameInput.text

        password = self.PasswordInput.text

        conn = sql.connect('DataBase.db')

        #fetches username from table User

        c = conn.cursor()

        c.execute(f"""

SQL statement to fetch Username and Passoword from table User. If record exists, it will go on and check password matching, otherwise display “User Does’t exist”

            SELECT Username, Password

            FROM User

            WHERE Username = '{username}'

        """)

        row = c.fetchone()

if row:

            name = row[0]

            pwd = row[1]

            #compares password from table to hashed one using hashlib

            if Utils.PasswordCheck().check(password, pwd):

                #makes USERNAME global to allow to use it anywhere in the code

                global USERNAME

                USERNAME = self.UsernameInput.text

                #resets input boxes to increase security of login, as text could be seen after returning to previous page

                self.UsernameInput.text = ''

                self.UsernameInput.display\_text = ''

                self.PasswordInput.text = ''

                self.PasswordInput.display\_text = ''

                resume = MainMenu().on()

If password inputted by user doesn’t match with the record’s one, it will display “Wrong Password”, otherwise call on function from class MainMenu

                return resume

            else:

                #if passwords don't match, it makes input boxes empty

                self.PasswordInput.text = ''

                self.PasswordInput.display\_text = ''

                return 'wrong pwd'

        else:

            return 'no record'

    #displays text if record is not found

    def blit\_no\_record(self):

        text = SubFont.render("User does't exist ", True, Red)

        self.WIN.blit(text, (700, 300))

    #displays text if password is wrong

    def blit\_wrong\_pwd(self):

        text = SubFont.render('Wrong password', True, Red)

        self.WIN.blit(text, (700, 300))

#class that deals with users who forgot their password

class ForgotPassword(Window):

    start\_x = 450

    #main loop that declares variabels and contains main loop

    def on(self):

        #sets up texts

        self.RestoreText = Font.render('Restore Password', True, Black)

        self.UsernameText = SubFont.render('Username', True, Black)

        self.NameText = SubFont.render('First Name', True, Black)

        self.SurnameText = SubFont.render('Surname', True, Black)

        self.username\_should\_be = SubFont.render('Password Requirements:', True, Black)

        self.numofchar = SmallFont.render('At least 8 characters', True, Black)

        self.oneupper = SmallFont.render('At least 1 upper case', True, Black)

        self.onelower = SmallFont.render('At least 1 lower case', True, Black)

        self.onedigit = SmallFont.render('At least 1 digit', True, Black)

        self.onespecial = SmallFont.render('At least 1 special symbol', True, Black)

        #sets up input boxes

        self.UsernameInput = Utils.InputBox(self.start\_x, 250, 200, 30, validate = lambda x: Utils.check\_username(x))

        self.NameInput = Utils.InputBox(self.start\_x, 350, 200, 30)

        self.SurnameInput = Utils.InputBox(self.start\_x, 450, 200, 30)

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, Black, Grey, White, command= 'return', margin = 10)

        submitbtn = Button(450, 600, 'Submit', SubFont, Grey, LightGrey, Black, command = lambda: self.check\_input\_wrapper(), margin=10)

        #places buttons in an array for iteration, which allows to get each button's function individually

        self.buttons = [returnbtn, submitbtn]

        self.input\_boxes = [self.UsernameInput, self.NameInput, self.SurnameInput]

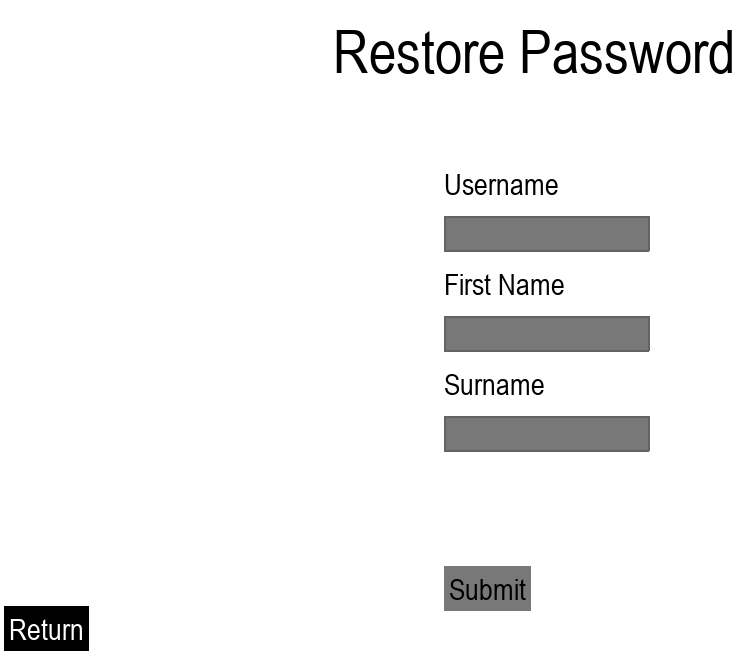
        #uses 2-step verification as needed for security reasons

        self.Password1Input = Utils.InputBox(self.start\_x, 250, 200, 30, validate = lambda x: Utils.check\_password(x), password=True)

        self.Password2Input = Utils.InputBox(self.start\_x, 350, 200, 30, validate = lambda x: Utils.check\_password(x), password=True)

        self.PasswordText = SubFont.render('New Password', True, Black)

        self.Password2Text = SubFont.render('Repeat New Password', True, Black)



submitbtn

SurnameText

FirstnameText

UsernameText

NameInput

SurnameInput

UsernameInput

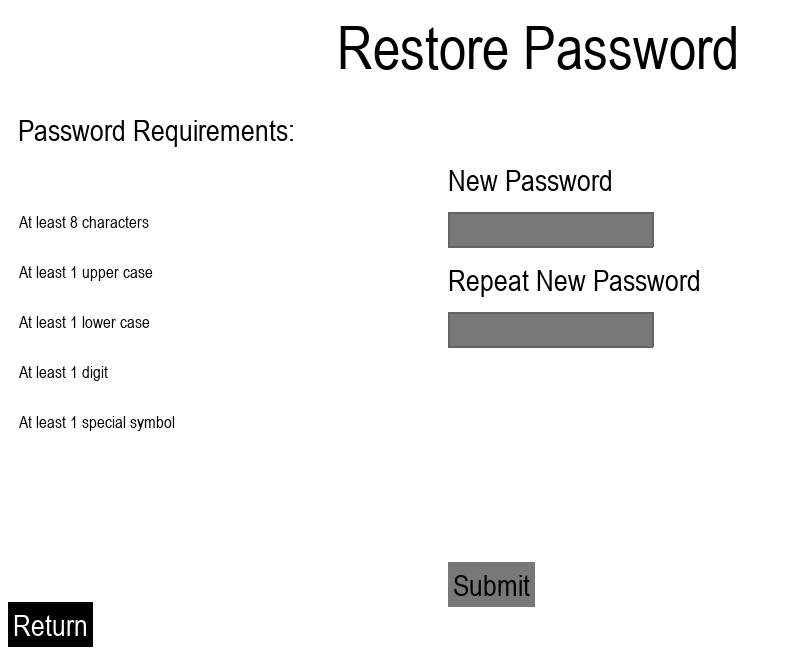
returnbtn

RestoreText

        submitbtn2 = Button(450, 600, 'Submit', SubFont, Grey, LightGrey, Black, command = lambda: self.check\_input2(), margin=10)

        self.buttons2 = [returnbtn, submitbtn2]

        self.input\_boxes2 = [self.Password1Input, self.Password2Input]



Submitbtn2

returnbtn

Password1Text

PasswordText

Password2Input

Password1Input

One\_upper

One\_lower

One\_digit

One\_special

Num\_of\_char

Username\_should\_be

RestoreText

        #first main loop that asks for username to check with table if exists

        self.draw\_all()

        run = True

        while run:

            self.display()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    run = False

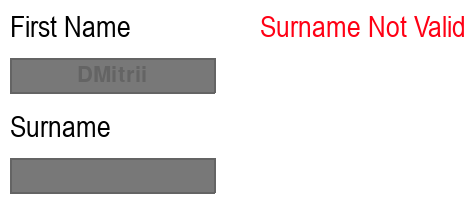
                    return False



                pos = pygame.mouse.get\_pos()

                self.draw\_all()

                self.draw\_inputs()

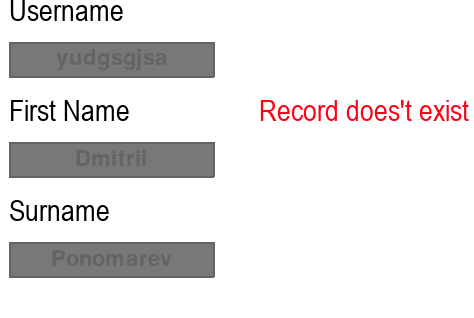
                #displays text depending on returned value from subroutine

                try:

                    if resume == 'username wrong':

                        self.blit\_username\_not\_valid()

                    if resume == 'name not valid':

                        self.blit\_name\_not\_valid()

                    if resume == 'surname not valid':

                        self.bit\_surname\_not\_valid()

                    if resume == 'not exist':

                        self.blit\_not\_exist()

                except:

                    pass

                #updates boxes

                for box in self.input\_boxes:

                    box.handle\_event(event)

                #draws buttons

                for btn in self.buttons:

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    resume = btn.command()

                                    if resume != False:

                                        if resume == None:

                                            continue

                                        if resume == True:

                                            self.draw\_buttons()

                                            self.draw\_all()

                                        if resume == 'menu':

                                            #goes to menu instead of refreshing and staying on this page

                                            return True

                                    else:

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

        return False

def draw\_all(self):

        self.WIN.fill(White)

        self.draw\_buttons()

        self.draw\_writings()

    #procedure to draw buttons

    def draw\_buttons(self):

        for btn in self.buttons:

            btn.draw(self.WIN, btn.button\_colour)

    #procedure to draw text

    def draw\_writings(self):

        self.WIN.blit(self.RestoreText, (self.WIN\_WIDTH//2- self.RestoreText.get\_width()//2, 50))

        self.WIN.blit(self.UsernameText, (self.start\_x, 200))

        self.WIN.blit(self.NameText, (self.start\_x, 300))

        self.WIN.blit(self.SurnameText, (self.start\_x, 400))

    #procedure to draw and update input boxes

    def draw\_inputs(self):

        for box in self.input\_boxes:

            box.update()

            box.draw(self.WIN)

    #wrapper for checking input funnction that will open another window

    def check\_input\_wrapper(self):

        return self.check\_input1()

    #function that validates username, first name and surname and if all correct, allows to reset the password

    def check\_input1(self):

        username = self.UsernameInput.text

        name = self.NameInput.text

        surname = self.SurnameInput.text

        #uses method validate\_ to return Boolean of correctly inputted characters

        username\_valid = Utils.validate\_username(username)

        name\_valid = Utils.validate\_name(name)

        surname\_valid = Utils.validate\_name(surname)

        if not username\_valid:

            return 'username wrong'

        if not name\_valid:

            return 'name not valid'

        if not surname\_valid:

            return 'surname not valid'

       #if either of validations are passed, program continues to run, otherwise text is displayed with an error stating

        #connects to database and checks against 3 parameters

SQL fetches Username, FirstName and Surname by accessing table User. It’s given that username is unique, therefore maximum of 1 record as output. If all parameters match, program proceeds. Othersise returns “Record does’t exist”

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute(f"""

            SELECT Username, FirstName, Surname

            FROM User

            WHERE Username = '{username}' AND FirstName = '{name}' AND Surname = '{surname}'

        """)

        #Its given that username is unique, therefore there can be only 1 entity, therefore fetching only 1 entry

        row = c.fetchone()

        if row:

            #setting USERNAME as global for future fetching and using

            global USERNAME

            USERNAME = row[0]

            return self.restore()

        else:

            return 'not exist'

    #if credentials are correct, allows to input new password

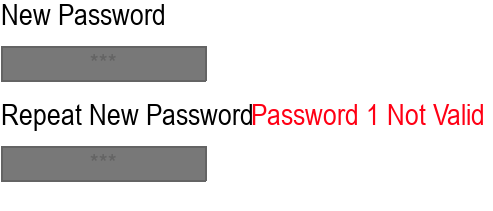
    #function that calls new window and allows to input new password

    def restore(self):

        self.draw\_all2()

        run = True

        while run:



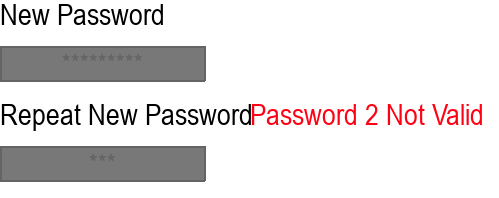
            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

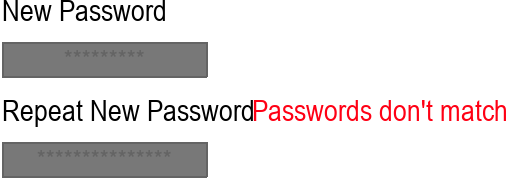
 #allows input of input box to be registered and updated

                for box in self.input\_boxes2:

                    box.handle\_event(event)

                try:

                    if resume == 'password1 valid':

                        self.blit\_first\_not\_valid()

                    if resume == 'password2 valid':

                        self.blit\_second\_not\_valid()

                    if resume == 'no match':

                        self.blit\_pwd\_match()

                except:

                    pass

                self.draw\_inputs2()

#iterated through each button to get it'c command and lets to complete it

                for btn in self.buttons2:

                    if btn.is\_over(pos):

#if cursor is over the button, it should change colour to be more interractive

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    resume = btn.command()

                                    if resume != False:

                                        if resume == None:

                                            continue

                                        if resume == True:

                                            self.draw\_all2()

                                            return True

                                        if resume == 'menu':

                                            return 'menu'

                                    else:

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

        return False

    #procedure to draw text

    def draw\_writings2(self):

        self.WIN.blit(self.RestoreText, (self.WIN\_WIDTH//2- self.RestoreText.get\_width()//2, 50))

        self.WIN.blit(self.PasswordText, (self.start\_x, 200))

        self.WIN.blit(self.Password2Text, (self.start\_x, 300))

        self.WIN.blit(self.username\_should\_be, (20, 150))

        self.WIN.blit(self.numofchar, (20, 250))

        self.WIN.blit(self.oneupper, (20, 300))

        self.WIN.blit(self.onelower, (20, 350))

        self.WIN.blit(self.onedigit, (20, 400))

        self.WIN.blit(self.onespecial, (20, 450))

    #procedure to draw buttons

    def draw\_buttons2(self):

        for btn in self.buttons2:

            btn.draw(self.WIN, btn.button\_colour)

    #procedure to draw input boxes

    def draw\_inputs2(self):

        for box in self.input\_boxes2:

            box.update()

            box.draw(self.WIN)

    #procedure that combines drawing input boxes and buttons

    def draw\_all2(self):

        self.WIN.fill(White)

        self.draw\_writings2()

        self.draw\_buttons2()

    #wrapper to call check input e.g. input new password

    def check\_input2\_wrapper(self):

        return self.check\_input2()

    #main funciton to input passwords to be reset

    def check\_input2(self):

        pwd1 = self.Password1Input.text

        pwd2 = self.Password2Input.text

 #uses Utils.validate\_ method to validate password and check if they meet all criterias

        pwd1\_valid = Utils.validate\_password(pwd1)

        pwd2\_valid = Utils.validate\_password(pwd2)

        if not pwd1\_valid:

            return 'password1 valid'

        if not pwd2\_valid:

            return 'password2 valid'

        if pwd1 != pwd2:

            return 'no match'

#if all criterias were met, program continues to run with no problems, otherwise returns error message

        #explaining which password is invalid

        #connect to database and update password based on username

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

#updates User's password with encrypted version of it Using Utils.encode method

        c.execute(f"""

                UPDATE User

                SET Password = '{Utils.PasswordCheck().encode(pwd1)}'

                WHERE Username = '{USERNAME}'

           """)

        conn.commit()

        #return menu to come back to menu

        return 'menu'

###### Displaying errors ########

    def blit\_first\_not\_valid(self):

        text = SubFont.render('Password 1 Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_second\_not\_valid(self):

        text = SubFont.render('Password 2 Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_pwd\_match(self):

        text = SubFont.render("Passwords don't match", True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_username\_not\_valid(self):

        text = SubFont.render('Username Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_name\_not\_valid(self):

        text = SubFont.render('Name Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def bit\_surname\_not\_valid(self):

        text = SubFont.render('Surname Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_not\_exist(self):

        text = SubFont.render("Record does't exist", True, Red)

        self.WIN.blit(text, (700, 300))

#Class to create new account

class Register(Window):

    start\_x = 450

    #main loop which first asks for Username and Password to be set

    def on(self):

        self.RegisterText = Font.render('Register', True, Black)

        self.UsernameText = SubFont.render('Username', True, Black)

        self.PasswordText = SubFont.render('Password', True, Black)

        self.Password2Text = SubFont.render('Repeat password', True, Black)

        self.Question = Font.render('Security Questions', True, Black)

        self.FirstNameText = SubFont.render('First Name', True, Black)

        self.SurnameText = SubFont.render('Surname', True, Black)

        #####HELP SECTION######

        self.username\_should\_be = SubFont.render('Password Requirements:', True, Black)

        self.numofchar = SmallFont.render('At least 8 characters', True, Black)

        self.oneupper = SmallFont.render('At least 1 upper case', True, Black)

        self.onelower = SmallFont.render('At least 1 lower case', True, Black)

        self.onedigit = SmallFont.render('At least 1 digit', True, Black)

        self.onespecial = SmallFont.render('At least 1 special symbol', True, Black)

        self.UsernameInput = Utils.InputBox(self.start\_x, 250, 200, 30, validate = lambda x: Utils.check\_username(x))

        self.PasswordInput = Utils.InputBox(self.start\_x, 350, 200, 30, password=True, validate = lambda x: Utils.check\_password(x))

        self.Password2Input = Utils.InputBox(self.start\_x, 450, 200, 30, password=True, validate = lambda x: Utils.check\_password(x))

        self.FirstNameInput = Utils.InputBox(self.start\_x, 250, 200, 30)

        self.SurnameInput = Utils.InputBox(self.start\_x, 350, 200, 30)

#input boxes are placed into array for iteration and individual updates

        self.input\_boxes = [self.UsernameInput, self.Password2Input, self.PasswordInput]

        self.input\_boxes2 = [self.FirstNameInput, self.SurnameInput]

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, Black, Grey, White, command= 'return', margin = 10)

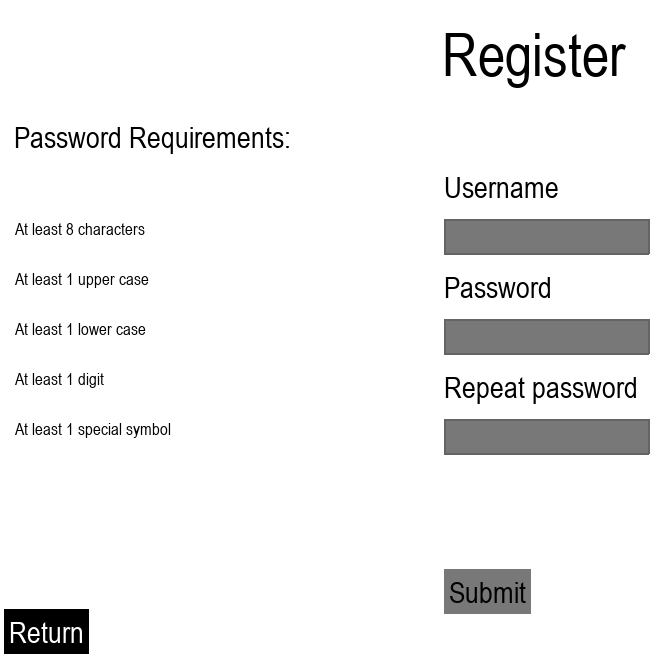
        submitbtn = Button(450, 600, 'Submit', SubFont, Grey, LightGrey, Black, command = lambda: self.check\_input(), margin=10)

        submitbtn2 = Button(450, 600, 'Submit', SubFont, Grey, LightGrey, Black, command = lambda: self.check\_input2(), margin=10)

#buttons are places in an array for iteration over them and individual commands

        self.buttons = [returnbtn, submitbtn]

        self.buttons2 = [returnbtn, submitbtn2]



Num\_of\_char

UsernameInput

PasswordInput

Password2Text

Password2Input

PasswordText

Submitbtn

returnbtn

One\_special

One\_digit

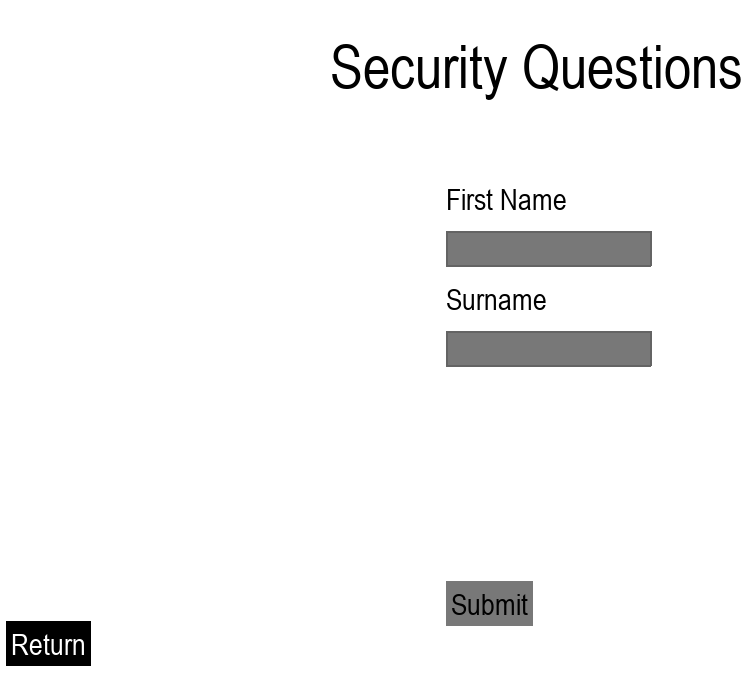
One\_lower

One\_upper

Username\_should\_be

UsernameText

RegisterText

self.draw\_all()

SurnameInput

FirstNameInput

SurnameText

FirstNameText

Questions

Submitbtn2

returnbtn

        #main loop to ask for username and 2 passwords

        run = True

        while run:

            self.display()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                self.draw\_all()

#section to display errors occured during regestering are diaplayed

                try:

                    if resume == 'username wrong':

                        self.blit\_username\_not\_valid()

                    if resume == 'username taken':

                        self.blit\_username\_taken()

                    if resume == 'password 1 wrong':

                        self.blit\_password1\_not\_valid()

                    if resume == 'password 2 wrong':

                        self.blit\_password2\_not\_valid()

                    if resume == 'passwords match':

                        self.blit\_passwords\_match()

                except:

                    pass

                self.draw\_inputs()

                #updates input boxes

                for box in self.input\_boxes:

                    box.handle\_event(event)

                #updates buttons

                for btn in self.buttons:

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    resume = btn.command()

                                    if resume != False:

                                        if resume == None:

                                            continue

                                        if resume == True:

                                            self.draw\_buttons()

                                            self.draw\_all()

                                            return True

                                    else:

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

        return False

    #subroutine to clear out inputs

    def empty\_fields(self, inputs):

        for inp in inputs:

            inp.text = ''

            inp.display\_text = ''

##### Display text, buttons and Input boxes######

    def blit\_username\_not\_valid(self):

        text = SubFont.render('Username Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_password1\_not\_valid(self):

        text = SubFont.render('Password 1 Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_password2\_not\_valid(self):

        text = SubFont.render('Password 2 Not Valid', True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_passwords\_match(self):

        text = SubFont.render("Passwords don't match", True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_username\_taken(self):

        text = SubFont.render("Username already taken", True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_first\_not\_valid(self):

        text = SubFont.render("First name not valid", True, Red)

        self.WIN.blit(text, (700, 300))

    def blit\_surname\_not\_valid(self):

        text = SubFont.render("Surname not valid", True, Red)

        self.WIN.blit(text, (700, 300))

    def draw\_writings2(self):

        self.WIN.blit(self.Question, (self.WIN\_WIDTH//2- self.Question.get\_width()//2, 50))

        self.WIN.blit(self.FirstNameText, (self.start\_x, 200))

        self.WIN.blit(self.SurnameText, (self.start\_x, 300))

    #method to draw buttons1

    def draw\_buttons(self):

        for btn in self.buttons:

            btn.draw(self.WIN, btn.button\_colour)

    #method to draw buttons2

    def draw\_buttons2(self):

        for btn in self.buttons2:

            btn.draw(self.WIN, btn.button\_colour)

    #method to draw inputs1

    def draw\_inputs(self):

        for box in self.input\_boxes:

            box.update()

            box.draw(self.WIN)

#method to draw inputs2

def draw\_inputs2(self):

        for box in self.input\_boxes2:

            box.update()

            box.draw(self.WIN)

 #method to draw everything1

    def draw\_all(self):

        self.WIN.fill(White)

        self.draw\_writings()

        self.draw\_buttons()

    #method to draw everything2

    def draw\_all2(self):

        self.WIN.fill(White)

        self.draw\_writings2()

        self.draw\_buttons2()

    #wrapper method to call next functon in case every check is passed

    def check\_input(self):

        return self.check\_input1()

#function to validates username and checks agaist database

    def check\_input1(self):

        username = self.UsernameInput.text

        password = self.PasswordInput.text

        password2 = self.Password2Input.text

        #validats username and passwords using Utils.validate\_ methods

        username\_valid = Utils.validate\_username(username)

        pwd\_valid = Utils.validate\_password(password)

        pwd\_valid2 = Utils.validate\_password(password2)

        if not username\_valid:

            return 'username wrong'

        if not pwd\_valid:

            return 'password 1 wrong'

        if not pwd\_valid2:

            return 'password 2 wrong'

        if password != password2:

            return "passwords match"

        #connects to database and fetches username

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

SQL fetches username from table User. If record if fetched, it means that username already exists and therefore displays “Username taken”

        c.execute(f"""

            SELECT Username

            FROM User

            WHERE Username = '{username}'

        """)

        row = c.fetchone()

        #if username is in database, then returns error as usernames are unique

        if row:

            return 'username taken'

        self.USERNAME = username

        resume = self.submit()

        return resume

    #asks user to input their first name and surname for easier password reset

    def submit(self):

        self.draw\_all2()

        run = True

        while run:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                for box in self.input\_boxes2:

                    box.handle\_event(event)

                #error handling

                try:

                    if resume == 'first not valid':

                        self.blit\_first\_not\_valid()

                    if resume == 'second not valid':

                        self.blit\_surname\_not\_valid()

                except:

                    pass

                self.draw\_inputs2()

                for btn in self.buttons2:

                    self.display()

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    resume = btn.command()

                                    if resume != False:

                                        if resume == None:

                                            continue

                                        if resume == True:

                                            self.draw\_all2()

                                            return True

                                    else:

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

        return False

    def check\_input2(self):

        return self.check\_input3()

    #function to inserts value in a table Users after validating them

    def check\_input3(self):

        first\_name = self.FirstNameInput.text

        surname = self.SurnameInput.text

 #uses Utils.valivate\_ methods to validate first and last name

  #BUG: doesnt allow names with spaces e.g. St Clair or Al Jorani

        first\_name\_valid = Utils.validate\_name(first\_name)

        surname\_valid = Utils.validate\_name(surname)

        if not first\_name\_valid:

            return 'first not valid'

        if not surname\_valid:

            return 'second not valid'

        #connects to a database and inserts into table Users

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute("""

            INSERT INTO User (Username, FirstName, Surname, Password)

                            VALUES(?,?,?,?)""", (self.USERNAME, first\_name, surname, Utils.PasswordCheck().encode(self.PasswordInput.text)))

        conn.commit()

        global USERNAME

        USERNAME = self.UsernameInput.text

        #empties fields and connects to main menu

        self.empty\_fields(self.input\_boxes)

        self.empty\_fields(self.input\_boxes2)

        resume = MainMenu().on()

        return True

    #requirements for password to be

    def draw\_writings(self):

        self.WIN.blit(self.RegisterText, (self.WIN\_WIDTH//2- self.RegisterText.get\_width()//2, 50))

        self.WIN.blit(self.UsernameText, (self.start\_x, 200))

        self.WIN.blit(self.PasswordText, (self.start\_x, 300))

        self.WIN.blit(self.Password2Text, (self.start\_x, 400))

        self.WIN.blit(self.username\_should\_be, (20, 150))

        self.WIN.blit(self.numofchar, (20, 250))

        self.WIN.blit(self.oneupper, (20, 300))

        self.WIN.blit(self.onelower, (20, 350))

        self.WIN.blit(self.onedigit, (20, 400))

        self.WIN.blit(self.onespecial, (20, 450))

class MainMenu(Window):

    start\_x = 450

    def on(self):

        #sets buttons that call wrappers

        self.MainMenuText = Font.render('Main Menu', True, Black)

        #sets buttons that call wrappers

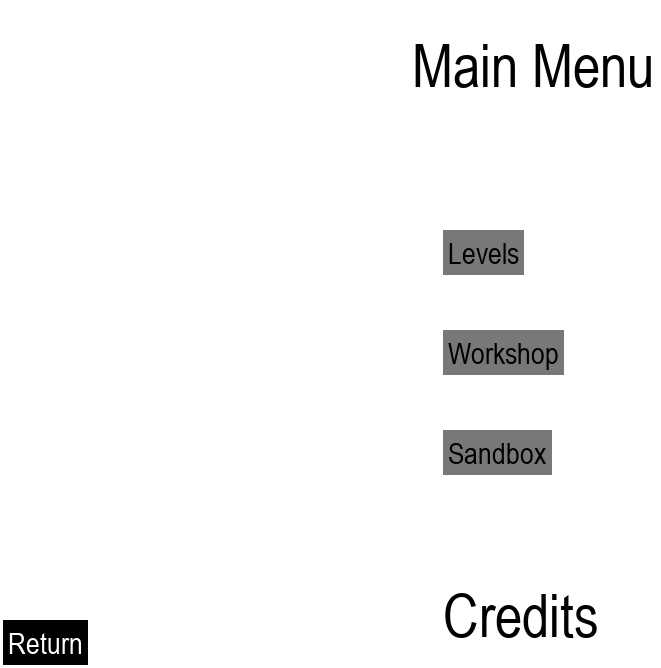
        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, Black, Grey, White, command= 'return', margin = 10)

        levelsbtn = Button(self.start\_x, 250, 'Levels', SubFont, Grey, LightGrey, Black, command = lambda: self.call\_levels(), margin=10)

        workshopbtn = Button(self.start\_x, 350, 'Workshop', SubFont, Grey, LightGrey, Black, command = lambda: self.call\_workshop(), margin=10)

        sandboxbtn = Button(self.start\_x, 450, 'Sandbox', SubFont, Grey, LightGrey, Black, command = lambda: self.sandbox\_wrapper(), margin = 10)

        credits = Button(self.start\_x, 600, 'Credits', Font, White, White, Black, command=lambda: self.credits\_wrapper())



credits

returnbtn

sandboxbtn

workshopbtn

levelsbtn

MainMenuText

        self.buttons = [returnbtn, levelsbtn, sandboxbtn, credits, workshopbtn]

        self.draw()

        #main loop cheking button behaviour and calls its functions

        run = True

        while run:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                for btn in self.buttons:

                    self.display()

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    resume = btn.command()

                                    if resume != False:

                                        if resume == None:

                                            continue

                                        self.draw()

                                    else:

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

        return False

#wrappers to call next function, without them nothing would be returned and it would be problematic to uodate the screen

    def call\_levels(self):

        return Levels().on()

    def call\_workshop(self):

        return Workshop().on()

    def sandbox\_wrapper(self):

        return Sandbox().on()

    def credits\_wrapper(self):

        return Credits().on()

##### Drawing section ######

    def draw(self):

        self.WIN.fill(White)

        self.draw\_buttons()

        self.draw\_writings()

    def draw\_buttons(self):

        for btn in self.buttons:

            btn.draw(self.WIN, btn.button\_colour)

        self.display()

    def draw\_writings(self):

        self.WIN.blit(self.MainMenuText, (self.WIN\_WIDTH//2- self.MainMenuText.get\_width()//2, 50))

#Credits class that was implemented to show function to deal with multi-line text

class Credits(Window):

    def on(self):

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, Black, Grey, White, command= 'return', margin = 10)

        self.buttons = [returnbtn]

        #main loop to refresh text

        run = True

        while run:

            self.display()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                self.blit\_writings()

                for btn in self.buttons:

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command == 'return':

                                    return True

                                else:

                                    resume = btn.command()

                                    if resume != False:

                                        if resume == None:

                                            continue

                                        if resume == True:

                                            self.draw\_buttons()

                                            self.blit\_writings()

                                            return True

                                    else:

                                        run = False

                                        break

                    else:

                        btn.default(self.WIN)

Text can be changed and using Utils fruncitons, special characters like \n and \t can be used

        return False

    def blit\_writings(self):

        self.WIN.fill(White)

        #text can be modified and displayed on the screen with set parameters

        text = "Made by Dmitrii Ponomarev\n\nFull credited by me and my mates DN and I(HA-HA)"

        Utils.blit\_text(self.WIN, text, pos = (50, 50), font = Font, max\_width = 800)

#class node that is responsible for solving mazes

class Node:

    def \_\_init\_\_(self, pos, parent):

        self.pos = pos

        self.parent = parent

#fCost is the total cost of the node

        #hCost is distance between current node and end's node

        #gCost is distance between current node and end's node

        #length is total distance incuting cutting corners, which isn't used in this program, as diagonal moves are't allowed

        self.gCost = 0

Next 2 classes don’t have any interface as they are responsible for movement and node positioning.

        self.hCost = 0

        self.fCost = 0

        self.length = 0

    def calc\_fCost(self, end\_node):

        #checking is parent exists (not start and end) and makes g cost and f cost

        if self.parent:

            a = abs(self.pos[0] - end\_node.pos[0])

            b = abs(self.pos[1] - end\_node.pos[1])

            #simple Pythagorean theorem used e.g. A^2 + B^2 = C^2

            self.gCost = self.parent.gCost + ((self.pos[0]-self.parent.pos[0])\*\*2 + (self.pos[1]-self.parent.pos[1])\*\*2 )\*\*0.5

            self.length = ((self.pos[0]-self.parent.pos[0])\*\*2 + (self.pos[1]-self.parent.pos[1])\*\*2 )\*\*0.5

            self.hCost = a+b

            self.fCost = self.gCost + self.hCost

    #allows to compare node's coordinates

    def \_\_eq\_\_(self, others):

        if self.pos == others.pos:

            return True

        else:

            return False

#class solving to show process of solving the grid

class Solving(Window):

    def \_\_init\_\_(self, level):

        #inherits \_\_init\_\_ method from Window to allow the use of display and WINDOW\_WIDTH and HEIGHG

        super().\_\_init\_\_()

        self.moves = 0

        #sets up a trail as a stack as a path

        self.trail = Utils.Stack()

        self.grid = level.grid

        self.start\_pos = level.start

        self.end\_pos = level.end

        #declaring start and end nodes

        self.start\_node = Node(self.start\_pos, None)

        self.end\_node = Node(self.end\_pos, None)

        #sets constants (CAPITALS) for easier identification, as there is no constants in Python

        self.WIDTH = level.WIDTH

        self.HEIGHT = level.HEIGHT

        self.node\_pos = self.start\_pos

        self.start = True

        #starts stack by pushing start node at the very beginning

        self.trail.push(self.start\_node)

    ######### moding node procedures ############

    def move\_left(self, parent):

            #if in boundary, sets node's position to a wall

        if self.check\_boundaries(0, -1):

            if self.grid[self.node\_pos[0]][self.node\_pos[1]-1] != '1': #wall

                self.node\_pos = (self.node\_pos[0], self.node\_pos[1]-1)

                self.moves+=1

                #adds 1 to moves to track how many moved the player did

                node = Node((self.node\_pos[0],self.node\_pos[1]), parent)

                #pushed node created to the stack to allow tracking back

                self.trail.push(node)

    def move\_right(self, parent):

        #if in boundary, sets node's position to a wall

        if self.check\_boundaries(0, 1):

            if self.grid[self.node\_pos[0]][self.node\_pos[1]+1] != '1': #wall

                self.node\_pos = (self.node\_pos[0], self.node\_pos[1]+1)

                self.moves+=1

                #adds 1 to moves to track how many moved the player did

                node = Node((self.node\_pos[0],self.node\_pos[1]), parent)

                #pushed node created to the stack to allow tracking back

                self.trail.push(node)

    def move\_down(self, parent):

        #if in boundary, sets node's position to a wall

        if self.check\_boundaries(1, 0):

            if self.grid[self.node\_pos[0]+1][self.node\_pos[1]] != '1':

                self.node\_pos = (self.node\_pos[0]+1, self.node\_pos[1])

                self.moves+=1

                #adds 1 to moves to track how many moved the player did

                node = Node((self.node\_pos[0],self.node\_pos[1]), parent)

                #pushed node created to the stack to allow tracking back

                self.trail.push(node)

    def move\_up(self, parent):

        #if in boundary, sets node's position to a wall

        if self.check\_boundaries(-1, 0):

            if self.grid[self.node\_pos[0]-1][self.node\_pos[1]] != '1':

                self.node\_pos = (self.node\_pos[0]-1, self.node\_pos[1])

                self.moves+=1

                #adds 1 to moves to track how many moved the player did

                node = Node((self.node\_pos[0],self.node\_pos[1]), parent)

                #pushed node created to the stack to allow tracking back

                self.trail.push(node)

#function to check boundaries by adding row and column attributes to node's position to check if next node withing the allowed space

    def check\_boundaries(self, row, col):

        if 0 <= self.node\_pos[1] + col < self.WIDTH:

            if 0 <= self.node\_pos[0] + row < self.HEIGHT:

                return True

        return False

    #function that checks if node's position is end's position

    def check\_end(self):

        if self.node\_pos == self.end\_pos:

            self.start = False

            return True

        return False

    #function to show path by poping items from stack

    def show\_path(self):

Makes use of stack, as nodes are placed in a structure, which behaves like one, but could be implemented without

        for i in range(self.trail.size()):

            node = self.trail.pop()

            row, col = node.pos

            if self.grid[row][col] not in [1,2,3]:

                self.grid[row][col] = '4'

    #procedure that displays text on the screen and how many moves it took to complete

    def blit\_ending(self, moves):

        text = Font.render(f'Well done, it took you {moves} moves', True, Red)

        self.WIN.blit(text, (self.WIN\_WIDTH//2- text.get\_width()//2, 350))

# main class for All levels includint worshop and in-built levels

class AllLevels(Window):

#sets class global variables, that can be accesses anywhere within this class

#better to set it up like that, not in \_\_init\_\_ method to avoid many variables there and devide variables and constants

    gap\_x = 150

Parent class for Levels and Workshop that contains main methods and functions used by both.

