2023

Sudoku Solver and Game

A program that is used to help people visualize and interact with a Sudoku solver, while also having the ability to generate and try to solve puzzles, and load previously played puzzles.

THEO EDWARD JOHN MANTEL-COOPER

Candidate number: 5122
Centre number: 14607
Hampton School

Contents

Analysis

The first Sudoku puzzle was published in Japan in 1984, and the word “Sudoku” is a registered trademark in Japan.

The problem

Sudoku has challenged people from all around the world for decades but creating and solving the puzzles has always been a problem. Some people hand-write puzzles, taking hours meticulously trying to put numbers in a grid and solving it by hand, hoping the permutation works and is solvable.

Background

Sudoku is played in a grid of 9x9, which is further split into nine 3x3 squares. Each square, row, and column, need to contain the numbers 1 through 9, without repeating any numbers in the row, column, or square. The puzzles come with a minimum of 17 pre-filled numbers for only one solution.

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

Figure 1: An example Sudoku puzzle

My solution

Therefore, the focus of this project is to build a program which:

- Allows a user to create a Sudoku board that they can challenge themselves on, of varying difficulty
- Gives the user a platform to solve their own problems.
  - These could be puzzles they created, or puzzles they have found from other sources (the internet, newspapers etc)
  - The user can input the numbers into the program, and it can give the user the solved board.

Backtracking Algorithm

The solving part of my program will utilise the backtracking algorithm to complete the board. The steps taken are as follows:

- Find an empty cell
- Starting from the number 1, try it in said square and check if it is valid with the rows, columns, and boxes
    - If that number doesn't work, go up by 1
- If that number works, put it in the empty cell and move onto the next empty cell
- If at any point none of the numbers from 1-9 work by the rules of Sudoku, it "backtracks" and goes to the last cell where it placed a value and goes up by 1 etc.

In a coding implementation of this algorithm, recursion is used to continuously call the function to backtrack and solve the puzzle.

Who will this be for

The main group to benefit from this program is anyone interested in Sudoku!

Some examples may be: anyone who wants a program to practise their Sudoku, anyone who is up for a challenge, anyone who wants an answer to an existing Sudoku, or anyone just interested in how a computer goes about solving a Sudoku puzzle. This may help people feel more confident in their Sudoku solving abilities and hopefully helps them solve future problems.

In particular, my sister has been doing Sudoku puzzles for as long as I can remember, and she often gets frustrated when she makes little progress and is stuck. She can use this as a tool to aid solving her puzzles, making and saving progress, and exploring the solver to see how a computer tackles the problem.

Anyone wanting to generate their own Sudoku puzzles will also benefit from this project, as it has a feature which allows for puzzle generation as well. My father can use this to aid running the COVID vaccination sites. Many people waiting for their vaccination feel uneasy and nervous, so my father has been searching for a solution to calm people down. Since Sudoku is such a well-known brain teaser, I have decided to implement this into my project to allow the vaccine sites to hand out little puzzles to calm people's minds before their vaccination.

Existing system

I visited the vaccination site where my father works and observed what happens on the site. The people waiting for their jabs come in, speak to the GP who asks them questions, and when they are through, they sit in a waiting room, waiting to be called up to get their vaccine. As I mentioned, people seem agitated and bored waiting, so I decided to give people some sample sudoku's I had brought from my house.

People appreciated the gesture, and even if they were stuck and could not solve the puzzle, it passed the time for them, and they felt more at ease when they eventually got called up to get their vaccination. This showed that people are less stressed and anxious when their mind is occupied, and sudoku is a perfect puzzle for that because everyone knows the rules, and it is equally stimulating, challenging, and rewarding.

I believe that people waiting for their vaccines can be given a laptop with my program on it, and as they are waiting for their vaccine they can play, experiment, and solve some of the challenges my program throws at them. This will help them feel more relaxed and they can even save progress to their account if they want to come back to the problem!

Sudoku solver 1

Back on the computer, I researched some online sudoku solver programs. I came across the website: https://sudokuspoiler.com/sudoku/sudoku9

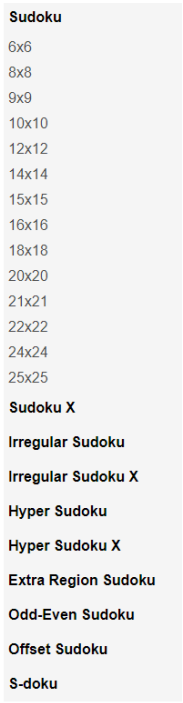


Figure 3: Online Sudoku solver 1 drop-down menu

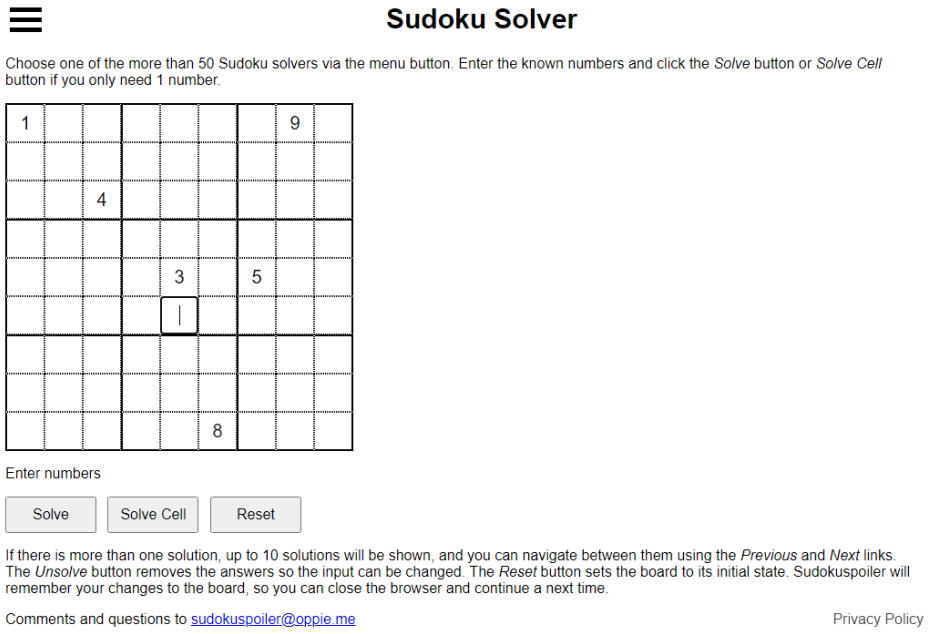


Figure 2: Online Sudoku solver 1

Features:

- Multiple different sized boards (Figure 3)
- Ability to type in your own problems
- Solve a cell, or the whole board

What I like:

- The reset key if you want to start again from scratch
- You can experiment and play around with different sized boards and try to create your own problems
- The UI seems friendly, not too confusing, and easy-to-use

What I would change:

- **Ability to create your own problems to solve yourself**
- Account sign-in to store previous solves etc
- The ability to actually play the game, not just solve your own problems

Sudoku solver 2

I then found another online Sudoku solver: https://www.sudoku-solutions.com/

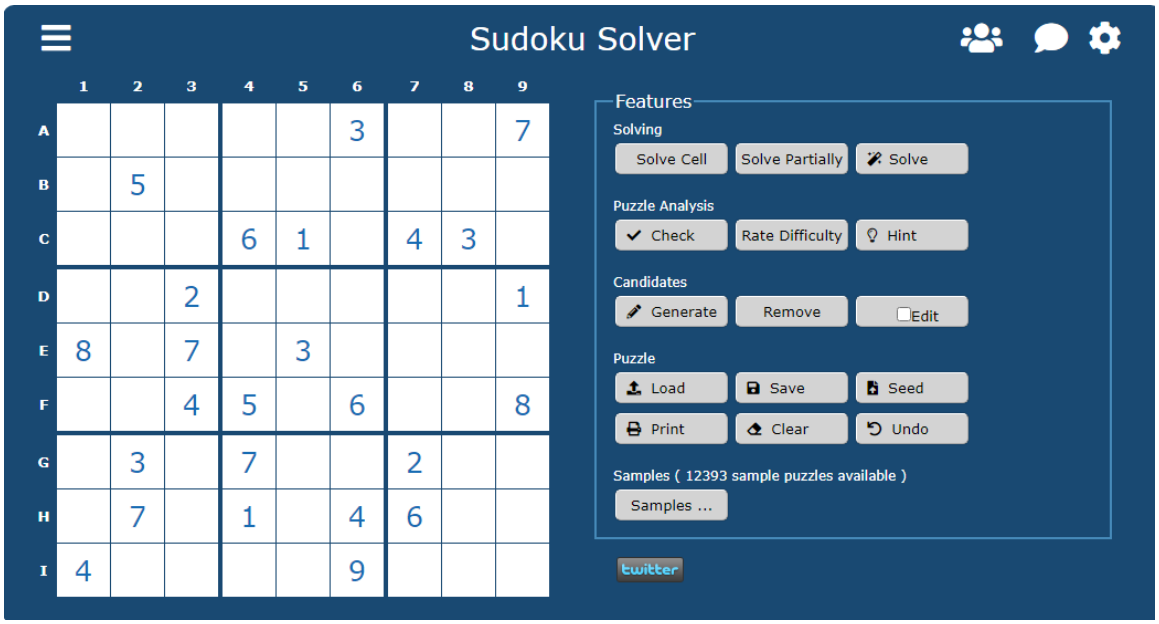


Figure 4: Online Sudoku solver 2

Features:

- Teaches you how to solve the puzzles using different methods
- Gives you hints
- Solves partially, solves individual cells, or the whole puzzle

What I like:

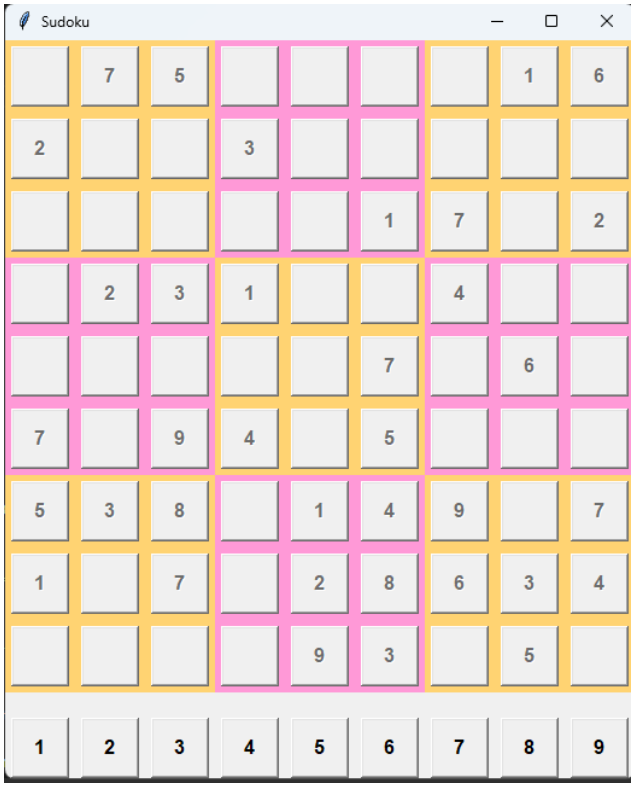
- Has built-in samples of varying difficulty
- Can save and load previous puzzles
- Undo button
- Can actually play Sudoku by placing your own numbers and “pencilling” in the small numbers if you are unsure if that is the final number for that box

What I would change:

- Simpler UI – I am a bit confused as to what all the buttons do and feel a bit overwhelmed by all the buttons
- Accounts – you can store and load puzzles, but accounts make it easier as they are tied to your account and only you can access them.

Prototypes

My project has been mainly procedural, editing as I go through, constantly improving the build. However, before I had all my windows interacting, I started off with a simple board using pre-generated puzzles as a proof of concept. Here is my earliest documented design of the build:



As you can see, the colour scheme is not amazing, and the UI is limited, or non-existent.

However, this early build taught me so much. For example, I knew that I had to add lots of buttons to help the user navigate through menus, different modes, such as New Game and Free Play. This also informed me that I need game actions, such as return to create a new game, undo, clear, and I needed user accounts for progress to be made, saved, and loaded.

However, I noticed that the actual game UI looked clear and established, so while I changed the colour scheme, I kept the layout of the buttons and 2 different colours to distinguish “blocks” in the Sudoku. I also liked the blurred out state of the game buttons so the user can identify that they are part of the game and should not be tampered with, and they **cannot** be tampered with as they are unclickable.

This early prototype, combined with my interview with my end user (below) ultimately influenced the requirements for this project.

Interview with end user – my sister (Alyssa)

Q1: When struggling on a sudoku, what do you generally do?

A1: I normally go onto my computer and look up a website or program to help me, any sudoku solving tool online is normally helpful

Abstracting A1: I need to create a tool that works on computers and is easily accessible

Q2: When using an online sudoku solving tool, what do you look for in a system?

A2: I like the programs and websites where I can input my own problems and it solves it for me. Sometimes when I do not have a physical Sudoku to hand, I like to solve the puzzles using the program

Abstracting A2: The user should be able to input their own numbers and solve their challenges using the program. The program should also generate puzzles so the user can play the game.

Q3: What features would you like in a computer program specifically designed to solve your Sudoku problems?

A3: Well, I would love to be able to input my problem obviously. Also, sometimes I cannot finish a puzzle in one go so I would like to have my own account and continue where I left off. Different difficulties to solve would be a good challenge as well.

Abstracting A3: There should be different difficulty puzzles generated, and usernames and accounts should be featured to save progress.

Q4: What would you want this program to look like?

A4: As long as it is simple to use, I don't think it has to be too fancy or complicated. Possibly just the board and a few buttons, well labelled to not confuse me or anyone else using it.

Abstracting A4: A simple user interface, but a GUI with buttons and the board

Objectives

After the discussion with my potential end user, and looking back at my original first prototype build, here are my proposed objectives for this program. Most of these objectives have been directly from the interview and discussion with my client, some are based off functions of existing systems showed previously, my first build, and how I would improve them.