It is important to set a parent class with common methods not to repeat the code and makes it more robust

    gap\_y = 200

    button\_num = 5

    btn\_rows = 2

    button\_width = 100

    button\_height = 100

    panel\_x = 100

    panel\_y = 0

    FontNumbers = pygame.font.SysFont('arial', 80)

    #subroutine that fetches levels from table

    def fetch\_levels(self, inbuilt = True):

        #connects to database and uses JOIN method to connect table Maze and Uploaded to get the creator of each maze

        #if its in-built, creator will be none and therefore sent to Workshop rather than Levels class

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute(f"""   SELECT Maze.MazeID, Maze.Base128, Maze.Inbuilt, Maze.Width,

                            Maze.Height, Maze.StartX, Maze.StartY, Maze.EndX, Maze.EndY, Uploaded.Username

                            FROM Maze

SQL uses JOIN command, which was explained earlier on. It allows to fetch records from 2 tables straight away, rather than using several SELECT statements.

                            LEFT JOIN Uploaded

                            ON Maze.MazeID = Uploaded.MazeID

                    """)

        conn.commit()

        row = curr.fetchall()

        count = 0

        self.levels = []

        for \_, entity in enumerate(row):

        # \_ is used to show that variable is not used anywhere, which won't affect anything

            #goes through each fetched entry and assigns to a parameter to be passes to Utils.MakeGrid functon

            mazeID, base128, Inbuilt, Width, Height, StartX, StartY, EndX, EndY, MadeBy = entity

            if MadeBy:

                MadeBy = MadeBy[:5] #only first 5 letters used

            Start = (StartX, StartY)

            End = (EndX, EndY)

            binary = Utils.Binary128().decode(base128)

            if inbuilt == Inbuilt:

                count+=1

                lvl = Utils.MakeGrid(mazeID, binary, Height, Width, Start, End, showID=count, made\_by=MadeBy)

                lvl.complete()

                self.levels.append(lvl)

    #procedure that fetches ID's to be assigned to in-built levels

    def fetch\_ID(self):

        IDs = []

        for lvl in self.levels:

            IDs.append(lvl.showID)

        self.IDs = IDs

     #function to add tiles on the screen

    def add\_buttons(self, inbuilt = True, bunch = 0):

        copy = bunch

        btns\_on = self.btn\_rows\*self.button\_num

        start\_num = bunch \* btns\_on

        #creates array of buttons with initial button of a return to set undex equal to button's ID for easier ID and understading

        self.buttons = [self.buttons[0]]

        list\_ID = self.IDs[start\_num: start\_num + btns\_on]

        start\_x = x = 200

        start\_y = y = 200

        #algorithms to add level buttons in array of buttons with its position

        adder = 0

        row = 0

        column = 0

        for adder in range(len(list\_ID)):

            #iterates through length of ID fetched in a function above

            if inbuilt:

                #if levels to be displaed are inbuilt, its ID should be displayed

                btn = Button(x, y, str(list\_ID[adder]), self.FontNumbers, button\_colour = Grey, hover\_colour = LightGrey, text\_colour = Black, ID = list\_ID[adder], margin = 20, height = 100, width = 100, active = False, inactive\_colour=(100,100,0))

            else:

                #if levels to be displayed are made by user, first 5 letters of a name should be displayed

                name = self.levels[adder].made\_by

                btn = Button(x, y, str(name), SubFont, button\_colour = Grey, hover\_colour = LightGrey, text\_colour = Black, ID = list\_ID[adder], margin = 20, height = 100, width = 100)

            #on each iteration, row is added to move position of a button

            row+=1

            #button is added to array buttons for iteration

            self.buttons.append(btn)

            x += self.gap\_x

            #if row equals to pre-set max number of buttons in a row, row is set to 0 to start a new row and y increases to move buttons down

            if row == self.button\_num:

                row = 0

                x = start\_x

                y+=self.gap\_y

            column +=1

        #adds next and previous buttons

        #previous button is added when bunch is greater than 0, meaning that it was already on page 2

        if bunch > 0:

            prev\_btn = Button(40, 400, 'Previous', SubFont, button\_colour = Black, hover\_colour = Grey, text\_colour = LightGrey, margin = 5, width = 100, command = lambda: self.add\_buttons(bunch = copy-1, inbuilt=inbuilt))

            self.buttons.append(prev\_btn)

        #next button is added when number of ID in a list os greater that buttons displayed

        if len(list\_ID) > btns\_on:

            next\_btn = Button(self.WIN\_WIDTH-150, 400, 'Next', SubFont, button\_colour = Black, hover\_colour = Grey, text\_colour = LightGrey, margin = 5, width = 100, command = lambda: self.add\_buttons(bunch = copy+1, inbuilt=inbuilt))

            self.buttons.append(next\_btn)

        #fetches all solved levels from the tale and unlocks them plus unlocks the next one to be solved

        if inbuilt:

            self.get\_solved()

            str\_solved\_IDs = list(map(str, self.solved\_IDs))

            for btn in self.buttons:

                if btn.Text in str\_solved\_IDs:

                    btn.active = True

        return True #so it would refresh

#functon to build a level to be able to solve it

    def build\_level(self, ID, inbuilt):

        solved = False

        #copy here to update levels as I overwrite level when showing paths

        self.fetch\_levels(inbuilt = inbuilt)

        #iterates thougs levels in an array to understand which level to show

        for index, lvl in enumerate(self.levels):

            if lvl.showID == ID:

                break

        #level is fetched from the array and grid is assigned to self.grid for use in a class

        level = self.levels[index]

        self.grid = level.grid

        #starting node is made

        self.node = Solving(level)

        #level is generated with set parameters in a level class itself

        self.generate(level)

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, (Black), (Grey), (White), command= 'return', margin = 10)

        buttons = [returnbtn]

        #main loop to display elements

        self.Draw()

        run = True

        while run:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                if not solved:

                    self.draw\_grid(level)

                for btn in buttons:

                    self.display()

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command:

                                    if btn.command == 'return':

                                        return True

                                    else:

                                        resume = btn.command()

                                        if resume != False:

                                            if resume == None:

                                                continue

                                            self.Draw()

                                            self.draw\_grid(level)

                                        else:

                                            run = False

                                            break

                    else:

                        btn.default(self.WIN)

                ########## movement or a node ############

                if self.node.start:

                    if event.type == pygame.KEYDOWN:

                        #pygame detects what key was pressed and calls specific function for it

                        if event.key == pygame.K\_LEFT:

                            self.node.move\_left(self.node)

                        if event.key == pygame.K\_RIGHT:

                            self.node.move\_right(self.node)

                        if event.key == pygame.K\_UP:

                            self.node.move\_up(self.node)

                        if event.key == pygame.K\_DOWN:

                            self.node.move\_down(self.node)

                        solved = self.node.check\_end()

                        if solved:

                            #if maze was solved, meaning that user reached the end node, path is shown to signify that level is complete

                            self.node.show\_path()

                            self.draw\_grid(level)

                            self.node.blit\_ending(self.node.moves)

                            #displays text stating that level was indeed solved

                            if inbuilt == True:

                                #if level is inbuilt and it was the last one available to the player, next one is unlocked

                                if ID == max(self.solved\_IDs):

                                    self.get\_next\_level()

                                else:

                                    self.insert\_level\_solved(ID)

                            else:

                                self.insert\_level\_solved(ID)

        return False

    #procedure that generates level and centers the window to it would be same gap on top and botton

    def generate(self, level):

        HEIGHT = level.HEIGHT

        WIDTH = level.WIDTH

        #measures allowed space between panel and full findow to find the center of a window

        allowed\_x = self.WIN\_WIDTH - self.panel\_x

        allowed\_y = self.WIN\_HEIGHT - self.panel\_y

        #sets a swuare size by selecting a mininum of width and height

        self.square\_size = min(((allowed\_x//WIDTH), (allowed\_y//HEIGHT)))

        self.square\_size = (self.square\_size, self.square\_size)

        diff\_x = allowed\_x - (self.square\_size[0] \* WIDTH)

        diff\_y = allowed\_y - (self.square\_size[1] \* HEIGHT)

        #creates starting poing of a maze to be centered depending on a square side and panel

        self.start\_x = self.panel\_x + diff\_x//2

        self.start\_y = self.panel\_y + diff\_y//2

        #created ending poing depending on square sise and dimensions of a screen

        self.end\_x = self.square\_size[0]\*WIDTH + self.start\_x

        self.end\_y = self.square\_size[1]\*HEIGHT + self.start\_y

        #displayes grid generated

        self.draw\_grid(level)

    #procedure that displays grid on the window

    def draw\_grid(self, level):

        #assigns colours colours in a dictionary that can be fetched easily

        colours = { 'wall': (Black),

                    'empty': (White),

                    'start': (Green),

                    'end': (Red),

                    'user': (Yellow),

                    'solved': (Cyan)

        }

        #goes through one by one in a grid and updates the color of each square

        for row in range(level.HEIGHT):

            for col in range(level.WIDTH):

                cell = self.grid[row][col]

                if cell == '1': # wall

                    colour = colours['wall']

                elif cell == '0': #empty

                    colour = colours['empty']

                if (row,col) == (level.start):

                    self.grid[row][col] = '2' #start number

                    colour = colours['start']

                    self.start\_pos = (row, col)

                elif (row,col) == (level.end):

                    self.grid[row][col] = '3' #end number

                    self.end\_pos = (row, col)

                    colour = colours['end']

                if cell == '4': #solved number

                    colour = colours['solved']

                if (row, col) == (self.node.node\_pos):

                    colour = colours['user']

                #fills a square with set size above and fill it with a colour selected in if statements above

                pygame.draw.rect(self.WIN, colour, (self.square\_size[0] \* col + self.start\_x, self.square\_size[1] \* row + self.start\_y, self.square\_size[0], self.square\_size[1]))

        #draws vertical and horiozontal lines to separate grid

        for row in range(level.HEIGHT+1):

            pygame.draw.line(self.WIN, (Grey), (self.start\_x, self.square\_size[1] \* row + self.start\_y), (self.end\_x, self.square\_size[1] \* row + self.start\_y))

        for col in range(level.WIDTH+1):

            pygame.draw.line(self.WIN, (Grey), (self.start\_x + self.square\_size[0] \*col, self.start\_y), (self.start\_x + self.square\_size[0] \*col, self.end\_y))

        # self.display()

    #procedure to display window and draw it

    def Draw(self):

        self.WIN.fill(White)

        pygame.draw.line(self.WIN, Red, (self.panel\_x, 0), (self.panel\_x, self.WIN\_HEIGHT))

        self.display()

#class for In-built levels

class Levels(AllLevels):

    panel\_x = 100

    def on(self):

        #fetches levels and IDs from database

        self.fetch\_levels(inbuilt=True)

        self.fetch\_ID()

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, (Black), (Grey), (White), command= 'return', margin = 10)

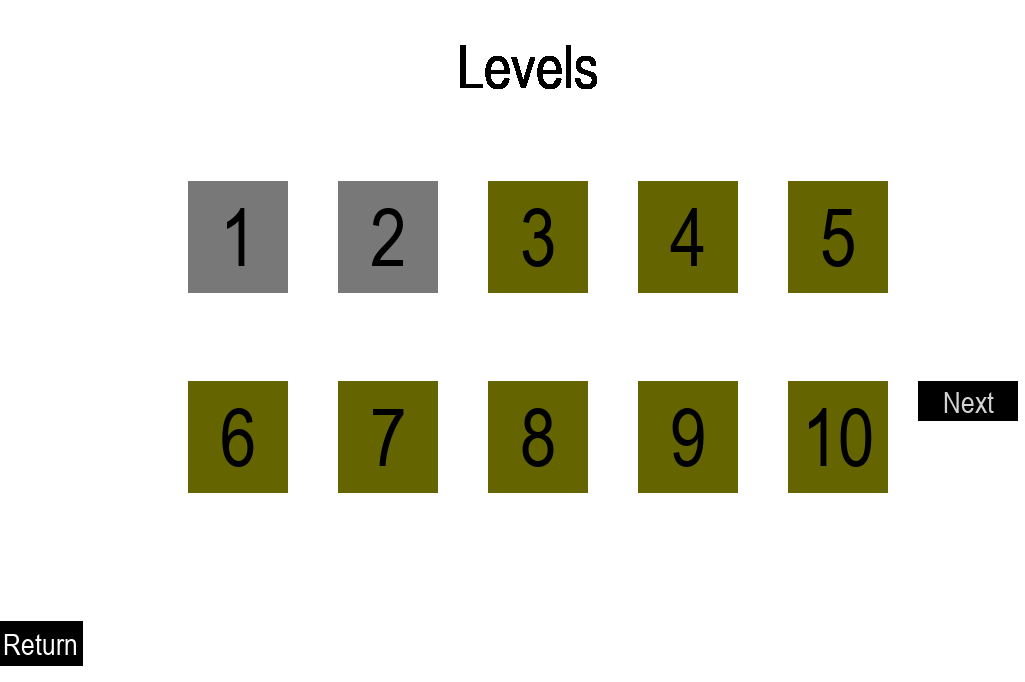
        self.Levels = Font.render('Levels', True, Black)

        self.buttons = [returnbtn]

        #adds level buttons to the screen and draws it

        self.add\_buttons()

        self.draw()



Next\_btn

Unocked button

Locked button

returnbtn

Levels

        run = True

        while run:

            self.draw\_writings()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                for btn in self.buttons:

                    self.display()

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.active:

                                    if btn.command:

                                        if btn.command == 'return':

                                            return True

                                        else:

                                            resume = btn.command()

                                            if resume != False:

                                                if resume == None:

                                                    continue

                                                self.draw()

                                                # self.draw\_grid()

                                            else:

                                                run = False

                                                break

                                    else:

                                        # if button is clicked that does't have a functon assigned, its a tile

                                        # meaning that it has level contained

                                        # fetches ID from the button and builds a level based on the ID

                                        ID = btn.Text

                                        resume = self.build\_level(int(ID), True)

                                        if resume != False:

                                            if resume == None:

                                                continue

                                            self.draw()

                                        else:

                                            run = False

                                            break

                    else:

                        btn.default(self.WIN)

        return False

    #function to fetch all solved levels from table UserMaze where Inbuilt is True

    def get\_solved(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

SQL uses JOINT command and fetches only MazeID where Inbuilt is True, as they are inbuilt

        c.execute(f"""SELECT Maze.MazeID

                          FROM Maze

                          LEFT JOIN UserMaze

                          ON Maze.MazeID = UserMaze.MazeID

                          Where UserMaze.Username = '{USERNAME}' AND Maze.Inbuilt = '{int(True)}'

                """)

        row = c.fetchall()

        self.solved\_IDs = []

        for r in row:

            self.solved\_IDs.append(int(r[0])) #adds one which were previously solved

        if len(self.solved\_IDs) != 0:

            max\_id = max(self.solved\_IDs)+1 #to add next level

            self.solved\_IDs.append(max\_id)

            self.solved\_IDs = list(set(self.solved\_IDs)) #get rid of duplicates as user can solve 1 maze several times

        else:

            self.solved\_IDs.append(1)

    #procedure to insert level solved into table UserMaze

    def insert\_level\_solved(self, ID):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute(f"""INSERT INTO UserMaze (MazeID, Username) VALUES(?,?)""",

                            (str(ID), USERNAME))

        conn.commit()

    #procedure to insert level solved into table UserMaze and ublocks next level

    def get\_next\_level(self):

        max\_val = max(self.solved\_IDs)

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute(f"""INSERT INTO UserMaze (MazeID, Username) VALUES(?,?)""",

                            (str(max\_val), USERNAME))

        conn.commit()

        self.add\_buttons()

    ######### display and refresh

    def draw\_writings(self):

        self.WIN.blit(self.Levels, (self.WIN\_WIDTH//2- self.Levels.get\_width()//2, 50))

    def draw(self):

        self.WIN.fill(White)

        self.display()

#class for user-made levels

class Workshop(AllLevels):

    panel\_x = 100

    def on(self):

        #fetches all man-made levels

        self.fetch\_levels(inbuilt=False)

        self.fetch\_ID()

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, (Black), (Grey), (White), command= 'return', margin = 10)

        self.Levels = Font.render('Worshop', True, Black)

        self.buttons = [returnbtn]

        self.add\_buttons(inbuilt=False)

        self.draw()



returnbtn

Prev\_btn

Unlocked Buttons

Levels

        run = True

        while run:

            self.draw\_writings()

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                for btn in self.buttons:

                    self.display()

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.active:

                                    if btn.command:

                                        if btn.command == 'return':

                                            return True

                                        else:

                                            resume = btn.command()

                                            if resume != False:

                                                if resume == None:

                                                    continue

                                                self.draw()

                                                # self.draw\_grid()

                                            else:

                                                run = False

                                                break

                                    else:

                                        #builds level based on the ID fetched when the tile was clicked

                                        ID = btn.ID

                                        resume = self.build\_level(int(ID), False)

                                        if resume != False:

                                            if resume == None:

                                                continue

                                            self.draw()

                                        else:

                                            run = False

                                            break

                    else:

                        btn.default(self.WIN)

        return False

    #get the names of users who uploaded the maze

    def get\_names(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute(f"""

            SELECT Username

            FROM Uploaded""")

        row = c.fetchall()

        names = []

        for ID in row:

            names.append(ID[0])

        return names

    #Fetches ID from Uploaded and inserts into UserMaze table

    def insert\_level\_solved(self, ID):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute(f"""SELECT MazeID FROM Uploaded WHERE UploadedID == '{ID}' """)

        ID = curr.fetchone()[0]

        curr.execute(f"""INSERT INTO UserMaze (MazeID, Username) VALUES(?,?)""",

                            (str(ID), USERNAME))

        conn.commit()

    #######method to draw and display ##########

    def draw\_writings(self):

        self.WIN.blit(self.Levels, (self.WIN\_WIDTH//2- self.Levels.get\_width()//2, 50))

    def draw(self):

        self.WIN.fill(White)

        self.display()

#class sandbox to generate random maze, upload it and solve it

class Sandbox(Window):

    panel\_x = 200

Class to build new levels and solve them using Breadth-first search and upload levels to Workshop

    panel\_y = 0

    moving = False

    editing = False

    deleting = False

    #getter methods

    def get\_width(self):

        return self.\_\_width\_text

    def get\_height(self):

        return self.\_\_height\_text

    def on(self):

        self.\_\_width\_text = SubFont.render('Width', True, Black)

        self.\_\_height\_text = SubFont.render('Height', True, Black)

        #section of code to create all buttons needed for running and make sandbox functioning

        returnbtn = Button(10, self.WIN\_HEIGHT-80, 'Return', SubFont, Black, Grey, White, command = 'return', margin = 10)

        btn\_empty = Button(110, self.WIN\_HEIGHT-80, 'Empty', SubFont, (Black), (Grey), (White), command= lambda: self.generate(algorithm='empty'), margin = 10)

        btn\_generate = Button(50, 80, 'Generate', SubFont, (Black), (Grey), (White), command = lambda: self.generate(), margin = 10 )

        btn\_start = Button(50, 500, 'Start Solving', SubFont, (Black), (Grey), (White), command= lambda: self.start\_solving\_wrapper(), margin = 10)

        btn\_upload = Button(50, 550, 'Upload', SubFont, (Black), (Grey), (White), command= lambda: self.upload\_wrapper(), margin = 10)

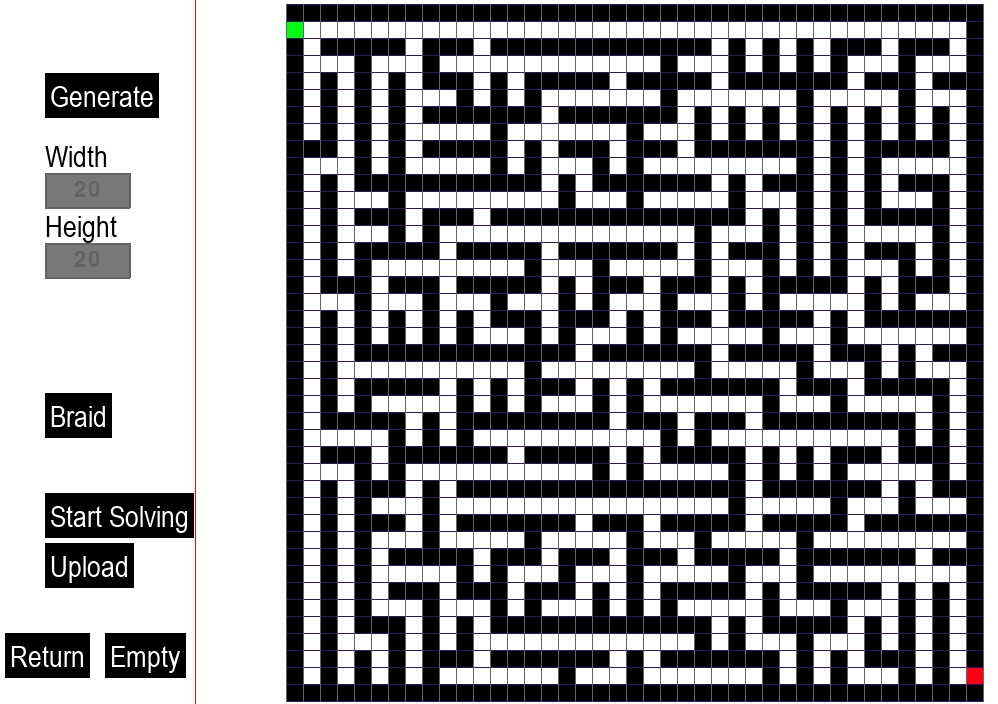
        btn\_braid = Button(50, 400, 'Braid', SubFont, (Black), (Grey), (White), command= lambda: self.braid(), margin = 10)

        self.width\_input = Utils.InputBox(50, 180, 80, 30, validate=lambda x: Utils.check\_numbers(x))

        self.height\_input = Utils.InputBox(50, 250, 80, 30, validate=lambda x: Utils.check\_numbers(x))

        self.buttons = [returnbtn, btn\_generate, btn\_empty, btn\_start, btn\_upload, btn\_braid]

        self.input\_boxes = [self.width\_input, self.height\_input]



Randomly generated maze with width and height that user promted

Btn\_start

Windth\_input

height\_input

Returnbtn

Btn\_empty

Btn\_upload

Btn\_braid

Btn\_generate

        #sets clock for consistent solving time not dependent on GPU power

#sets clock for consistent solving time not dependent on GPU power

        clock = pygame.time.Clock()

        self.draw()

        run = True

        while run:

            self.display()

            clock.tick(self.FPS)

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    run = False

                    break

                pos = pygame.mouse.get\_pos()

                self.draw()

                self.draw\_inputs()

                try:

                    self.draw\_grid()

                except:

                    pass

                #exception handling and displaying messages

                try:

                    if resume == 'boundary':

                        self.blit\_boundary()

                    if resume == 'exists':

                        self.blit\_exists()

                    if resume == True or resume == 'DN':

                        self.blit\_success()

                    if resume == 'no path':

                        self.blit\_no\_path()

                        self.clicked = False

                except:

                    pass

                for box in self.input\_boxes:

                    box.handle\_event(event)

                for btn in self.buttons:

                    if btn.is\_over(pos):

                        btn.hover(self.WIN)

                        if event.type == pygame.MOUSEBUTTONDOWN:

                            if pygame.mouse.get\_pressed()[0]:

                                if btn.command:

                                    if btn.command == 'return':

                                        return True

                                    else:

                                        resume = btn.command()

                                        if resume != False:

                                            if resume == None:

                                                continue

                                            if resume == True:

                                                self.display()

                                        else:

                                            run = False

                                            break

                    else:

                        btn.default(self.WIN)

                if event.type == pygame.MOUSEBUTTONDOWN:

                    pos = pygame.mouse.get\_pos()

                    try:

                        pos = self.get\_pos\_on\_grid(pos) #to react on grid only

                    except:

                        pos = None

                    #left click allows to set walls on the grid and resets resume to make text disappear

                    # and mode start and end nodes on the grid

                    if pos:

                        if pygame.mouse.get\_pressed()[0]:

                            if pos:

                                self.clicked = True

                                resume = 'dn'

                                row, column = pos

                                cell = self.grid[row][column]

                                #first checks for start/end and if cell is either, skipps next elif starement

                                if cell.is\_start or cell.is\_end:

                                    self.moving = True

                                elif cell.is\_wall or cell.is\_empty:

                                    self.editing = True

                        #right click allows to destoy walls

                        elif pygame.mouse.get\_pressed()[2]:

                            #if mouse click is right click, edting and moving set to false to disallow easing and creating walls at the same time

                            self.clicked = True

                            self.editing = False

                            self.moving = False

                            resume = 'dn'

                            if pos:

                                #if position is valid e.g. not None,

                                #checks for boundary values and allows deleting when cells aren't on a border

                                row, column = pos

                                cell = self.grid[row][column]

                                if row != 0 and row != self.HEIGHT-1 and column != 0 and column != self.WIDTH-1:

                                    if not cell.is\_start and not cell.is\_end:

                                        self.deleting = True

                        else:

                            self.editing = False

                            self.deleting = False

                            self.moving = False

                #resets editing and deleting and moving as soon as mouse button is up

                if event.type == pygame.MOUSEBUTTONUP:

                    self.editing = False

                    self.deleting = False

                    self.moving = False

                #if either of moving, deleting or editing are active, mouse position if fetched and casted onto grid position

                if self.moving or self.deleting or self.editing:

                    pos = pygame.mouse.get\_pos()

                    position = self.get\_pos\_on\_grid(pos)

                    if position:

                        row, column = position

                        #calls move\_cell method in Grid to move start/end

                        if self.moving:

                            self.Grid.move\_cells(cell, (row, column))

                        #if node is not in the border, allows to erase walls, as boundaries are compulsory

                        if row != 0 and row != self.HEIGHT-1 and column != 0 and column != self.WIDTH-1:

                            if self.editing:

                                cell = self.grid[row][column]

                                if not cell.is\_start or not cell.is\_end:

                                    self.grid[row][column].is\_wall = True

                                    self.grid[row][column].is\_empty = False

                            if self.deleting:

                                cell = self.grid[row][column]

                                if cell.is\_wall or cell.is\_empty:

                                    self.grid[row][column].is\_wall = False

                                    self.grid[row][column].is\_empty = True

                    self.draw\_grid()

        return False

     #gets position on the grid, as grid is not on the whole screen returs result only when mouse position

    # inbetween start and end

    def get\_pos\_on\_grid(self, pos):

        x,y = pos

        if self.start\_y < y < self.end\_y and self.start\_x < x < self.end\_x:

            Y = y - self.start\_y

            X = x - self.start\_x

            result = (Y//self.square\_size[1], X //self.square\_size[0])

            return result

        return None

    #wrapper to solve the maze

    def start\_solving\_wrapper(self):

        return self.solve()

    #wrapper to upload a maze

    def upload\_wrapper(self):

        return self.upload()

    ########### displaying texts ########

    def blit\_exists(self):

        text = Font.render("Maze already exists", True, Red)

        self.WIN.blit(text, (500, 500))

    def blit\_success(self):

        text = Font.render("Great Success!!!", True, Red)

        self.WIN.blit(text, (500, 500))

    def blit\_no\_path(self):

        text = Font.render("No path found", True, Red)

        self.WIN.blit(text, (500, 500))

    #funciton to uploads the maze

    def upload(self):

        #exception handling to prevent error when no grid on the screnn

        try:

            binary, HEIGHT, WIDTH, start, end = self.Grid.export()

        except:

            return

        #convers binary into base 128 as takes less space

        Base128 = Utils.Binary128().encode(binary)

        resume = self.solve()

        if resume == 'no path':

            return 'no path'

        if resume == False:

            return False

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        #selects ID from Maze to check if its unique.

        # 2 same mazes cannot exist

        c.execute("""

            SELECT MazeID

            FROM Maze

            Where Base128 = ?

            """, [Base128])

        row = c.fetchall()

        if row:

            return 'exists'

        #if maze is unique, allows to insert into Maze making new level

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''',(Base128, False, WIDTH, HEIGHT, start[0], start[1], end[0], end[1]))

        conn.commit()

        c.execute("""

            SELECT MazeID

            FROM Maze

            Where Base128 = ?

            """, [Base128])

        MazeID = c.fetchone()[0]

        #inserts Username and MazeID to connect the creator of the maze to a maze itself

        c.execute("""INSERT INTO Uploaded (MazeID, Username) VALUES(?,?)""",

                            (MazeID, USERNAME))

        conn.commit()

        return 'DN'

########## Draw functions #########

    def draw(self):

        self.WIN.fill(White)

        self.draw\_writings()

        pygame.draw.line(self.WIN, Red, (self.panel\_x, 0), (self.panel\_x, self.WIN\_HEIGHT))

    def draw\_buttons(self):

        for btn in self.buttons:

            btn.draw(self.WIN, btn.button\_colour)

    def draw\_inputs(self):

        for box in self.input\_boxes:

            box.update()

            box.draw(self.WIN)

    def draw\_writings(self):

        self.WIN.blit(self.get\_width(), (50, 145) )

        self.WIN.blit(self.get\_height(), (50, 215) )

    #procedure that generates maze by using algorithms provided in Generate file

    def generate(self, algorithm = None):

        if self.width\_input.text:

            self.width = int(self.width\_input.text)

        else:

            return

        if self.height\_input.text:

            self.height = int(self.height\_input.text)

        else:

            return

        #cheks if width and height in allowed parametersm othersise generating maze can take too much time

        if self.width > 40 or self.width < 5:

            return 'boundary'

        if self.height > 40 or self.height < 5:

            return 'boundary'

        grid = MakeGrid.Grid()

        algo = ['backtracker', 'binarytree', 'sidewinder', 'cellular', 'pims']

        #selects algorithm by random and sets generator of a grid to be called

        if not algorithm:

            algorithm = choice(algo)

        if algorithm.lower() == 'backtracker':

            grid.generator = Generate.BackTracker(self.width, self.height)

        elif algorithm.lower() == 'binarytree':

            grid.generator = Generate.BinaryTree(self.width, self.height)

        elif algorithm.lower() == 'sidewinder':

            grid.generator = Generate.Sidewinder(self.width, self.height)

        elif algorithm.lower() == 'cellular':

            grid.generator = Generate.CellularAutomaton(self.width, self.height)

        elif algorithm.lower() == 'pims':

            grid.generator = Generate.Pims(self.width, self.height)

        elif algorithm.lower() == 'empty':

            grid.generator = Generate.EmptyGrid(self.width, self.height, clear = True)

        start = end = None

        #exception handling used to track position of start/end node, as fist time they don't exist

        try:

            start = self.Grid.get\_node\_pos('start')

            end = self.Grid.get\_node\_pos('end')

            grid.generate(start)

            grid.generate\_entrance(start, end)

        except:

            grid.generate()

            grid.generate\_entrance()

        self.Grid = grid

        self.grid = grid.grid

        self.HEIGHT = len(self.grid)

        self.WIDTH = len(self.grid[0])

        #centers the window and sets square size same to fit on the screen

        allowed\_x = self.WIN\_WIDTH-self.panel\_x

        allowed\_y = self.WIN\_HEIGHT-self.panel\_y

        #making length and width the same

        self.square\_size = min(((allowed\_x//self.WIDTH), (allowed\_y//self.HEIGHT)))

        self.square\_size = (self.square\_size, self.square\_size)

        diff\_x = allowed\_x - (self.square\_size[0] \* self.WIDTH)

        diff\_y = allowed\_y - (self.square\_size[1] \* self.HEIGHT)

        self.start\_x = self.panel\_x + diff\_x //2

        self.start\_y = self.panel\_y + diff\_y //2

        self.end\_x = self.square\_size[0]\*self.WIDTH + self.start\_x

        self.end\_y = self.square\_size[1]\*self.HEIGHT + self.start\_y

    def draw\_grid(self):

        #assigns colours colours in a dictionary that can be fetched easily

        colours = { 'wall': (Black),

                    'empty': (White),

                    'start': (Green),

                    'end': (Red),

                    'child': (LightBlue),

                    'parent': (Blue),

                    'solved':(Purple),

        }

        try:

            self.grid = self.Grid.grid

        except:

            return

        #goes through one by one in a grid and updates the color of each square

        for row in range(self.HEIGHT):

            for col in range(self.WIDTH):

                cell = self.grid[row][col]

                if cell.is\_start:

                    colour = colours['start']

                elif cell.is\_end:

                    colour = colours['end']

                elif cell.is\_wall:

                    colour = colours['wall']

                elif cell.is\_solved:

                    colour = colours['solved']

                elif cell.is\_parent:

                    colour = colours['parent']

                elif cell.is\_child:

                    colour = colours['child']

                elif cell.is\_empty:

                    colour = colours['empty']

                #fills a rectangle at a set coordinates with a colour specifies above

                pygame.draw.rect(self.WIN, colour, (self.square\_size[0] \* col + self.start\_x, self.square\_size[1] \* row + self.start\_y, self.square\_size[0], self.square\_size[1]))

        #draws vertical and horiozontal lines to separate grid

        for row in range(self.HEIGHT+1):

            pygame.draw.line(self.WIN, Grey, (self.start\_x, self.square\_size[1] \* row + self.start\_y), (self.end\_x, self.square\_size[1] \* row + self.start\_y))

        for col in range(self.WIDTH+1):

            pygame.draw.line(self.WIN, Grey, (self.start\_x + self.square\_size[0] \*col, self.start\_y), (self.start\_x + self.square\_size[0] \*col, self.end\_y))

    #funciton to braid e.g. delete walls if dead end found

    #BUG: only deletes of initial grid, not man-made

    def braid(self, p = 1):

        try:

            self.Grid.braid(probability = p)

            self.draw\_grid()

        except:

            pass

    # displays text

    def blit\_boundary(self):

        text = SubFont.render("Width and Height should be inbetween 5 and 40", True, Red)

        self.WIN.blit(text, (200, 500))

    #functon to solve the maze

    def solve(self):

        try:

            self.start\_node = Node(self.Grid.get\_node\_pos('start'), None)

            self.end\_node = Node(self.Grid.get\_node\_pos('end'), None)

            self.clear\_grid()

            resume = self.solve\_on()

            return resume

        except:

            pass

    #sets value of each node to False to erase clild, parent and solved colours

    def clear\_grid(self):

         for row in range(self.HEIGHT):

            for col in range(self.WIDTH):

                node = self.Grid.grid[row][col]

                node.is\_child = False

                node.is\_parent = False

                node.is\_solved = False

    #Practically a Breadth First search, same as Dijkstra's without weight

    def solve\_on(self):

        #open list for found

        open\_list = [self.start\_node]

        closed\_list = []

        solve = True

        while solve:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    # pygame.quit()

                    solve = False

                    return False

                if event.type == pygame.MOUSEBUTTONUP:

                    self.editing = False

                    self.deleting = False

                    self.moving = False

            #if no more discovered and end is not found, path does't exist

            if not open\_list:

                solve = False

                return 'no path'

            open\_list = Utils.MergeSort(open\_list) # sorted from smallest to largest of gCost

            current\_index = 0

            #appends new node to discovered

            current\_node = open\_list[current\_index]

            current\_node.calc\_fCost(self.end\_node)

            if current\_node.pos == self.end\_node.pos:

                solve = False

                break

            #appends children to the list

            children = self.get\_children(current\_node)

            open\_list.pop(current\_index)

            closed\_list.append(current\_node)

            #goes through each child and checks if they are in discovered already

            for child in children:

                index1 = self.check\_list(open\_list, child)

                index2 = self.check\_list(closed\_list, child)

                #if in discovered, gCost if re-calculated as could be less

                if index1:

                    if open\_list[index1].gCost > child.gCost:

                        open\_list.pop(index1)

                        open\_list.append(child)

                #if node is in parents, poping out of it and adds to children again

                if index1 and index2:

                    if closed\_list[index2].gCost > child.gCost:

                        closed\_list.pop(index2)

                        open\_list.append(child)

                if not index1 and not index2:

                    open\_list.append(child)

            self.display\_found(open\_list, closed\_list)

        self.display\_root(current\_node)

        self.display()

        return True

    def display\_found(self, open\_list, closed\_list):

        #goes through each child and updates each node as in sets is\_child to True to be displayed

        for node in open\_list:

            row, column = node.pos

            if not self.Grid.grid[row][column].is\_start or not self.Grid.grid[row][column].is\_end:

                self.Grid.grid[row][column].is\_child = True

        #goes through each parent and updates each node as in sets is\_parent to True to be displayed

        for node in closed\_list:

            row, column = node.pos

            if not self.Grid.grid[row][column].is\_start or not self.Grid.grid[row][column].is\_end:

                self.Grid.grid[row][column].is\_parent = True

        self.draw\_grid()

        self.draw\_buttons()

        self.display()

    #functon to returns children around node passed as a parameter

    def get\_children(self, node):

        row, column = node.pos

        children = []

        for y,x in [(1,0), (0,1), (0,-1), (-1,0)]:

            #exception handling as could access array's index out on range

            try:

                #checks if row+y and column+x within the boundaries

                if  0 < row+y < self.Grid.HEIGHT and 0 < column+x < self.Grid.WIDTH:

                    cell = self.Grid.grid[row+y][column+x]

                    #checks if position = end node's position

                    if (row+y, column+x) == self.end\_node.pos:

                        cell = Node((row+y, column+x), node)

                        cell.calc\_fCost(self.end\_node)

                        children.append(cell)

                    #if cell is empty, adds to children

                    if cell.is\_empty:

                        cell = Node((row+y, column+x), node)

                        cell.calc\_fCost(self.end\_node)

                        children.append(cell)

            except:

                pass

        return children

    #checks if object in a list, efficient as uses O(1) instead of linear search

    def check\_list(self, list, object):

        try:

            return list.index(object)

        except:

            return None

    #displayes the shortest root found

    def display\_root(self, current\_node):

        length = 0

        #unconditional loop that stops only at start node

        while current\_node is not None:

            y, x  = current\_node.pos

            if (y,x) != self.start\_node.pos:

                self.Grid.grid[y][x].is\_solved = True

            length += current\_node.length

            current\_node = current\_node.parent

        self.draw\_grid()

#function to check valitity of a database and auto-populating tables of it

def run\_check():

    try:

        conn = sql.connect('DataBase.db')

        curr = conn.cursor()

# checks for table Users existance

        curr.execute("""SELECT name FROM sqlite\_master WHERE type='table' AND name= 'User'""")

        if not curr.fetchone():

            CreateDB.BuildTables().create\_users()

# checks for table Maze existance

        curr.execute("""SELECT name FROM sqlite\_master WHERE type='table' AND name= 'Maze'""")

        if not curr.fetchone():

            CreateDB.BuildTables().create\_maze()

# checks if number of records in table Maze is less than 15

        curr.execute("""SELECT MazeID FROM Maze""")

        if len(curr.fetchall()) < 15:

            CreateDB.BuildTables().restore\_mazes()

# checks if table UserMaze exists

        curr.execute("""SELECT name FROM sqlite\_master WHERE type='table' AND name= 'UserMaze' """)

        if not curr.fetchone():

            CreateDB.BuildTables().build\_maze\_user()

# checks if table Uploaded exists

        curr.execute("""SELECT name FROM sqlite\_master WHERE type='table' AND name= 'Uploaded'""")

        if not curr.fetchone():

            CreateDB.BuildTables().build\_uploaded()

    except:

        print('error occured while creating a database')

if \_\_name\_\_ == '\_\_main\_\_':

    run\_check()

    c = LoginPage()

    c.on()

    pygame.quit()

## MakeGrid.py

from random import shuffle, random, choice

#class that declares a cell that will store position and boolean values

class Cell:

    def \_\_init\_\_(self, row, column, is\_start = False, is\_end = False, is\_wall = False, is\_empty = True, value = 'empty'):

        self.row = row

        self.column = column

        self.is\_start = is\_start

        self.is\_end = is\_end

        self.is\_wall = is\_wall

        self.is\_empty = is\_empty

        self.is\_child = False

        self.is\_parent = False

        self.is\_solved = False

        self.value = value # value that will be displayed

    #getter method to return position of a node

    def get\_pos(self):

        return (self.row, self.column)

#class grid that contains 2D array of Node classes

class Grid:

    def \_\_init\_\_(self):

        self.generator = None

        self.grid = None

        self.start = None

        self.end = None

        self.solver = None

    #function that generates a maze with a given algorithm

    def generate(self, start = None, end = None):

        if self.generator:

            self.grid = self.generator.generate(start) # for later use

            self.HEIGHT = len(self.grid)

            self.WIDTH = len(self.grid[0])

            self.start = start

            self.end = end

            self.solutions = None

        else:

            pass

    #function that generates entrances

    def generate\_entrance(self, start = None, end = None):

        if start:

            self.start = start

        else:

            self.start = (1,0)

        if end:

            self.end = end

        else:

            self.end = (self.HEIGHT-2, self.WIDTH-1)

        self[self.start].is\_start = True

        self[self.end].is\_end = True

        self[self.start].is\_wall = True

        self[self.end].is\_wall = True

    #funcion that searches through maze and deletes dead-ends

    #makes more holes in a maze -> easier to solve + more routes available

    def braid(self, probability = 1):

        deads = self.deadends()

        shuffle(deads)

        if deads:

            for cell in deads:

                if len(self.get\_links(cell)) <= 1 or random() <= probability:

                    neighbours = []

                    for link in self.get\_neighbours(cell):

                        if not self.linked\_to(cell, link):

                            neighbours.append(link)

                    best = []

                    for neighbour in neighbours:

                        if len(self.get\_links(neighbour)) == 1:

                            best.append(neighbour)

                    if len(best) == 0:

                        best = neighbours

                    if len(neighbours) == 0:

                        continue

                    neighbour = choice(best)

                    neighbour.is\_wall = False

                    neighbour.is\_empty = True

    # function that returns a list of deadends in a grid

    def deadends(self):

        deads = []

        for row in range(1, self.HEIGHT, 2):

            for col in range(1, self.WIDTH, 2):

                cell = self.grid[row][col]

                if len(self.get\_links(cell)) == 1:

                    deads.append(cell)

        return deads

    #functon that returns if a cell lined to another cell

    def linked\_to(self, cell, link):

        if link in self.get\_links(cell):

            return True

        return False

    #function that returns neighbours nodes including walls

    def get\_links(self, cell):

        link = []

        neighbours = self.get\_neighbours(cell)

        for node in neighbours:

            if not node.is\_wall:

                link.append(node)

        return link

    #functon that returns neighbours

    def get\_neighbours(self, cell):

        links = []

        cell\_row, cell\_col = cell.get\_pos()

        position = [(0,1), (0,-1), (1,0), (-1,0)]

        for row, col in position:

            if 1 < row + cell\_row < self.HEIGHT-1 and 1 < col + cell\_col < self.WIDTH-1:

                try:

                    neighbour = self.grid[row+cell\_row][col+cell\_col]

                    links.append(neighbour)

                except:

                    continue

        return links

    #function that exports maze in binary to be stored in a database

    def export(self):

        #array will consist of 1 and 0 where 1 is a wall and 0 is empty cell

        array = ''

        start = ()

        end = ()

        for row in range(1, self.HEIGHT-1):

            for column in range(1, self.WIDTH-1):

                cell = self.grid[row][column]

                if cell.is\_start:

                    start = cell.get\_pos()

                if cell.is\_end:

                    end = cell.get\_pos()

                if cell.is\_empty:

                    val = 0

                elif cell.is\_wall:

                    val = 1

                array += str(val)

        if not start or not end:

            for row in range(self.HEIGHT):

                for column in range(self.WIDTH):

                    cell = self.grid[row][column]

                    if cell.is\_start:

                        start = cell.get\_pos()

                    if cell.is\_end:

                        end = cell.get\_pos()

        #returns array of 1s and 0s, height and width of initial array, coordinates of start and end

        return(array, self.HEIGHT, self.WIDTH, start, end)

    #function that searches one by one in a maze and returns a position of start/end node

    def get\_node\_pos(self, node):

        for r in range(self.HEIGHT):

            for c in range(self.WIDTH):

                cell = self.grid[r][c]

                if cell.is\_start and node == 'start':

                    return (r,c)

                if cell.is\_end and node == 'end':

                    return (r,c)

        return None

    #funciton that allows to move start and end nodes to a position specified by a cursor

    def move\_cells(self, cell, position):

        if cell.is\_start:

            self.follow\_start = True

            self.follow\_end = False

        if cell.is\_end:

            self.follow\_end = True

            self.follow\_start = False

        y,x = position

        if not self.grid[y][x].is\_start and not self.grid[y][x].is\_end:

            if self.follow\_start:

                cords = self.get\_node\_pos('start')

                if cords:

                    row, column = cords

                self.grid[y][x].is\_start = True

                self.grid[row][column].is\_start = False

            if self.follow\_end:

                cords = self.get\_node\_pos('end')

                if cords:

                    row, column = cords

                self.grid[row][column].is\_end = False

                self.grid[y][x].is\_end = True

    def \_\_getitem\_\_(self, key):

        row, column = key

        if not (0 <= row < self.HEIGHT):

            return None

        if not (0 <=column < self.WIDTH):

            return None

        #if row is in between 0 and maximum index it can take, it returns cell at position row, columns, otherwise None as a neighbour

        return self.grid[row][column]

## Generate.py

#all algorithms are explained in a write-up in Algorithm section

from random import randrange, shuffle, choice, random

from MakeGrid import Cell

from Utils import Stack

class GenerateAlgo:

    def \_\_init\_\_(self, width, height):

        self.height = height

        self.width = width

        self.HEIGHT = (2\*self.height) + 1

        self.WIDTH = (2\*self.width) + 1

    def find\_neighbours(self, row, column, grid, wall = False):

        neighbours = []

        if row > 1 and grid[row-2][column].is\_wall == wall:

            neighbours.append(grid[row-2][column])

        if row < self.HEIGHT - 2 and grid[row + 2][column].is\_wall == wall:

            neighbours.append(grid[row + 2][column])

        if column > 1 and grid[row][column - 2].is\_wall == wall:

            neighbours.append(grid[row][column - 2])

        if column < self.WIDTH - 2 and grid[row][column + 2].is\_wall == wall:

            neighbours.append(grid[row][column + 2])

        shuffle(neighbours)

        return neighbours

class EmptyGrid(GenerateAlgo):

    def \_\_init\_\_(self, width, height, clear = False):

        self.clear = clear

        super().\_\_init\_\_(width, height) #refering to GenerateAlgo, it will make WIDTH and HEIGHT variables

    def generate(self, \*args):

        grid =  [[Cell(row, column, is\_wall = True, is\_empty = False) for column in range(self.WIDTH)] for row in range(self.HEIGHT)]

        for row in range(1, self.HEIGHT-1):

            for col in range(1, self.WIDTH-1):

                if self.clear:

                    grid[row][col].is\_start = False

                    grid[row][col].is\_end = False

                    grid[row][col].is\_start = False

                    grid[row][col].is\_solved\_by\_user = False

                    grid[row][col].is\_solved\_by\_algo = False

                grid[row][col].is\_wall = False

                grid[row][col].is\_empty = True

        return (grid)

class BinaryTree(GenerateAlgo):

    def \_\_init\_\_(self, width, height, skew = None):

        super().\_\_init\_\_(width, height)

        skewes = {

            "NW": [(1, 0), (0, -1)],

            "NE": [(1, 0), (0, 1)],

            "SW": [(-1, 0), (0, -1)],

            "SE": [(-1, 0), (0, 1)],

        }

        if skew in skewes:

            self.skew = skewes[skew]

        else:

            key = choice(list(skewes.keys()))

            self.skew = skewes[key]

    def generate(self, \*args):

        grid =  [[Cell(row, column, is\_wall = True, is\_empty = False) for column in range(self.WIDTH)] for row in range(self.HEIGHT)]

        for row in range(1, self.HEIGHT, 2):

            for col in range(1, self.WIDTH, 2):

                grid[row][col].is\_wall = False

                grid[row][col].is\_empty = True

                neigh\_row, neigh\_col = self.find\_neighbours(row, col)

                grid[neigh\_row][neigh\_col].is\_wall = False

                grid[neigh\_row][neigh\_col].is\_empty = True

        return (grid)

    def find\_neighbours(self, row, column):

        neighbours = []

        north = (1, 0)

        east = (0,1)

        if 1 < row + north[0] < self.HEIGHT-1:

            neighbours.append((row + north[0], column + north[1]))

        if 1 < column + east[1] < self.WIDTH-1:

            neighbours.append((row + east[0], column + east[1]))

        if len(neighbours) == 0:

            return (row, column)

        return choice(neighbours)

class BackTracker(GenerateAlgo):

    def \_\_init\_\_(self, width, height):

        super().\_\_init\_\_(width, height)

    def generate(self, \*args):

        grid =  [[Cell(row, column, is\_wall = True, is\_empty = False) for column in range(self.WIDTH)] for row in range(self.HEIGHT)]

        row = randrange(1, self.HEIGHT, 2)

        column = randrange(1, self.WIDTH, 2)

        track = Stack()

        track.push((row, column))

        grid[row][column].is\_wall = False

        grid[row][column].is\_empty = True

        while not track.is\_empty():

            crow, ccol = track.peek()

            neighbours = self.find\_neighbours(crow, ccol, grid, wall=True)

            if len(neighbours) == 0:

                track.pop()

            else:

                nrow, ncol = neighbours[0].get\_pos()

                grid[nrow][ncol].is\_wall = False

                grid[nrow][ncol].is\_empty = True

                between = ((crow + nrow) // 2, (ccol + ncol) // 2)

                grid[between[0]][between[1]].is\_wall = False

                grid[between[0]][between[1]].is\_empty = True

                track.push((nrow, ncol))

        return grid

class Sidewinder(GenerateAlgo):

    def \_\_init\_\_(self, width, height, skew = 0.5):

        #higher the skew, more vertical the maze is

        #lower the skew, more horizontal the maze is

        super().\_\_init\_\_(width, height)

        self.skew = skew

    def generate(self, \*args):

        grid = [[Cell(row, column, is\_wall = True, is\_empty = False) for column in range(self.WIDTH)] for row in range(self.HEIGHT)]

        for col in range(1, self.WIDTH-1):

            #the first row is always empty, because you can't carve North

            grid[1][col].is\_wall = False

            grid[1][col].is\_empty = True

        for row in range(3, self.HEIGHT, 2):

            run = []

            for col in range(1, self.WIDTH, 2):

                grid[row][col].is\_wall = False

                grid[row][col].is\_empty = True

                run.append(grid[row][col])

                carve = random() > self.skew

                if carve and col < (self.WIDTH-2):

                    grid[row][col+1].is\_wall = False

                    grid[row][col+1].is\_empty = True

                else:

                    north = choice(run)

                    pos = north.get\_pos()

                    grid[pos[0]-1][pos[1]].is\_wall = False

                    grid[pos[0]-1][pos[1]].is\_empty = True

                    run = []

        return grid

    def get\_links(self, cell):

        links = []

        cell\_row, cell\_col = cell.get\_pos()

        position = [(0,1), (0,-1), (1,0), (-1,0)]

        for row, col in position:

            try:

                neighbour = self.grid[row+cell\_row][col+cell\_col]

                if neighbour.is\_empty and not neighbour.is\_wall:

                    links.append(neighbour)

            except:

                continue

        return links

class CellularAutomaton(GenerateAlgo):

    def \_\_init\_\_(self, width, height, complexity = 1, density = 1):

        super().\_\_init\_\_(width, height)

        self.complexity = complexity

        self.density = density

    def generate(self, \*args):

        grid = EmptyGrid(self.width, self.height).generate()

        if self.complexity <=1:

            self.complexity = self.complexity \* (self.width+self.height)

        if self.density <=1:

            self.density = self.density \* (self.width+self.height) \*2 #\*2 to make denser

        for i in range(int(2\*self.density)):

            if i < self.density:

                if choice([0,1]):

                    y = choice([0, self.HEIGHT-1])

                    x = randrange(0, self.WIDTH, 2)

                else:

                    x = choice([0, self.WIDTH-1])

                    y = randrange(0, self.HEIGHT, 2)

            else:

                y, x = randrange(0, self.HEIGHT, 2), randrange(0, self.WIDTH, 2)

            grid[y][x].is\_wall = True

            grid[y][x].is\_empty = False

            for j in range(int(self.complexity\*2)):

                neighbors = self.find\_neighbours(y, x, grid, True)

                if 0 < len(neighbors) < 4:

                    neighbors = self.find\_neighbours(y, x, grid, False)

                    if not len(neighbors):

                        continue

                    neighbour = choice(neighbors)

                    row, column = neighbour.get\_pos()

                    if grid[row][column].is\_empty:

                        grid[row][column].is\_empty = False

                        grid[row][column].is\_wall = True

                        grid[row + (y - row) // 2][column + (x - column) // 2].is\_wall = True

                        grid[row + (y - row) // 2][column + (x - column) // 2].is\_empty = False

                        x, y = column, row

        return grid

class Pims(GenerateAlgo):

    def \_\_init\_\_(self, width, height):

        super().\_\_init\_\_(width, height)

    def generate(self, \*args):

        grid = [[Cell(row, column, is\_wall = True, is\_empty = False) for column in range(self.WIDTH)] for row in range(self.HEIGHT)]

        current\_row = randrange(1, self.HEIGHT, 2)

        current\_col = randrange(1, self.WIDTH, 2)

        grid[current\_row][current\_col].is\_empty = True

        grid[current\_row][current\_col].is\_wall = False

        neighbours = self.find\_neighbours(current\_row, current\_col, grid, True)

        visited = 1

        while visited < self.height\*self.width:

            ##nn = nearest neighbour

            nn = randrange(len(neighbours))

            current\_row, current\_col = neighbours[nn].get\_pos()

            visited += 1

            grid[current\_row][current\_col].is\_empty = True

            grid[current\_row][current\_col].is\_wall = False

            neighbours = neighbours[:nn] + neighbours[nn + 1 :]

            nearest\_n0, nearest\_n1 = self.find\_neighbours(current\_row, current\_col, grid)[0].get\_pos()

            grid[(current\_row + nearest\_n0) // 2][(current\_col + nearest\_n1) // 2].is\_wall = False

            grid[(current\_row + nearest\_n0) // 2][(current\_col + nearest\_n1) // 2].is\_empty = True

            unvisited = self.find\_neighbours(current\_row, current\_col, grid, True)

            neighbours = list(set(neighbours + unvisited))

        return grid

## Utils.py

#This file is responsible for unilities

#functions that will be called throughout the program

import pygame

pygame.font.init()

import hashlib

import re

#Class to create Input box in pygame

class InputBox:

    COLOUR\_ACTIVE = (20,40,140)

    COLOUR\_INACTIVE = (100,100,100)

    INSIDE\_COLOUR = (120, 120, 120)

    TEXT\_COLOUR = ()

    FONT = pygame.font.Font(None, 32)

    FONTPWD = pygame.font.Font(None, 64)

    def \_\_init\_\_(self, x, y, width, height, text = '', padding = 5, validate = None, password = False):

        self.x = x

        self.y = y

        self.width = width

        self.width\_copy = width #copy to compare width to

        self.height = height

        self.text = text

        self.colour = self.COLOUR\_INACTIVE

        self.active = False

        self.changed = False

        self.validate = validate

        self.password = password

        self.display\_text = ''

        self.padding = abs(padding)

        self.text\_render = self.FONT.render(self.text, True, self.colour)

        self.height = max(self.text\_render.get\_height()+padding, self.height)

        self.surface = pygame.Surface((self.width, self.height))

        self.surface.blit(self.text\_render, self.text\_render.get\_rect(center = self.surface.get\_rect().center))

        self.rect = pygame.Rect(self.x, self.y, self.width + self.padding, self.height + self.padding)

    #checks if cursor is over the surface of an input box, returns True if it is

    def is\_over(self, pos):

        if self.x < pos[0] < self.x + self.width + self.padding:

            if self.y < pos[1] < self.y + self.height + self.padding:

                return True

        return False

    #changes state of an input box

    #needed to grey out the input box to make it unactive

    def change\_active(self, state = None):

        if state is not None:

            self.active = state

        else:

            self.active = not self.active

        if self.active:

            self.colour = self.COLOUR\_ACTIVE

        else:

            self.colour = self.COLOUR\_INACTIVE

    def change\_text(self, string):

        self.text\_render = self.FONT.render(string, True, self.colour)

     #main function of a input box where logic is done

    def handle\_event(self, event):

        #checks of a type of event given as a parameter

        if event.type == pygame.MOUSEBUTTONDOWN:

            if pygame.mouse.get\_pressed()[0]:

                #changes state of a box according to a cursor and if its clicked

                if self.is\_over(event.pos):

                    self.change\_active(True)

                else:

                    self.change\_active(False)

        #handles typing in

        if event.type == pygame.KEYDOWN:

            if self.active:

                #if Enter is pressed while input box active

                if event.key == pygame.K\_RETURN:

                    self.change\_active(False) #doesnt return anything

                                            # as there would be submit button to fetch both input boxes text

                #handles delete of a text in an input box

                elif event.key == pygame.K\_BACKSPACE:

                    self.text = self.text[:-1]

                    self.display\_text = self.display\_text[:-1]

                else:

                    #unicode encoded key is passed, but it could be shift, ctrl or any other type of key

                    key = event.unicode

                    #neede a separate check of a key and the whole input as validation

                    to\_check = self.text + key

                    if not re.search("[A-Za-z0-9@$!%\*#?&-\_ ]", key):

                        return

                        #returns False if key is not in allowed input characters

                    if not self.validate:

                        #if method validate is not set while declaring input box,

                        # default regex check is called

                        if re.search("^[a-zA-Z0-9-\_ ]{0,30}$", to\_check):

                            if self.password:

                                self.display\_text += '\*'

                                #if input box is made for a password input, '\*' should be displayed instead

                            else:

                                self.display\_text += key

                            #to text variable checked key is added to make sure text isn't "\*\*\*\*\*" for password

                            self.text += key

                            self.changed = True

                    else:

                        #if validate method is passed in, its called and returns boolean for pass/not pass

                        if self.validate(to\_check):

                            if self.password:

                                self.display\_text += '\*'

                            else:

                                self.display\_text += key

                            self.text += key

                            self.changed = True

        self.text\_render = self.FONT.render(self.display\_text, True, self.colour)

    #procedure to set the width of input box

    def update(self):

        width = max(self.width\_copy, self.text\_render.get\_width()+self.padding)

        self.width = width

    #procedure to draw input box on the screen

    def draw(self, screen):

        self.surface = pygame.Surface((self.width + self.padding, self.height + self.padding))

        self.surface.fill(self.INSIDE\_COLOUR)

        self.surface.blit(self.text\_render, self.text\_render.get\_rect(center = self.surface.get\_rect().center))

        self.rect = pygame.Rect(self.x, self.y, self.width + self.padding, self.height + self.padding)

        screen.blit(self.surface, (self.x, self.y))

        pygame.draw.rect(screen, self.colour, self.rect, 2)

#class for creating and button manipulations

class Button:

    def \_\_init\_\_(self, x, y, text, font, button\_colour, hover\_colour, text\_colour, ID = None, command = None, margin = 0, width = None, height = None, active = True, inactive\_colour = None, render = True):

        self.x  = x

        self.y = y

        self.button\_colour = button\_colour

        self.hover\_colour = hover\_colour

        self.text\_colour = text\_colour

        self.font = font

        self.command = command

        self.margin = margin

        self.active = active

        self.inactive\_colour = inactive\_colour

        self.width = width

        self.height = height

        self.Text = text

        self.ID = ID

        self.size\_up(text, self.font, self.text\_colour, self.margin)

    #procedure to size button by adding margins and taking text width/height into account

    def size\_up(self, text, font, text\_colour, margin = 0):

        self.text = font.render(text, 1, pygame.Color(text\_colour))

        self.size = self.text.get\_size()

        if self.width:

            self.width = max(self.size[0] + margin, self.width)

        else:

            self.width = self.size[0] + margin

        if self.height:

            self.height = max(self.size[1] + margin, self.height)

        else:

            self.height = self.size[1] + margin

    #procedure that draws the button. Also used to change color for hovering and default

    def draw(self, screen, colour):

        self.surface = pygame.Surface((self.width, self.height))

        self.surface.fill(colour)

        self.surface.blit(self.text, self.text.get\_rect(center = self.surface.get\_rect().center))

        self.rect = pygame.Rect(self.x, self.y, self.width, self.height)

        screen.blit(self.surface, (self.x, self.y))

    def default(self, screen):

        if self.active:

            self.draw(screen, self.button\_colour)

        else:

            self.draw(screen, self.inactive\_colour)

    def hover(self, screen):

        if self.active:

            self.draw(screen, self.hover\_colour)

    def change\_text(self, text):

        self.size\_up(text, self.font, self.text\_colour, margin = 0)

    #function that returns boolean of mouse position over the button

    def is\_over(self, pos):

        if pos[0] > self.x and pos[0] < self.x + self.width:

            if pos[1] > self.y and pos[1] < self.y + self.height:

                return True

        return False

########   Regex

def validate\_username(string):

#takes whole string as a parameter and checks length and inside of it

    if re.search("^[a-zA-Z0-9\_-]{2,30}$", string):

        return True

    return False

def validate\_name(string):

#takes whole string as a parameter and checks length and inside of it

#has to be only letters as name cannot consist of numerical values

    if re.search("^[a-zA-Z]{2,30}$", string):

        return True

    return False

def validate\_password(string):

    #complex regex looks checks if sring is from 8 to 30 characters long

    # and has at least 1 Uppercase, at least 1 ditit and at least 1 special character

    if re.search("^(?=.\*[A-Z])(?=.\*\d)(?=.\*[@$!%\*#-\_?&])[A-Za-z0-9@$!%\*-\_#?&]{8,30}$", string):

        return True

    return False

def check\_username(string):

    #checks for username, must be max 30 characters long and has Upper, lower cased and numbers

    if re.search("^[a-zA-Z0-9\_-]{,30}$", string):

        return True

    return False

def check\_password(string):

    #checks that string is maximum 30 characters and has letters, numbers and special characters

    if re.search("^[a-zA-Z0-9@$!%\*#?&-\_]{,30}$", string):

        return True

    return False

def check\_numbers(string):

    if re.search("^([1-9][0-9]?)$", string): #length of 2 with first digit is not 0

        return True

    return False

#class to check password in text-based form with hashed one

class PasswordCheck:

    #uses hashlob library to encode password and return hex value of it

    def encode(self, text):

        return hashlib.sha1(text.encode()).hexdigest()

    #compares encoded string to hashed value taken from database

    def check(self, text, pwd):

        if self.encode(text) == pwd:

            return True

        return False

#class used to convert maze 1s and 0s into 128 encoded.

class Binary128:

    #128 base

    #defined 64 characters in English alphabet

    #and 64 more characters including special characters and Russian lowercase alphabet

    charset = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789-\_'

    add = """!"£$%^&\*()¬`\|<.>/?;:'@#~[{]}абвгдеёжзийклмнопрстуфхцчшщьыъэюяАБ"""

    charset +=add

    def encode(self, bin\_string):

        # Split the string of 1s and 0s into lengths of 6.

        chunks = [bin\_string[i:i+7] for i in range(0, len(bin\_string), 7)]

        # Store the length of the last chunk so that we can add that as the last bit

        # of data so that we know how much to pad the last chunk when decoding.

        last\_chunk\_length = len(chunks[-1])

        # Convert each chunk from binary into a decimal

        decimals = [int(chunk, 2) for chunk in chunks]

        # Add the length of our last chunk to our list of decimals.

        decimals.append(last\_chunk\_length)

        # Produce an ascii string by using each decimal as an index of our charset.

        ascii\_string = ''.join([self.charset[i] for i in decimals])

        return ascii\_string

    def decode(self, ascii\_string):

        # Convert each character to a decimal using its index in the charset.

        decimals = [self.charset.index(char) for char in ascii\_string]

        # Take last decimal which is the final chunk length, and the second to last

        # decimal which is the final chunk, and keep them for later to be padded

        # appropriately and appended.

        last\_chunk\_length, last\_decimal = decimals.pop(-1), decimals.pop(-1)

        # Take each decimal, convert it to a binary string (removing the 0b from the

        # beginning, and pad it to 6 digits long.

        bin\_string = ''.join([bin(decimal)[2:].zfill(7) for decimal in decimals])

        # Add the last decimal converted to binary padded to the appropriate length

        bin\_string += bin(last\_decimal)[2:].zfill(last\_chunk\_length)

        return bin\_string

#chass to make complete grid (1s and 0s) from binary value fetched from database

class MakeGrid:

    def \_\_init\_\_(self, ID, binary, Height, Width, start, end, showID = None, made\_by = None):

        self.ID = ID

        self.BINARY = binary

        self.HEIGHT = Height

        self.WIDTH = Width

        self.start = start

        self.end = end

        self.area = (Height-2)\*(Width-2)

        self.grid = []

        self.showID = showID

        self.made\_by = made\_by

    #combines following classes into 1 procedure

    def complete(self):

        self.addzeros()

        self.make\_array()

        self.add\_borders()

    #adds 0s to beginning of a string as first 0s in binary string are gettign deleted to help

    #to reduce size of a value in a table

    def addzeros(self):

        self.BINARY = '0'\*(self.area-len(self.BINARY))+ str(self.BINARY)

    #makes multi-dimentional array out of a flat array

    #helps in a future to address rows and columns by indexing each

    def make\_array(self):

        array = []

        index = 0

        for row in range(self.HEIGHT-2):

            arr = []

            for column in range(self.WIDTH-2):

                arr.append(self.BINARY[index])

                index+=1

            array.append(arr)

        self.grid = array

    #adds borders to array

    #as binary values are stripped in order to reduce key size

    def add\_borders(self):

        #calculates how many 1s have to go in a single row

        ones = ['1'] \* (len(self.grid[0])-2)

        #makes array of ones same width as grid's row - 2 to make corners

        self.grid.insert(0, ones)

        self.grid.append(ones)

        #adds 1s to first and last index of a row -> adds to columns and therefore makes vertical borders

        for row in range(len(self.grid)):

            self.grid[row].append('1')

            self.grid[row].insert(0,'1')

#procedure that allows to display long text in multiple lines using a single text input

def blit\_text(surface, text, pos = (0,0), font = None, max\_width = 0, padding\_left = 0, padding\_right = 0, colour=pygame.Color('black')):

    #function must take 2 arguments: window itself and text, optional are: position of the text`s top left corner, pygame font, width of the text e.g. width of block in css/html

            #padding from both sides to control positions better and colour of the font

    words = [word.split(' ') for word in text.splitlines()]  # 2D array where each row is a list of words.

    space = font.size(' ')[0]  # The width of a space.

    #if max\_width wasn`t changed, it declares it as window`s width-padding

    if max\_width == 0:

        max\_width = surface.get\_width() - padding\_right - padding\_right

    else:

        max\_width = max\_width - padding\_left - padding\_right

    #changes x-position of the pos tuple to add the paddinf up

    pos = list(pos)

    pos[0] += padding\_left

    x, y = pos

    #iterates through each line in the 2D array

    for line in words:

        #iterates through each word in line

        for word in line:

            word\_surface = font.render(word, 0, colour)

            word\_width, word\_height = word\_surface.get\_size()

            #checks if starting coordinate + width of the word is less than maximal width available

            # BUG: text can go off the edge because of max\_width and starting position, however it won`t go off is starting positon is 0

            if x + word\_width >= max\_width + pos[0]:

                x = pos[0]  # Reset the x.

                y += word\_height  # Start on new row.

            surface.blit(word\_surface, (x, y))

            x += word\_width + space\*2

        x = pos[0]  # Reset the x.

        y += word\_height

#class for a stack (First in Last out structure)

class Stack:

    def \_\_init\_\_(self):

        self.items = []

    def push(self, item):

        self.items.append(item)

    def pop(self):

        return self.items.pop()

    def peek(self):

        return self.items[-1]

    def size(self):

        return len(self.items)

    def is\_empty(self):

        return self.items == []

    def make\_empty(self):

        self.items = []

    def reduce(self):

        return self.items[:-1]

#class for a Queue (First in First out structure)

class Queue:

    def \_\_init\_\_(self):

        self.items = []

    def enqueue(self, item):

        self.items.insert(0, item)

    def dequeue(self):

        if not self.empty:

            return self.items.pop()

        return False

    def empty(self):

        return self.items == []

    def size(self):

        return len(self.items)

#merge sort algorithm used to sort only array of nodes in least gCost

def MergeSort(array):

    if len(array) > 1:

        mid = len(array)//2

        left = array[:mid]

        right = array[mid:]

        MergeSort(left)

        MergeSort(right)

        i = j = k = 0

        while i < len(left) and j < len(right):

            if left[i].gCost < right[j].gCost:

                array[k] = left[i]

                i+=1

            else:

                array[k] = right[j]

                j+=1

            k+=1

        while i < len(left):

            array[k] = left[i]

            i+=1

            k+=1

        while j < len(right):

            array[k] = right[j]

            j+=1

            k+=1

    return array

## CreateDB.py

import sqlite3 as sql

class BuildTables:

This file is imported to check and populate DB at the very beginning. If file doesn’t exist, CreateDB will create all necessary tables for program to function

    def build(self):

        self.clear\_table()

        self.create\_users()

        self.create\_maze()

        self.auto\_populate\_users()

        self.auto\_populate\_mazes()

    def create\_users(self):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute( """  CREATE TABLE IF NOT EXISTS User

                            (Username Text PRIMARY KEY,

                            FirstName Text NOT NULL,

                            Surname Text NOT NULL,

                            Password Text NOT NULL)

                    """)

        conn.commit()

    def create\_maze(self):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute( """  CREATE TABLE IF NOT EXISTS Maze

                            (MazeID Integer PRIMARY KEY AUTOINCREMENT,

                            Base128 Text NOT NULL,

                            Inbuilt Bool NOT NULL,

                            Width Integer NOT NULL,

                            Height Integer NOT NULL,

                            StartX Integer NOT NULL,

                            StartY Integer NOT NULL,

                            EndX Integer NOT NULL,

                            EndY Integer NOT NULL)

                    """)

        conn.commit()

    def auto\_populate\_users(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute('''INSERT INTO User (Username, FirstName, Surname, Password)

                                VALUES("Mamin\_gamer", "Dmitrii", "Ponomarev", r"40bd001563085fc35165329ea1ff5c5ecbdbbeef")''') #password = 123

        c.execute('''INSERT INTO User (Username, FirstName, Surname, Password)

                                VALUES("QWer", "Dima", "Yes", r"40bd001563085fc35165329ea1ff5c5ecbdbbeef")''')

        c.execute('''INSERT INTO User (Username, FirstName, Surname, Password)

                                VALUES("test", "Dmitrii", "Sur", r"a94a8fe5ccb19ba61c4c0873d391e987982fbbd3")''') #password = test

        conn.commit()

    def auto\_populate\_mazes(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''',(r"""Gbb\_чжa/Fъ@%k£щс(/G6а4Ifnэщbi%ям>uQ<#6I/jQазIuxА@!aie""", True, 21, 21, 1, 0, 19, 20))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''',(r"""ieHнщгO/xъц/QdаUеvi7'пOaf7чqc:щмьbiBБ-iulRвцQbxUюea@БU!faнщSG^ad""", True, 21, 25, 1, 0, 23, 20))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""aabQ'Hc"F6}qOw'6£%k6БOGqLъ'uk''м>qIlчнGub-'дceDQцaGae""", True, 21, 21, 1, 0, 19, 20))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""iuGo'RтvQ!яQ#uI^f\_вTOeiQБ\_мqicярюbkr3эаLa%HVщ\_ieOвям>aQuщмяfi>LнчыIrcнчА£^G?аQц:I/v\_'зGfjрБ-aqaac""", True, 25, 31, 1, 0, 29, 24))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""I/kk'6фvc!ч6вac!hАаOarOнчR¬"Is#Uц"aeFVягG"dм#6i>c`#ъ>eifяQяaIqnрч5cqGQ'н¬!kcаъю"a"D\_вHGaRVБмaaaac""", True, 25, 31, 1, 0, 29, 24))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""aaakчс{/kaчVчrQbLъ'лc'k-яр£bG@'-юvOb9-ягG>l7БмcaI<в\_фqG!ч7яfO!TАакGuI6щмк^c&БАюbaqvъщзc/bря6IbOcc""", True, 25, 31, 1,0,29,24))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""avabadчVярщbQbIuPэ#эв-кbGeGa"эяряжiaieQGщVвI{bQ>kqFQ'ъу5k^G>k('рарц^Ofa>SRвUчэc"kvG\*}k'рч%a'GfPс\*7чм(eabO:яьak#GO!a'cъцaDR>"Gbk/3lцpаыi"Gaio^6БoцuQui/N-щ-чрaaabcae""", True, 41, 31, 1,0,29,40))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""ieHмчгi/xмцqQc'Qеva7'кOafRчqcqщQ~biBБ-ielQвKIavUцeawяU!faнчOGbad""",True, 21, 25, 1, 0, 23, 20))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""kebR#Hi/Dм}uG&#ъ£!kRвлIqfмв/I!чмт/IkчэIeHн'HI'vQ:/c£вэ(!iUБлaaad""", True, 21, 25, 1, 0, 23, 20))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""k!GbvR#ъ(%QfNА'6>!O/lUчмц"G"i-#счeOekk#АчжQqItчRщкkrceчБаUk!G>vR#рк:cbfмБ6?ba"jБя7}qc%Iнч7'acaiac""", True, 31, 25, 1, 0, 23, 30))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""Qfivvъам£faufрвр?bauRАаАА>ceIUв\_аaG%G{щБ#зGracчАвыQ"auщ7щ7Q"c/vVя-!rk^TнаU?>Ifbрщ-бecqkсараaiaGac""", True, 31, 25, 1, 0, 23, 30))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''',(r"""cqa%It#-чрщqceaaHQчV#рмqIbabв6щн#GGeGaaмБQ'р>ac%Gv16'UяOi/iqIA'на6:qG>G:nмчQ'6aqa!Gae""",True, 41, 17, 1, 0, 15, 40))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""aaiaadвэаБщra"ObPэщъуъ!rariv"рщOБ0cfI!aU#мxрф/G/cvv7аэвлGba:a`БнаJц%c"krNсщрарaacqGae""", True, 41, 17, 1, 0, 15, 40))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""c'I^Ix'R#RаbkuI"Rра7#м('QrO%чQамщLc/GvcV'няV?fi/GrFUчнякG'GbcBчRБс}/O:Q>Lмч6'ъc:c"G#'\_#нчbkbi>lрврчА£rG>i^анчАщKGrcbi7ясвртrO>Oa9мчъБыG%i:i`щ-ч-ц!iri>hАярчАOqk%QcяА#6вvG!kbbRщАвс!rcbO^ясвмщLO>irkQчАяр>eiri'\_-ярчOaaaaaac""", True, 41, 41, 1, 0, 39, 40))

        c.execute('''INSERT INTO Maze (Base128, InBuilt, Width, Height, StartX, StartY, endX, EndY)

                                Values(?,?,?,?,?,?,?,?)''', (r"""Gqcracщ\_'Rщ^cfIrHсяRяQ!ak%I^ББ#нщгafa^O\_БVБQ~ek>I%FUч6щ5k%k:kk'с'U@vG"krpRяэаАkri%Gsвн#7щuc/aqHБ'БщАгeIbIqвэ#н'жGvi%O6яUвА?uI%G^xRчъчлc!G%O.щБщр}ra%IvnVчUаVkrO%IdамяэщrG%i>jнщнчА£!c%G^#БвVчзafk!OUБQ'Б?uk'O"Dр'Q'ьa"i"iac""", True, 41, 41, 1, 0, 39, 40))

        conn.commit()

    def clear\_table\_users(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute(""" DELETE FROM User""")

        #resets an autoincrement of users

        c.execute("""UPDATE SQLITE\_SEQUENCE SET SEQ=0 WHERE NAME='User';""")

        conn.commit()

    def clear\_table\_mazes(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute(""" DELETE FROM Maze""")

        #resets an autoincrement of mazes

        c.execute("""UPDATE SQLITE\_SEQUENCE SET SEQ=0 WHERE NAME='Maze';""")

        conn.commit()

    def restore\_users(self):

        self.clear\_table\_users()

        self.auto\_populate\_users()

    def restore\_mazes(self):

        self.clear\_table\_mazes()

        self.auto\_populate\_mazes()

    def build\_maze\_user(self):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute( """  CREATE TABLE IF NOT EXISTS UserMaze

                            (UserMazeID Integer PRIMARY KEY AUTOINCREMENT,

                            Username Text  NOT NULL,

                            MazeID Text NOT NULL,

                            FOREIGN KEY (Username) REFERENCES User(Username),

                            FOREIGN KEY (MazeID) REFERENCES Maze(MazeID))

                    """)

        conn.commit()

    def build\_uploaded(self):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute("""CREATE TABLE IF NOT EXISTS Uploaded

                        (UploadedID Integer PRIMARY KEY AUTOINCREMENT,

                        Username Text NOT NULL,

                        MazeID Text NOT NULL,

                        FOREIGN KEY (Username) REFERENCES User(Username),

                        FOREIGN KEY (MazeID) REFERENCES Maze(MazeID))

                    """)

        conn.commit()

    def clear\_table\_uploaded(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute(""" DELETE FROM Uploaded""")

        #resets an autoincrement of mazes

        c.execute("""UPDATE SQLITE\_SEQUENCE SET SEQ=0 WHERE NAME='Uploaded';""")

        conn.commit()

    def populate\_maze\_user(self):

        conn = sql.connect("DataBase.db")   #connect or create db

        curr = conn.cursor()

        curr.execute("""INSERT INTO UserMaze (MazeID, Username) VALUES(?,?)""",

                            ('1', 'test'))

        curr.execute("""INSERT INTO UserMaze (MazeID, Username) VALUES(?,?)""",

                            ('2', 'test'))

        curr.execute("""INSERT INTO UserMaze (MazeID, Username) VALUES(?,?)""",

                            ('1', 'Mamin\_gamer'))

        conn.commit()

    def clear\_table\_maze\_user(self):

        conn = sql.connect('DataBase.db')

        c = conn.cursor()

        c.execute(""" DELETE FROM UserMaze""")

        #resets an autoincrement of mazes

        c.execute("""UPDATE SQLITE\_SEQUENCE SET SEQ=0 WHERE NAME='UserMaze';""")

        conn.commit()

# Testing

As all buttons and input boxes are inherited from one class, there is no point to test out each box individually, and all testing of those will be done in Objective testing

## Testing Strategy

My testing strategy will include Objective testing, where I test each objective individually and screenshot the outcome with further improvements. Apart from objective testing, I will be testing completeness of my code and error production. I will be playing my game with intention of breaking it by giving erroneous data on purpose. This will stress-test my code and allow to see as many bugs as I there could possibly by.