1. Upon opening, user should be presented with a window which welcomes them and has 3 buttons:
    1.1 New game (bullet point 2)
    1.2 Free play (bullet point 4)
    1.3 How to play (bullet point 5)

2. New Game
    2.1 Opens a select difficulty page which has 3 options:
        - Easy
        - Medium
        - Hard

3. Game window
    3.1 Once a difficulty has been selected, a new window will open with a generated board of the selected difficulty
        3.1.1 A 9x9 grid of buttons
        3.1.2 Some numbers already in place from the generated puzzle
            3.1.2.1 These numbers' buttons will be unclickable, so the user cannot overwrite them
        3.1.3 A list of the numbers 1-9 beneath the board

    3.2 The user enters a number onto the board by first pressing the number button from the list of numbers beneath the board and then pressing a button on the board.
        3.2.1 If the user has not entered a number, an error should show up

    3.3 This will input the number onto the board interface.

    3.4 Sign in button
        3.4.1 Opens a new window which prompts the user to enter their:
            - Username
            - Password

    3.5 Forgotten password Window
        3.5.1 Allows the user to enter their email
        3.5.2 Email must be valid
        3.5.3 User receives an email with a code which they have to input

3.5.4 If code is valid, user can create a new, valid password which is then updated to their account.

3.5.5 User then prompted to sign in again.

3.6 There is a create account button if the user does not have an account yet

3.6.1 User enters:

- Email
- Username
- Password
- Password again for security

3.7 There must be text in every space

3.8 Usernames must be unique

3.8.1 When the user attempts to create an account, the program will check if the inputted username is already used in the database

3.8.2 If it is, the user will have to choose a new username

3.8.3 Usernames must be valid (only containing letters, numbers and "_-."

3.9 Email addresses must be valid

3.10 Passwords must be 8 characters long and contain a special character:
" #?!@$%^&*-_"

3.10.1 Both passwords must match

3.11 Passwords will be put through a hashing algorithm and stored in a database alongside the entered username and email

3.12 When a user attempts to sign in, their inputted password will be hashed, compared to the hashed password matching to the username in the database and if they match, the user will be signed in.

3.13 If they do not match, or the username is not found in the database, an incorrect username / password error window will pop up.

3.14 The user's username will be displayed. The sign in button will then change to say: "Sign out", which will sign out the user upon being clicked

3.15 The program will now know there is an active signed in user

3.16 Save button

3.16.1 The user must be signed in to use this feature

3.16.2 If they are not, an error window will pop up telling them they must be signed in

3.16.3 Opens a new window

3.16.4 Gives the user a unique Save ID which they are prompted to remember

3.16.4.1 The saved puzzle will contain the original board, and the user's inputs

3.17 Load button

3.17.1 The user must be signed in to use this feature and they must be signed into the account which saved the puzzle.

3.17.2 If they are not, an error window will pop up telling them they must be signed in

3.17.3 They can enter their Save ID they were given when saving a puzzle

3.17.4 This will then place the puzzle and their old edits onto the board, and they will be able to keep playing the puzzle until they want to save again.

3.18 Load previous starting puzzles button

3.18.1 The load previous puzzles will allow any user to load any previously generated starting boards, just by inputting the Puzzle ID

3.19 Clear button

3.19.1 Clears the board of any numbers inputted by the user, but keeps the pre-generated puzzle board numbers

3.20 Undo button

3.20.1 Undoes the previous number placed by the user

3.20.2 The row and column values of the cell placed, and the number placed by the user is pushed and stored in a stack

3.20.3 When undo is clicked, the stack is popped, and the cell with that co-ordinate is cleared

3.20.4 Can press as many times as the user wants

3.20.5 If the user has undone so many that there are no more moves to undo, or the user has pressed the undo button without placing a number first, an error will pop up saying there are no moves to undo

3.21 If user loses all 5 lives, they lose and return to the select difficulty page

3.22 Return button

3.22.1 Takes the user back to the New Game window (bullet point 2), and when they select a difficulty it generates a new puzzle

4. Free play
    4.1 A board is generated including:
        4.1.1 An empty 9x9 grid of buttons
        4.1.2 A list of the numbers 1-9 beneath the board

    4.2 Solve button
        4.2.1 The program will check if the board is valid to be solved according to the rules of Sudoku
        4.2.2 If it is valid, the board will be solved through the use of the backtracking algorithm (as discussed earlier)
        4.2.3 This will be visualised to the user, and they will see the program trying numbers and the program "backtracking" and changing its answers
        4.2.4 If the board cannot be solved because the way the user inputted numbers, an error window will pop up saying the board is invalid
    4.3 Instant solve button
        4.3.1 The program will check if the board is valid to be solved according to the rules of Sudoku
        4.3.2 If it is valid, the board will be solved, and the GUI will just update straight away to the user with the solved numbers in green.

    4.4 Undo button with the exact same functionality as before

    4.5 Wipe button
        4.5.1 Clears the entire board
        4.5.2 Gets rid of everything the user has inputted
        4.5.3 Essentially, making the board return to the start state

    4.6 Return button
        4.6.1 Takes the user back to the Welcome window (bullet point 1)

5. How to play
    5.1 A new window will open which will explain the rules of Sudoku so new players can understand the rules before jumping into a puzzle

6. Quit button
    6.1 Any time the “Quit” button is pressed, the program should terminate.

Design

Chosen Programming Language

Python

I have chosen to use Python for this project for several reasons.

Python can definitely handle this project, thanks to its built in GUI modules. Choosing between them was a challenge, as I have had experience with many of them (such as Pygame, Tkinter, PySimpleGUI). However, I decided on Tkinter for this project due to its simplistic syntax, yet scalable and advanced features.
Python also has a plethora of in-built modules to help my program, such as the "random" module used for generating random combinations of integers, the "requests" module to call APIs, and the "re" module to manage regular expressions for accepting valid email addresses, usernames, and passwords.

Python has in-built SQL modules, such as sqlite3 which meant writing and using my SQL databases with Python was a lot easier to manage, and I didn't have to install lots of external programs and modules. This also meant that I could write my queries inside my Python code and use variables etc to manipulate and view the correct data with high precision. I utilised the formatted string in Python in order to execute SQL commands inside a Python string, making it easy to add parameters into the query. An example of this is below:

```
def get_starting_board(self, saveid):
    command = (f"SELECT startingBoard FROM save, puzzle WHERE save.PuzzleID = puzzle.PuzzleID AND save.SaveID={saveid}")
    self.cur.execute(command)
```

Python is an interpreted language, which means that single lines and functions of code can be easily tested and modified without needing to recompile the whole source code, which makes debugging and editing quicker, easier, and more efficient. As the code is executed line-by-line, breakpoints can be used and I utilised VS code's feature of seeing variables and their values during runtime, which definitely cuts down on debugging time and helps me understand what problems are happening where, and how to fix them.

Finally, I have been coding in Python for several years and it is the one that I am most fluent in. I decided to use it for this project as I wanted to be coding in a language that I felt comfortable with, and one that has high levels of readability and is relatively simple to understand.

Programming techniques

I will be using Object-Oriented Programming throughout this project, using classes and objects to model parts of the program, such as the board having its own class, and each window having a class.

I will use features of OOP such as inheritance, by creating template classes and then inheriting them in other parts of the program to avoid rewriting programs, and then the

inherited classes can extend the original template class, and use polymorphism to assign new functionality to an already existing method.
Another benefit of OOP is that there is a sense of encapsulation so the users will not be able to see data about the classes, or edit them.

Also, as my finished code will be very lengthy, OOP helps to structure the code and break it up into classes which can be developed and tested alone before being integrated into the final program.

Finally, I will divide the problem into multiple Python files and each file will contain at least one class which can then be called and passed into other files to maximise readability and help the debugging stages, as files can be tested independently.

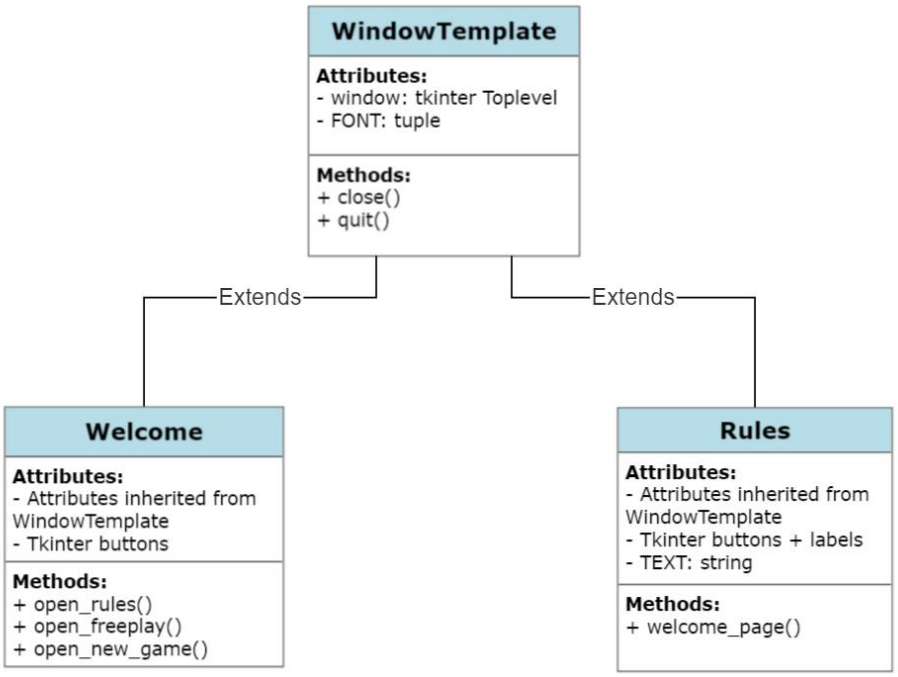
Limitations of Python

As mentioned earlier, Python is an interpreted language which also comes with some downsides. For example, the speed of running the program again and again, such as the recursive nature of the Sudoku solving algorithm proves slower compared to compiled languages such as C and C++, however I feel that this difference is minute and with optimisations, will have minimal effects on the user.

I strongly feel that even though Python has performance issues compared to other languages, the cleanliness and readability of code, combined with the in-built modules shows that Python proves to be a strong language choice and one that I am very happy with.

Class Diagrams

Opening Windows

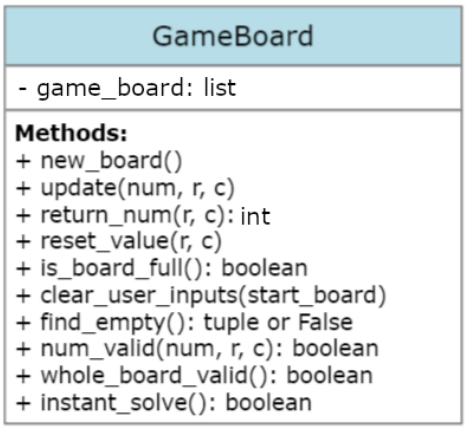


The "WindowTemplate" class serves as a template class for "Welcome" and "Rules" to inherit from. This class creates the tkinter Toplevel which is just a new window, and also stores a constant which is the font used in the text.

The "Welcome" class inherits the Toplevel from the "WindowTemplate" class, while creating its own buttons needed for the player to start the game - see the UI diagrams for the Welcome page. It also creates some methods, which each correspond to a button, so the "Rules", "Free Play" and "New Game" buttons call the "open_rules", "open_freeplay", and "open_new_game" functions respectively.

The "Rules" class also inherits the tkinter Toplevel and the FONT constant from the "WindowTemplate" class, while also having attributes such as the text to display on this window, called from an API which just stores the text. There is also a return button for the user to return back to the welcome page, which the "welcome_page" method handles by creating an instance of the "Welcome" class.

Board class



This class is the actual backend board. "game_board" is stored as a 2D list, shown in the "Internal Data Storage" section of this Design section. The methods in this class are used for manipulation of the game board, and so other parts of the program can modify / access information about the current board, stored as a 2D list.

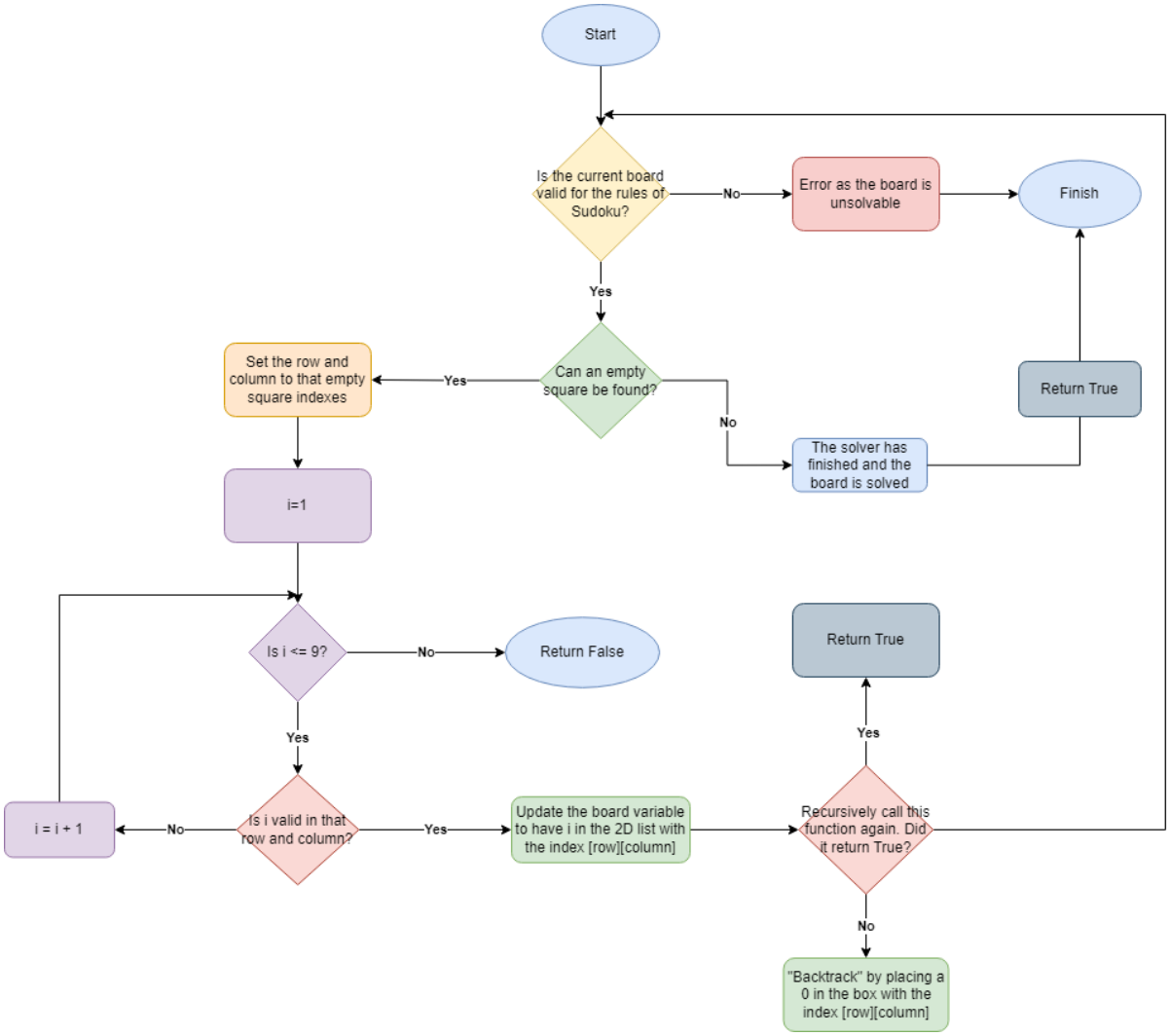
- "new_board" is used to create a new board, for example if the user creates another puzzle.
- "update" is used to place a new number in the board, as the call is required to pass a number, row, and column for manipulation of the board.
- "return_num" returns the number at those co-ordinates. This is used to check if the user is overwriting a number on the board by comparing the inputted number to the existing number at that index.
- "reset_value" is used to reset whatever number is at the specified row and column back to 0 (the default value for an empty square)
- "is_board_full" returns either True or False depending on whether there is a number in every index of the 9x9 2D list. This is used to check when the board is completed by the user to congratulate them on solving the puzzle.
- "clear_user_inputs" is used to clear only the modifications made by the user, but by requiring a "start_board" variable, the current board is set to the starting board, essentially wiping whatever numbers the users inputted. This is used when the user presses the "Clear" button
- "find_empty" is used to either return a tuple of an empty square in the 2D list board or return False to suggest that there are no empty squares. This is used by the solver to decide where to place numbers
- "num_valid" is used to check whether the given number is valid in the given indexes. Used by the solver to check if each value is valid for the rules of Sudoku in that current index in the 2D list. Returns a Boolean.
- "whole_board_valid" is used to check whether the whole board at the moment is valid according to the rules of Sudoku. Is used before the solver starts to check if the board can actually be solved, and also used when the user has finished the puzzle to determine if their solution was correct.
- "instant_solve" is the implementation of the solving algorithm, demonstrated on the flowchart.

Algorithms

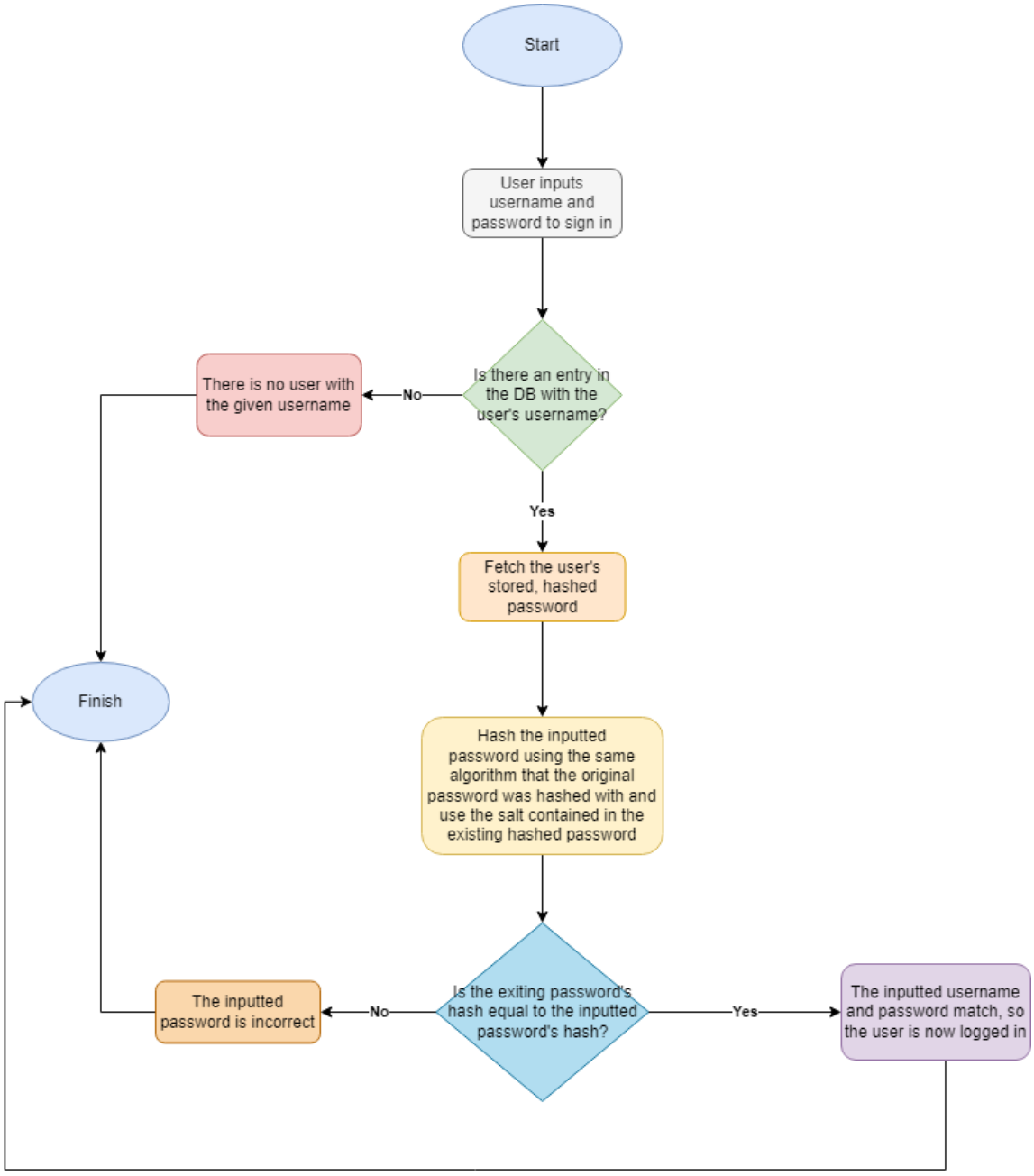
A few of the algorithms implemented throughout my program

Sudoku Solver algorithm

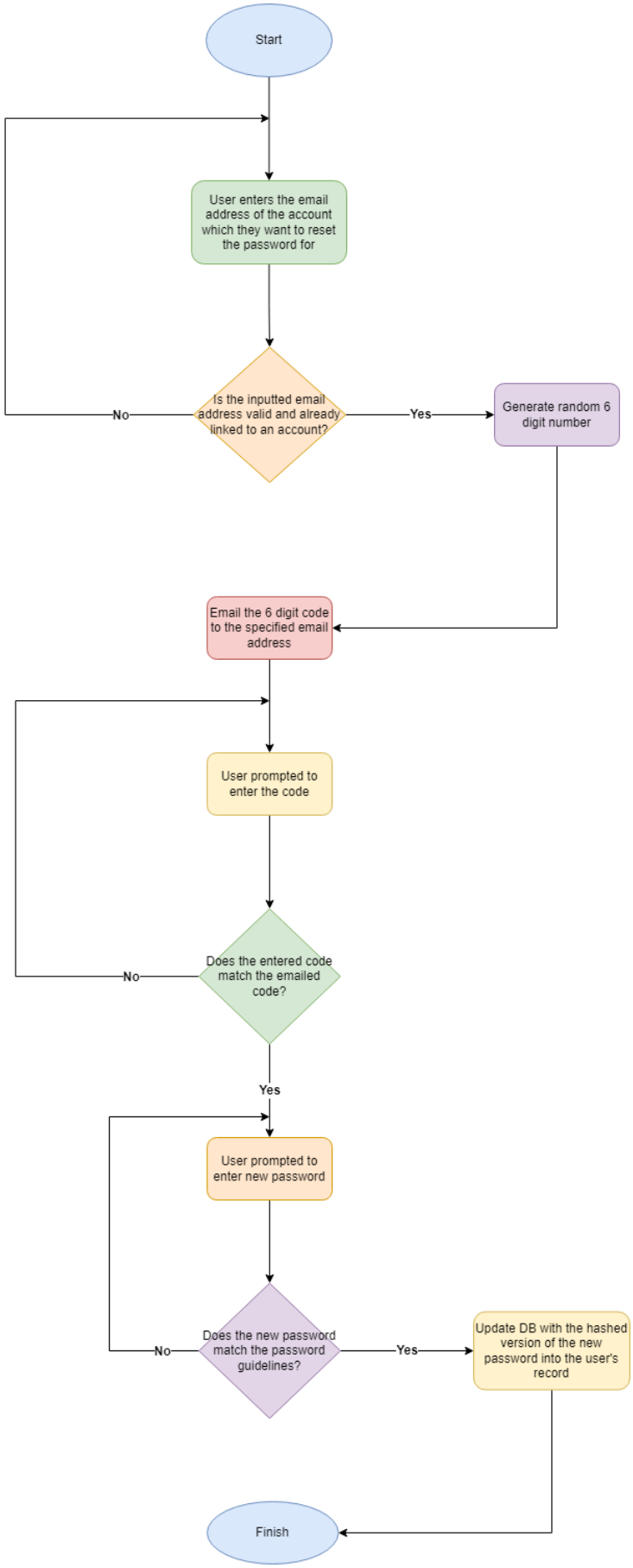
This algorithm is the most complicated, as it uses recursion to effectively "brute force" the Sudoku board by checking every number, starting with 1 and up to 9, in the first empty square, and if no numbers are valid, the solver "backtracks" by placing a 0 and returning to the previous square and it continues trying higher numbers, moving backwards and forwards as the program traverses the board.



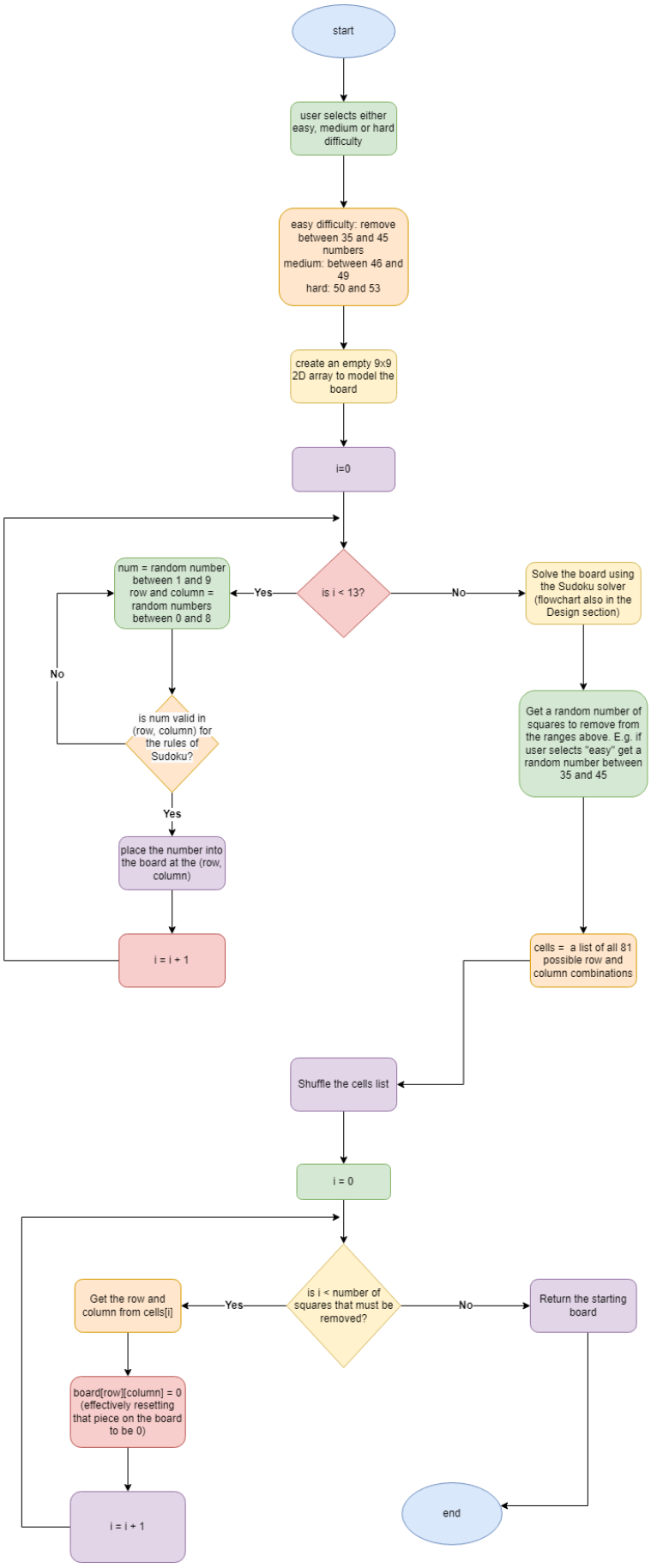
Password verification algorithm



Forgotten password algorithm



Puzzle generation algorithm



Explanation

13 random, valid numbers placed onto the board:

```
0 0 0  | 0 0 0  | 0 3 0
2 0 0  | 0 0 0  | 0 0 0
0 5 0  | 0 0 0  | 6 0 0
- - - - - - - - - - - -
6 0 0  | 0 0 0  | 0 0 2
0 0 0  | 0 0 0  | 0 8 0
0 2 0  | 0 0 0  | 7 0 0
- - - - - - - - - - - -
0 0 0  | 0 0 0  | 2 0 1
0 4 0  | 0 0 9  | 0 0 0
0 0 0  | 0 0 0  | 0 0 0
```

The solver solves the board from the previous state creating this solved board:

```
1 6 4  | 2 5 7  | 8 3 9
2 3 7  | 6 9 8  | 1 4 5
8 5 9  | 1 3 4  | 6 2 7
- - - - - - - - - - - -
6 1 5  | 8 7 3  | 4 9 2
4 7 3  | 9 1 2  | 5 8 6
9 2 8  | 5 4 6  | 7 1 3
- - - - - - - - - - - -
3 9 6  | 4 8 5  | 2 7 1
5 4 1  | 7 2 9  | 3 6 8
7 8 2  | 3 6 1  | 9 5 4
```

43 random squares have been set to 0 (as this is an easy puzzle, between 35 and 45 numbers could be removed):

```
0 0 4  | 2 0 0  | 8 0 0
0 3 7  | 0 9 0  | 1 0 0
8 5 9  | 0 3 0  | 0 0 7
- - - - - - - - - - - -
6 1 0  | 8 7 0  | 0 0 2
0 0 0  | 9 1 0  | 0 0 6
9 2 0  | 0 4 6  | 0 1 0
- - - - - - - - - - - -
3 9 0  | 0 0 5  | 0 7 0
0 0 0  | 7 0 0  | 3 6 0
0 0 0  | 3 6 1  | 9 5 4
```

This is the starting, easy difficulty puzzle which is then placed onto the GUI for the user to play. The 0's are represented as blank buttons that the user can put their numbers on, and the other numbers are placed on the GUI as disabled buttons with the number on them, so the user knows the difference between their placed numbers and the starting game numbers.