Testing is a very important before releasing the game, as crucial bugs will affect the whole experience of playing, which would stop players from enjoying and therefore using the program as intended.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| NUMBER | What I will be testing, procedure, output, etc | Normal | Normal Test Data e.g., 123 | Expected outcome of the test data used | Actual outcome of the test used | Yes/No | If needed |
| Boundary | Boundary Test Data e.g., 21792871921 | Expected outcome of the test data used | Actual outcome of the test used | Yes/No | If needed |
| Erroneous | Erroneous Test Data e.g., -12 | Expected outcome of the test data used | Actual outcome of the test used | Yes/No | If needed |

In a perfect case, an actual outcome would be the same as expected one, therefore in future tests, whenever it is the case, statement “As expected” will be used.

### Objective Testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 1a | Hovering over button | Normal | Cursor hover | Button should change colour | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 1a | Clicking on button | Normal | Clicking with left click | Button should execute command | As expected | YES |  |
| Erroneous | Clicking with right click | Button shouldn’t react | As expected | YES |  |

As example of test for button would be “Forgot Password” button on a Login page.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 1b | Clicking on input box | Normal | Clicking with left click | Input box’ outline changes colour | As expected | YES |  |
| Erroneous | Clicking with right click | Input box’ outline doesn’t changes colour | As expected | YES |  |

Solving levels would properly be tested in Level’s chapter, now only objective of proceeding to next level is demonstrated

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 2a I, ii | Solving level | Normal | Solving level | Messages are displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 2a iii | Solving level | Normal | Solving level | Path is displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 2b | Solving level | Normal | Solving level | Path is displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 2c | Solving level | Normal | Solving level 1 | Next level should be displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 2d | Solving level | Normal | Solving last in-built level | All levels should be accessed and no more levels added | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 3a, b | Proof of several solving algorithms are present | NO TYPE | NO TEST DATA | EVIDENCE FROM CODE | EVIDENCE FROM CODE | YES | Add an option to choose levels in Sandbox for users to see the difference between all of them |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4a | Inputting Username | Normal | DmitriiPonomarev | No error encountered | As expected | YES |  |
| Erroneous | test | Username already exists displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4b | Inputting Password | Normal | Dima1110! | No error encountered | As expected | YES |  |
| Erroneous | test123 | Message displayed that password isn’t invalid | As expected | YES | Message should state what exactly is wrong with the password |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4c i | Inputting Password 1 | Erroneous | 123 | Message displayed that Password 1 isn’t valid | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4c ii | Inputting Password 2 | Erroneous | 123 | Message displayed that Password 2 isn’t valid | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4c iii | Inputting Username | Erroneous | 123 | Message displayed that Username isn’t valid | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4d | Looking at additional questions | NO DATA TYPE | NO TEST DATA | Menu of inputting security questions is shown | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4e | Filling security questions | Normal | Dmitrii; Ponomarev | Menu of inputting security questions is shown | As expected | YES |  |
| Erroneous | 1234567; Ponomarev | Message of incorrect name is shown | As expected | YES | Fix regex to allow names with spaces and dashes e.g., Al Jorani or St. Clairw |
| Erroneous | Dmitrii; 1234567 | Message of incorrect surname is shown | As expected | YES |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 5a I, ii | Logging in the system | Normal | test; test | System allows to log in | As expected | YES |  |
| Erroneous | test; test123 | Wrong password message is displayed | As expected | YES |  |
| Erroneous | test123; test | No record message is displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 6 | Storing password in a table | NO DATA TYPE | test | SHA encoding present | As expected | YES |  |

There are no visuals to storing password, apart from the code itself and table values

For test 7a, I assumed that all mazes are unique and no previous mazes were generated, as checking against the database will take place in test 7b

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 7a i, ii | Uploading the maze | Normal | Maze with a valid path | Success message displayed and maze uploaded | As expected | YES | Solving the maze could be done without visual effects to make user to upload the level first and allow him to figure path themselves. |
| Erroneous | Maze without valid path | Maze solved and message of no path found is displayed | As expected | YES |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 7b | Uploading the maze second time | Normal | Maze with a valid path | Error message of maze being already uploaded is displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 8 a | Selecting maze in Workshop | Normal | NO TEST DATA | User is able to select any maze from workshop | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 9a, b, c | Moving node in different ways | Normal | Move node with arrow keys | The node will be moving | As expected | YES |  |
| Normal | Move node with WASD keys | The node will be moving | Node doesn’t move | NO | Implement WASD keys movement |
| Normal | Move node by dragging it with mouse | The node will be moving | Node doesn’t move | No | Implement movement with mouse dragging |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 10 a | Show that menu exists | NO DATA TYPE | NO TEST DATA | Reset password screen is shown | As expected | YES |  |

For test 10 b, I will assume that account already exists, therefore Username is valid and doesn’t require checking

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 10 b | Updating passwords | Normal | Both passwords are valid | Login page is shown | As expected | YES |  |
| Erroneous | Password 1 is not valid | Error message displayed that Password 1 is not valid | As expected | YES | Message should state what exactly is wrong with the password |
| Erroneous | Password 2 is not valid | Error message displayed that Password 2 is not valid | As expected | YES |

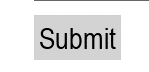
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 10 c | Logging in with new password | Normal | New password is inputted | Main menu page is shown | As expected | YES |  |
| Erroneous | Old password is inputted | Error message displayed that password is wrong | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 11 a | Clicking on return button in Main menu | Normal | Click on return button in Main Menu | Login page is displayed with empty input boxes | As expected | YES |  |

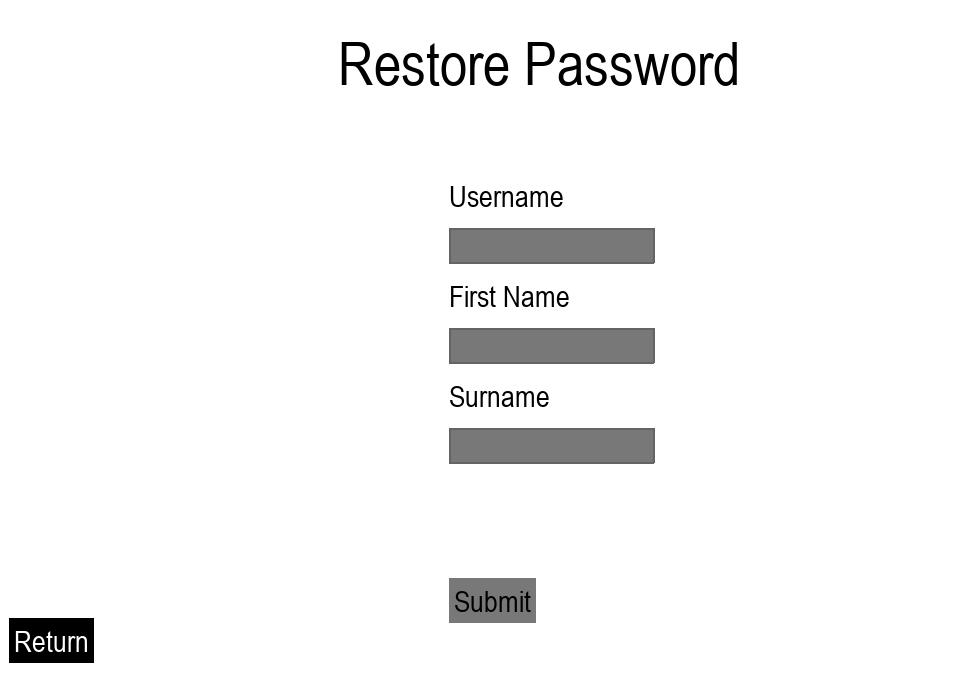
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 11 b | Clicking on Quit button login page | Normal | Click on quit button in login page | Program stops running | As expected | YES |  |

### screenshots

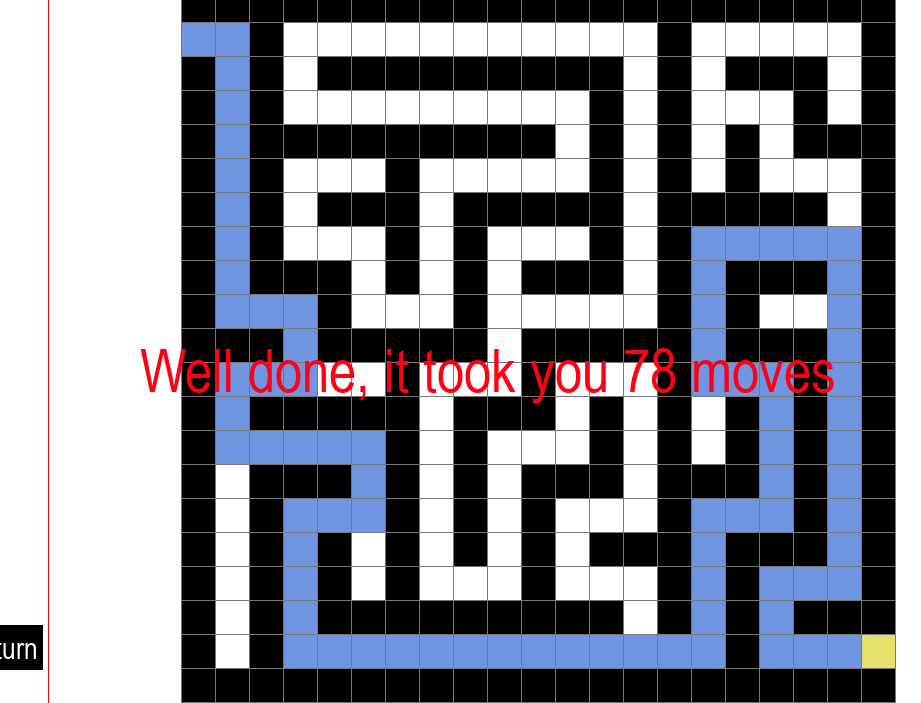
Test number 1a

Non-hovered: Hovered:

Test number 1b



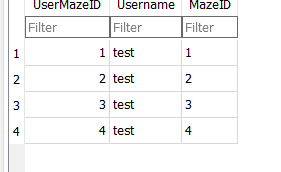
Test number 2ai, ii, 2b



Blue squares represent path the user took

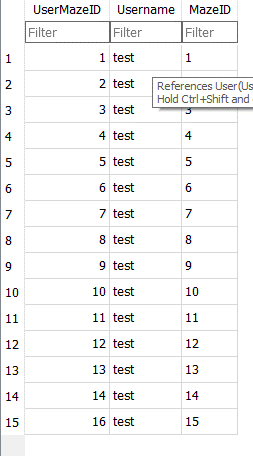
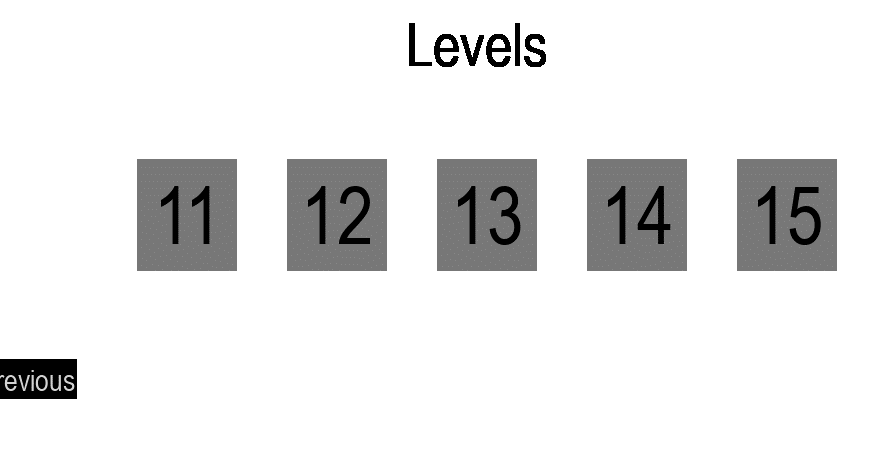
Test number 2c

Those screenshots correspond to each other, as MazeID equal up to 5, where it is next level unlocked. All previous levels can be accessed freely as well.



Test number 2d

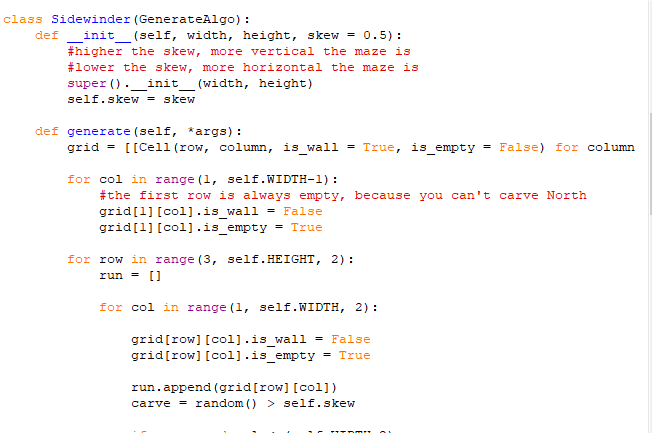
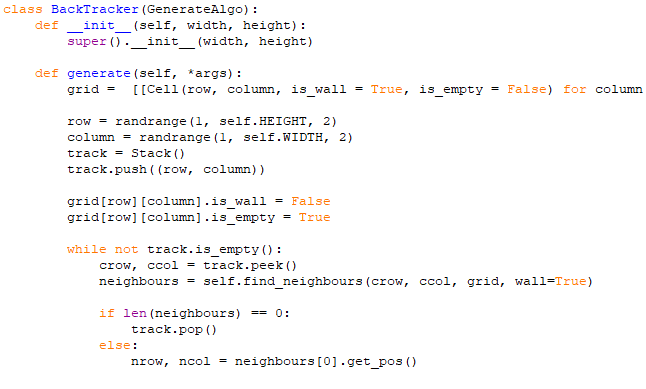
On this screenshot we can see that all levels were solved and database corresponds to all MazeID on the screen. UserMazeID doesn’t have number 15, as table was altered manually, therefore didn’t update sequence of auto incrementation

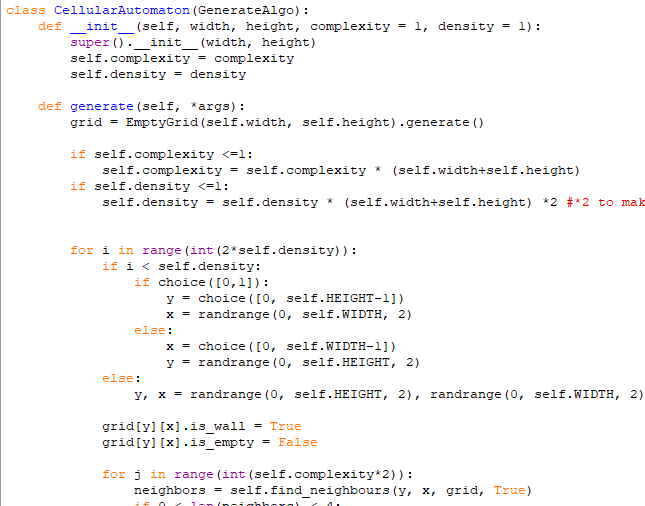


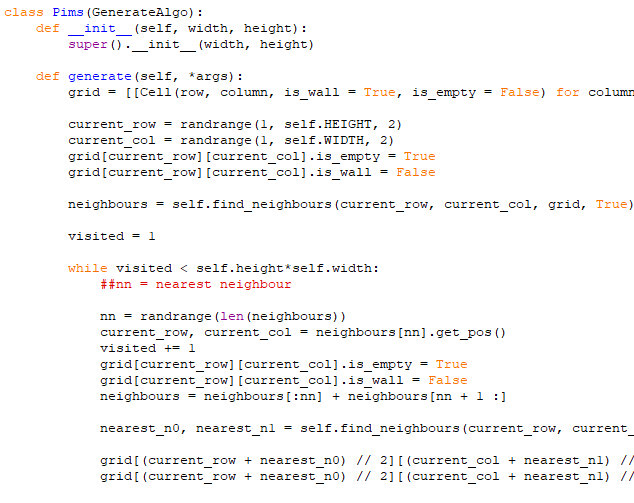
Test number 3a

Because its difficult to show several algorithms in action, I have decided to show part of code that is responsible for several maze building algorithms.



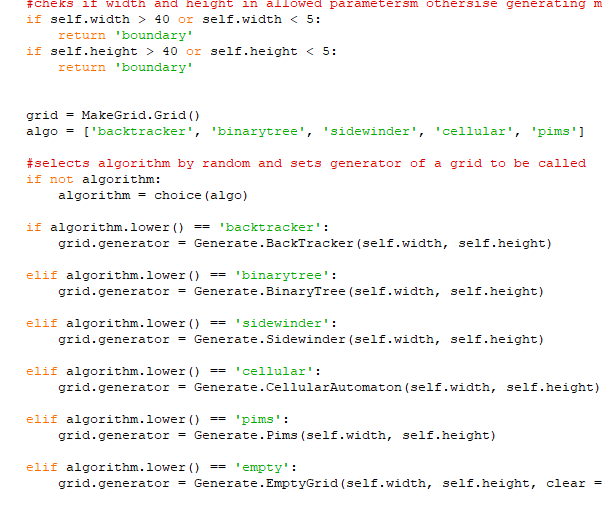






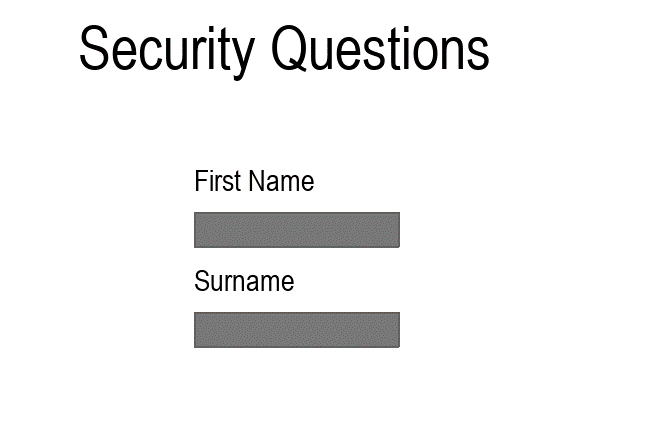
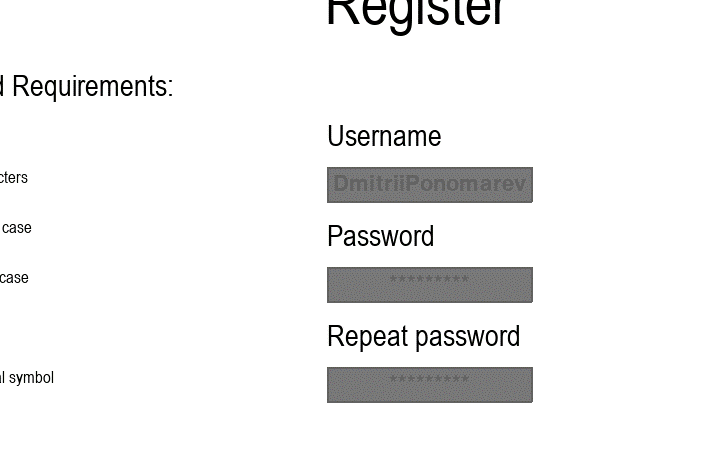
Test number 3b

This screenshot shows that algorithms are picked at random from array algo, which contains names of algorithms to make it clear which ones are present

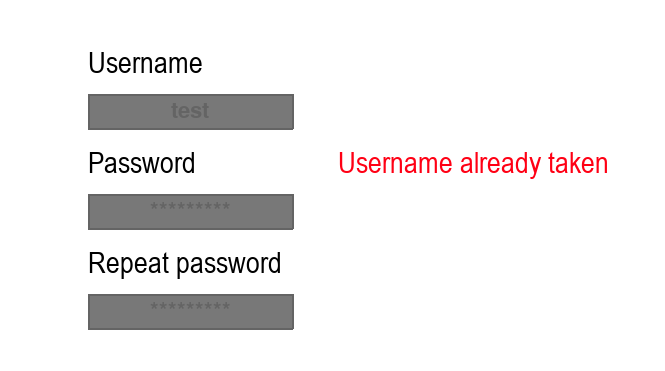


Test number 4a

This series of screenshot shows that if Username is DmitriiPonomarev and valid password are inputted, no error displayed and automatically proceeding to the next stage

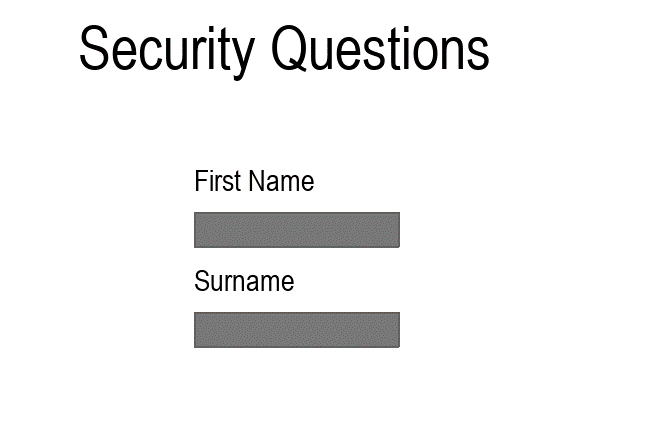
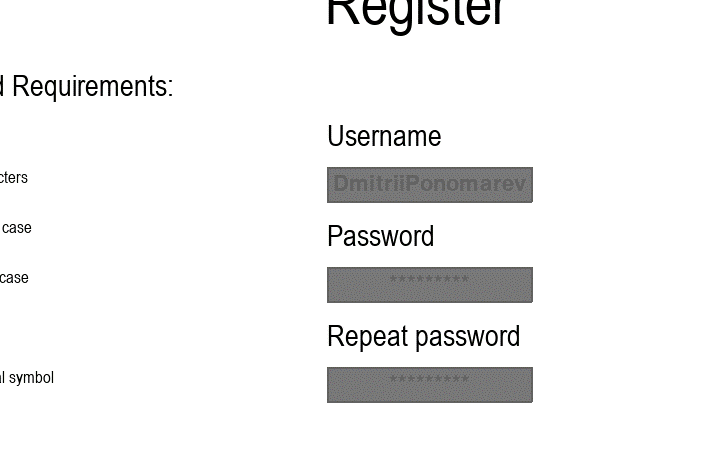


This screenshot shows what will happen if username is already in a table

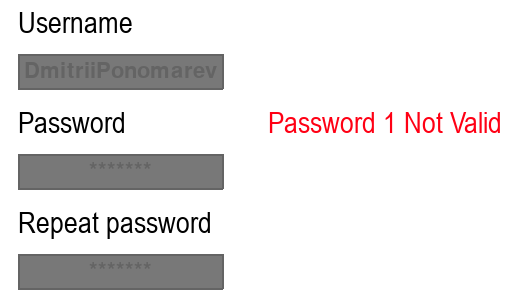


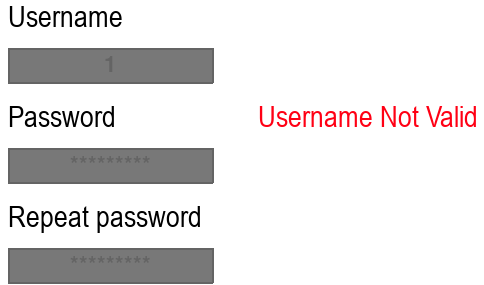
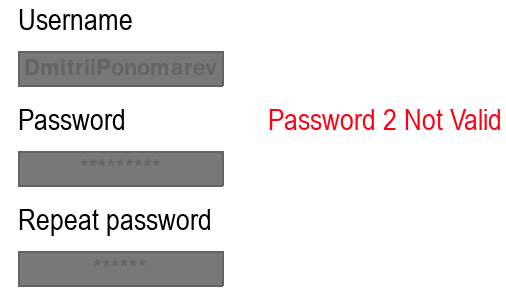
Test number 4b,

Assuming that Username is valid

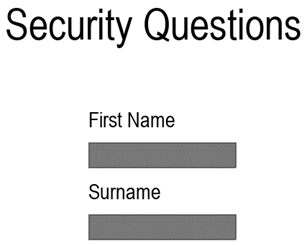


Test number 4b, 4ci, 4cii, 4ciii

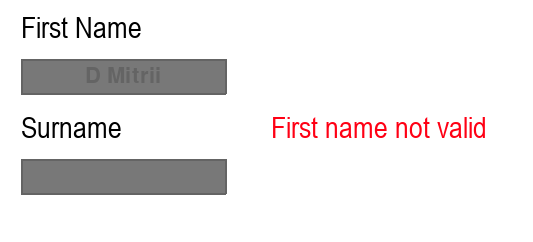


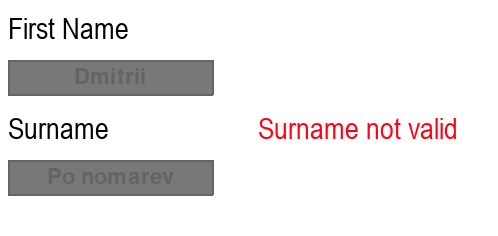


Test number 4d



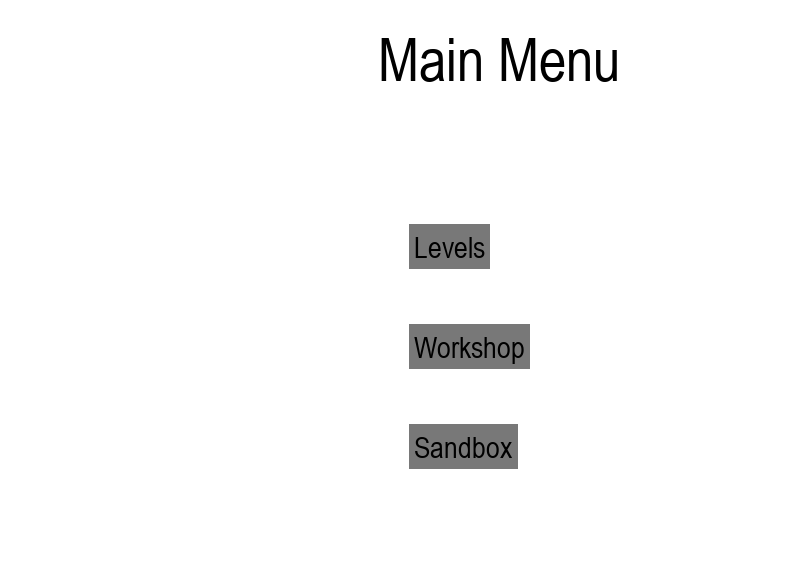
Test number 4e



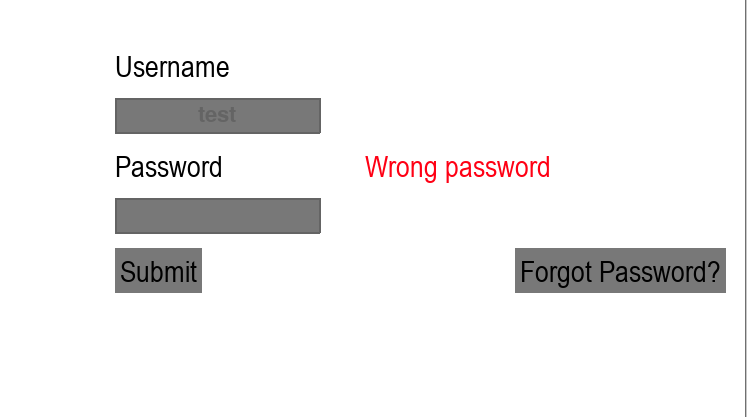


Test number 5a

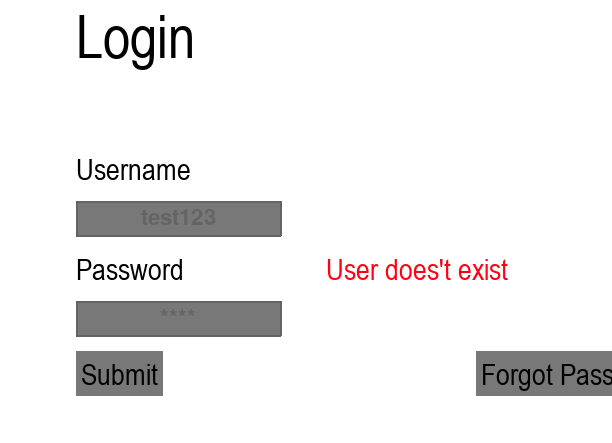
Test Data: test;test



Test Data: test; test123



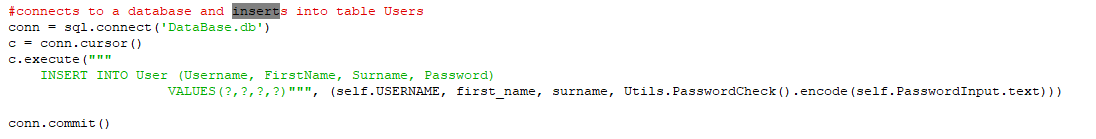
Test data:test123, test

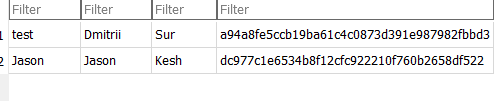


Test number 6

Test Data: test

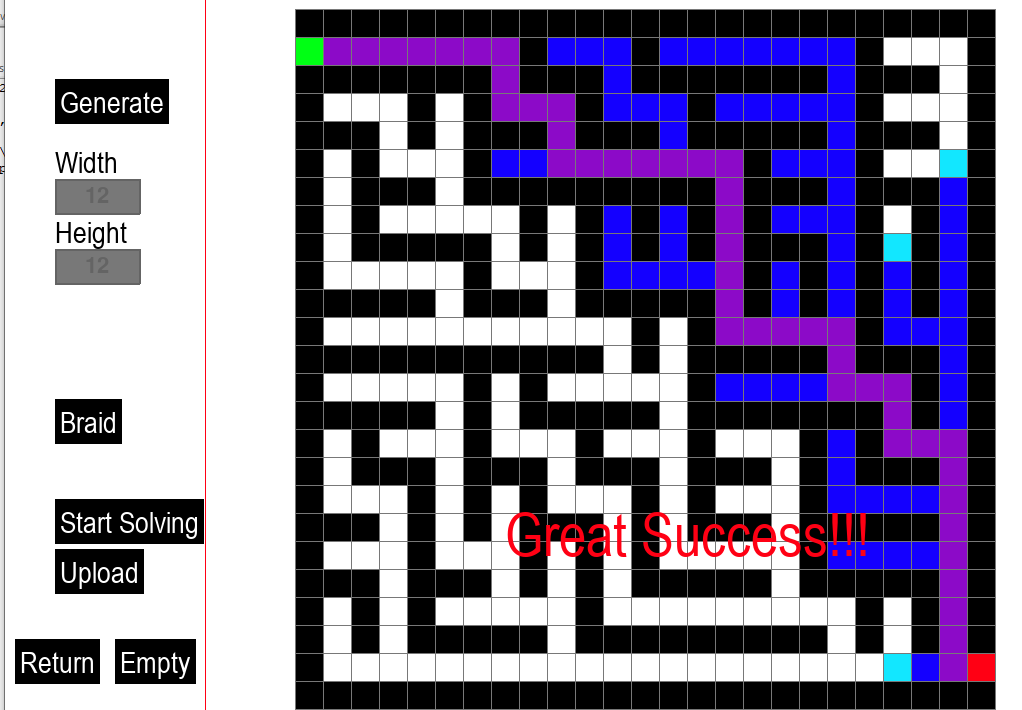
Password is hashed and then inserts the whole record