Number validation algorithm

The “num_valid” function takes in 3 inputs: the number, the row, and the column. It has to do 3 tasks to check if a number is valid for the rules of Sudoku: check there are none of the same numbers in the row, column, or 3x3 “box”.

Row

I loop through every number in the current row by using the index [row][i] with “i” being the looping variable and check if the current number is equal to the inputted number. Make sure the current square we are on is not the inputted square (because you could place e.g. 5 over an existing 5 on the same square which would be valid but according to this function, without the final check in the if loop, would deem this to be invalid). Return false if a number is found to match.

```
# Check if a given number is valid in the given cell
def num_valid(self, num, r, c):
    # Check row
    for i in range(len(self.game_board[0])):
        if self.game_board[r][i] == num and (r,i) != (r,c): # check if there are duplicate nums in the row and "i" is not the same as the column
            return False
```

Column

Similarly, loop through every number in the column by using the index [i][column] with “i” being the looping variable. Check if the current number is equal to the inputted number. Also make sure the current square is not the inputted square for the reason just provided. Return false if a number is found to match.

```
# Check column
for i in range(len(self.game_board)):
    if self.game_board[i][c] == num and (i,c) != (r,c): # check if there are duplicate nums in the column and "i" is not the same as the row
        return False
```

3x3 box

This one is more challenging, as it requires splitting the board into 3 boxes, in both the x and y, either (0, 1 or 2). The row and column is assigned a box by integer diving the row and column by 3 (as row and column will be between 0 and 8). I then loop through the box, by multiplying up by 3, and then adding 3 in both x and y to create a 3x3 grid to loop through, and again check if the current number is equal to the number inputted, while making sure we are not on the inputted square. Return False if a number is found to match.

If all of these checks are completed successfully, and no number is found to be the same in either the row, column, or 3x3 box, then return True as the inputted number is valid in that row and column.

```
for i in range(box_y * 3, box_y*3 + 3):
    for j in range(box_x * 3, box_x*3 + 3):
        if self.game_board[i][j] == num and (i,j) != (r,c):
            return False

return True
```

To check validity of the whole board, I loop through every number in the board, note the number, row, and column and pass them into this num_valid function. If every number returns True, the whole board is valid. If any number returns False, the board is not valid for the rules of Sudoku.

Implementation of the stack

The stack is implemented as a singly linked list, with the head being the most recent data item pushed onto the stack (the top item)

My stack is used so the user can "undo" previous moves, to remove previously placed numbers off the board, so the stack is popped, and GUI updated.

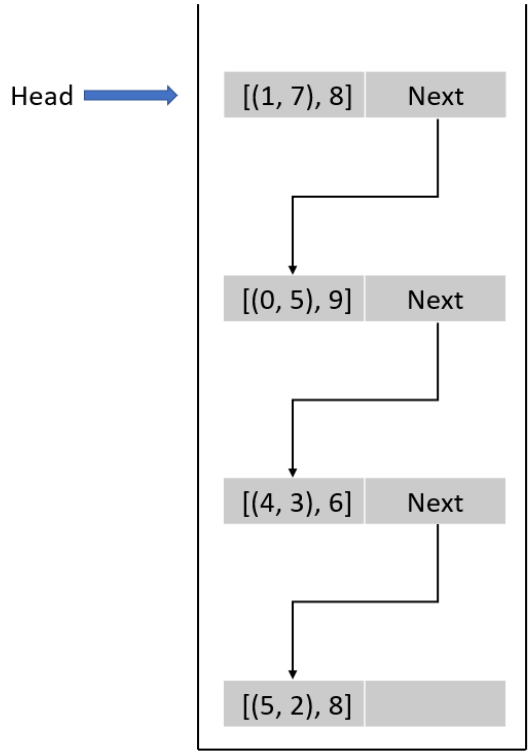
Data stored in the stack

Basic stack structure: [(row, column), number]

The row and column of the number placed is stored in a tuple as the first element in the list stored in the stack, and the second item is the number the user placed. This list is then popped or pushed from the stack accordingly.

Visual diagram of the stack implemented as a linked list

Some data has been added to the stack, as a visual of what a stack may look like during a user's attempt at playing the Sudoku. (the head is the latest number placed onto the board)



Each time a new node is pushed to the stack, the new node's next is "pointed to" the current head, and then the head is changed to be the latest node pushed to the stack.

Stack code

Node class

```
# A class for a single node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None
```

Stack class

```
class Stack:
    def __init__(self):
        self.head = None


    # Checks if stack is empty
    def is_empty(self):
        return self.head == None


    # Push something onto the stack
    def push(self, data):
        if self.is_empty():
            self.head = Node(data)
        else:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node


    # Pop the top element off the stack
    def pop(self):
        if self.is_empty():
            return None
        else:
            popping = self.head.data
            self.head = self.head.next
            return popping
```

Stack reallocation algorithm

Every move that the user places on the Sudoku board is stored in a stack, so the user can undo all of their previous moves. When an undo is done, the stack is popped so their last move is removed, and the GUI is updated. However, this runs into some issues.

- When a user clears a square with the erase button, that number may have been pressed first, or at any time so you cannot just pop the stack, you need to remove only that element
- When a user overwrites a previous square, if the old square is kept on the stack essentially they will have to press undo more times than squares on the board because their old entries will still be on the stack.

Therefore, I created a solution to remove a given element from my stack. Since I implemented the stack using a linked list, not for example a list, I cannot just use the index of the element and remove it.

My solution is outlined below:

- The stack works as such:
    - When the user places a number on the board, it is added into the stack like so: [(row, column), number]
- We "pop" the highest element from the stack and add it to a temporary stack.
- We keep doing this until we reach the element which we want to remove from the stack altogether, and pop this element, but do not push it onto the temporary stack.
- We then loop through the temporary stack, and "pop" each element and push it onto the original game stack, therefore removing the element which we wanted to remove while keeping the rest of the stack in exactly the same order.
    - If this was used as a means to remove a number, i.e. If the user cleared that square then we leave the stack as is
    - If this was used because the user overwrote a previous number they inputted, the user's new number's row, column, and value will then be pushed to the top of the game stack.

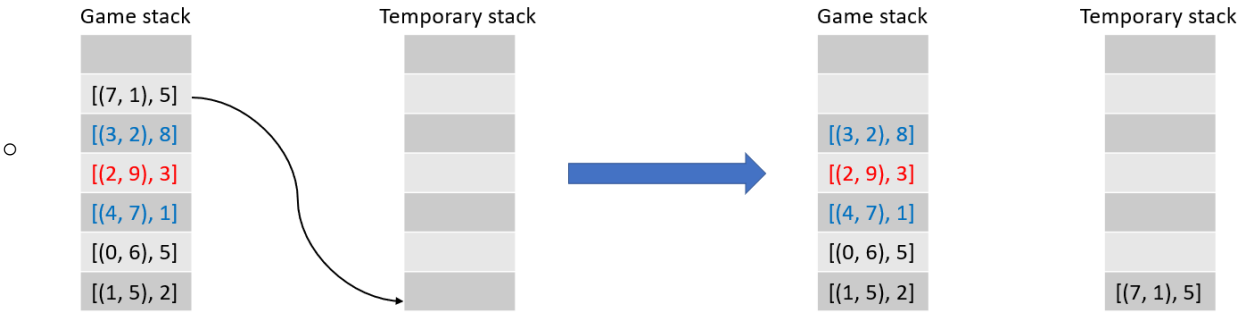
- This is an example stack if the user placed those numbers at those indexes on the board.

- Starting Game Stack (temporary stack is empty at this point):

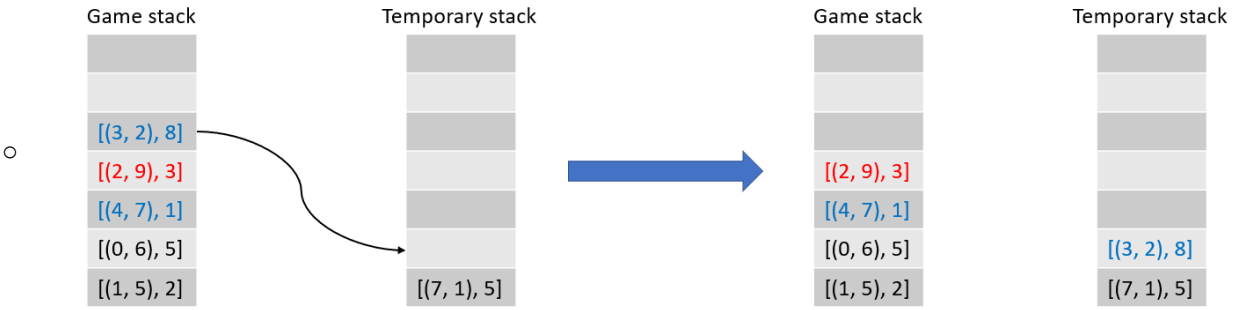
Game stack
[(7, 1), 5]
[(3, 2), 8]
[(2, 9), 3]
[(4, 7), 1]
[(0, 6), 5]
[(1, 5), 2]

- The red element is the number that we want to clear from the stack
- The blue elements are either side of the element that we want to remove so it is easier to follow
- **The process step-by-step is on the left, and to the right of the arrow shows the stacks after that process has taken place**

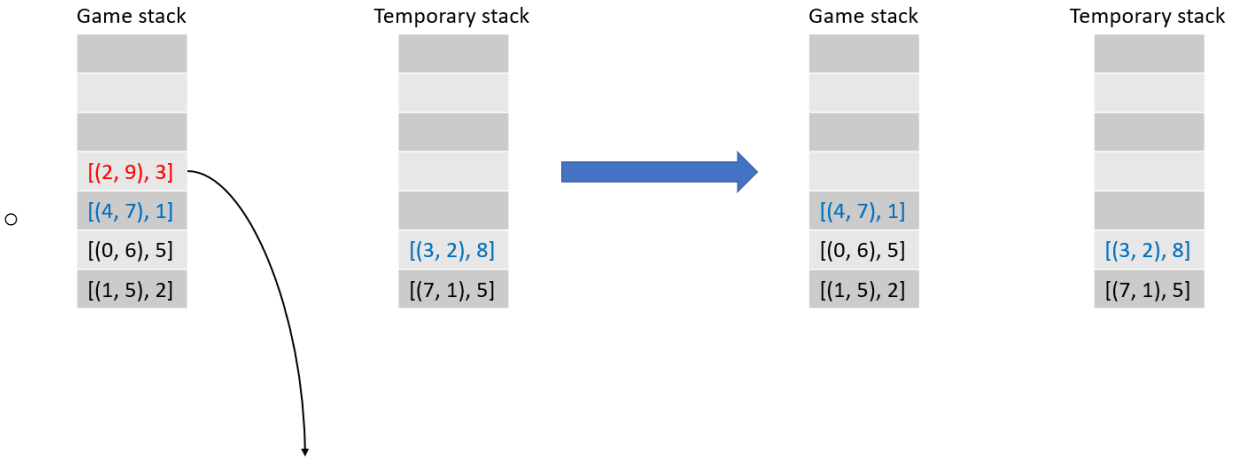
- First Element popped from main stack



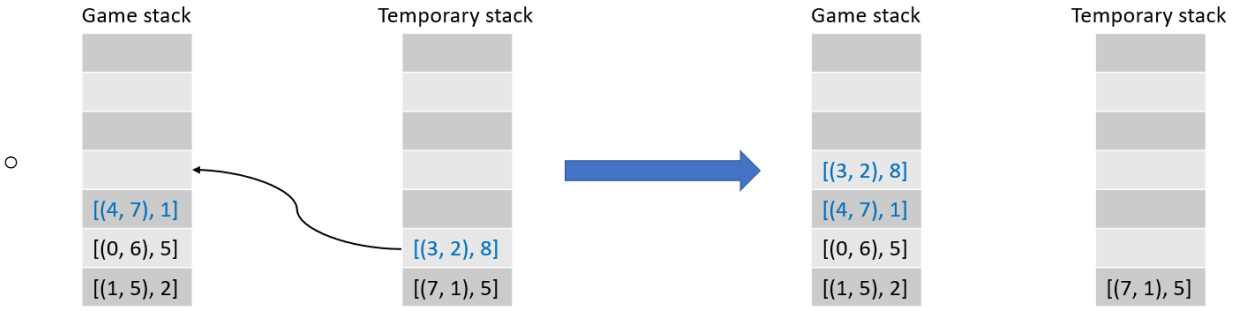
- Next element popped



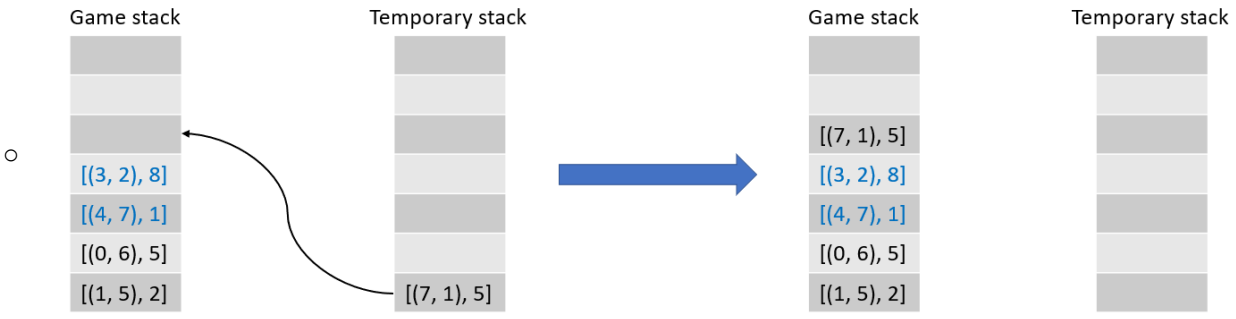
- Value that we want to get rid of is popped, and discarded



- As the value we want to go has been discarded, values from temporary stack are popped from the temporary stack and pushed onto the game stack.



- Finally, another value pushed from the temporary stack back onto the game stack



- What the game stack looked like before and after the stack re-allocation algorithm was executed:

Starting game stack
[(7, 1), 5]
[(3, 2), 8]
[(2, 9), 3]
[(4, 7), 1]
[(0, 6), 5]
[(1, 5), 2]

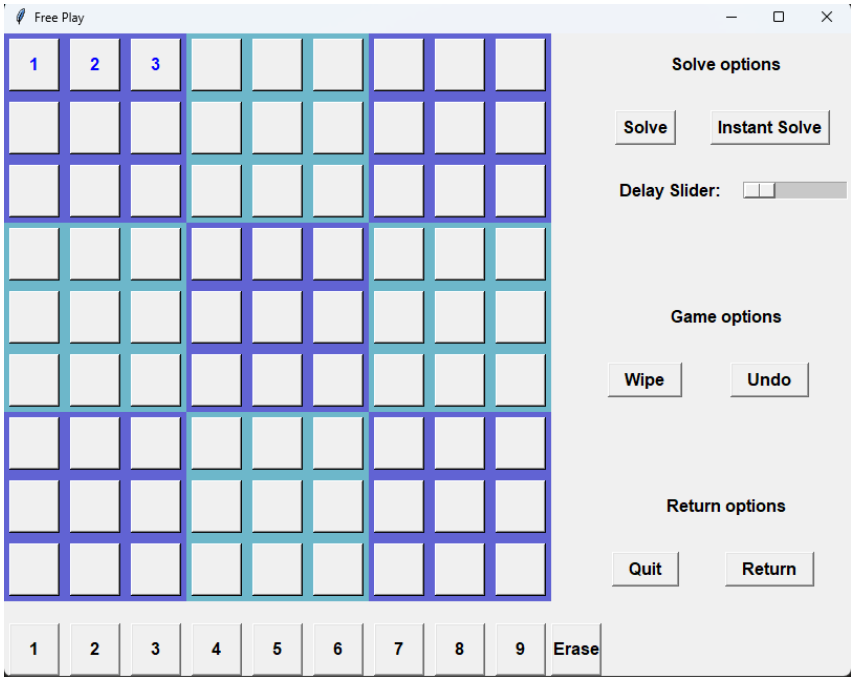
Final game stack
[(7, 1), 5]
[(3, 2), 8]
[(4, 7), 1]
[(0, 6), 5]
[(1, 5), 2]

- As you can see, the red value which we wanted to get rid of has been removed, and every other element is in the stack in the exact same order.

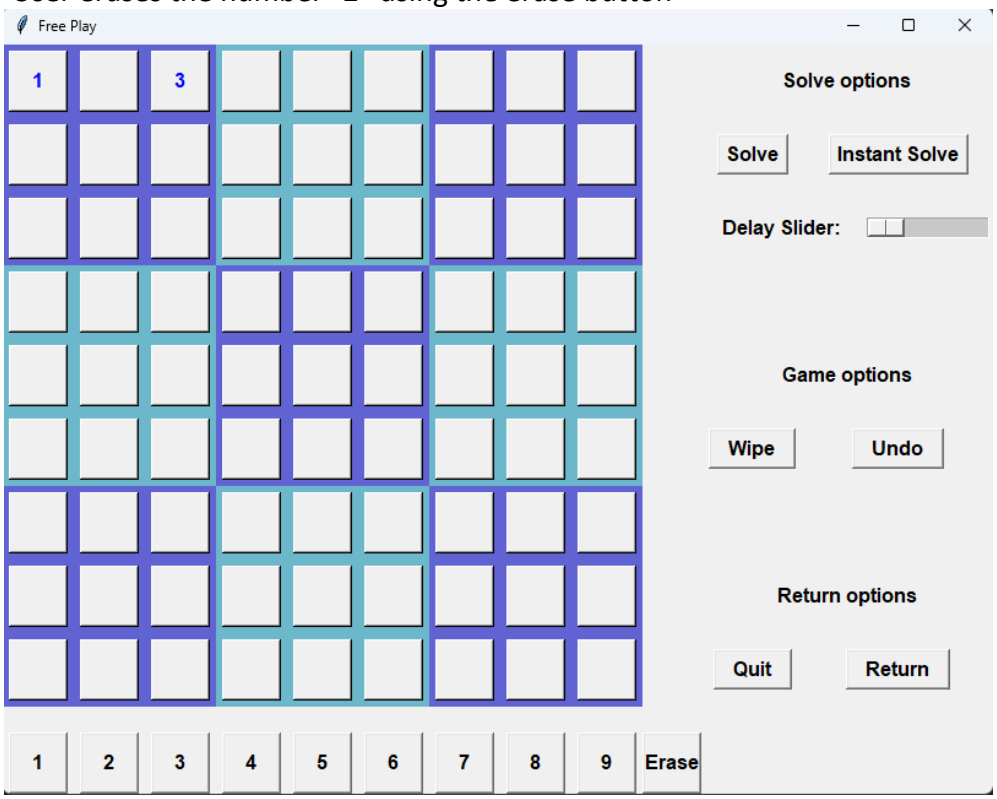
- Values would be popped and pushed to and from the temporary and game stack as many times as required until the desired element is removed, in this case only two values had to be placed onto the temporary stack and pushed back onto the game stack, but the algorithm keeps doing it until the element is removed, and then pushes every element back onto the game stack.

Stack reallocation algorithm in action

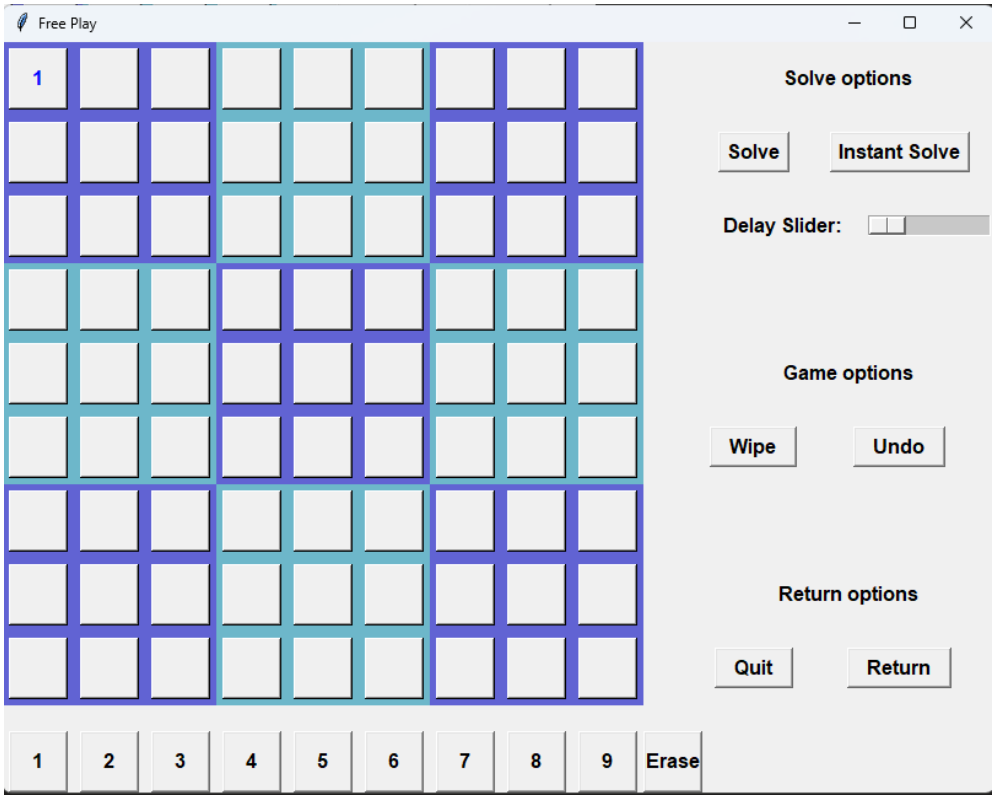
User places 3 numbers in the order: 1, 2, 3



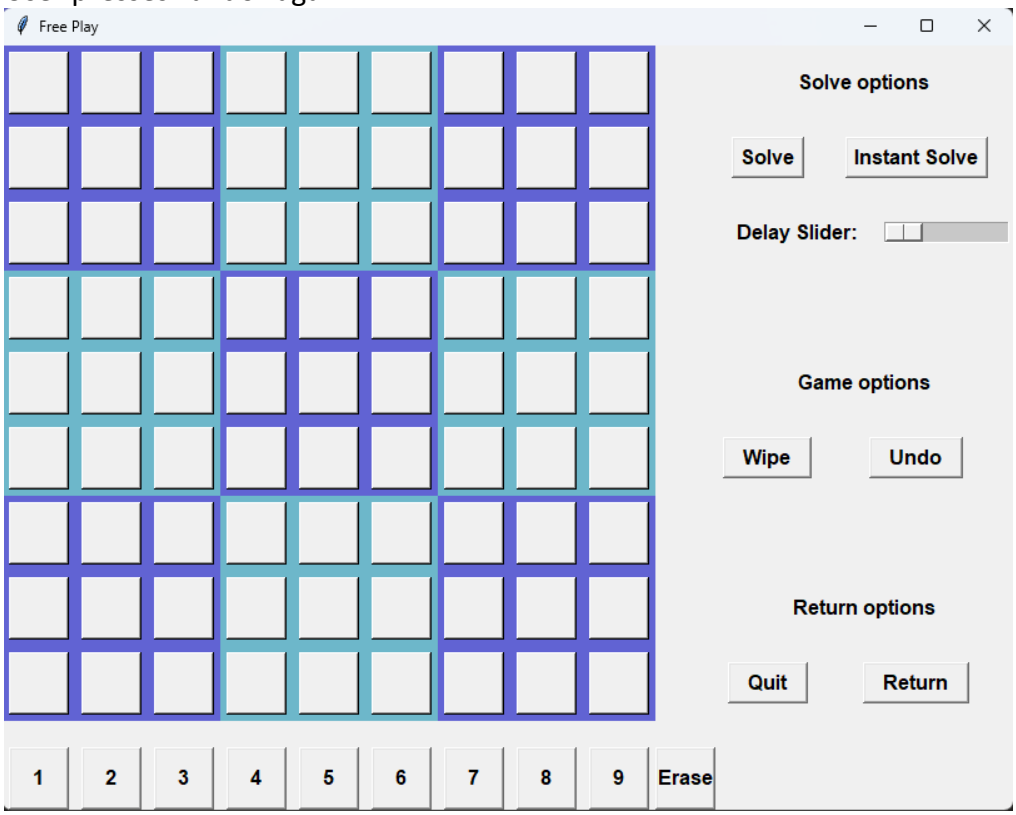
User erases the number "2" using the erase button



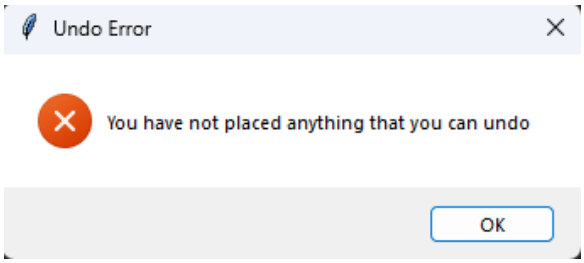
User presses "undo" once



User presses "undo" again



User presses "undo" again



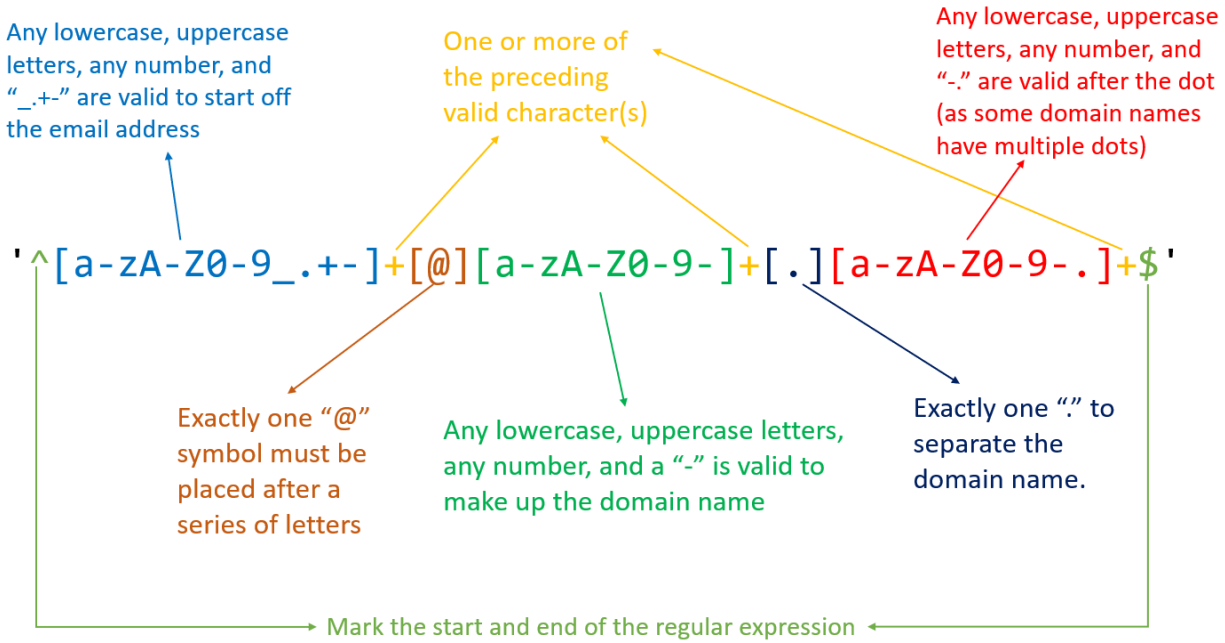
Therefore, the stack reallocation algorithm removed the "2" button press from the stack, while keeping the other buttons in order, so the undo functionality works even when numbers have been erased.