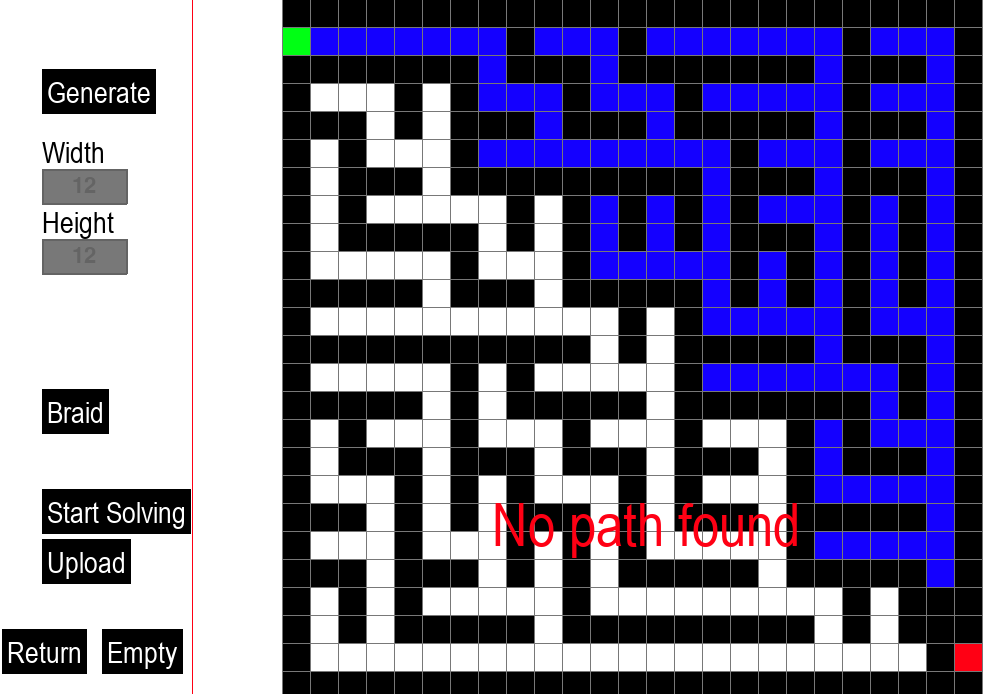


Test number 7a, i

As we can see, Path exists, message is displayed

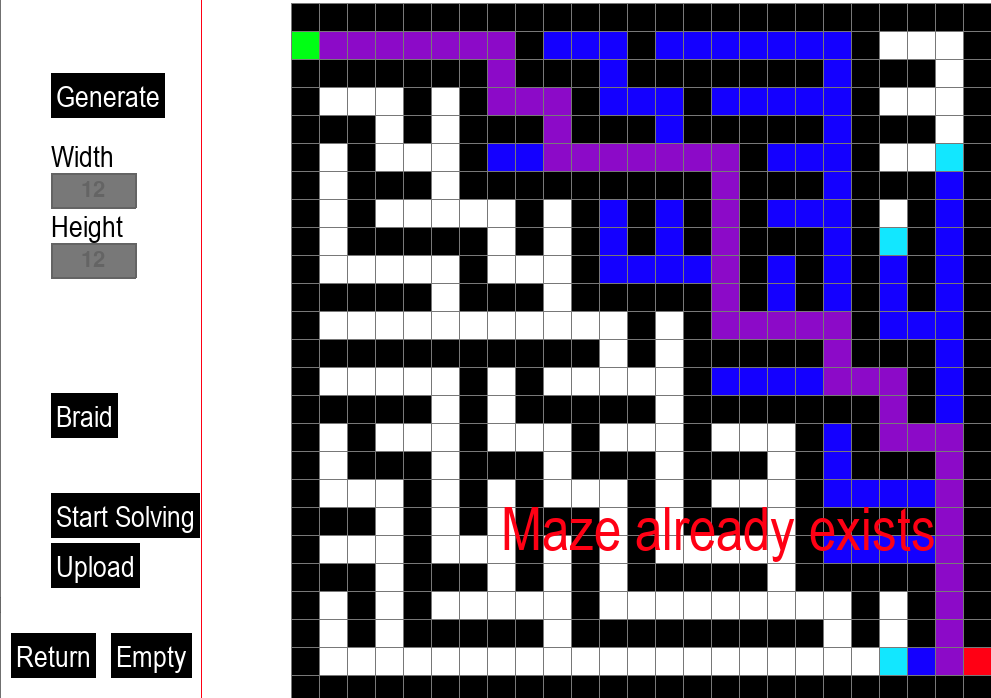


Test number 7a ii



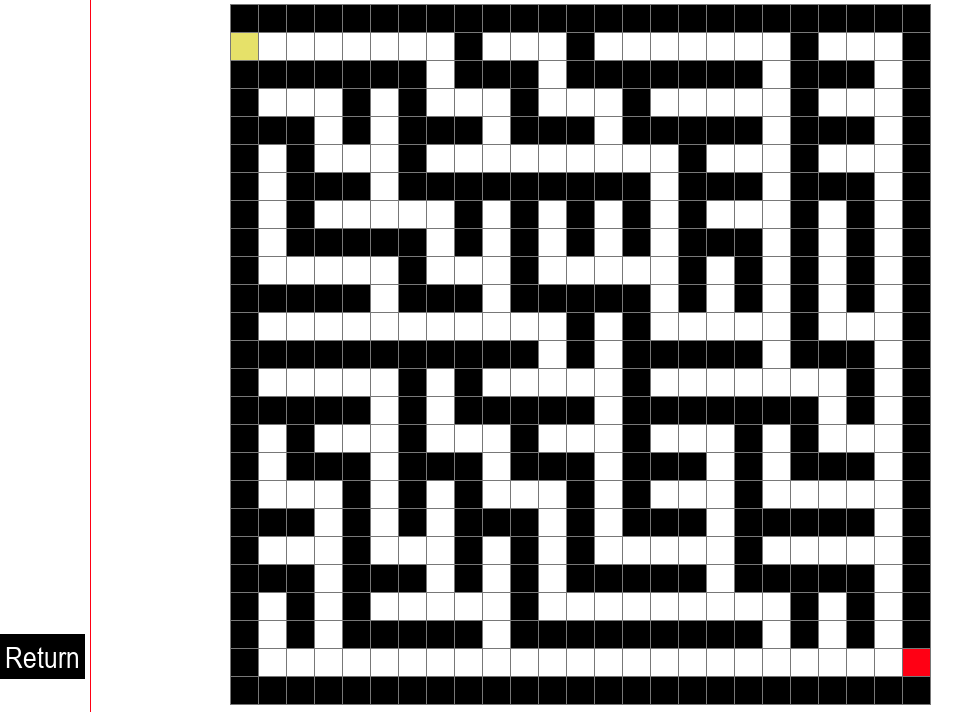
Test number 7b

Message displayed that maze already exists, therefore doesn’t let user to upload it



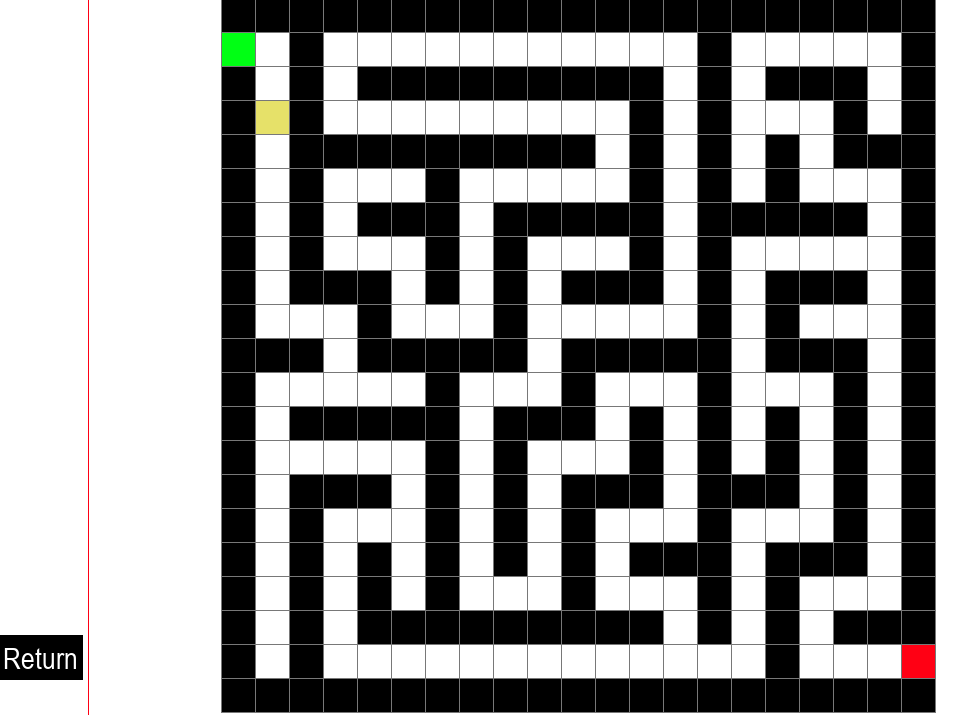
Test number 8a

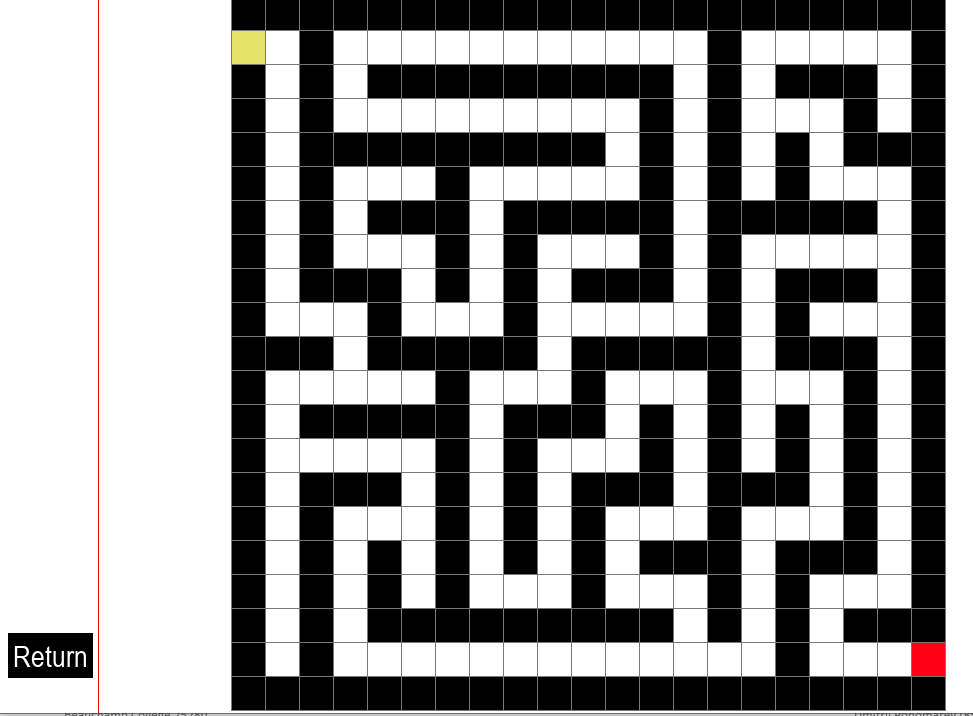
The maze uploaded in test 7 was used to show both uploading and selection correctness



Test number 9a

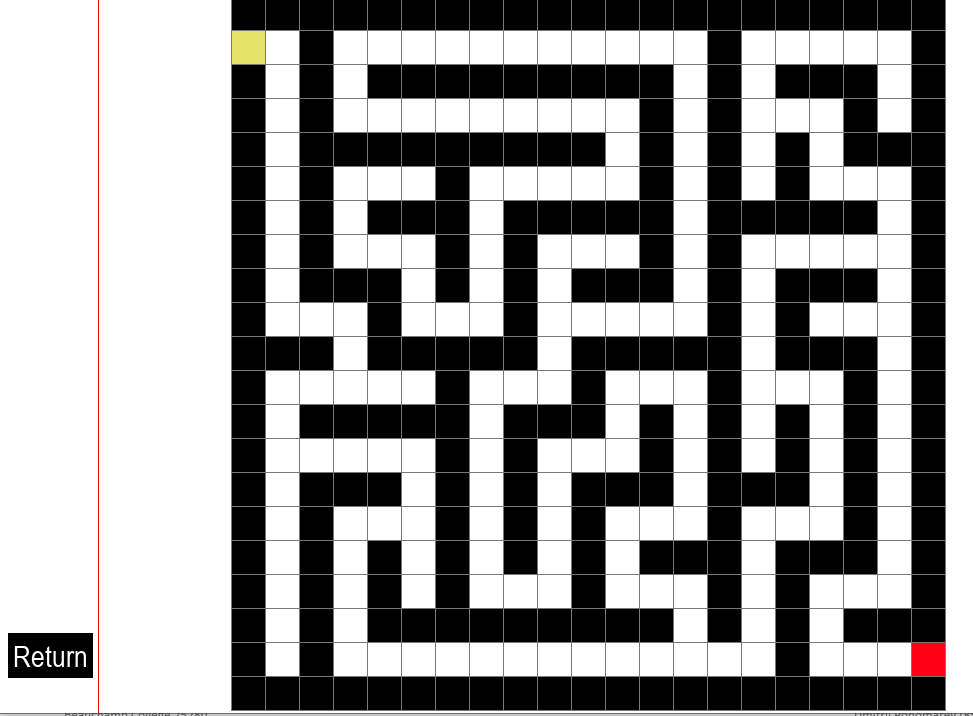
Screenshot shows the position of a node after sequence of buttons: left, down, down



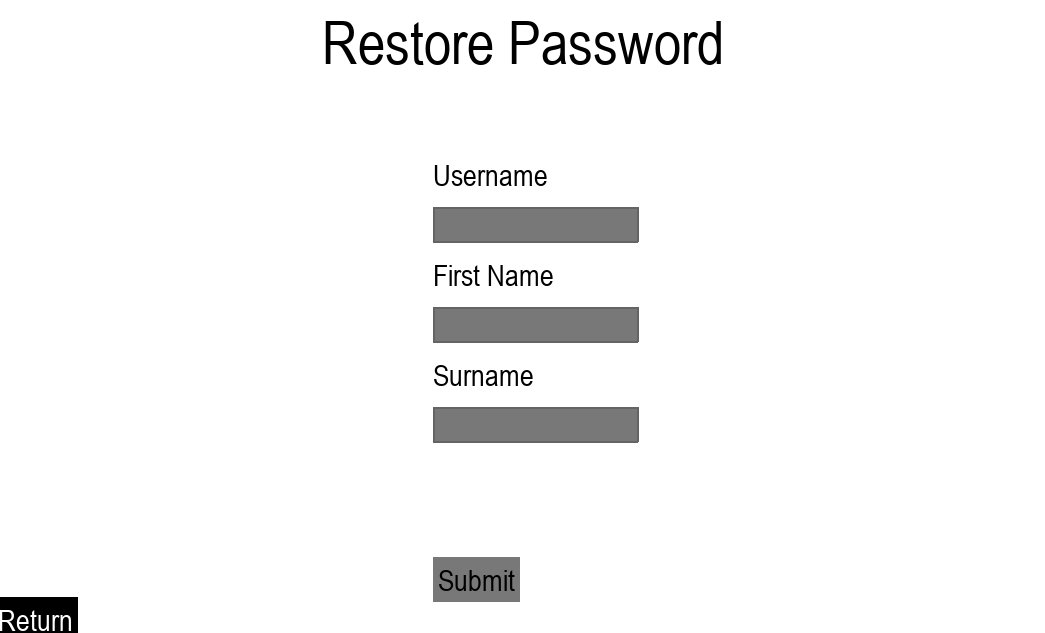
Screenshot shows the position of a node after sequence of buttons: D, S, S

Test number 9c

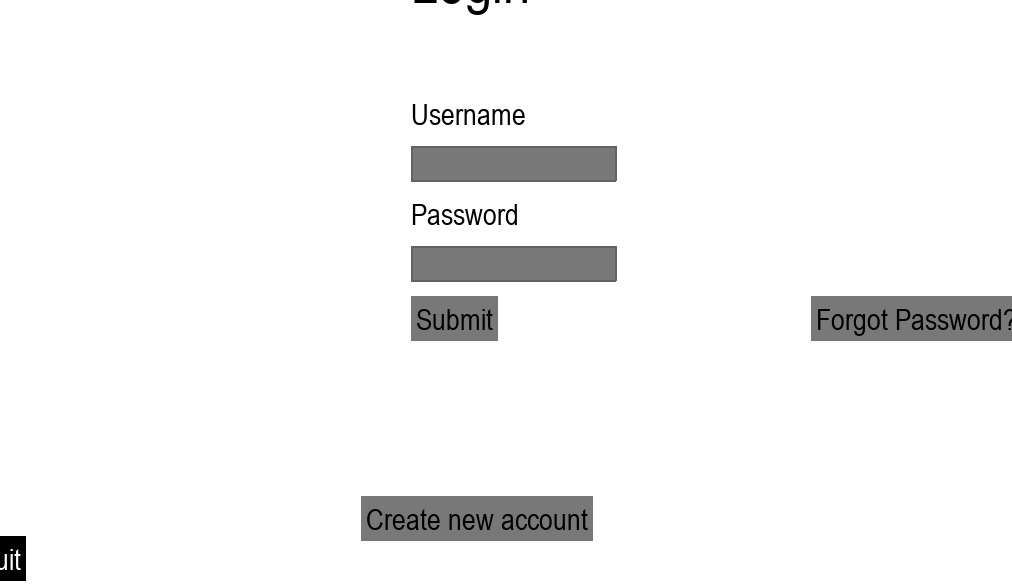
Screenshot shows the position of a node after dragging node with a cursor to position shown in test 9a

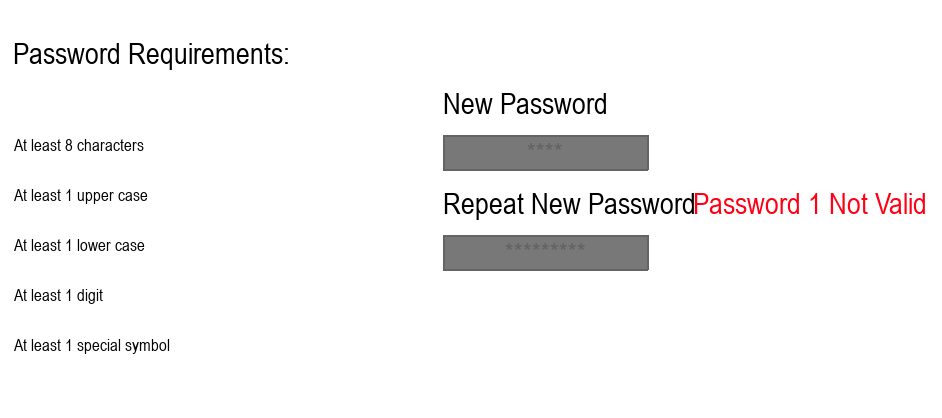


Test number 10a



Test number 10b

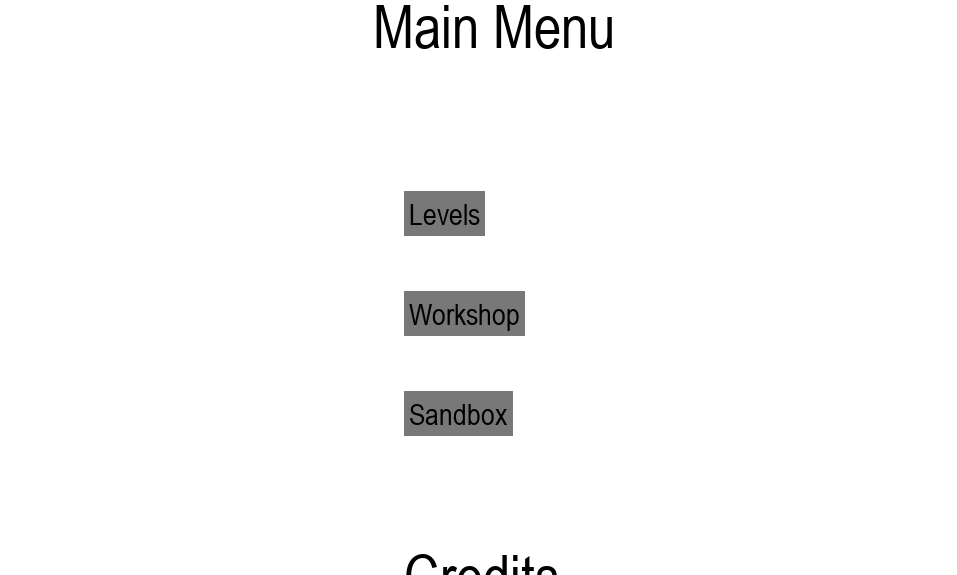




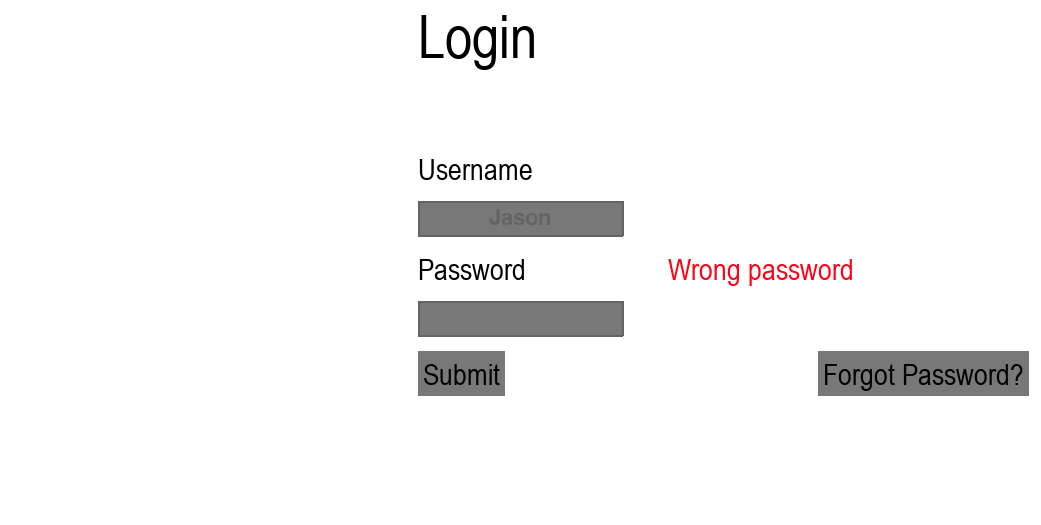


Test number 10c

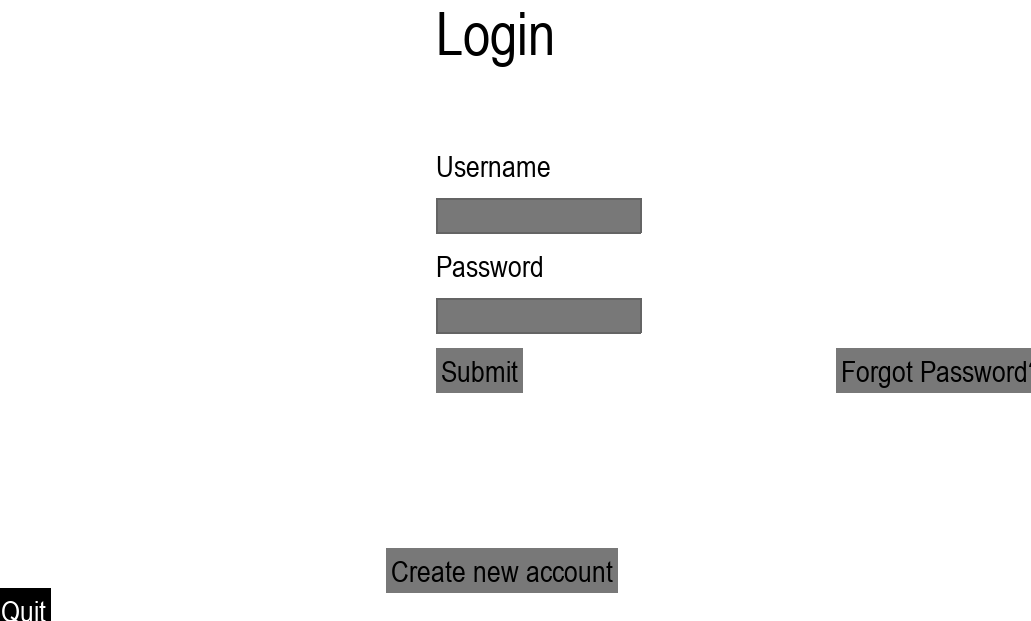
New password is used



Old password is used

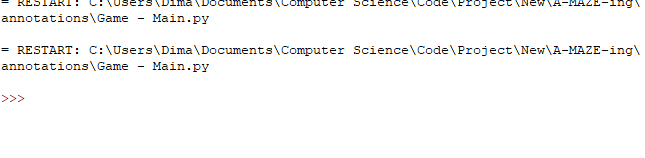


Test number 11a



Test number 11b

Quit button is pressed



## Login Page Testing

As input boxes and buttons are called from 1 class with same default reaction parameters, their testing will be done in here. All Input box’ text is limited to 30 characters, apart from several, which will be tested separately. It is done to prevent input box going off the screen while still holding normal data in there.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 1 | Input of characters in the input boxes to see their display and stretch of input box | Normal | Test | Word “Test” would be displayed | As expected | YES |  |
| Boundary | TestTestTestTestTestTestTest12 (30 characters) | Whole sequence would be displayed | As expected | YES |  |
| Erroneous | TestTestTestTestTestTestTestTest (32 characters) | Only first 30 characters would be displayed | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 2 | Pressing Quit button | Normal | NO TEST DATA | Program would stop running | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 3 | Login into the system by inputting details | Normal | Correct credentials (test; test) | Main menu page shown | As expected | YES |  |
| Erroneous | Correct login and incorrect password (test; test1234) | Error message shown stating incorrect password | As expected | YES |  |
| Erroneous | Incorrect login and password (test123; test123) | Error message shown stating no user found | As expected | YES |  |

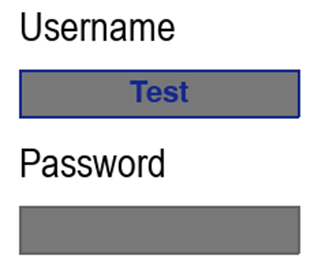
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 4 | Pressing Forgot Password button | Normal | NO TEST DATA | Forgot Password page should be displayed | As expected | YES |  |

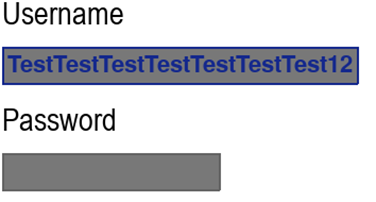
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 5 | Pressing on Create New account button | Normal | NO TEST DATA | Register page should be displayed | As expected | YES |  |

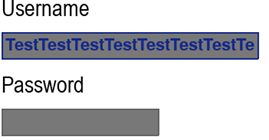
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 6 | Hovering over any button | Normal | NO TEST DATA | Button should change colour | As expected | YES |  |

### Screenshots

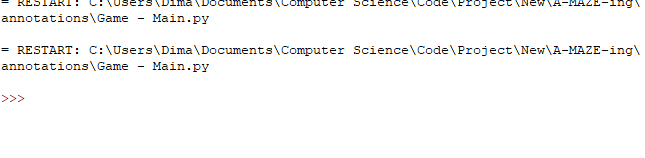
Test number 1



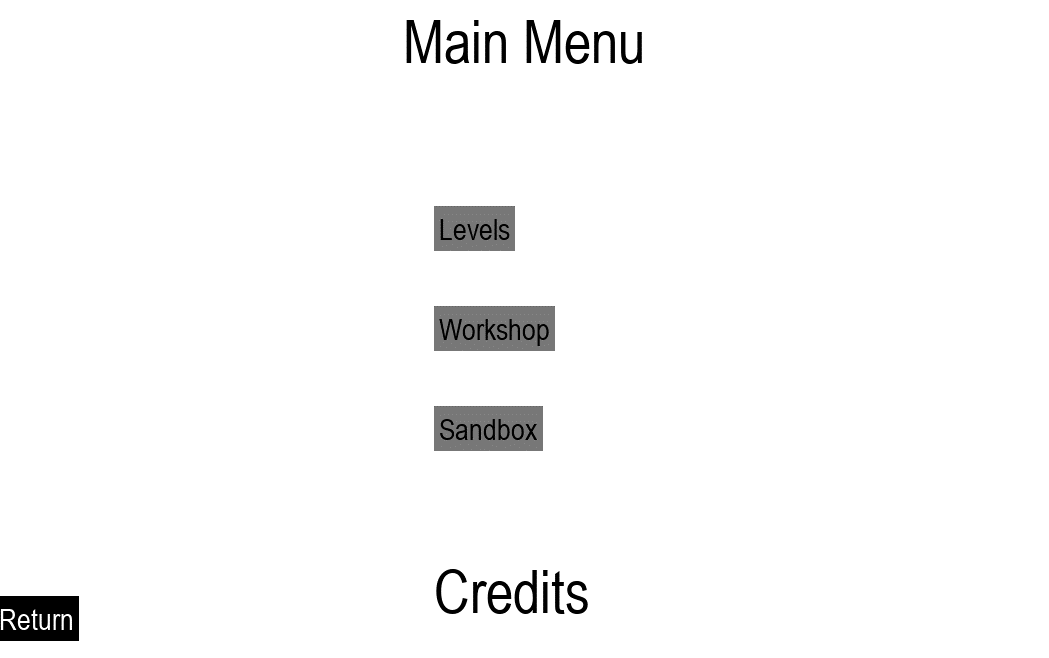


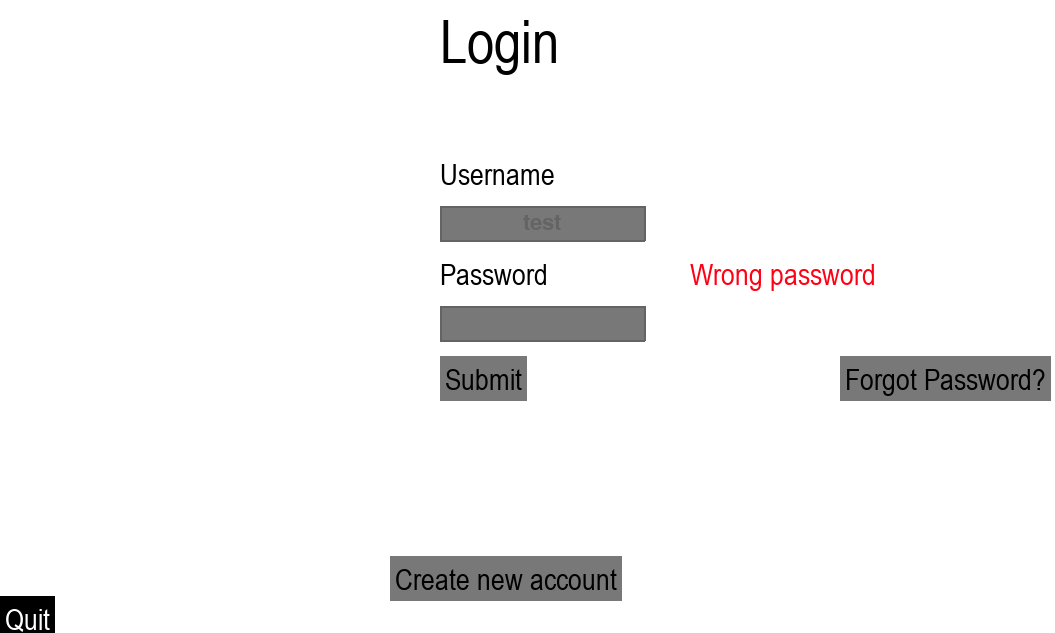


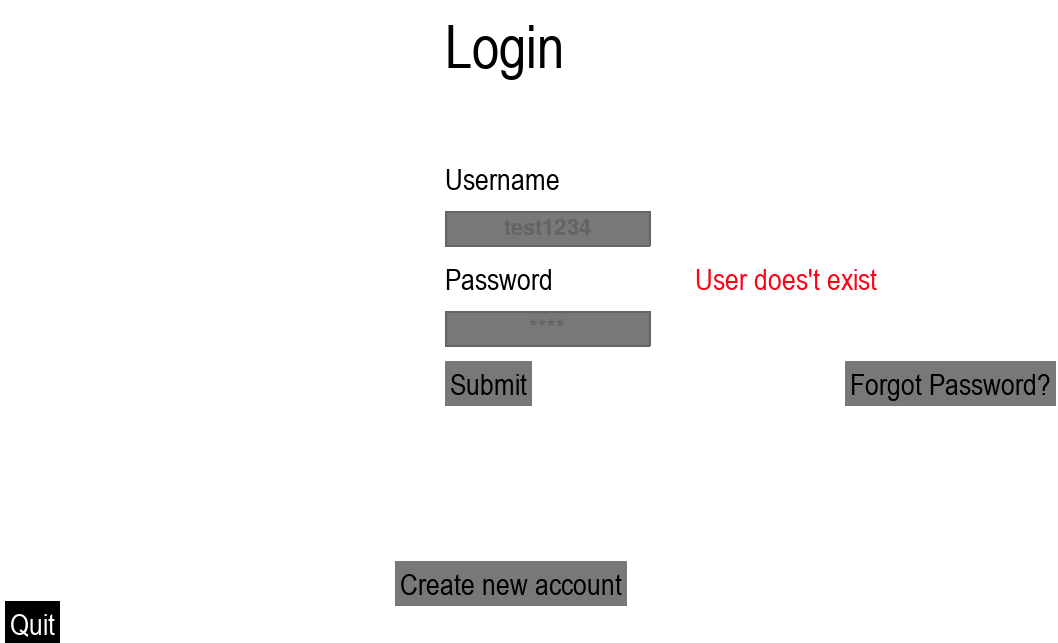
Test number 2



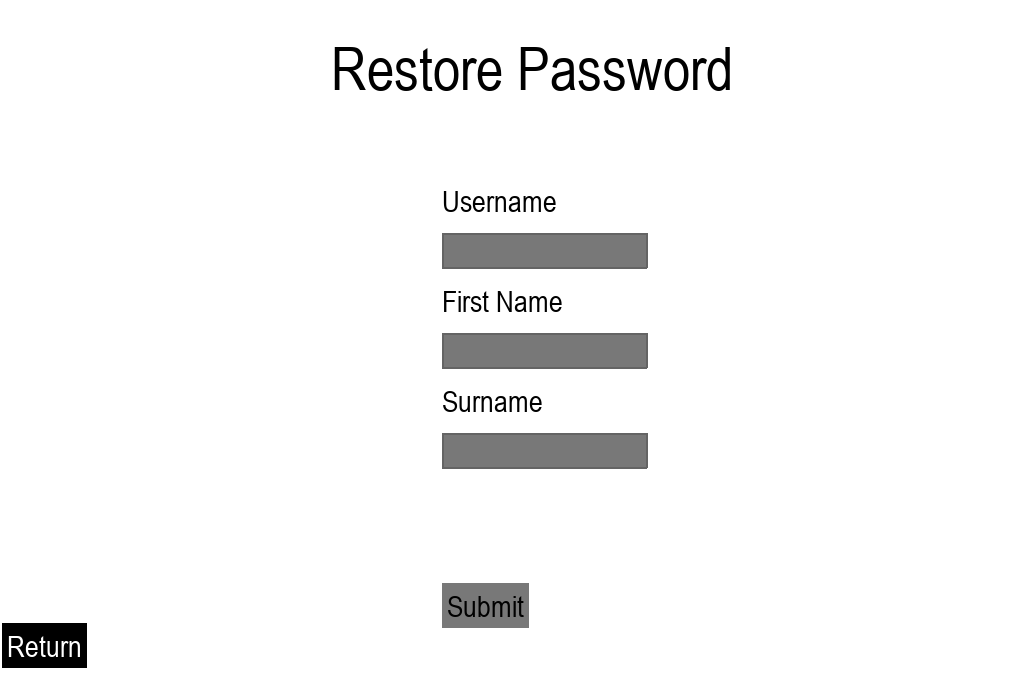
Test number 3







Test number 4

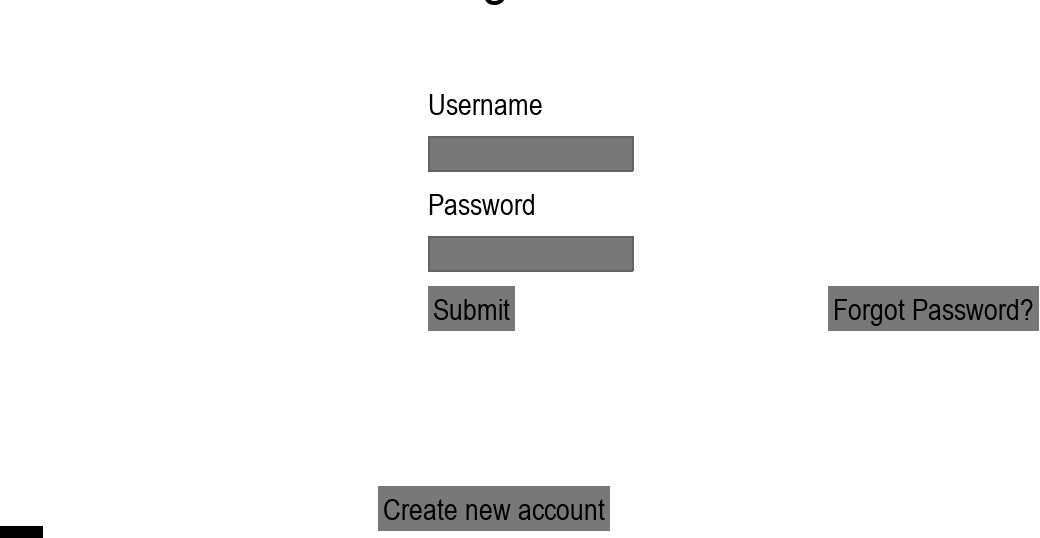


Test number 5

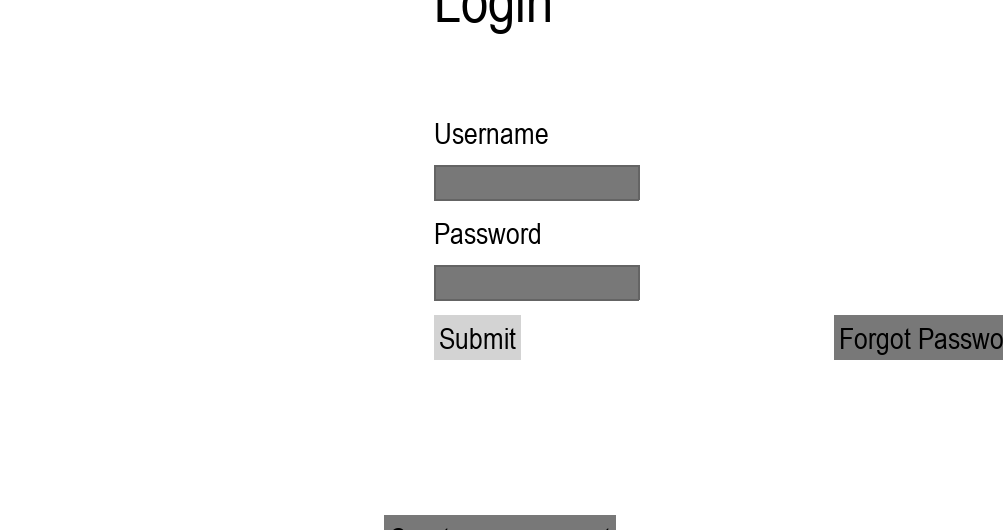


Test number 6

No hovering:



Hovering over submit button



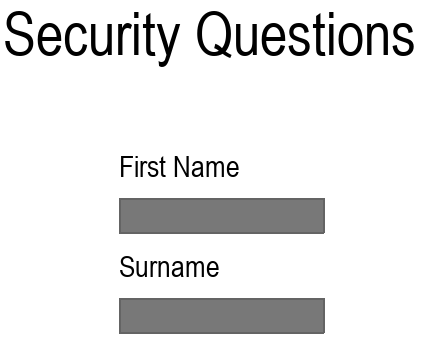
## Register Testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 7 | Registering new account | Normal | Clicking on button “Create new account” | Next window of register would appear | As expected | YES |  |
| Erroneous | Type username that is already in a database | Display error stating user is already registered | As expected | YES |  |
| Erroneous | Incorrect Password 1 and correct password 2 | Display error stating Password 1 is incorrect | As expected | YES |  |
| Erroneous | Correct password 1 and incorrect password 3 | Display error stating Password 2 is incorrect | As expected | YES |  |
| Erroneous | Password 1 and Password 2 don’t match | Display error stating passwords don’t match | As expected | YES |  |

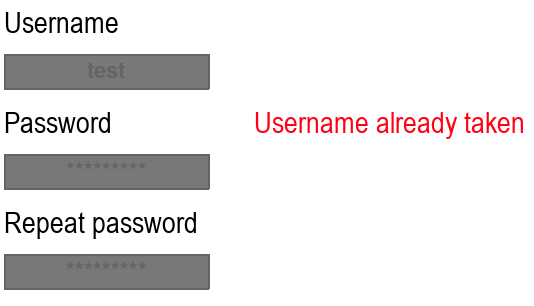
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 8 | Credentials input into input boxes | Normal | Inserting valid Name and Surname | Main Menu window would appear | As expected | YES |  |
| Boundary | Name/Surname is too short | Display error of invalid Name/Surname | As expected | YES |  |
| Erroneous | Name/Surname contains spaces | Display error of invalid Name/Surname | As expected | YES | Allows spaces in the box, but because of regex doesn’t allow them to pass |

### Screenshots

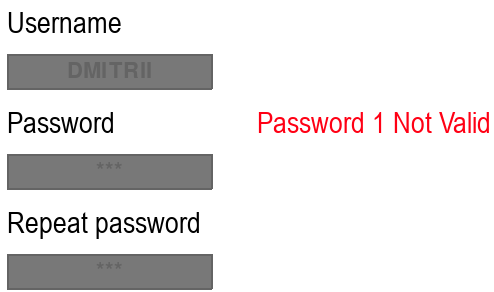
Test 7.1



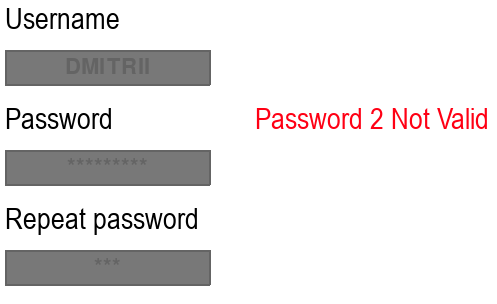
Test 7.2



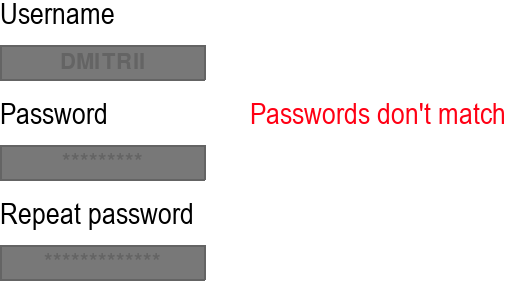
Test 7.3



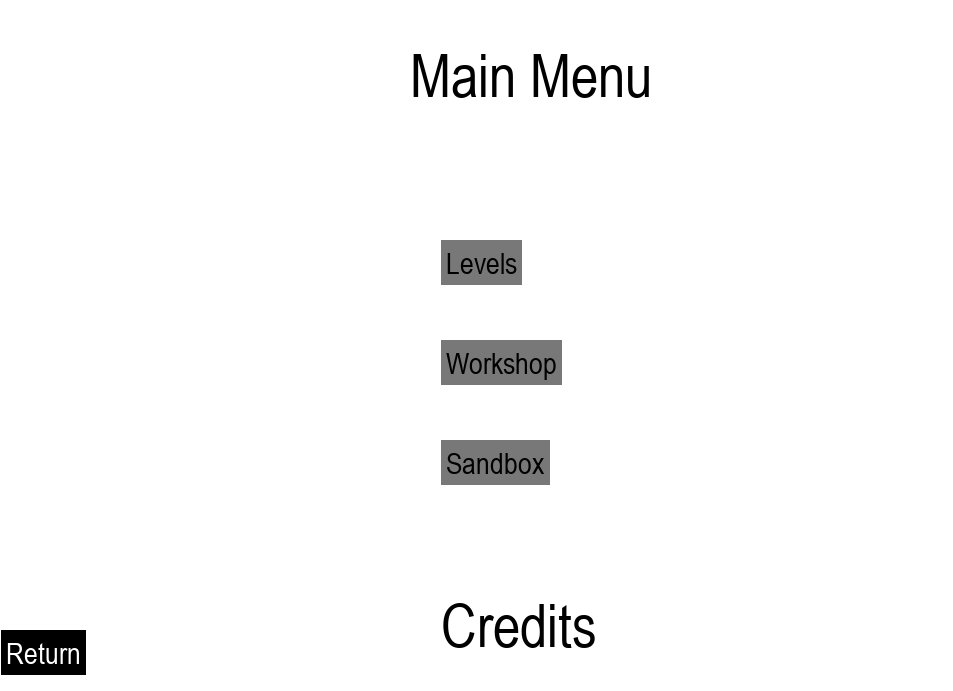
Test 7.4



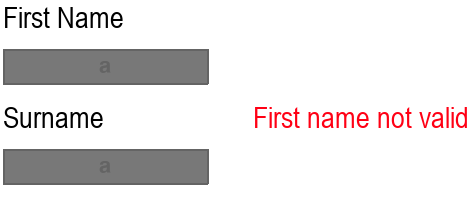
Test 7.5



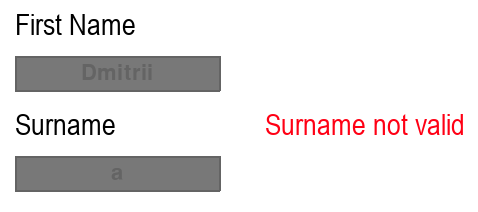
Test number 8.1



Test number 8.2



Test number 8.3



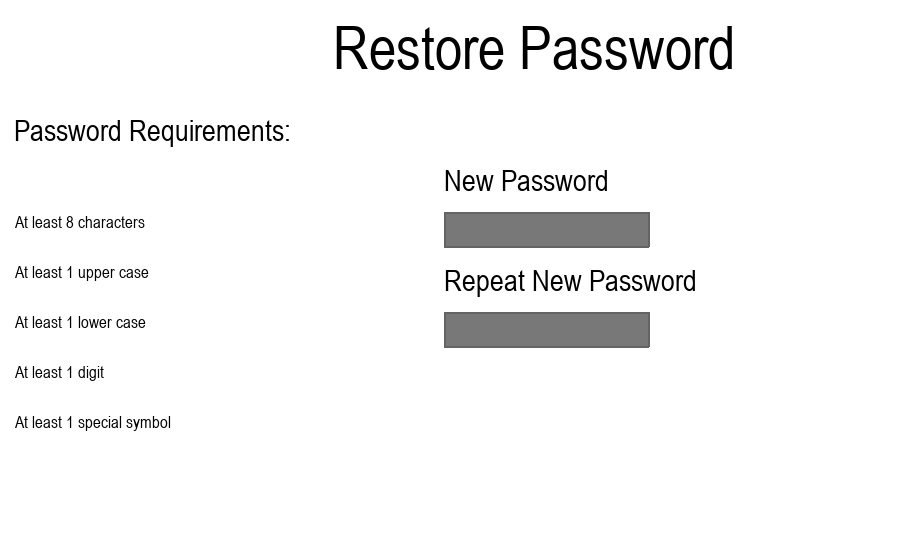
## Forgot Password Testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 9 | Inputting Username, First name and password | Normal | Input valid username and password according to writings on the left | Second window of restore password would appear | As expected | YES |  |
| Boundary | Inputting username which is too short(less than 3 characters) | Display error stating username is too short | As expected | YES |  |
| Boundary | Name that has spaces | Display error stating that Name is invalid | As expected | YES |  |
| Boundary | Surname that has spaces | Display error stating that Surname is invalid | As expected | YES |  |
| Erroneous | Inputting username that is not in a database | Display error stating that Record doesn't exist | As expected | YES |  |

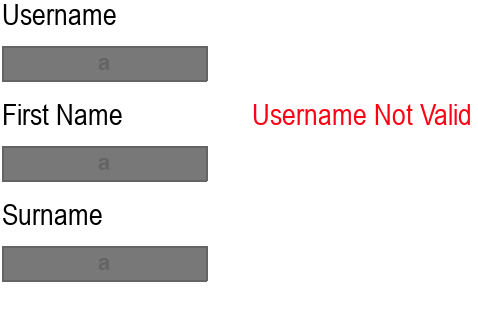
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 10 | Repeating password | Normal | Inputting 2 valid passwords | Login window would appear with empty input boxes | As expected | YES |  |
| Erroneous | Password 1 is invalid and Password 2 is valid | Stating error that Password 1 invalid | Password 1 invalid displayed | YES |  |
| Erroneous | Password 1 is valid and Password 2 is invalid | Stating error that Password 2 invalid | Password 2 invalid displayed | YES |  |
| Erroneous | Password 1 and Password 2 are both valid, but don’t match | Stating error that Passwords don’t match | Surname not valid displayed | YES |  |

### Screenshots

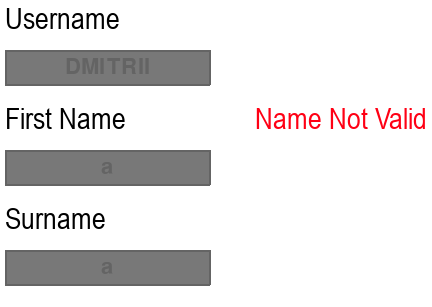
Test number 9.1



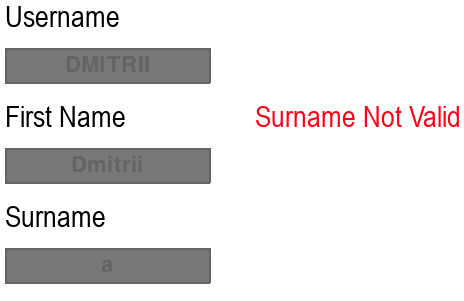
Test number 9.2



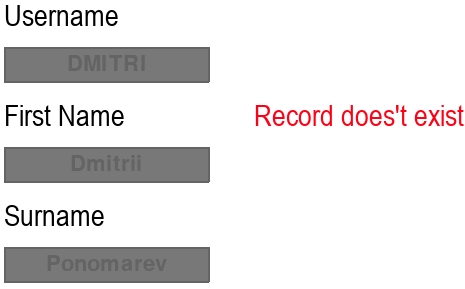
Test number 9.3



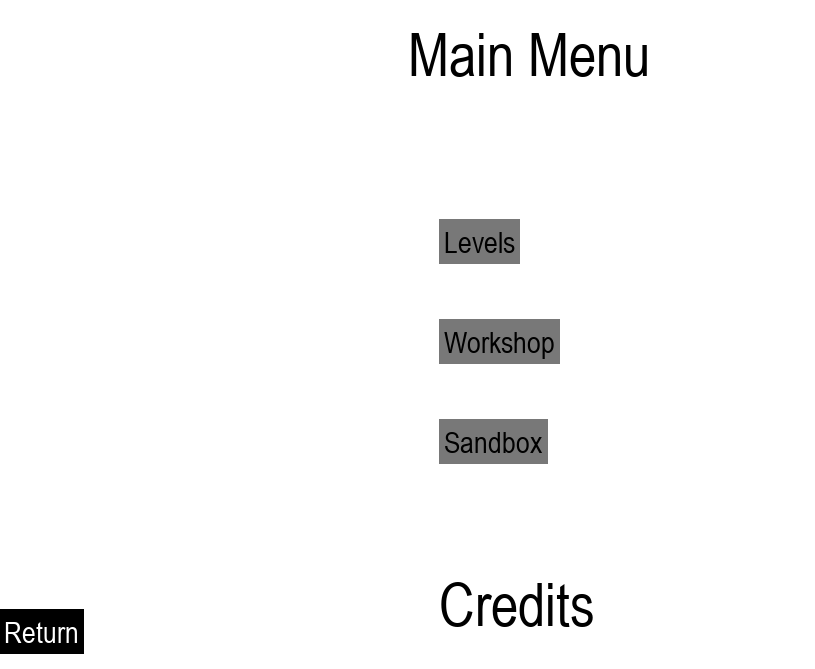
Test number 9.4



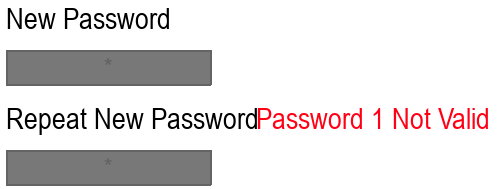
Test number 9.5



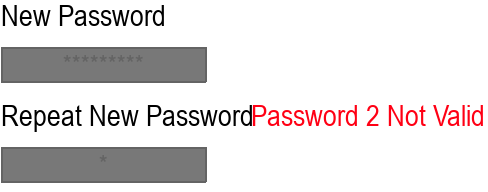
Test number 10.1



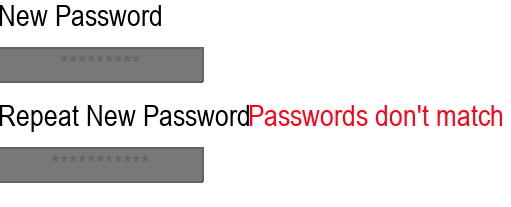
Test number 10.2



Test number 10.3



Test number 10.4



## Main Menu testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 11 | Clicking on Levels button | Normal | Levels Menu should be called | As expected | YES |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 12 | Clicking on Workshop button | Normal | Workshop Menu should be called | As expected | YES |  |

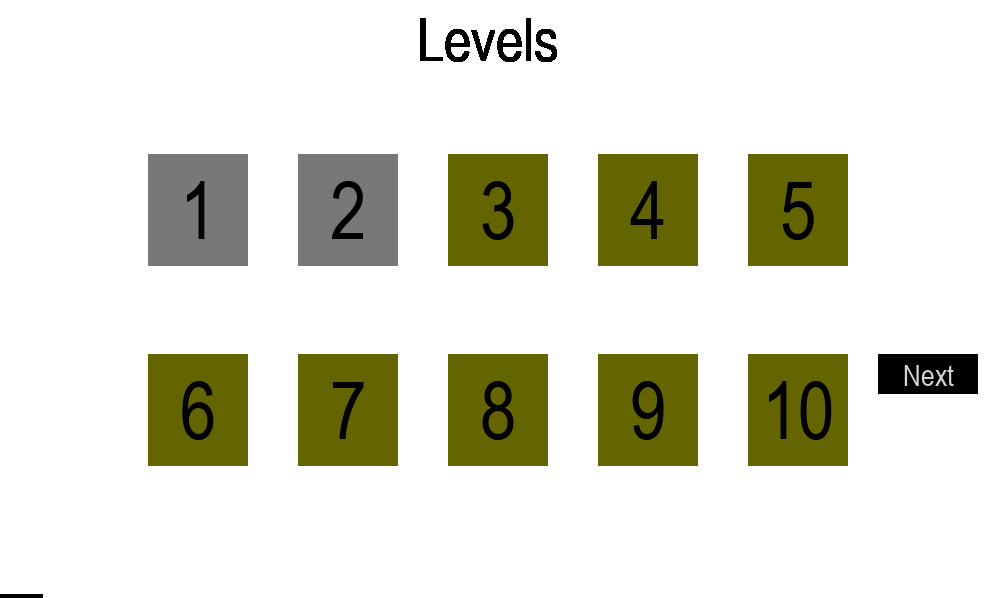
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 13 | Clicking on Sandbox button | Normal | Sandbox Menu should be called | As expected | YES |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 14 | Clicking on Credits button | Normal | Credits Menu should be called | As expected | YES |  |

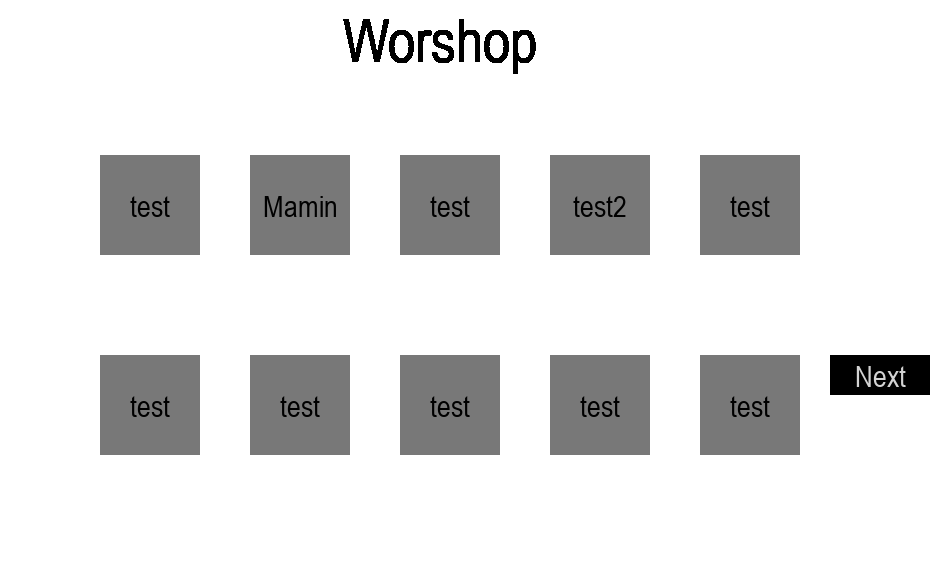
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 15 | Clicking on return button | Normal | Program should go back to login stage | As expected | YES |  |

### Screenshots

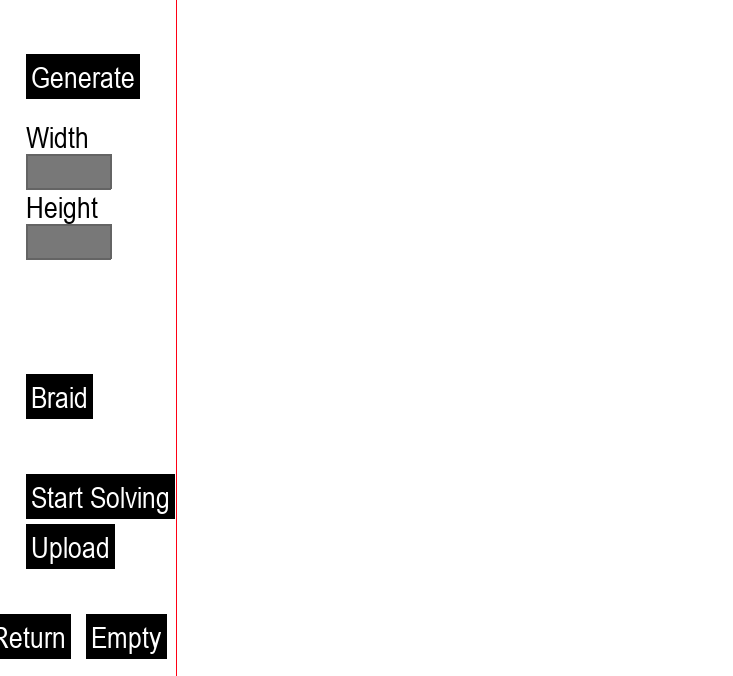
Test number 11



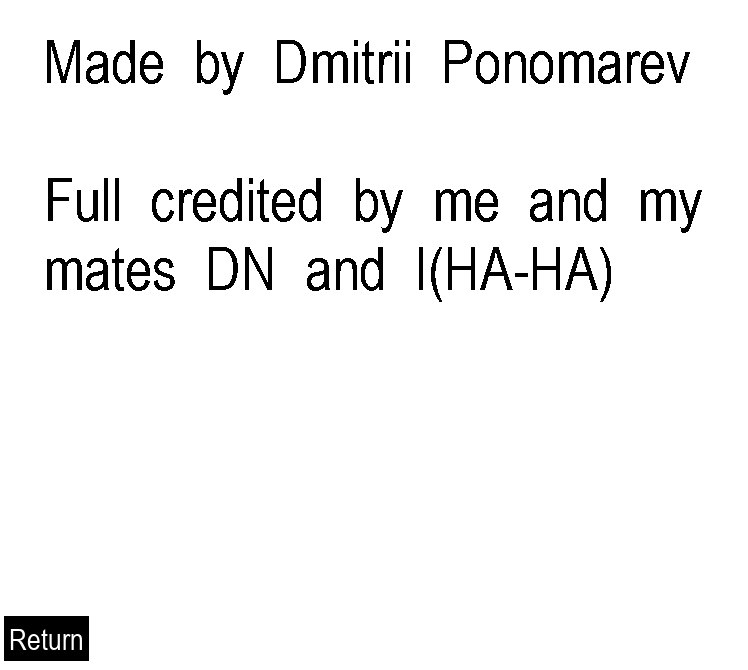
Test number 12



Test number 13



Test number 14



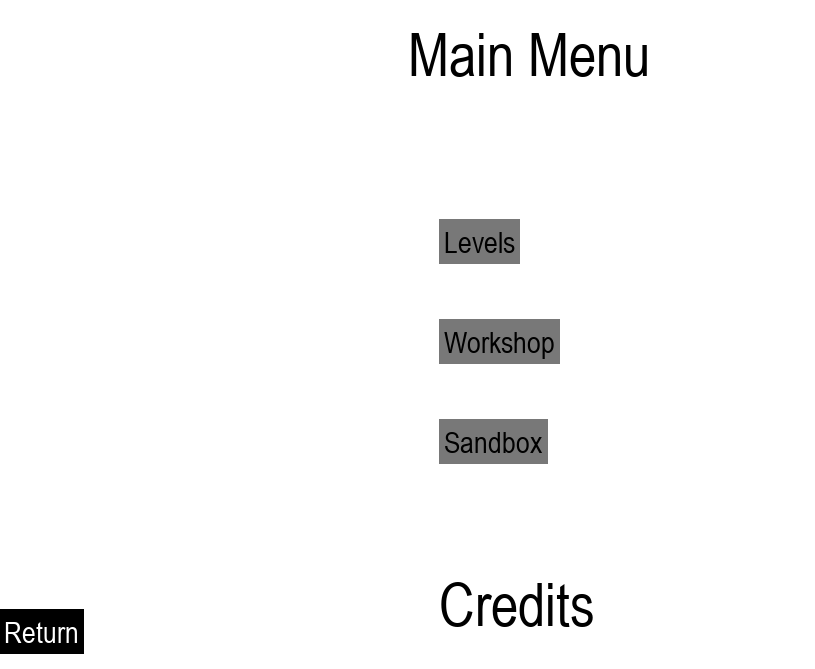
## Credits Testing

Each return button returns 1 stage behind. In this case it goes back to Main Menu window, therefore no more testing of return button will be made.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 16 | Clicking on return button | Normal | Program should go back to Main menu | Main menu is displayed | YES |  |

### Screenshots

Test number 16



## Levels Testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 17 | Clicking on first level | Normal | Program display maze of the first level | As expected | YES |  |

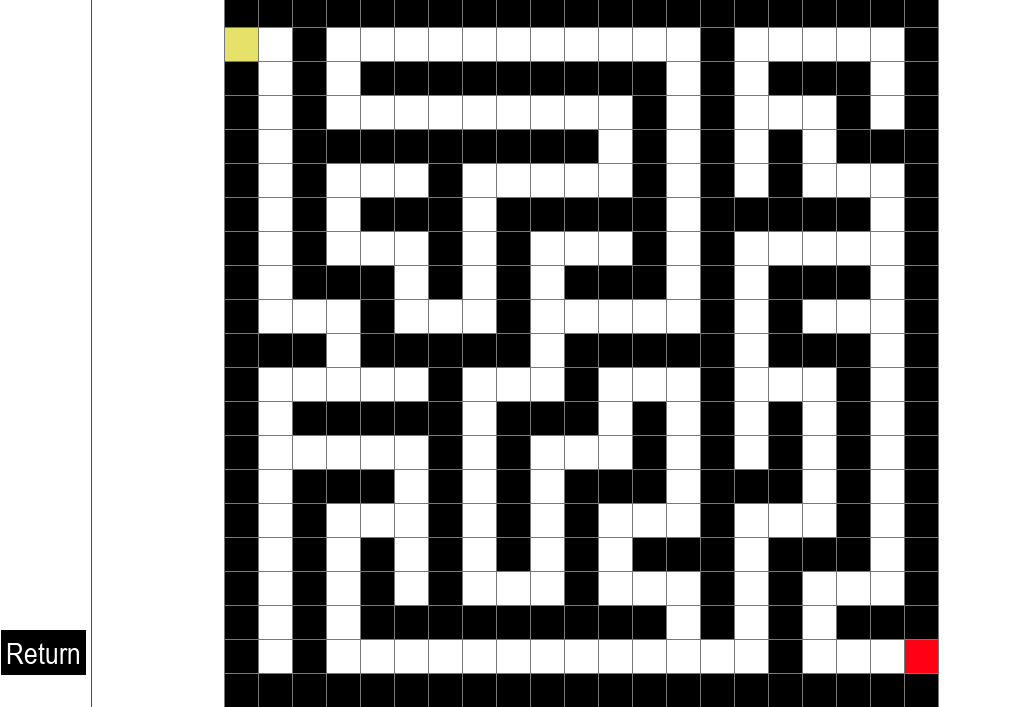
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 18 | Clicking on next button | Normal | Next bunch of levels should be displayed | As expected | YES |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 19 | Clicking on previous button | Normal | Previous bunch of levels should be displayed | As expected | YES |  |

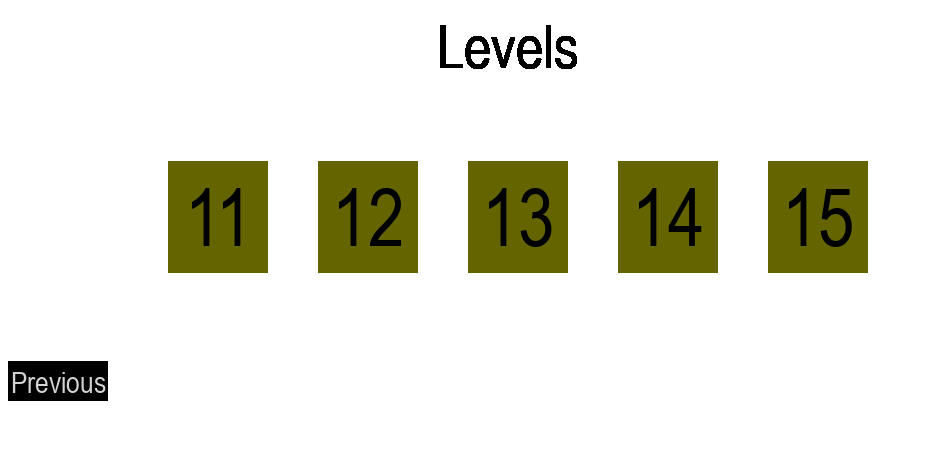
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 20 | Clicking on locked button | Boundary | Nothing should happen | As expected | YES |  |

### Screenshots

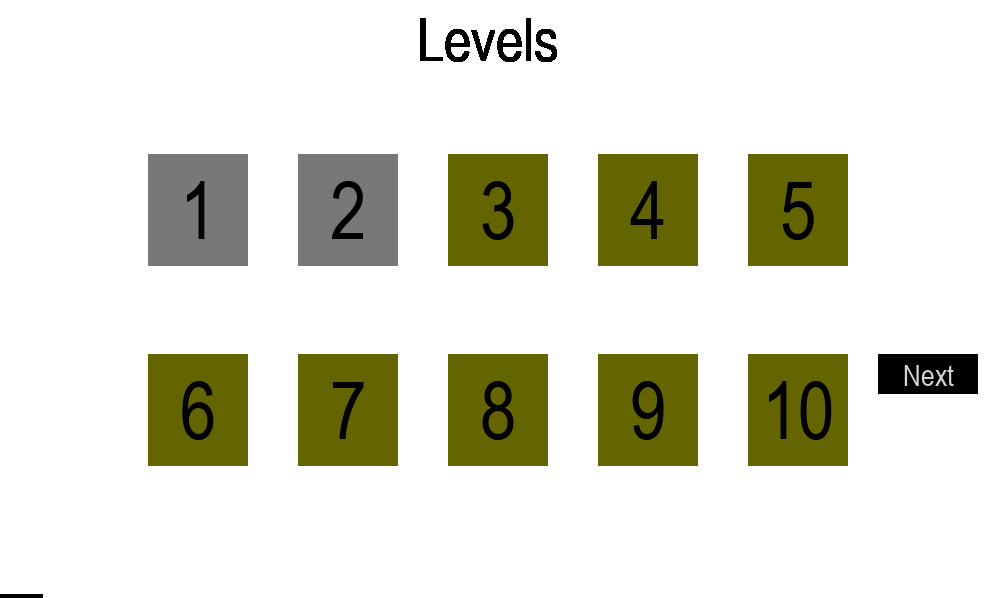
Text number 17



Test number 18



Test number 19

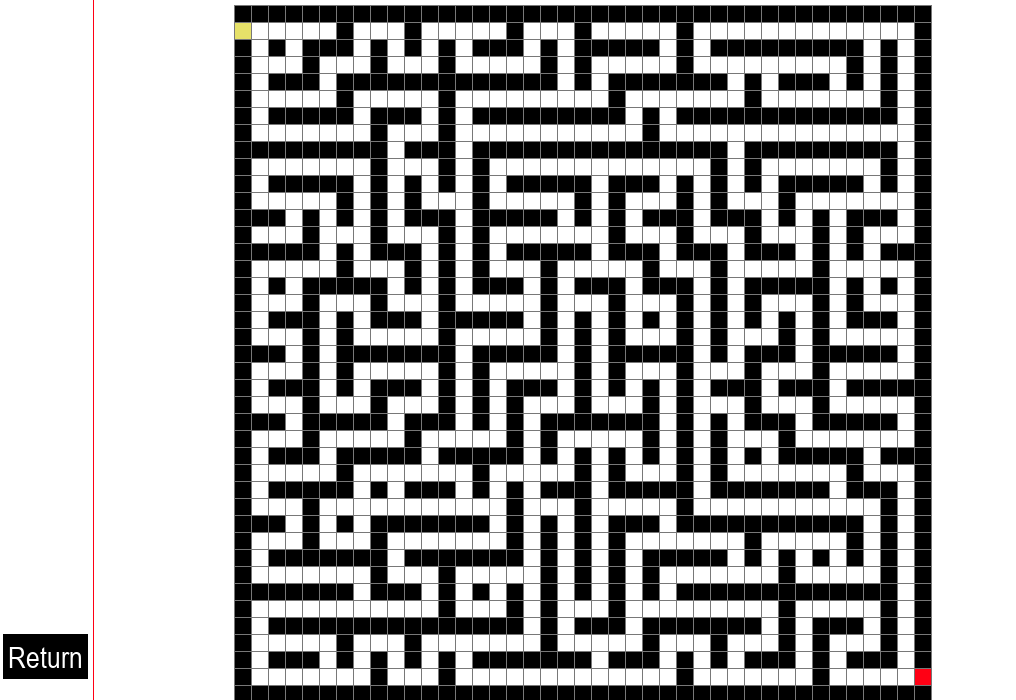


## Workshop testing

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 21 | Clicking on level button | Boundary | Level should appear | As expected | YES |  |

### screenshots

Test number 21



## Solving Maze Testing

Because solving maze is common for both Levels and Sandbox, there would be only 1 testing on level 1 in Levels due to its simplicity and not to spoil any other levels.

Its also hard to show erroneous data, especially moving to the wall, as nothing is displayed, therefore there would be no screenshots for some of the tests, as it would be impossible to tell if its correct.

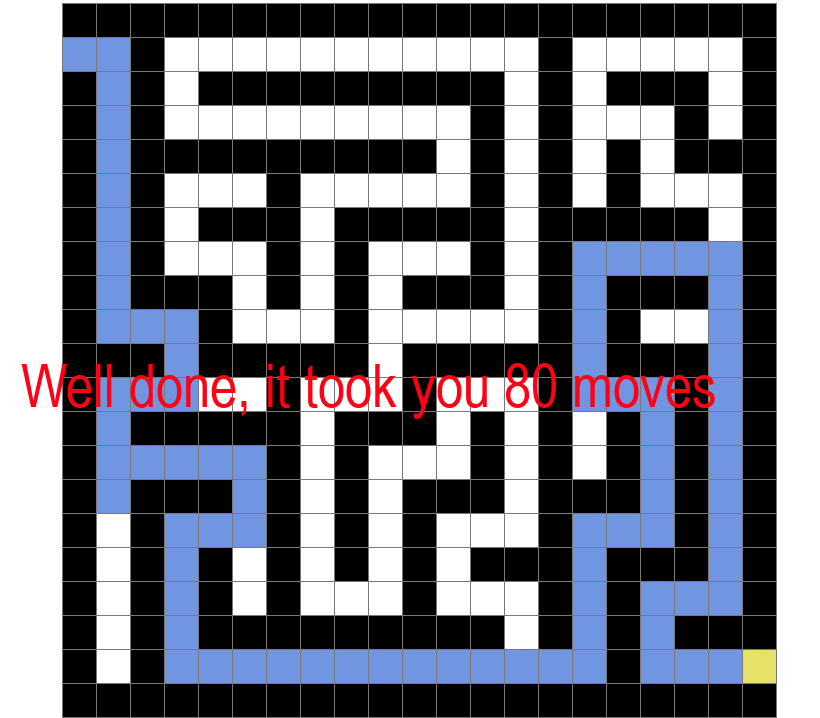
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 22 | Solving the maze | Normal | User moved freely | Whatever way user went should appear | As expected | YES | Stack could be used to hide all loops and tracking back, but since maze is about solving and going to dead ends, I decided not to implement it |
| Erroneous | User tries to move node into wall | Node doesn’t move | As expected | YES |  |
| Boundary | User goes back and forth several times | Final path displays with all moves the user did | As expected | YES |  |
| Normal | User solves new given level | Next level should unlock, unless it’s the very last level | As expected | YES |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 23 | Reloading the level after completing it | Normal | Clear level should appear | As expected | YES |  |

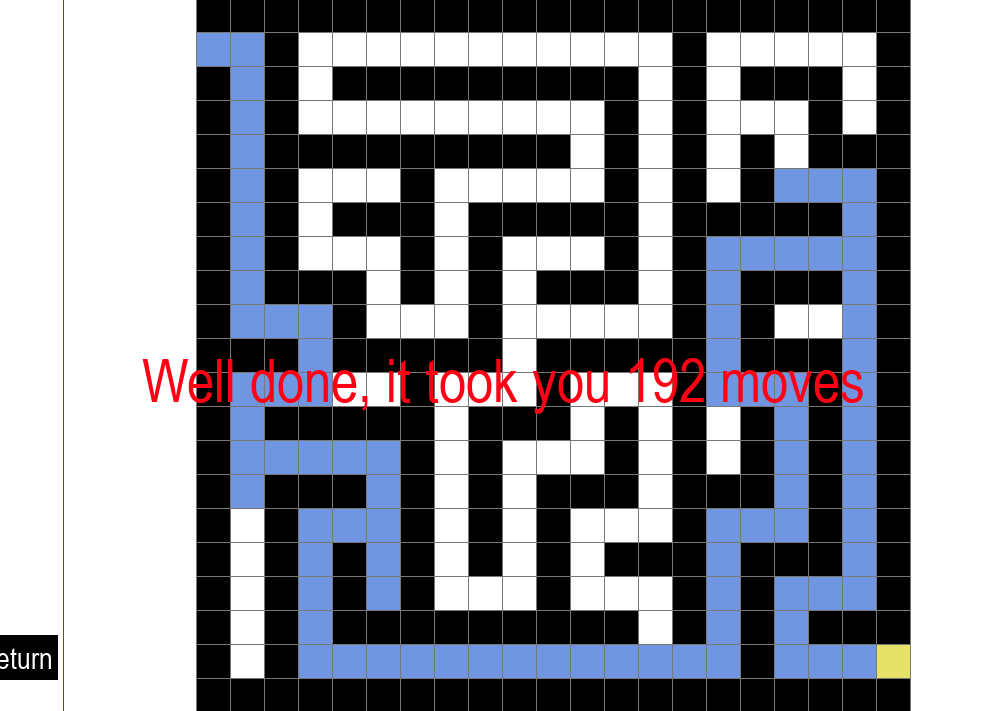
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 24 | Dragging the start node to solve the maze | Erroneous | Level should get solved | Node didn’t move from its initial position | No | Implement solving by cursor and key arrows |