Also, this is enforced if the user overwrites a number, for example if they had a number "2" on the board, and then put a "7" on the same square, pressing undo once would clear it from the board and pressing undo again would result in the error above, as the reallocation algorithm is enforced there. (Provided they had placed no prior numbers elsewhere on the board).

If a user presses a button multiple times, for example just clicks 10 times on the same square with the same number it only registers as one number in the stack, and pressing undo would remove it from the board, and pressing undo a second time would result in the error above, as well. (Provided they had placed no prior numbers elsewhere on the board).

Regular Expressions

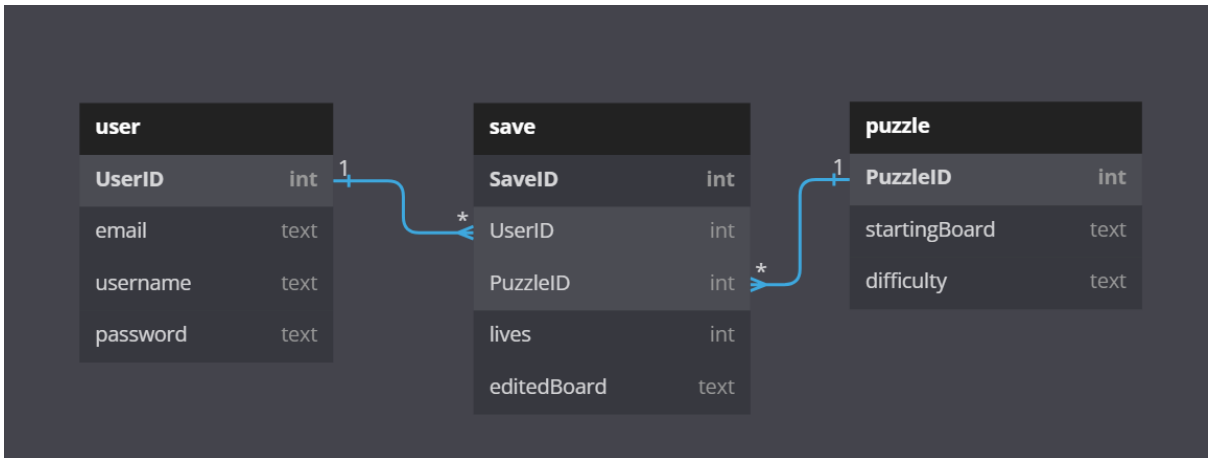
Email Verification



Databases Mock-up

Entity-Relationship Diagram

Database tables



I will have 3 tables:

- **user Table**
    - UserID (primary key) - a unique integer ID which is assigned to each user
    - Email - an email address used to allow the user to change their password. Must be a valid email address, see the regex diagram for the email address validation.
    - Username - whatever the user wants, has to be unique and can only contain letters, numbers, and the characters: "_.-"
    - Password - stored as a hashed password, using the bcrypt hashing algorithm. This hash also stores the salt, so a separate column is not needed to store the salt.

- **save table**
    - SaveID (primary key)
    - UserID (foreign key)
    - PuzzleID (foreign key)
    - lives
    - editedBoard

    - This table is used to save edited boards, so users can go back to their progress.
    - editedBoard - a 2D list of all the numbers that will be shown to the user. The numbers on here not on the StartingBoard will be edits the user made.
    - Lives stores how many lives the user had when they saved the puzzle, so if they had placed 4 incorrect guesses before saving, when they load the puzzle back up, they only have one life left
    - As there is no SQLite data type for storing a 2D list, I have decided to store the boards as a string of text in the database and convert it back into a list in python using the "Json" module.

- **puzzle table**
    - PuzzleID (primary key) - a unique integer ID which is assigned to each puzzle saved
    - startingBoard - a 2D list of what the starting board of the puzzle is to load starting boards from and is used when an edited puzzle is loaded to tell what edits the user made and what numbers were already on the starting board. Stored as a string for the reason just mentioned
    - Difficulty stores what difficulty the puzzle was.
    - This allows players to load puzzles created by anyone, and 2 people can load the same starting puzzle.
        - E.g. If player "Bob" generates a medium difficulty puzzle, with Puzzle ID 5, he can say to user "Jeff" 'I did a good Sudoku today, medium puzzle 5', and Jeff could go and load the starting board of that puzzle, by inputting Puzzle ID 5 into the “load pre-generated puzzles” window.

This database is normalised, in order to avoid redundancies and duplicated data, and allow easy access to the data I am looking for. I have a Python class called “Sql” where all my queries are made, each stored in a function so I can pass the parameters into each function, and the SQL query is then executed. For example:

```
def return_password(self, username):
    command = (f"SELECT password FROM user WHERE username='{username}'")
    self.cur.execute(command)

    result = self.cur.fetchall()
    return result[0][0]
```

This function is used to return the password by inputting a username, which is used to compare whether a user’s entered (hashed) password when signing in is the same as their stored (hashed) password, attached to their account in the “user” table in my database.

Loading and saving

Every time the user starts a game, by selecting a difficulty, the starting board will be saved into the "puzzle" table, along with the difficulty of the puzzle. Any users can then load any previously generated starting board, even if they did not generate it, just by using the PuzzleID.

Players have the ability, once signed in, to save their progress on a given puzzle. A unique Save ID is generated, which the user is prompted to memorise. Once the user is signed in again, for example on a different session on a different day, they can load their previously saved progress on the puzzle by inputting their given Save ID into the "Load" window, allowing them to continue where they left off.

These two features allow players to share tough or interesting puzzles, as anyone can have a go at it just by loading the starting board by inputting the Puzzle ID, and individual users can save and load their progress on puzzles, making this Sudoku program more like a "game", where you can challenge your friends by sharing some hard puzzles, while also being able to go back to your previous edits and finish off the puzzle at any given time.

What the database looks like with data in it

User Table

UserID	username	email	password
Filter	Filter	Filter	Filter
1	GUIuser	[redacted]@gmail.com	$2b$12$M80FL0RBZAeP/...
2	TheoMantel-Cooper	[redacted]@gmail.com	$2b$12$ZBVggx4sfI7z3HVhRVJx4.m
3	Theo	[redacted]@gmail.com	$2b$12$Zpu5nvP/...

Obviously, any valid email address will work, I have just used real Gmail addresses as I needed access to the email addresses to send the forgotten password email.

Save table

SaveID	UserID	PuzzleID	lives	editedBoard
Filter	Filter	Filter	Filter	Filter
2038	5	4	1	[[2, 2, 1, 2, 5, 8, 6, 2, 2], [0, 0, 0, 0, 0, 9, 1, 0, ...
3554	3	23	4	[[1, 4, 5, 2, 0, 7, 0, 6, 0], [3, 0, 6, 1, 4, 8, 7, 0, ...
4516	3	34	5	[[1, 4, 5, 0, 0, 7, 0, 6, 0], [3, 0, 6, 1, 4, 8, 7, 0, ...
5747	3	8	4	[[1, 2, 4, 5, 0, 8, 0, 0, 0], [0, 0, 8, 0, 3, 0, 0, 0, ...

Puzzle table

PuzzleID	startingBoard	difficulty
Filter	Filter	Filter
16	[[0, 3, 4, 0, 0, 0, 0, 0, 9], [1, 5, 8, 0, 7, 4, 0, 3, ...	easy
17	[[0, 1, 0, 0, 0, 0, 0, 0, 5], [4, 6, 0, 0, 2, 9, 0, 0, ...	hard
18	[[0, 2, 3, 5, 0, 9, 0, 7, 4], [0, 0, 6, 2, 0, 7, 0, 0, ...	medium
19	[[1, 0, 3, 4, 0, 9, 0, 0, 0], [4, 9, 0, 1, 7, 6, 2, 0, ...	medium
20	[[0, 5, 0, 0, 4, 0, 0, 0, 0], [0, 0, 6, 7, 0, 9, 1, 5, ...	medium
21	[[1, 0, 0, 0, 0, 6, 0, 0, 8], [3, 6, 0, 0, 0, 0, 0, 0, ...	medium
22	[[0, 0, 2, 4, 6, 8, 7, 0, 0], [0, 6, 0, 9, 0, 0, 0, 3, ...	medium
23	[[1, 4, 5, 0, 0, 7, 0, 6, 0], [3, 0, 6, 1, 4, 8, 7, 0, ...	easy
24	[[0, 1, 0, 0, 0, 0, 7, 0, 0], [3, 4, 0, 0, 7, 0, 0, 0, ...	hard

Hashing algorithm

I was originally going to create my own hashing algorithm from scratch, and while it is possible, it is heavily advised against due to the inherently insecure way of storing passwords. This is because if I create my own algorithm it will likely have exploits and be not secure, thus potentially risking user's personal details and account information. After researching different hashing algorithms, I decided on the "Bcrypt" algorithm.

Bcrypt

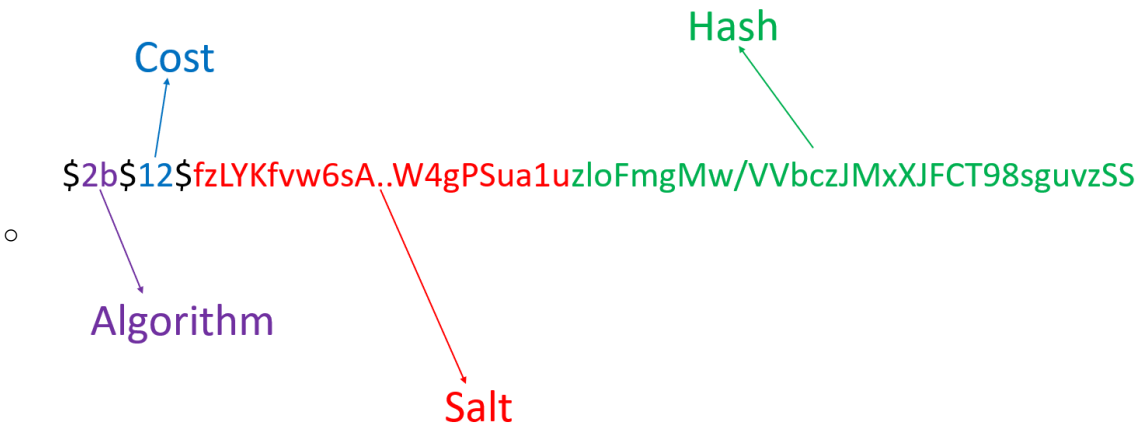
Bcrypt is a hashing algorithm based on the Blowfish cipher. It is an adaptive function; over time, the number of iterations can be increased in order to make the execution slower, which in turn means it is more resistant against brute-force attacks. This means that it is CPU intensive, and thus computationally intensive to brute force. Since its introduction in 1999, it has become one of the most widely used password hashing algorithms used in industry.

How it works

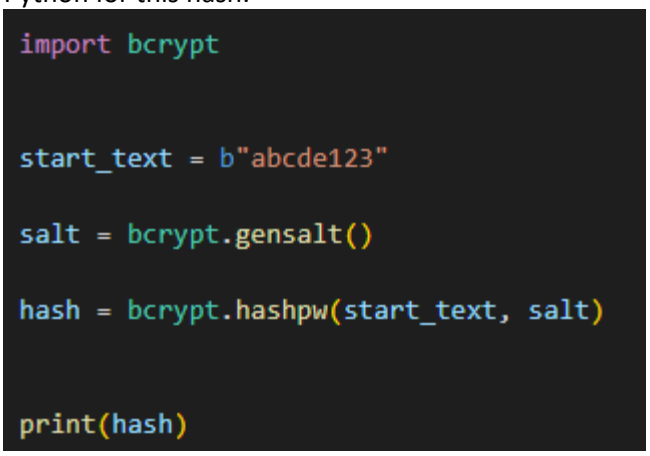
The function takes in a string of bytes, up to 72 bytes, a numeric cost, and a 16-byte salt value. This is usually randomly generated. This is then turned into a "work factor", which is passed through the Blowfish cipher multiple times, finally outputting the final hash value.

For example:

- Input password: "abcde123"
- Random salt
- Cost: 12

- Outputs:
  - 

- A cost of 12 means 2^{12} rounds, so 4096 rounds.

- Code in Python for this hash:
    - 

The salt is used so if 2 users have the same password, which is not uncommon, then their hashes will be completely different. Work factor is the computational effort needed to generate the hash value, so the higher it is the higher the CPU will work, but the harder it will be to brute force.

Advantages

- Implements random salting, which prevents against rainbow tables being used to crack common hashes.
- Computationally expensive to brute force. The algorithm is designed to be slow and computationally expensive.
- Flexible. It is available and easy to integrate into multiple programming languages, such as Python, and can customise the work factor to make it more / less computationally intensive to hash.

Disadvantages

- CPU intensive so may be slow on older hardware which slows down execution of a program
- Blowfish cipher has vulnerabilities, such as side-channel attacks.

Comparisons

- **Bcrypt vs SHA-1 and MD5**
    - MD5 and SHA-1 are older algorithms, and are not typically used in industry, as although they are fast, have big security issues which could lead to potential vulnerabilities with storing passwords

- **Bcrypt vs SHA-256**
    - SHA-256 is a newer algorithm, and one of the most common hashing algorithms today. However, it is very susceptible to brute force attacks due to its quick speed.

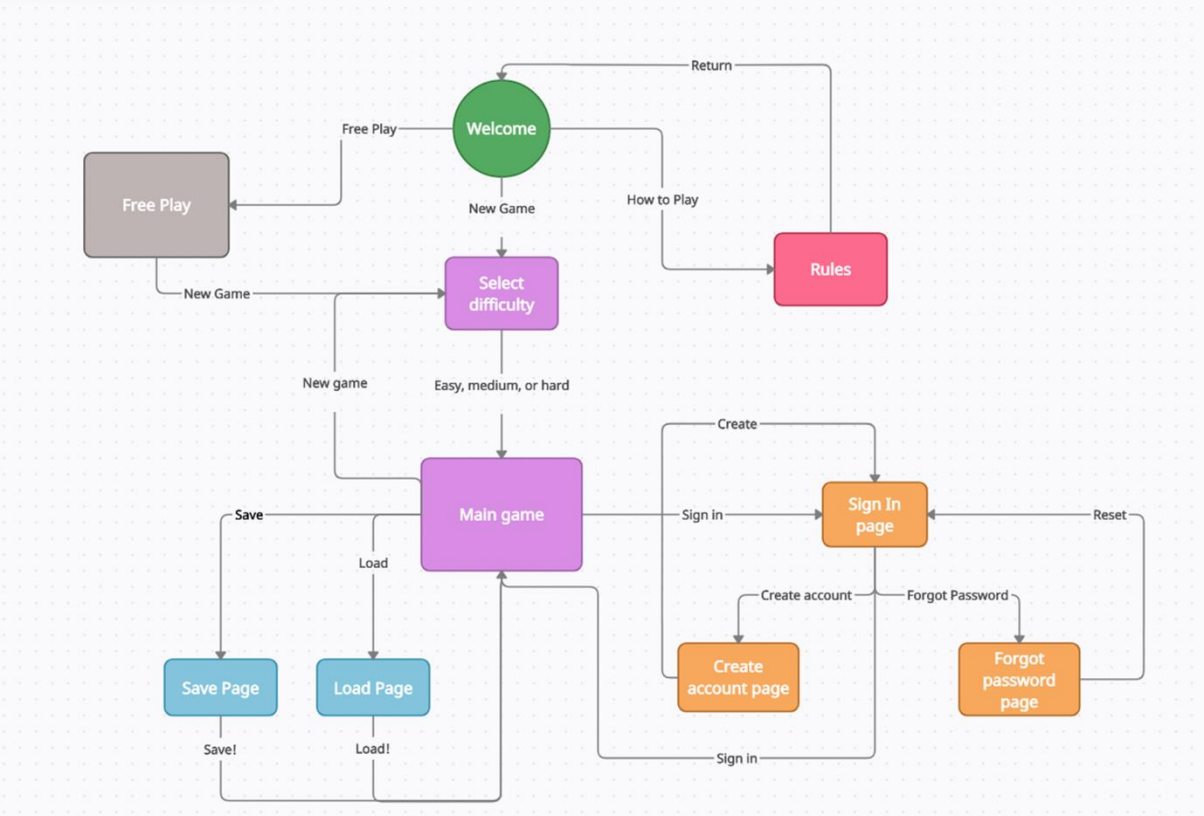
Bcrypt on the other hand, has a variable work factor, so the implementation can make the algorithm slower, more difficult to brute force, and thus more secure than SHA-256.

- **Bcrypt vs Argon2**
  - Argon2 is a newer hashing algorithm and is designed to be more brute-force resistant than Bcrypt, however it is more computationally intensive to use. I found that the Bcrypt algorithm was easier to implement into Python.

Overall, Bcrypt was the hashing algorithm I decided to implement because of the advantages just mentioned, and its ease of use within Python, as you can easily import the module, hash and compare passwords.

I think that it will be more than enough for its use in this Sudoku program, and its security vs hardware utilisation trade-off is the best out of all the aforementioned algorithms in my opinion, therefore it was the obvious choice for me to use in this project.

Page navigation



Video walkthrough

This is a video walkthrough of the UI, and how everything in the program interacts with one another. There is no audio or speaking, just me going through the program.

URL (if QR code does not work): https://youtu.be/s_9-5z5eUsw

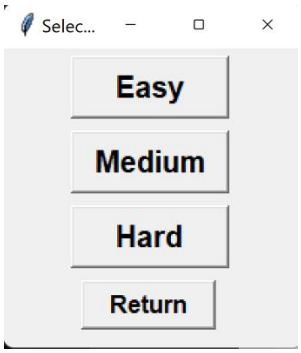
QR code:

UI Mock-up

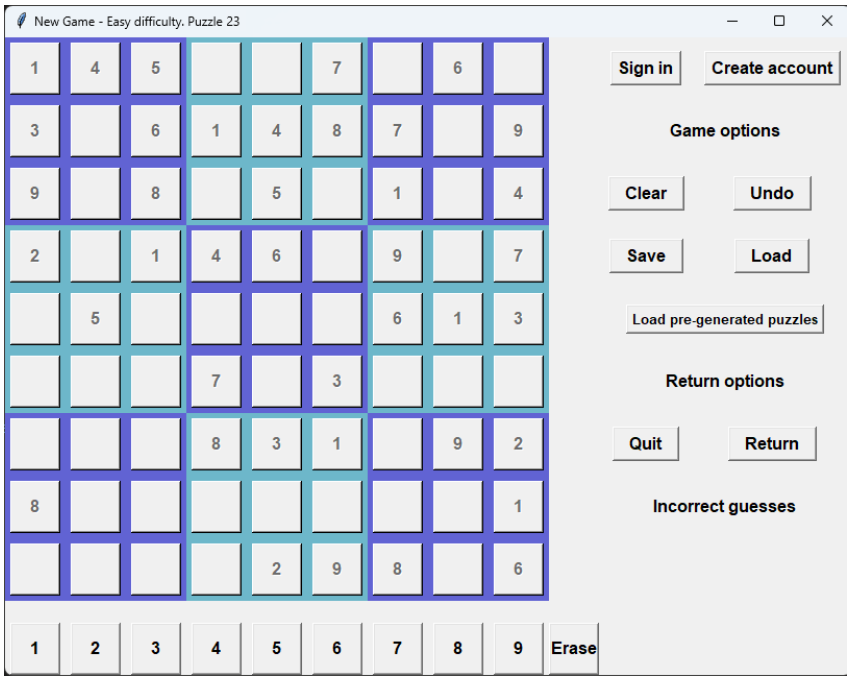
This is just the essential, biggest UI parts of the project, for all of the intricate error prevention, sign-in windows etc, refer to the video above, or the testing section.

Welcome Page

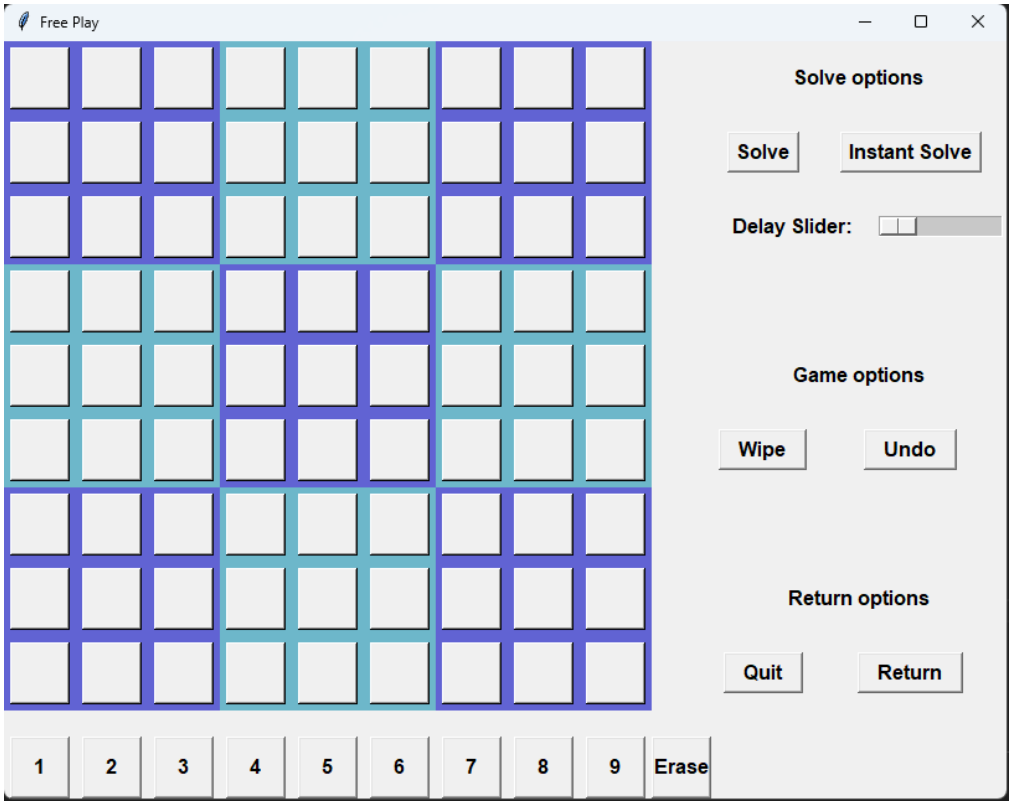
New game – Select difficulty page



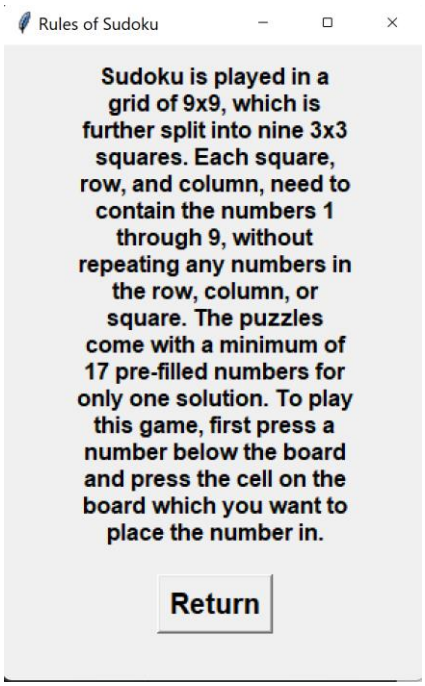
New game - after selecting difficulty



Free Play



Rules page


Rules of Sudoku
Sudoku is played in a grid of 9x9, which is further split into nine 3x3 squares. Each square, row, and column, need to contain the numbers 1 through 9, without repeating any numbers in the row, column, or square. The puzzles come with a minimum of 17 pre-filled numbers for only one solution. To play this game, first press a number below the board and press the cell on the board which you want to place the number in.
Return

Internal data storage

Constants

Throughout the program I utilise constants, to save re-writing code, and avoid mistakes re-using the same values.
For example,

```
self.FONT = ("Arial", 15, "bold") # Default font
```

To define the default font for the majority of buttons and text in my program.

```
try: # School WiFi doesn't allow API calls
    API_TEXT = requests.get(url="https://api.npoint.io/683413d787a24bac2915").json()
    TEXT = API_TEXT["Sudoku"]["Rules"]  # Stores the text in a constant
except:
    TEXT = ("Sudoku is played in a grid of 9x9, which is further spit into nine 3x3 squares. "
            "Each square, row, and column, need to contain the numbers 1 through 9, without "
            "repeating any numbers in the row, column, or square. The puzzles come with a minimum "
            "of 17 pre-filled numbers for only one solution. To play this game, first press a "
            "number below the board and press the cell on the board which you want to place the number in.")
```

This "TEXT" constant is used to store the text used in the rules page of the program. It is in a try except statement, as the text was stored in an API which returns the text when called, which worked fine however upon running the code at school there was an issue arising from the school's Wi-Fi firewalls so in the except statement it has the text written out in case the API call does not work. Both of these constants are used in the code frequently and never changed, as they are constants.

Board

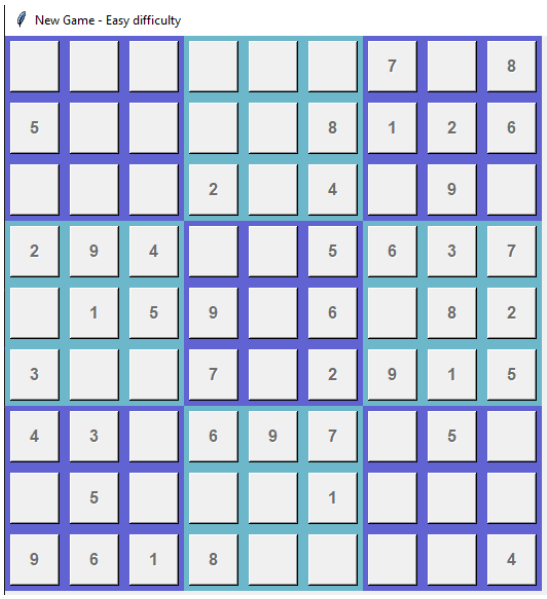
The board will be stored as a 2D array, with 9 arrays with 9 elements in each, to create 9 rows and 9 columns to mimic a Sudoku board. The notation to reference a single box is board[row-1][column-1], because the values of a list in Python start at 0.
The default value for an empty box is 0, and then a placed number will be referenced by inputting that number onto the board instead of a 0. E.g. If a user inputted the number 2 on the second row and third column, board[1][2] == 2.
For example, a default board can be written as:

```
board = [
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
        [0,0,0,0,0,0,0,0,0],
       ]
```

Another example, can be shown graphically:

This is an example of an "Easy" puzzle starting board

```
board = [
    [0,0,0,0,0,0,7,0,8],
    [5,0,0,0,0,8,1,2,6],
    [0,0,0,2,0,4,0,9,0],
    [2,9,4,0,0,5,6,3,7],
    [0,1,5,9,0,6,0,8,2],
    [3,0,0,7,0,2,9,1,5],
    [4,3,0,6,9,7,0,5,0],
    [0,5,0,0,0,1,0,0,0],
    [9,6,1,8,0,0,0,0,4],
   ]
```



Technical solution

I am just pasting screenshots of the code, as they are commented to explain what each function does.