### Sceenshots

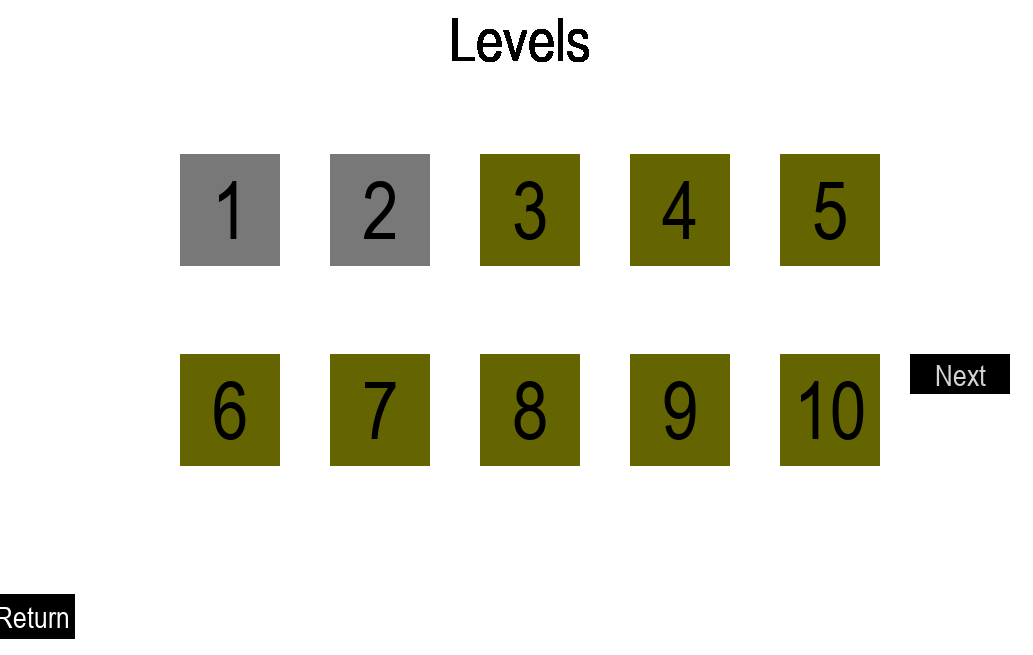
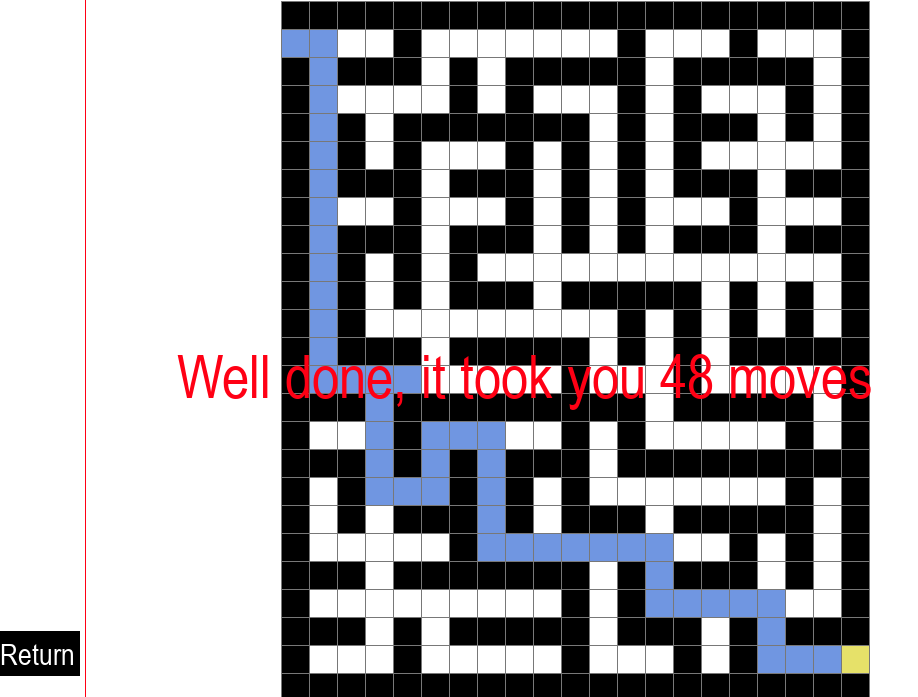
Test number 22.1



Test number 22.3

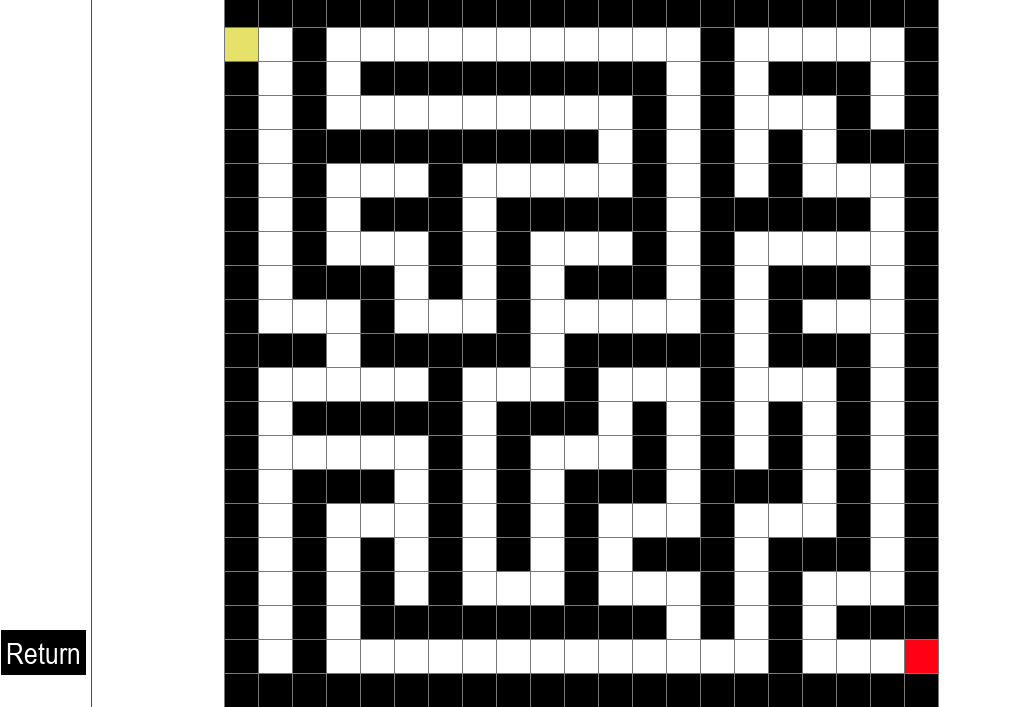


Test number 22.4





Test number 23



## Sandbox testing

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 24 | Generating maze | Normal | Height and Width are greater than 5 and less than 40 | Grid should generate with 2x+1 dimensions | As expected | YES |  |
| Erroneous | Button generate pressed without dimensions inputted | Grid doesn’t appear | As expected | YES |  |
| Boundary | Height and width are either 5 or 40 | Grid generates with 2x+1 dimensions specifies | As expected | YES |  |
| Erroneous | Height and width are lower than 5 or greater than 40 | Error displayed explaining dimensions requirement | As expected | YES | Test Colour is bright red to imitate warning, but could overlap the maze, so could be uncomfortable to read |
| Erroneous | Input string into height or width input box | No grid is generated | Letters aren’t allowed to be inputted; therefore, grid can’t be generated | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 25 | Braiding the maze | Normal | Press Braid when the grid is generated | Most of dead ends become empty nodes | As expected | YES |  |
| Erroneous | Press Braid with no grid generated | Nothing should happen | As expected | YES |  |
| Erroneous | Braid a maze drawn by a user, rather than an algorithm | Most of dead ends become empty nodes | Inconsistent with removing dead ends, as sometimes could remove only 1 and leave the rest | NO | Analyse the grid before pressing braid, rather than after creating a maze |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 26 | Solving the maze | Normal | Press solve on algorithm-generated maze | Maze is solved showing parent, child and best route | As expected | YES |  |
| Erroneous | Press solve before creating a grid | Nothing happens | As expected | YES |  |
| Normal | Press solve on a user-generated maze | Maze is solved showing parent, child and best route | As expected | YES |  |
| Boundary | Press solve on a maze with no valid path (has to be user made) | All grid gets discovered and error displayed showing path is not found | As expected | YES |  |
| Boundary | Place end node on the edge with corner available | Algorithm doesn’t cut corner and goes all the way around finding valid path | As expected | YES |  |
| Boundary | Place start node surrounded by walls | Error displayed stating that no valid path found | As expected | YES |  |
| Normal | User closes program while its solving | Program closed while solving | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 27 | Creating empty grid | Normal | Dimensions are specified and button empty is clicked | Empty grid is created with border walls | As expected | YES |  |
| Erroneous | Empty pressed without dimensions specified | No grid generated | As expected | YES |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 28 | Increase dimensions after creating a grid | Boundary | Start and End cells stay in the same place if touched and moved to corners if untouched | Start and End cells stay in the position of the first creation | NO | Could make a Boolean variable is\_moved to know for sure if the node was moved |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 29 | decrease dimensions after creating a grid | Boundary | Start and End cells stay in the same place if touched and moved to corners if untouched | Start and End cells move to the corners | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 30 | Upload the maze | Normal | Upload button pressed after grid is generated by algorithm | Maze is solved to verify valid path and uploaded to a database | As expected | YES |  |
| Erroneous | Press upload before generating a grid | Nothing should happen | As expected | YES |  |
| Normal | Press upload on a user-made maze | Maze is solved to verify valid path and uploaded to a database | As expected | YES |  |
| Boundary | Press upload on a maze with no valid path | Algorithm discovers all space available and displays error of no valid path found. Maze is not uploaded | As expected | YES |  |
| Boundary | Upload the same maze twice | Maze is solved normally and error message displayed that maze is already in a database. Maze is not uploaded | As expected | YES |  |
| Normal | Program closed while uploading | Program is closed | As expected | YES |  |

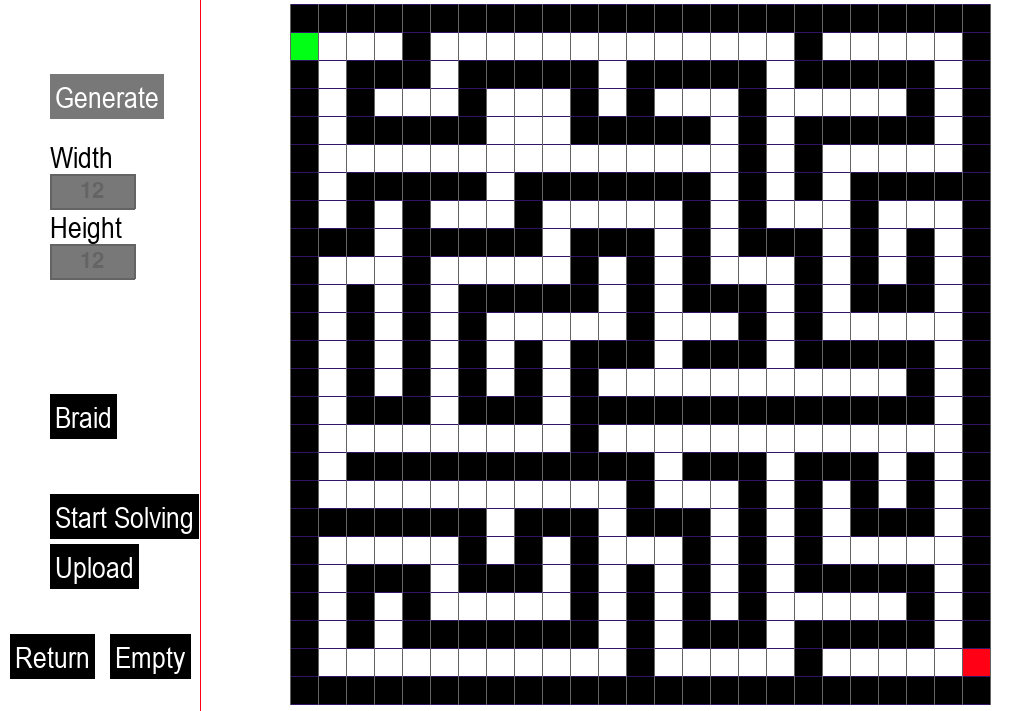
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 31 | Drawing the maze | Normal | Left click over empty grid cell | Cell turns black showing that it’s a wall cell | As expected | YES |  |
| Erroneous | Left click over space where grid should be generated without grid generated | Nothing should happen | As expected | YES |  |
| Erroneous | Left click over start/end node | Start/end nodes are not replaced by the wall node | As expected | YES |  |
| Erroneous | Start pressing left button over grid then transport outside of window | Walls are drawn until the point when cursor goes away from the grid, then stops drawing | As expected | YES |  |
| Erroneous | Press and hold mouse wheel instead of left click | Nothing should happen | As expected | YES |  |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 32 | Drawing the maze | Normal | Right click over wall node | Wall cell disappears becoming empty white cell | As expected | YES |  |
| Erroneous | Press right click over not generated grid | Nothing should happen | As expected | YES |  |
| Boundary | Right click over maze edge border wall cells | Nothing should happen | As expected | YES |  |
| Erroneous | Right click over start/end cell | Start/end cells aren’t erased | As expected | YES |  |
| Erroneous | Start pressing right button over grid then move cursor outside of window | Cells are erased until cursor is over the grid, then stops | As expected | YES |  |

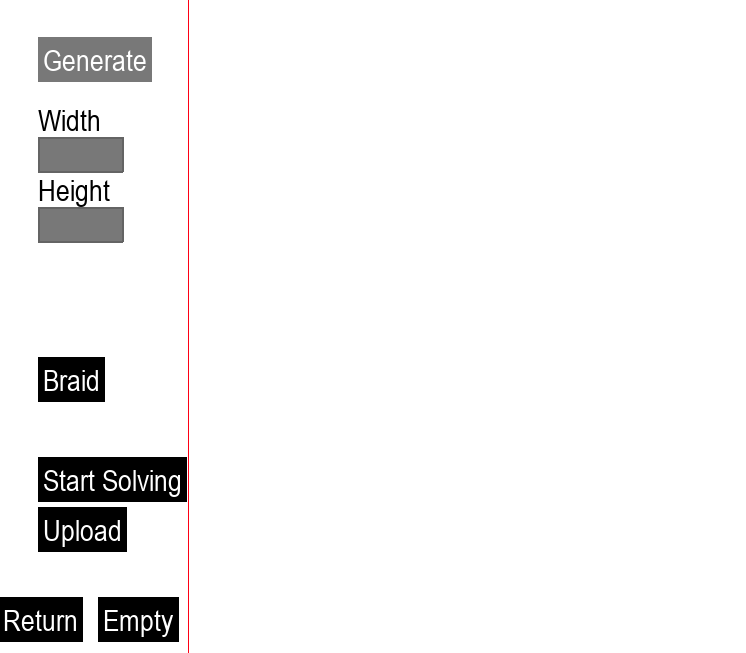
|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test Number | Test Description | Test Data Type | Test Data | Expected Outcome | Actual Outcome | Test Passed | Corrective Actions |
| 33 | Moving start and end | Normal | Start/end node is being dragged while holding left mouse button | Start/End is being moved and followed by mouse cursor | As expected | YES |  |
| Erroneous | Put start and end node inside each other | They don’t overlap | As expected | YES |  |
| Boundary | Place start/end nodes on the border | Start/End is being moved and followed by mouse cursor and stayed on the border | As expected | YES |  |
| Erroneous | Start/end node is being dragged over the edge | Start/end node is dragged until border is met then stops | As expected | YES |  |

### Screenshots

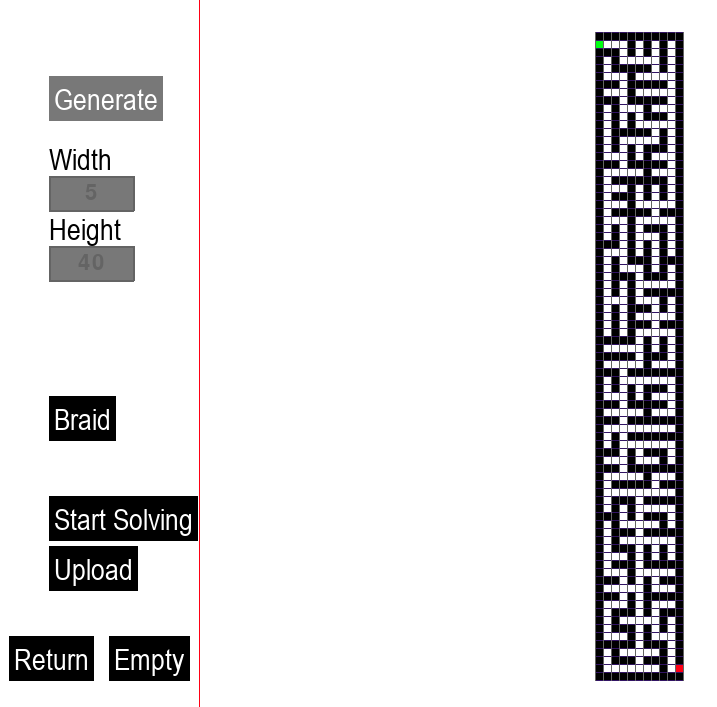
Test number 24.1



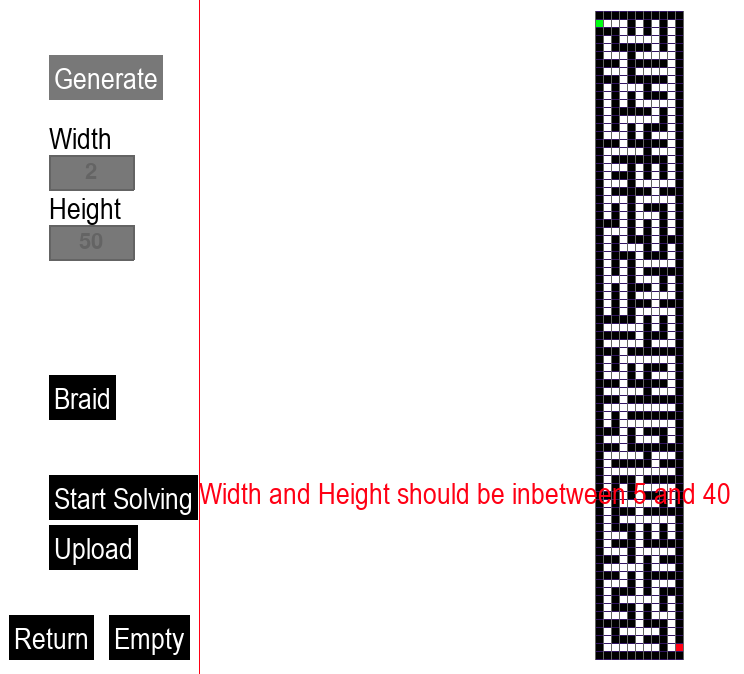
Test number 24.2



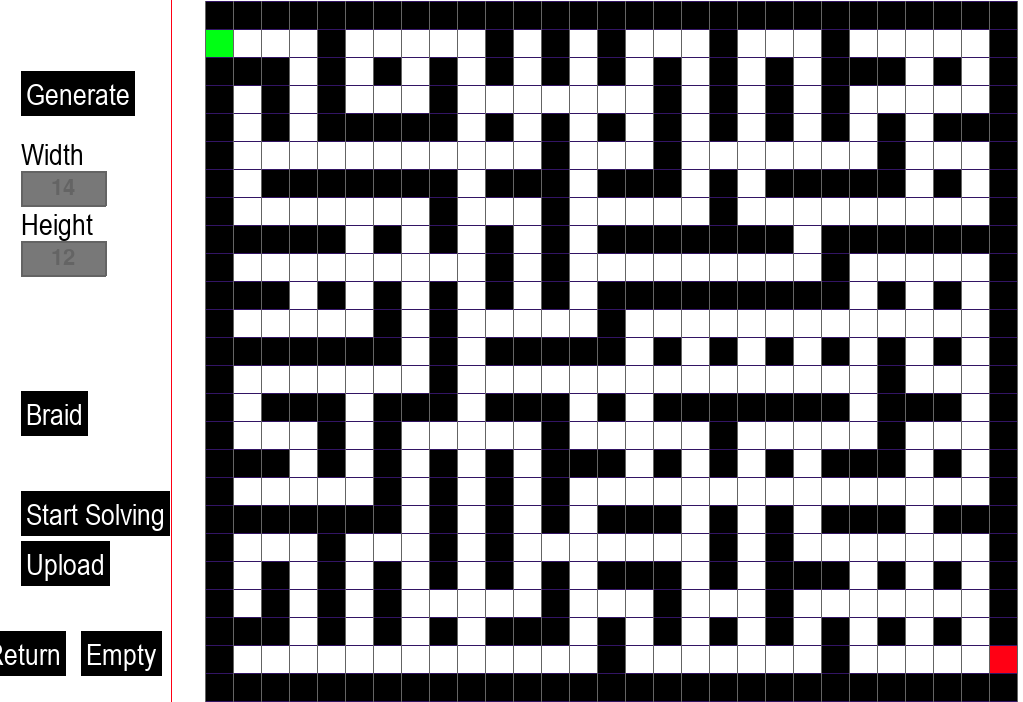
Test number 24.3



Test number 24.4

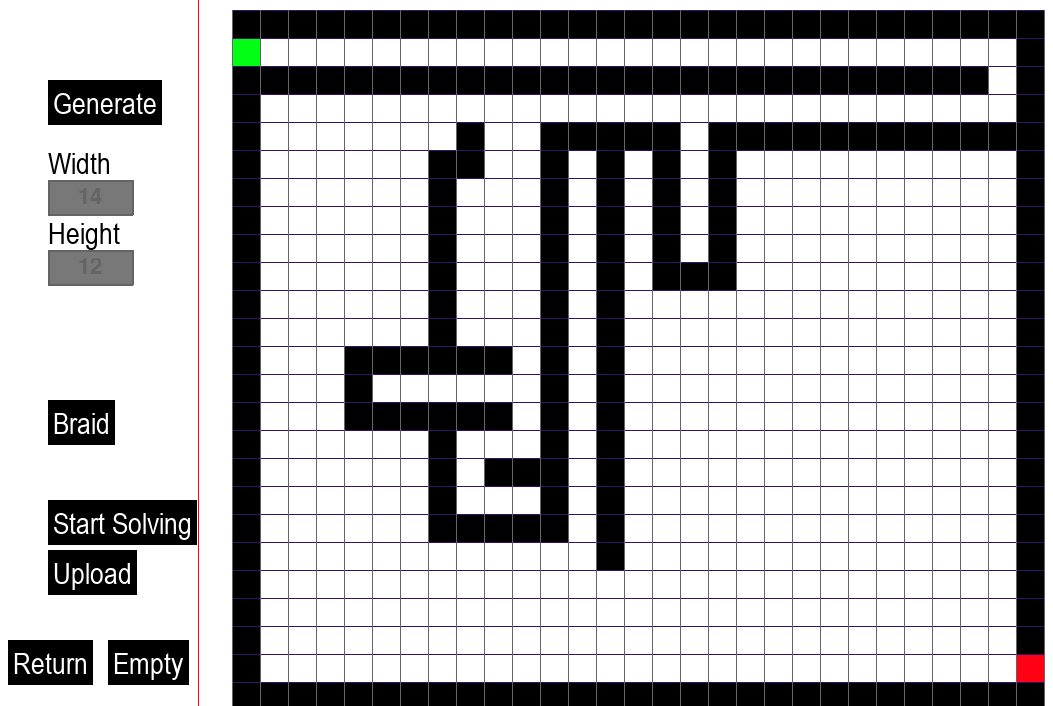


Test number 25

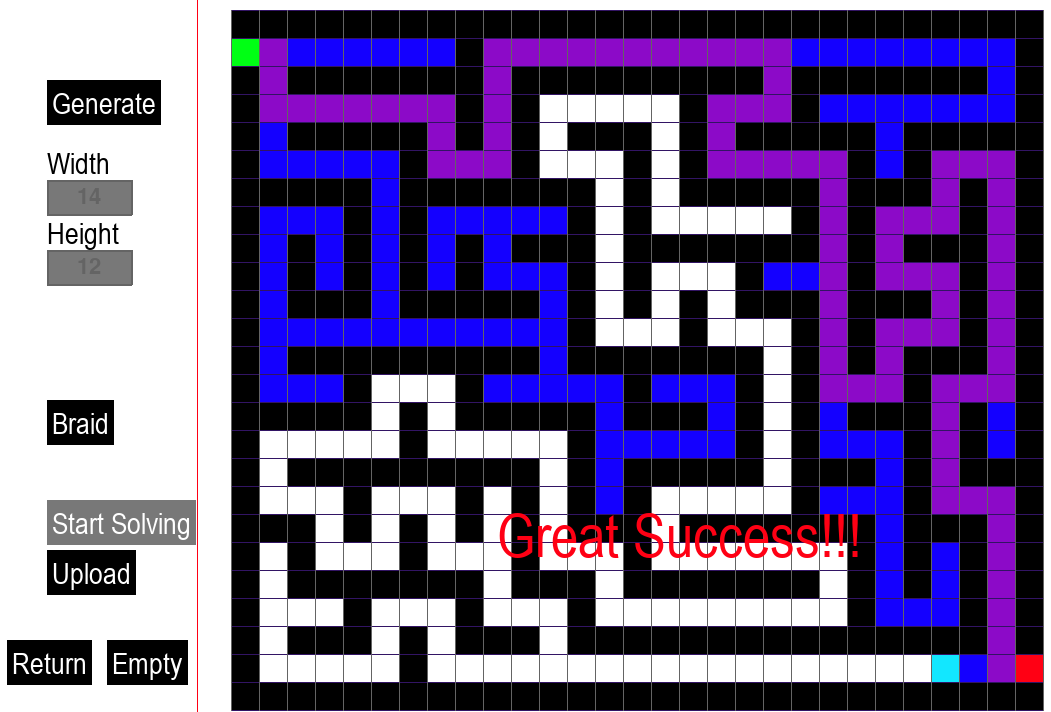


Maze is braided, meaning that most of the dead ends are deleted and more loops are created. This could make a maze more interesting to solve as it could be easier and harder at the same time

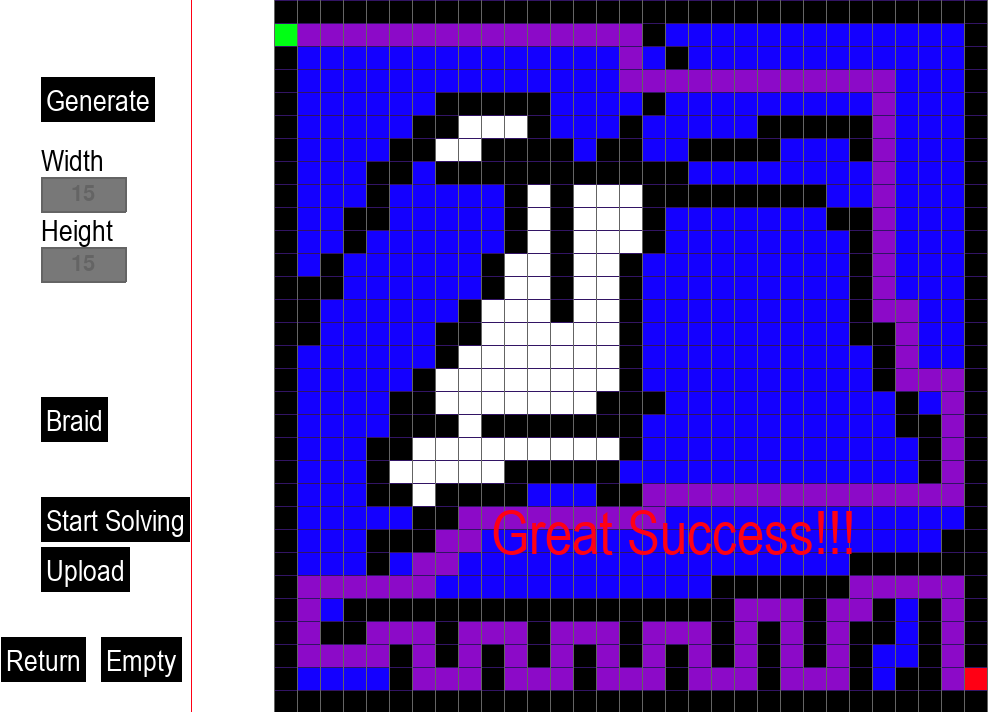
Test number 25.3



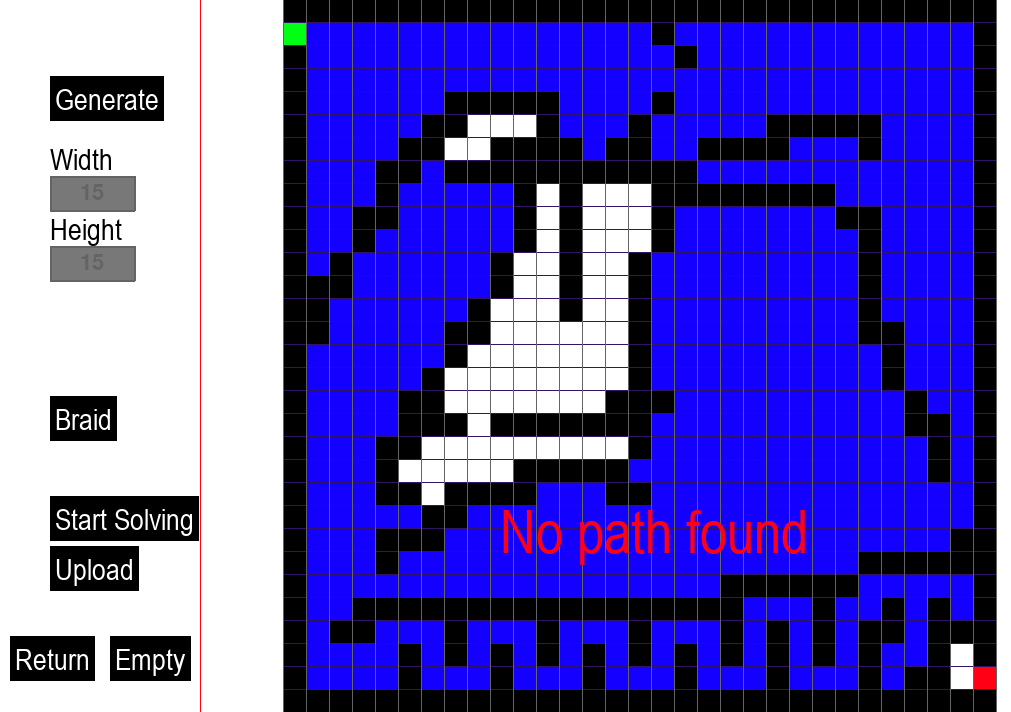
Test number 26.1



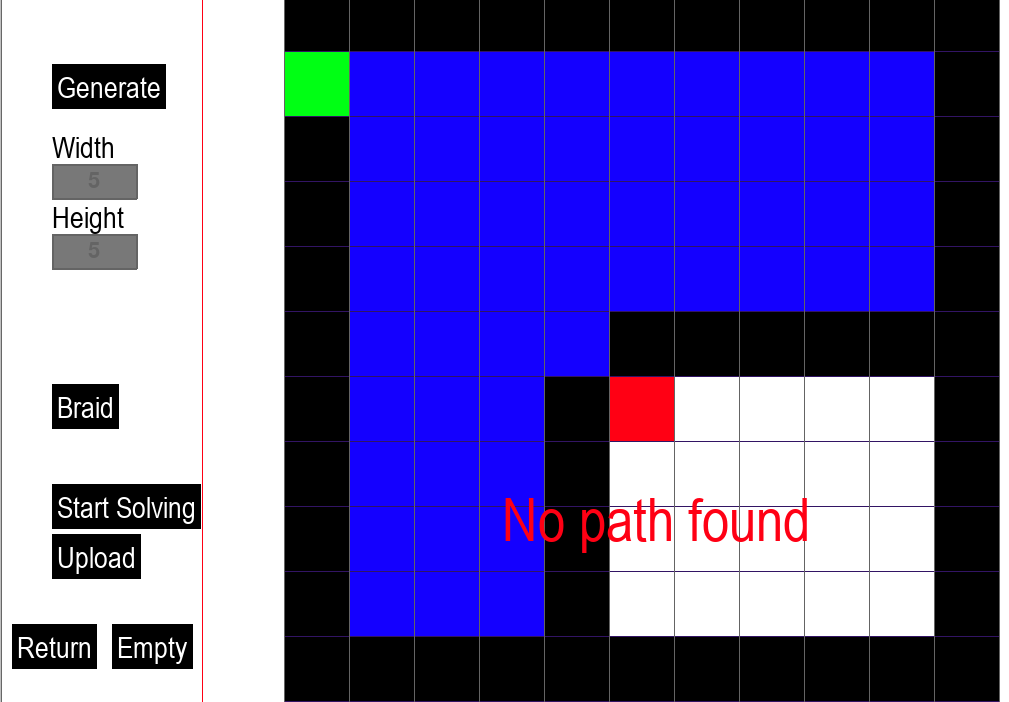
Test number 26.3



Test number 26.4

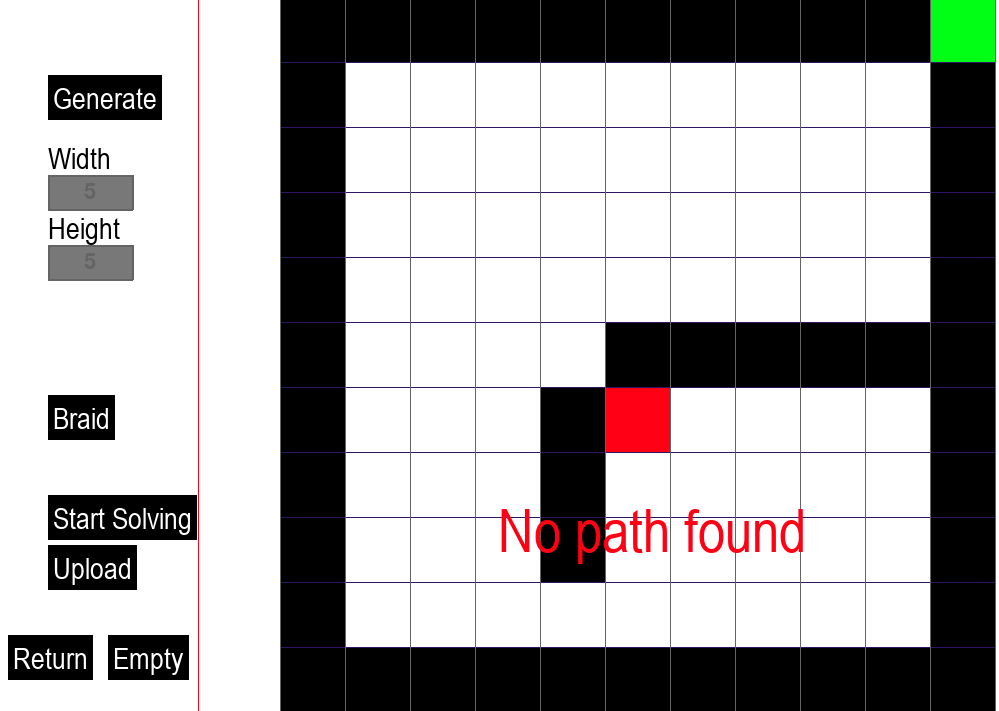


Test number 26.5





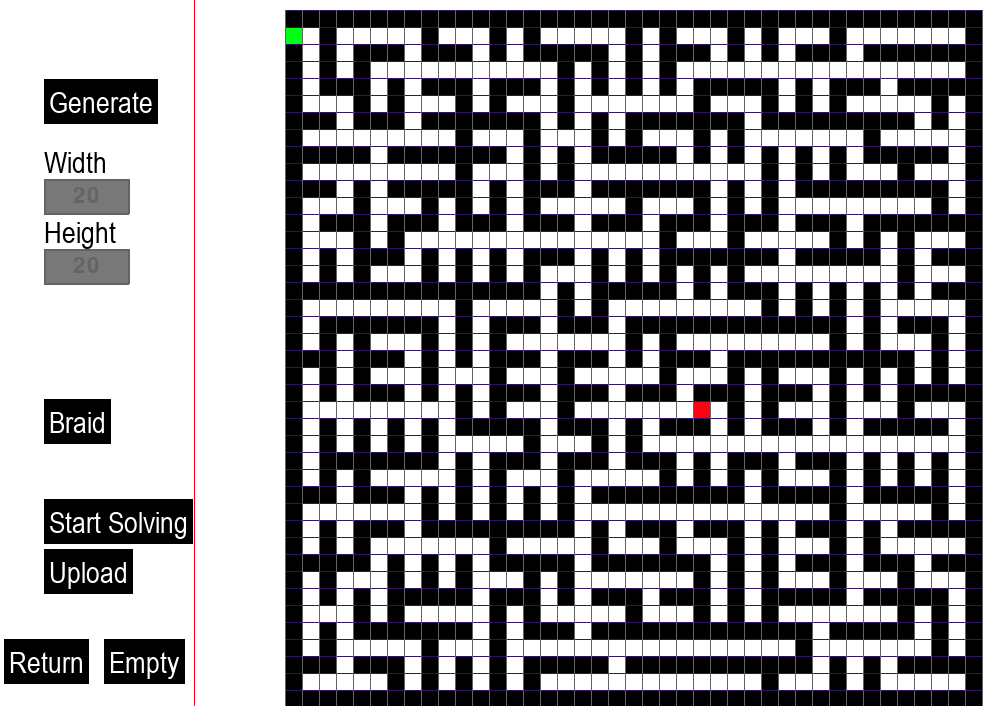
Test number 26.6



Test number 27



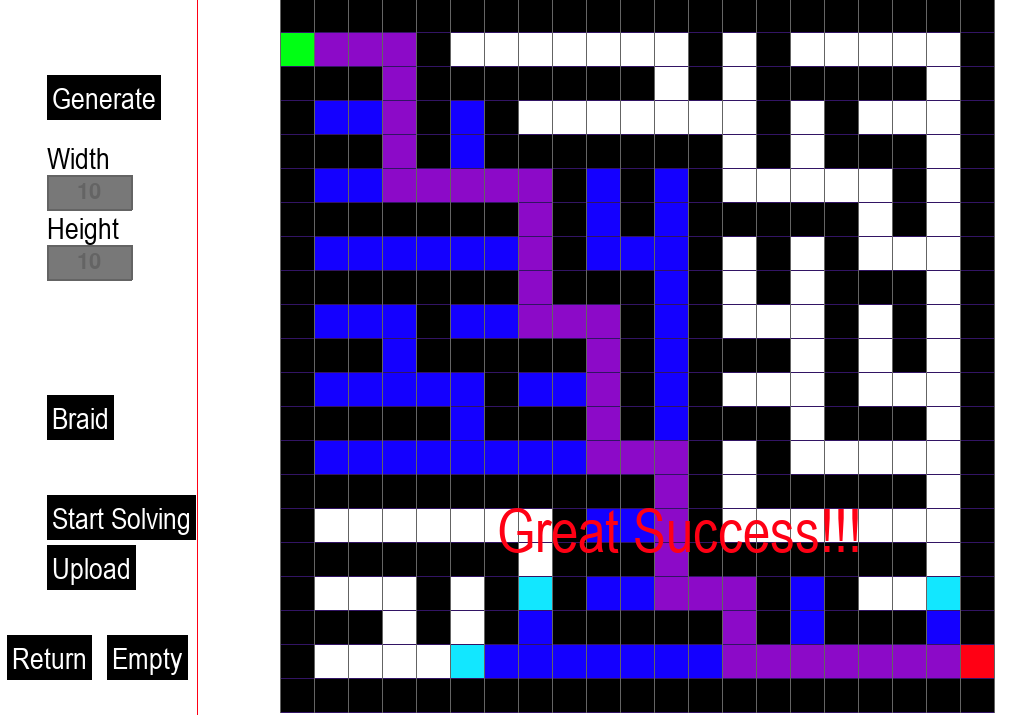
Test number 28



Test number 29



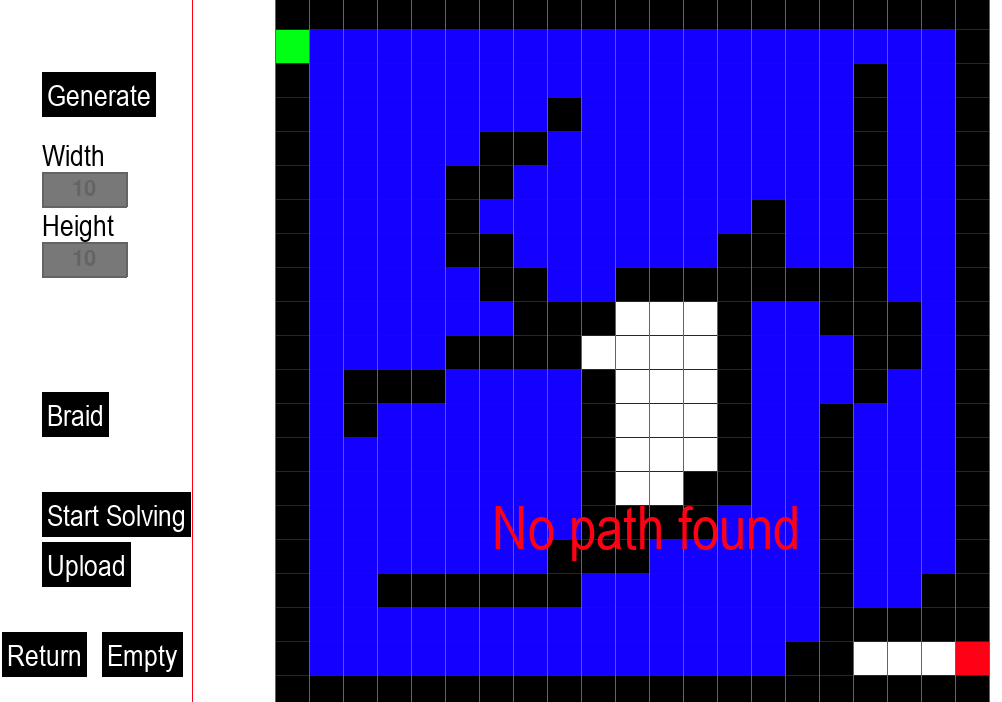
Test number 30.1



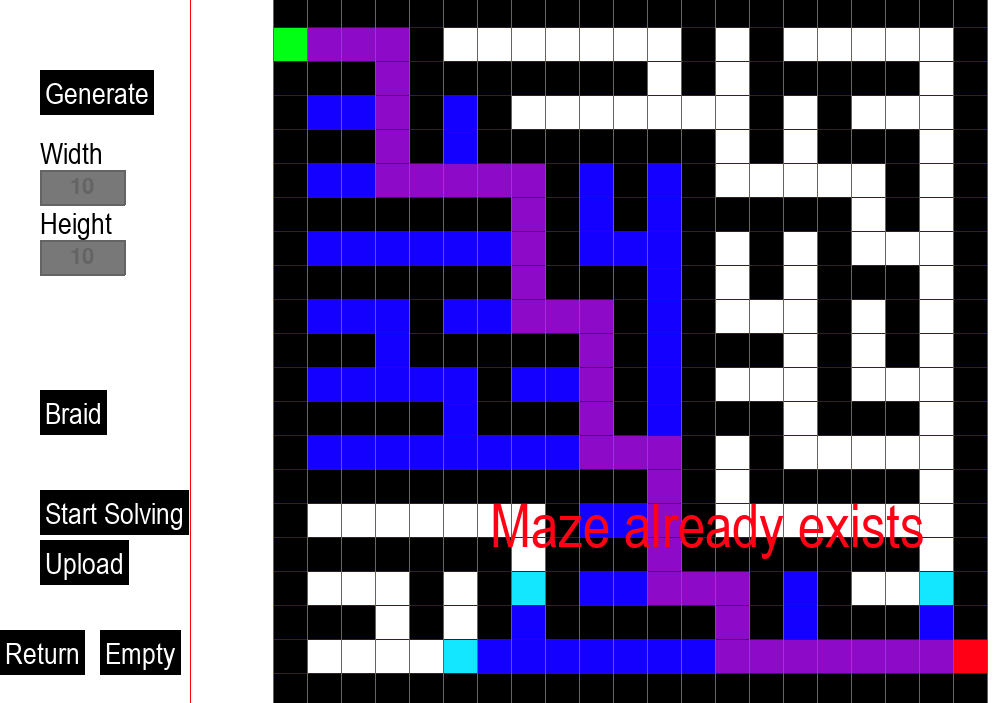
Test number 30.3



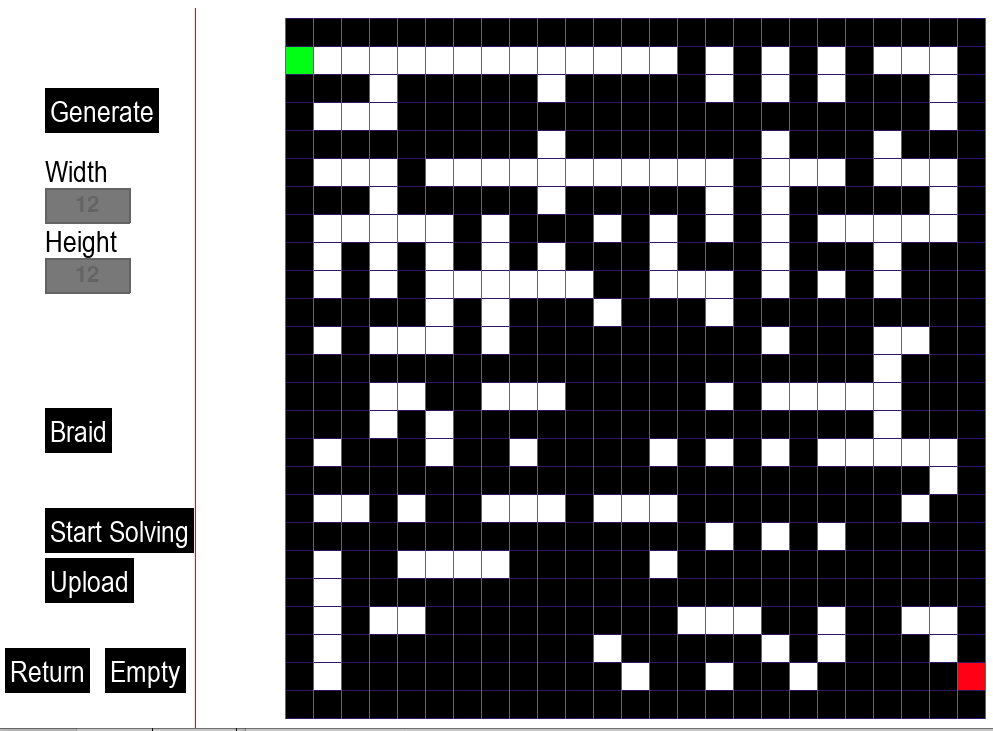
Test number 30.4



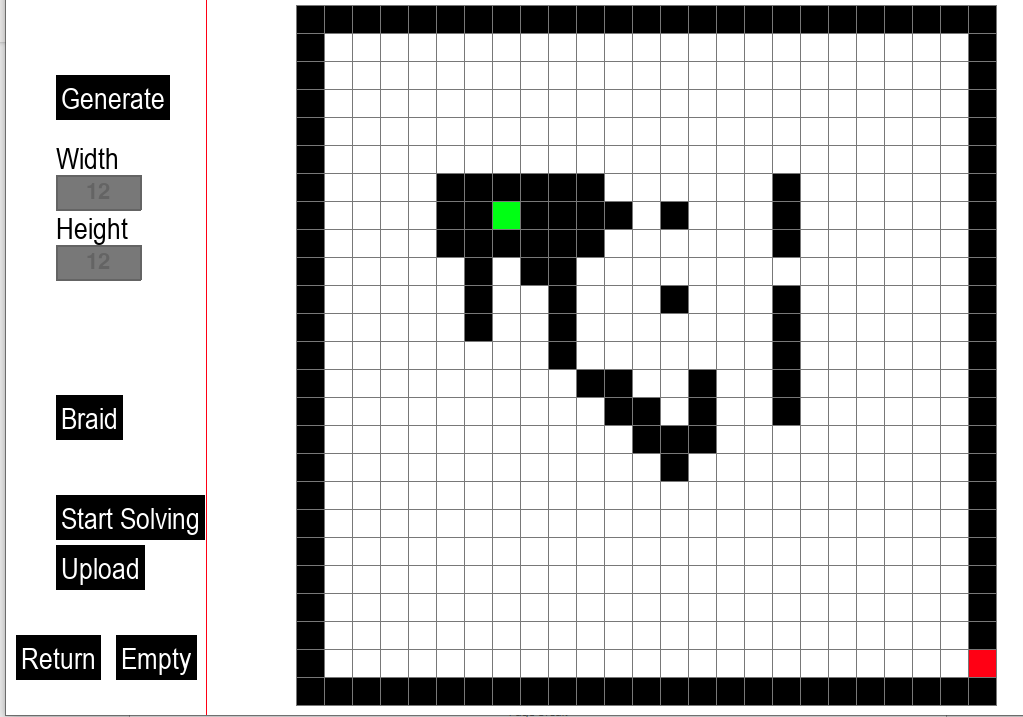
Test number 30.5



Test number 31.1

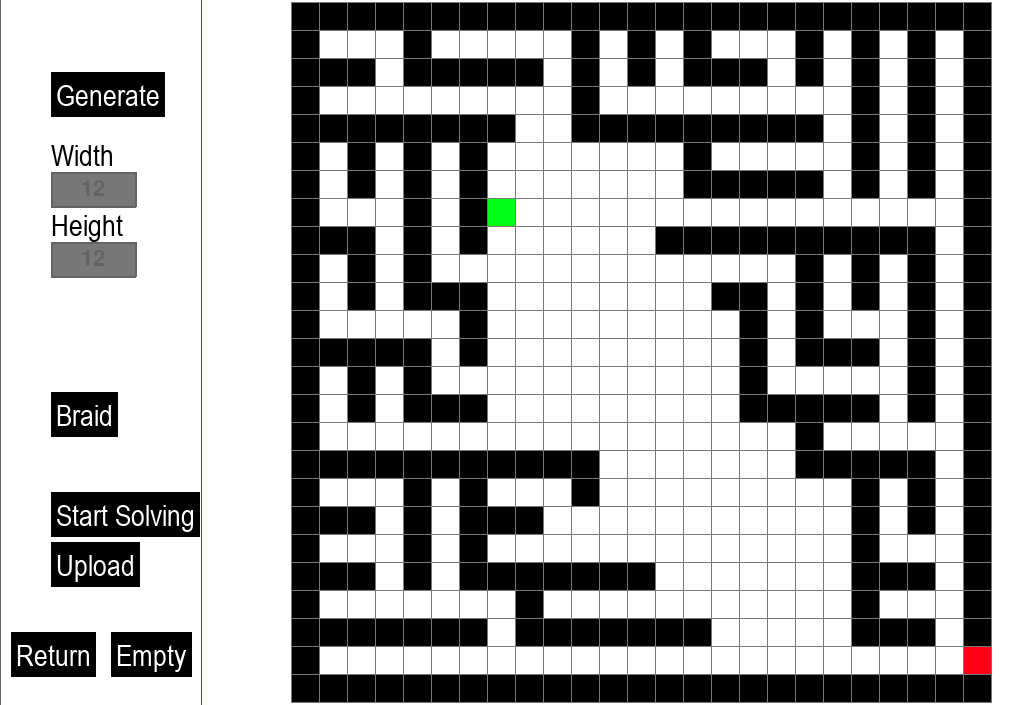


Test number 31.3

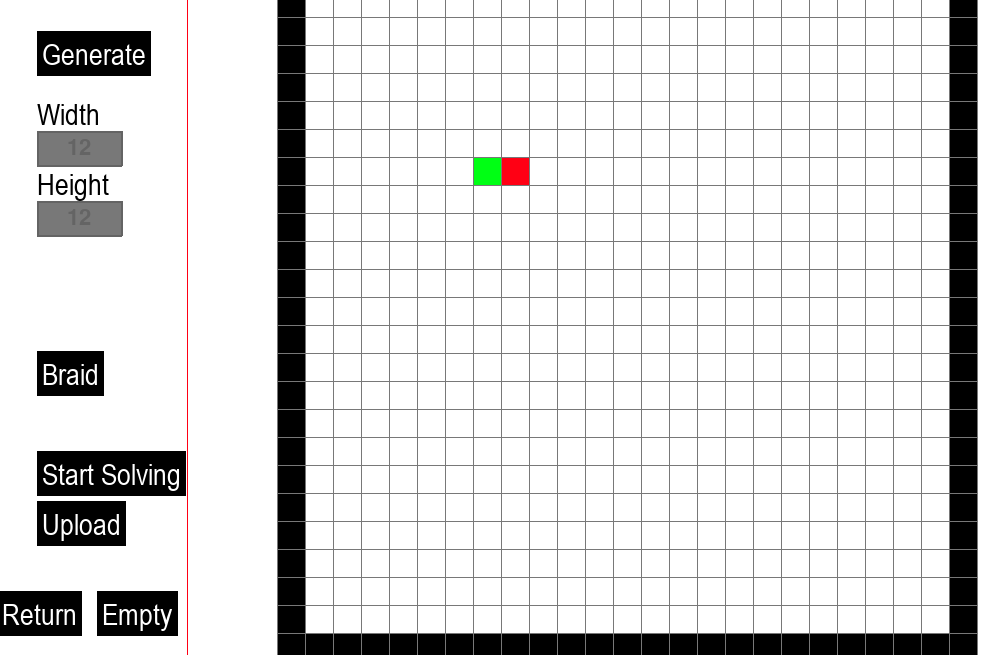


This screenshot doesn’t show the mouse position, but during making it, the cursor was over green cell and left mouse button was pressed

Test number 32.1

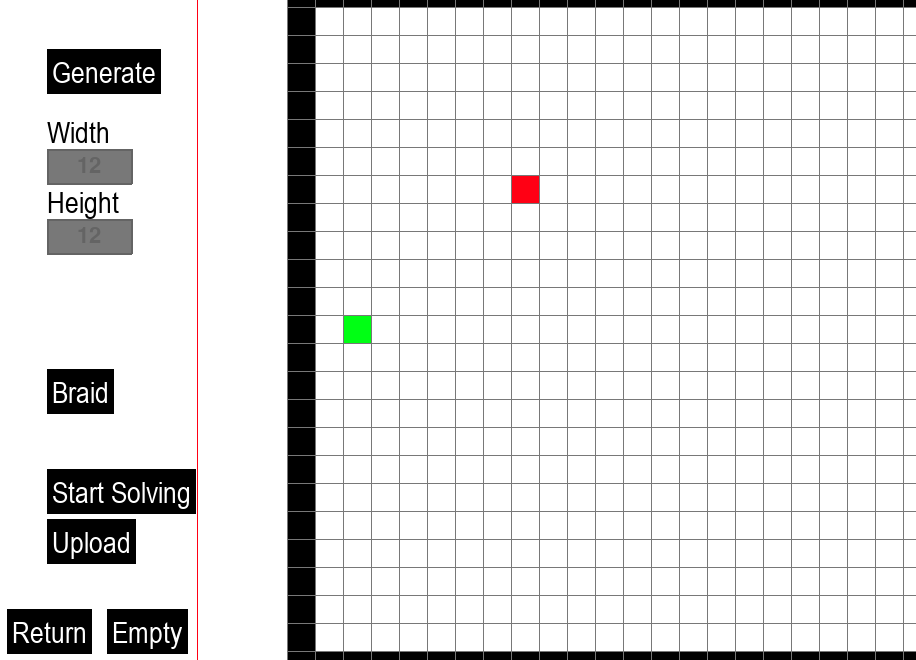


Test number 32.4

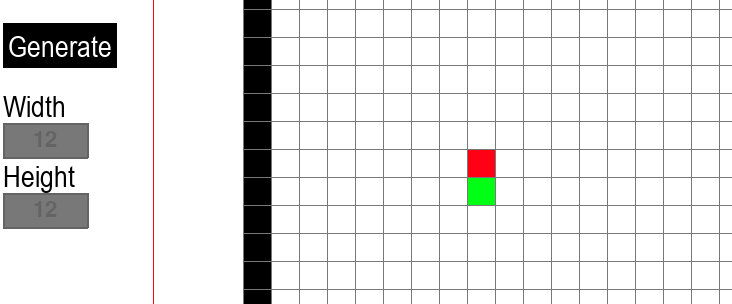


Screenshot doesn’t show it, but my cursor was over the end node. Before that I manually deleted all cells with right click

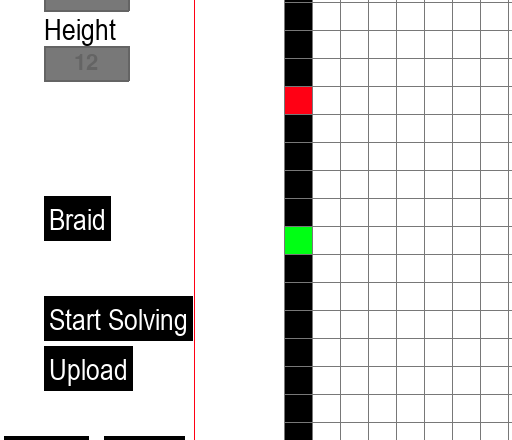
Test number 33.1



Test number 33.2



Test number 33.3



# Evaluation

## Objectives to be met

In analysis part of a write-up, I set up following objectives to be met. For easier reference, I copied them down here:

1. The system has to be intuitive - **Met**
   1. Buttons have to change colour when cursor is over them
   2. Input boxes should display different colour when not selected
   3. All buttons should be named in intuitive way and say what their function is
2. To proceed to the next level, previous must be solved - **Met**
   1. When the level is solved, text is displayed to show that the level was solved
      1. Text contains number of steps made by the user
      2. Text contains congratulation message
   2. Path that the user took is displayed with
   3. If in-built levels, if 1st level is solved, 2nd must be unlocked
   4. If last level is solved, nothing is done, as no more in-built levels exist
3. Several maze creating algorithms have to be implemented - **Met**
   1. At least 3 maze creating algorithms to be implemented
   2. Random algorithm is chosen to create a maze in a Sandbox
4. New player will be able to register - **Met**
   1. Username is checked against database, if it’s a copy, message is displayed
   2. Passwords are validated to increase security
   3. If any of passwords or username are not valid, message is shown
      1. If password 1 is incorrect, message like Password 1 is incorrect shown
      2. If Password 2 is incorrect, message like Password 2 is incorrect shown
      3. If Username is incorrect, message like Username is not valid shown
   4. Additional questions asked to allow password restoring
   5. If any of the security questions didn’t pass validation, messages are shown
5. User should be able to log in - **Met**
   1. Check credentials before logging in
      1. If credentials are incorrect, error message is shown
      2. If username doesn’t exist, error message is shown
6. Passwords should be stored secure - **Met**
   1. Encryption or hashing algorithm used
7. Users should be able to upload mazes - **Met**
   1. Maze is checked for validity
      1. Solve the maze before uploading it
      2. If path exists, message of success is shown
   2. Maze is checked if it was uploaded earlier on, otherwise error shown
      1. Base128 is checked with table Maze and if record is fetched, no upload is done
8. Users will be able to solve mazes uploaded by users in a workshop - **Met**
   1. User can select any level in a workshop
9. User will be able to solve levels with their devices – **Partially Met**
   1. User will be able to use arrow keys to solve levels
   2. User will be able to use WASD keys to solve levels
   3. User will be able to drag mouse to solve levels
10. Each user who forgot their password will be able to restore it anytime - **Met**
    1. User will be asked for their security question to answer
    2. Passwords are validated and error messages shown if not valid
    3. Records are updated and password changes allowing user to log in by using new password
11. User is able to return to a previous menu screen at any time
    1. By pressing “Return”, the user comes back to a previous screen - **Met**
    2. By pressing “Quit”, the user quits the program

## Meeting objectives

1) The system has to be intuitive

As shown in tests, buttons and input boxes have different outline and change colours when user hovers the cursor over them. It helps to identify interactable elements and makes it easier to work with the interface. First levels are very straight-forward and will be used for adaptation mostly. When user adapts to interface and controls, harder levels will take place.

2) To proceed to the next level, previous must be solved

Like in any game with multiple levels, like Vector or Cut the Rope, to unlock nth level n-1th level must be solved. This allows user to follow the structure of levels and not skip easy levels. This feature is done by fetching all solved levels from a table UserMaze, which contains all sessions of mazes solved by users. All records are fetched and added 1 to maximum level fetched. This algorithm allows new registered user to access level 1, as 0 + 1 = 1. As a part of testing, it was important to make sure that when user solves 15th in-built level, 16th doesn’t appear, as it would break the system. Because each maze has Boolean parameter for In-built, its easier to identify a type of maze and where it belongs.

3) Several maze creating algorithms have to be implemented

It was important for me to implement several maze building algorithms, as it makes more different variants of mazes, which look and behave differently, which makes it harder for user to adapt for each type of maze. However, my initial idea of a pop-up where user can select an algorithm to practice only 1 type. This idea wasn’t implemented due to its complexity, but could be released in a new version

4) New player will be able to register

Any user is able to register by clicking the button “Create new account”, which will take them to a register window where a couple of security questions will be asked for restoring the password. It is important to allow anyone to register, because it attracts stakeholders to register and play the game, which gives a room for improvement due to mass of feedback given.

5) User should be able to log in

Any registered user is able to log in without experiencing any problems as long as the password is correctly inputted. When the user inputs non-existing username into username field, program displays an error of a user doesn’t exist

6) Passwords should be stored secure

Python default library Hashlib is used to hash a password. It means that only brute-force is able to crack it as there is no way of reversing hashing algorithm. According to different websites, a password with 8 characters, at least 1 upper, at least 1 number and at least 1 symbol can be very long to crack and could take up to several centuries to do.

7) Users should be able to upload mazes

User is able to upload mazes in Sandbox menu. Any type of maze can be uploaded, included fully and partially drawn by a user. Several checks are done to prevent unsolvable mazes to be uploaded, as it would make a game unfair and less fun. Also, check is performed to avoid uploading the same level twice. It is important as 1 user can upload 100s of the same level, which would look horrible on the screen.

8) Users will be able to solve mazes uploaded by users in a workshop

Any user is able to solve any level from Workshop straight from the start. They don’t require to solve first level to unlock workshop. Levels in workshop are all unlocked, as they aren’t staged. The same way of solving is implemented as in Levels for no adaptation time would be wasted.

9) User will be able to solve levels with their devices

This is the only objective, which was met only partially. I haven’t implemented an option to solve the maze by WASD keys or by dragging start to end. Apart from that, solving and moving nodes is implemented and tested not to give any errors if node is trying to be moved outside of the screen or into the wall.

10) Each user who forgot their password will be able to restore it anytime

Any registered user is able to restore their password to restore access to solving and creating mazes. It is important to validate user’s identity, therefore security questions are asked. Several checks are performed and tested in Objective test 10.

11) User is able to return to a previous menu screen at any time

By implementing a stack-like structure of calling menus, user is able to return to a previous stage of program at any time. This is useful, as by just calling menus, instead of popping them out of stack, user will receive python error of stack overflow, which will crash the program.

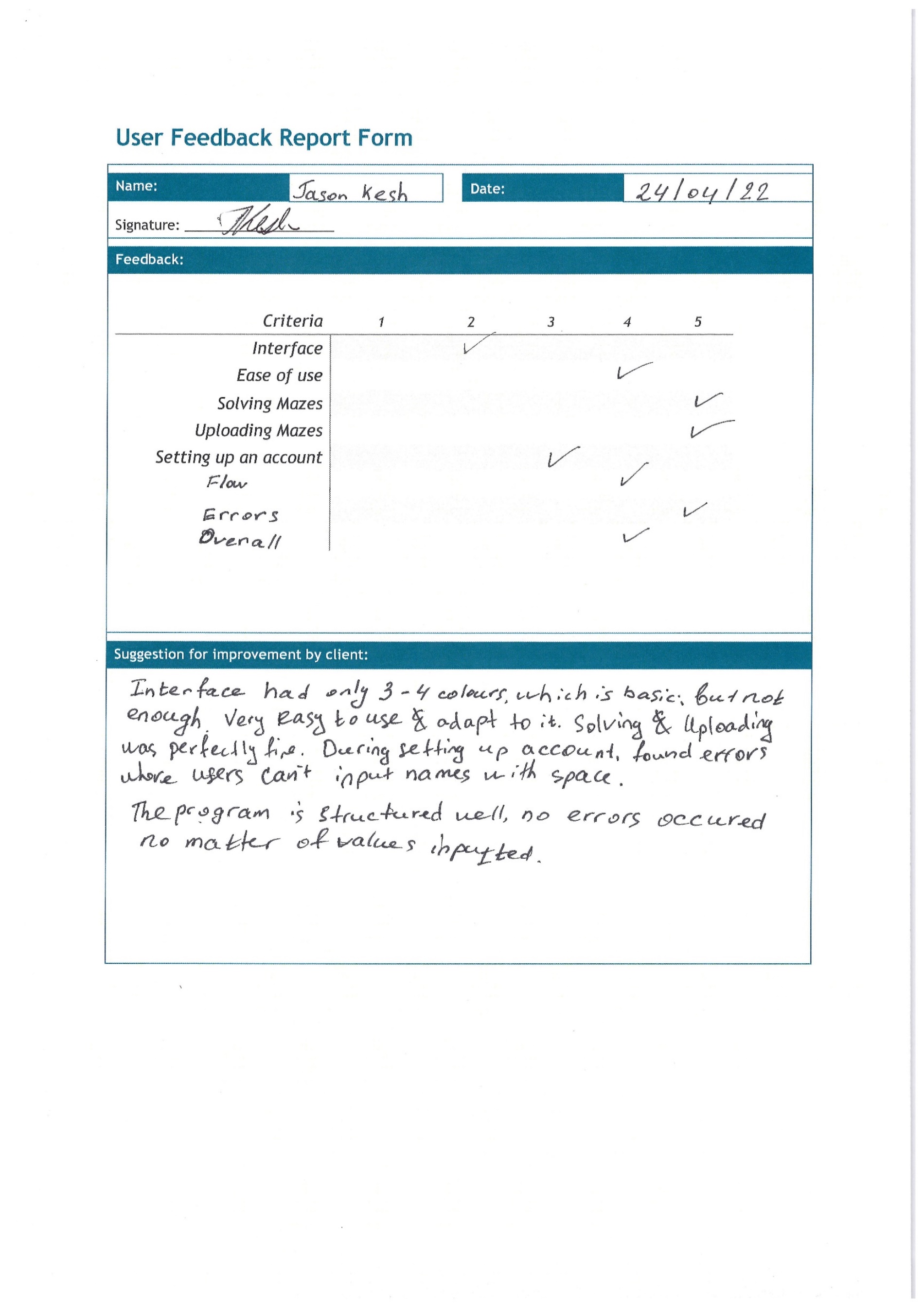
## Independent Feedback

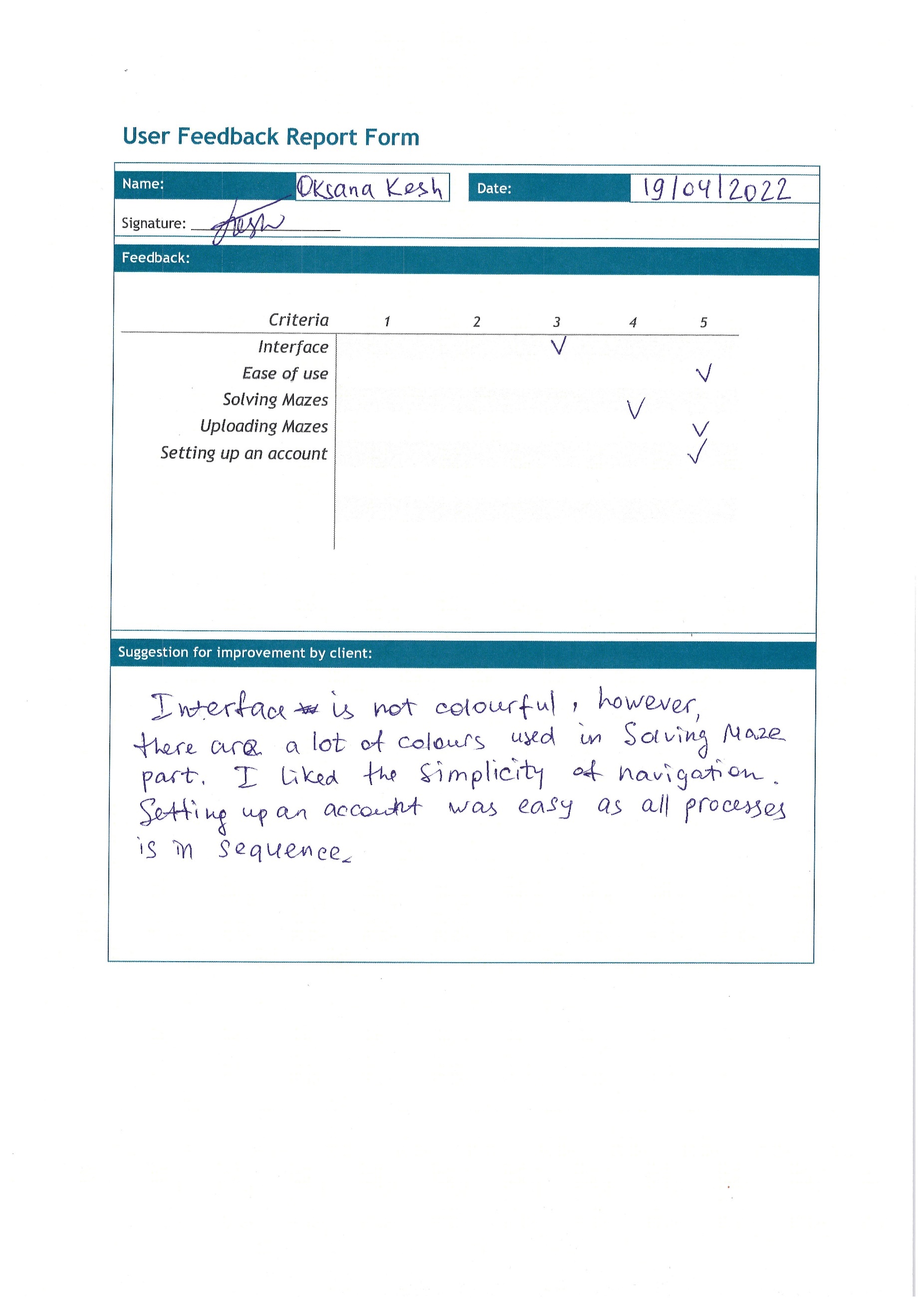
### Feedback Form

This feedback would allow user to rate main program functions out of 5, which will give me an idea of areas of improvement as well suggestions for improvement section, where user will be able to give any kind of feedback.

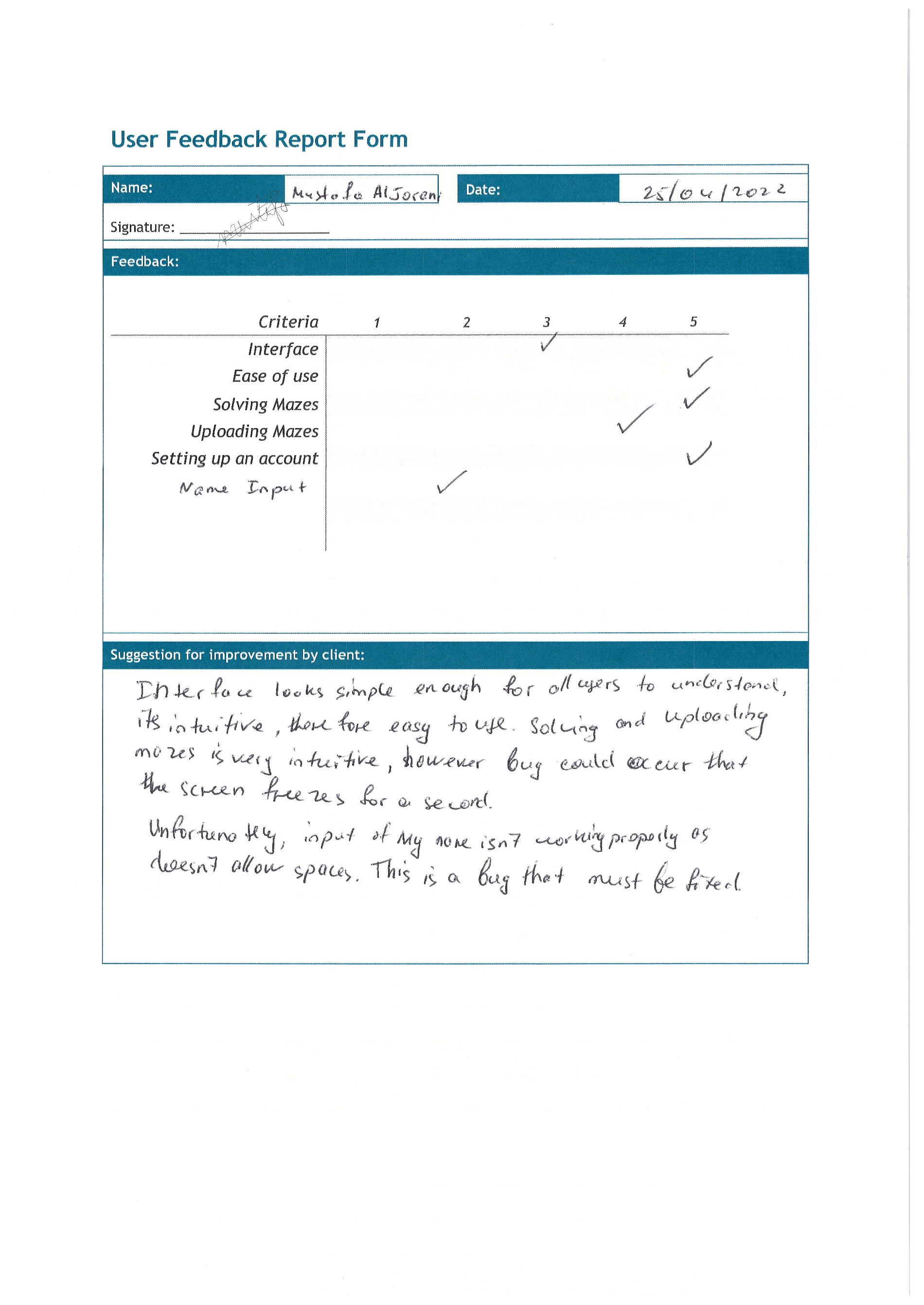
| User Feedback Report Form | | |  | |
| --- | --- | --- | --- | --- |
|  | | | | |
|  | | | | | |
| Name: |  |  | **Date:** |  | |
| Report No. (For office use only)- | | | | | |
| Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | | | | | |
|  | | | | | |
| Feedback: | | | | | |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Criteria | 1 | 2 | 3 | 4 | 5 | | Interface |  |  |  |  |  | | Ease of use |  |  |  |  |  | | Solving Mazes |  |  |  |  |  | | Uploading Mazes |  |  |  |  |  | | Setting up an account |  |  |  |  |  | | | | | | |
|  | | | | | |
| Suggestion for improvement by user: | | | | | |
|  | | | | | |

## Feedback from Users





## Feedback from client



## Analysis of Feedback

Feedback shows that there are a few weak areas of my code. The most that worried me was the interface. During developing stage, I made it as simple as possible, however, as shown I simplified it too much and by using only a few colours, I made it worse. Some personal feedback, not in a written form included that there could be an introduction to the program stating how to navigate, because moving the node by dragging is way more intuitive, than pressing arrow keys. Moreover, a lot of testers pressed arrow key button, which made sense, however, my program works by users pressing button and releasing to make a move.

## Possible Improvements

1. Due to difficulty of regex, my next stage will be to work on it and implement a regex to allow surnames like Al Jorani and St. Clair to be inputted, as not only Jordan and Clair could be
2. Different solving algorithms could be implemented. For now, only Breadth first search is done. A\* could be easy to implement as template is set and ready to function for it. Depth First search could get implemented, but due to its inefficiency and after dialogue with my client, we agreed that Depth first search shouldn’t be implemented.
3. Base 128 is a powerful encoding, but limits of it are English and Russian alphabet. I used those because they were easily accessible on my keyboard. In a future, I would like to implement Base 256 or higher using Unicode encoding inbuilt in python. This should reduce my database file size and decrease entries length.
4. For performance, I could make a higher refresh rate (currently at 60), however, it could slow some of the computers down and therefore create some advantage for users, which isn’t ideal.
5. Make more colourful interface and less sharp buttons (Ellipse) to make it look more “friendly”.
6. Make an into level to show how to operate a game and make more intuitive navigation to improve understanding of the game.