They are all in different files, so I will have a heading for each file, and I will be putting the files in as much order as makes sense, although they all interact and are used at different times (e.g. lots of files use the SQL file). To launch the program, you run the windows.py file.

windows.py

windows.py > ...

```python
from tkinter import *
import requests
import free_play
import new_game


# A template that the windows can inheret from that include close and quit functions, and the default font
class WindowTemplate:
    def __init__(self):
        self.window = Toplevel() # This creates a  new Tkinter window

        self.FONT = ("Arial", 15, "bold") # Default font

    # Just closes the current window
    def close(self):
        self.window.destroy()

    # Exits the whole program
    def quit(self):
        exit()



# Hidden window to run in the background because one window needs a "mainloop" and tkinter does not like
# windows constantly opening and closing each with their own mainloop so this windows is always open
# but hidden in the background
class hidden:
    def __init__(self):
        self.window = Tk()
        self.window.withdraw() # Hides the window

        Welcome() # Calls the main Welcome class
        self.window.mainloop()


```

```python
class Welcome(WindowTemplate):
    def __init__(self):
        super().__init__()
        self.window.title("Welcome!")
        self.window.geometry("225x200") # Sets the size of the window

        # "New Game" button
        self.new_game_but = Button(self.window, text="New game", font=self.FONT, width=10, command=self.open_new_game).place(x=45, y=5)

        # "Free play" button
        self.free_play_but = Button(self.window, text="Free play", font=self.FONT, width=10, command=self.open_freeplay).place(x=45, y=55)

        # "Rules" Button
        self.rules_but = Button(self.window, text="How to play", font=self.FONT, width=10, command=self.open_rules).place(x=45, y=105)

        # Button to quit the program
        self.quit_but = Button(self.window, text="Quit", font=("Arial", 12, "bold"), width=8, command=self.quit).place(x=65, y=155)


    # Function to close the current window and open the rules page by calling the "Rules" class
    def open_rules(self):
        self.close()
        Rules()


    # Function to close the current window and open the FreePlayWindow class in the "free_play.py" file
    def open_freeplay(self):
        self.close()
        free_play.FreePlayWindow()

    # Function to close the current window and open the SelectDifficulty class in "new_game.py" file
    def open_new_game(self):
        self.close()
        new_game.SelectDifficuly()

```

```python

# Calls my API which holds the data for the rules page
# This is outside the class because it only has to run once at the start of the program
# Instead of running every time the "Rules" page is opened which takes time


try: # School WiFi doesn't allow API calls
    API_TEXT = requests.get(url="https://api.npoint.io/683413d787a24bac2915").json()
    TEXT = API_TEXT["Sudoku"]["Rules"]  # Stores the text in a constant
except:
    TEXT = ("Sudoku is played in a grid of 9x9, which is further spit into nine 3x3 squares. "
            "Each square, row, and column, need to contain the numbers 1 through 9, without "
            "repeating any numbers in the row, column, or square. The puzzles come with a minimum "
            "of 17 pre-filled numbers for only one solution. To play this game, first press a "
            "number below the board and press the cell on the board which you want to place the number in.")



# Rules page class
class Rules(WindowTemplate):
    def __init__(self):
        super().__init__()
        self.window.title("Rules of Sudoku") # Sets the title of the window
        self.window.geometry("300x450") # Sets the size of the window



        # The text which has been collected from the API
        self.label = Label(self.window, text=TEXT, font=("Arial", 12, "bold"), wraplength=200).place(x=50, y=10)

        # A button so the user can return to the Welcome page
        self.return_but = Button(self.window, text="Return", font=self.FONT, command=self.welcome_page).place(x=110, y=375)


    # A function to close the current window and return to the original welcome page
    def welcome_page(self):
        Welcome()
        self.close()



# This starts the program
# You start the program by running this file.
if __name__ == "__main__":
    hidden()
```

board_class_file.py

board_class_file.py > ...

```python
# Board class used in the Free Play and New Game modes
class GameBoard:
    def __init__(self):
        self.new_board() # Call the new_board function when starting this class


    def new_board(self):
        self.game_board = [[0 for _ in range(9)] for _ in range(9)] # Create empty 9x9 2D list full of zeroes


    def update(self, num, r, c): # Update the board with the number supplied
        self.game_board[r][c] = num


    def return_num(self, r, c): # Returns the number at the co-ords on the board
        return self.game_board[r][c]


    def reset_value(self, r, c): # Reset the number at the co-ordinates supplied
        self.game_board[r][c] = 0


    def is_board_full(self): # Used in the New Game mode to congratulate the user upon completion of the puzzle
        for r in range(9):
            for c in range(9):
                if self.game_board[r][c] == 0:
                    return False      # The board still has empty numbers on it

        return True   # The board is full


    def clear_user_inputs(self, start_board): # Used in the New Game mode, where the numbers the user inputted
        # should be cleared, but the starting board numbers should remain
        for r in range(9):
            for c in range(9):
                if self.game_board[r][c] != 0 and start_board[r][c] == 0: # If there is a number the user entered
                    # and there is no number that is on the starting board, clear the number
                    self.game_board[r][c] = 0


    def find_empty(self): # Find an empty cell and return the row and column of it
        for r in range(9):
            for c in range(9):
                if self.game_board[r][c] == 0:
                    return (r, c)

        return False # There are no empty cells

```

```python

    # Check if a given number is valid in the given cell
    def num_valid(self, num, r, c):
        # Check row
        for i in range(len(self.game_board[0])):
            if self.game_board[r][i] == num and (r,i) != (r,c): # check if there are duplicate nums in the row and "i" is not the same as the column
                return False

        # Check column
        for i in range(len(self.game_board)):
            if self.game_board[i][c] == num and (i,c) != (r,c): # check if there are duplicate nums in the column and "i" is not the same as the row
                return False

        # Check 3x3 cube for duplicate number
        box_y = r // 3
        box_x = c // 3

        for i in range(box_y * 3, box_y*3 + 3):
            for j in range(box_x * 3, box_x*3 + 3):
                if self.game_board[i][j] == num and (i,j) != (r,c):
                    return False

        return True


    def whole_board_valid(self): # Checks if the whole current board is valid
            for r in range(9):
                for c in range(9):
                    num = self.game_board[r][c]
                    if num != 0:
                        if self.num_valid(num, r, c) == False:  return False

            return True
```

```
82
83      # Instant solve function
84      def instant_solve(self):
85          find = self.find_empty() # Find empty slot
86          if find == False:
87              # Board is solved
88              return True
89          else:
90              row, col = find[0], find[1]
91
92          for i in range(1, 10): # Checks each number
93              if self.num_valid(i, row, col):
94                  self.game_board[row][col] = i # Set the number on the board if that number is valid
95
96                  if self.instant_solve(): # Recursive call of this function
97                      return True
98                  else:
99                      self.game_board[row][col] = 0 # Backtrack by setting the number on the board to 0
100
101         return False
102
103
104     # A function used to print the board, *only used for testing* as it prints out in the terminal
105     def print_board(self):
106             print()
107             print()
108             for i in range(len(self.game_board)):
109                 if i % 3 == 0 and i != 0:
110                     print("- - - - - - - - - - - - ")
111
112                 for j in range(len(self.game_board[0])):
113                     if j % 3 == 0 and j != 0:
114                         print(" | ", end="")
115
116                     if j == 8:
117                         print(self.game_board[i][j])
118                     else:
119                         print(str(self.game_board[i][j]) + " ", end="")
```

stack.py

stack.py > ...

```python
# A class for a single node
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None


# This is my implementation of a stack using a "Linked List" data structure with a head marking the top of the stack
class Stack:
    def __init__(self):
        self.head = None


    # Checks if stack is empty
    def is_empty(self):
        return self.head == None


    # Push something onto the stack
    def push(self, data):
        if self.is_empty():
            self.head = Node(data)
        else:
            new_node = Node(data)
            new_node.next = self.head
            self.head = new_node


    # Pop the top element off the stack
    def pop(self):
        if self.is_empty():
            return None
        else:
            popping = self.head.data
            self.head = self.head.next
            return popping


    # Clears the stack by "popping" all the elements until it is empty
    def clear_stack(self):
        while not self.is_empty():
            self.pop()



    # Stack reallocation algorithm
    # Removes the element in the stack with the given co-ords by removing everything from the stack,
    # holding it in a temporary stack and putting it back in the original one
    def remove_element(self, co_ords):
        temp = Stack()

        while self.head and self.head.data[0] != co_ords: # Add every number into the temporary stack until you reach the desired value
            temp.push(self.pop())

        if self.head: # If there is an item at the head, pop it (this is the desired item we want to remove)
            self.pop()

        while temp.is_empty() == False: # while there are numbers in the temporary stack
            self.push(temp.pop()) # Add them into the game stack

```

game_template.py

```python
from tkinter import *
from tkinter import messagebox
import board_class_file
import stack


# This is the template class for the "free play" and "new game" modes
class GameTemplate:
    def __init__(self):
        self.window = Toplevel()

        self.FONT = ("Arial", 12, "bold") # Sets the default font constant
        self.BUTTON_BG_COLOUR = "#f0f0f0"

        # dictionaries so all certain buttons can be activated / deactivated / modified
        self.cells_dict = {}
        self.nums_dict = {}
        self.utilities_dict = {}

        # creates an instance of the "GameBoard" class to use its methods to modify the board
        self.board_class = board_class_file.GameBoard() # This is composition (creating an instance of GameBoard which only exists inside this object)

        # Create a stack using my "Stack" class
        self.numbers_stack = stack.Stack()


        # Draws the board
        for row in range(9):
            for col in range(9):
                # To get the colours in the correct places and make the board look like it is in 3x3 "boxes"
                if row in (0,1,2,6,7,8) and col in (3,4,5) or row in (3,4,5) and col in (0,1,2,6,7,8):
                    colour = "#6DB7CA"
                else:
                    colour = "#6163D3" # outisde - you want darker on the outside


                # create a tkinter frame to store the buttons in
                frame = Frame(self.window, width=10, height=10, padx=5, pady=5, bg=colour)
                frame.grid(row=row, column=col)

                # Creates the game buttons
                # The lambda function allows the rows and columns to be stored in the dictionary as well (refer to line 54)
                game_buttons = Button(frame, justify="center", width=4, height=2, padx=0, pady=0, foreground="blue", bg=self.BUTTON_BG_COLOUR, font=self.FONT,
                                      command=lambda row=row, col=col: self.player_update_num(row, col))
                game_buttons.pack()
                self.cells_dict[(row, col)] = game_buttons

```

```python

        # Empty space between game buttons and the numbers at the bottom of the board
        empty_space = Label(self.window, text="")
        empty_space.grid(row=10, column=1)


        # Create the numbers that the user uses to select their number
        for i in range(1,10):
            num_button = Button(self.window, width=4, height=2, padx=0, pady=0, text=i, font=self.FONT, command=lambda i=i: self.set_selected_num(i))
            num_button.grid(row=11, column=i-1)
            self.nums_dict[i-1] = num_button
            # lambda i=i means: it stores the value of i at the time the lambda is defined, instead of waiting to look up the value of i later when it will be equal to 9 every time.


        # A button to clear a single button on the cell
        single_button_clear = Button(self.window, width=4, height=2, padx=0, pady=0, text="Erase", font=self.FONT, command=lambda: self.set_selected_num(0))
        single_button_clear.grid(row=11, column=9, columnspan=2)
        self.nums_dict[9] = single_button_clear


    # Set the users selected number
    def set_selected_num(self, num):
        self.selected_num = num


    # Quit the program
    def quit(self):
        exit()


    # Close the current window
    def close(self):
        self.window.destroy()

```

```
    # Undo function
    def undo(self):
        # Checks if the user has placed a number because the stack cannot pop something if it is empty if the user hasn't placed any numbers
        try:
            row, col = self.numbers_stack.pop()[0]
        except:
            messagebox.showerror(title="Undo Error", message="You have not placed anything that you can undo")
        else:
            self.board_class.reset_value(row, col) # Reset the actual board's value
            self.cells_dict[(row, col)].config(text="", bg=self.BUTTON_BG_COLOUR)  # Update the GUI


    # Function to be used after the computer has solved the board so user cannot edit the board
    def disable_num_buttons(self):
        for button in self.nums_dict:
            self.nums_dict[button]["state"] = DISABLED

        self.selected_num = None


    # Function to enable the player to edit the board
    def enable_num_buttons(self):
        for button in self.nums_dict:
            self.nums_dict[button]["state"] = NORMAL


    # Function to disable the "utilities" buttons when solving is taking place
    def disable_utilities(self):
        for button in self.utilities_dict:
            self.utilities_dict[button]["state"] = DISABLED

        self.disable_num_buttons() # Disables the number buttons as well


    # Function to enable the "utilities" buttons when solving is finished
    def enable_utilities(self):
        for button in self.utilities_dict:
            self.utilities_dict[button]["state"] = NORMAL

        self.enable_num_buttons() # Enables the number buttons as well

```

```
    # Funciton to disable all the game cells when solving is taking place
    def disable_game_cells(self):
        for cell in self.cells_dict:
            self.cells_dict[cell]["state"] = DISABLED


    # Function to enable all the game cells when solving is finished
    def enable_game_cells(self):
        for cell in self.cells_dict:
            self.cells_dict[cell]["state"] = NORMAL


    # Wipe the entire board
    def wipe(self):
        self.board_class.new_board() # Create a new board from the "board_class_file.py"

        self.enable_num_buttons()  # Make the number buttons enabled
        self.enable_game_cells()   # Make the game cells enabled

        # Make every button have no text on it
        for cell in self.cells_dict:
            self.cells_dict[cell].config(text="", bg=self.BUTTON_BG_COLOUR)

        # Clears stack when the board is cleared
        self.numbers_stack.clear_stack()

```

generate_board.py

generate_board.py > ...

```python
import random

# This is a constant, and a dictionary which stores the range of numbers that correlates to difficulty
# The "keep" comments shows the range of values that will be kept on the board
DICT_NUMS_TO_BE_REMOVED = {
    "easy": (35, 45), # Keep: (36, 46)
    "medium": (46, 49), # Keep: (32, 35)
    "hard": (50, 53) # Keep: (28, 31)
}


# A class to generate a random board that takes a "difficulty" input as a parameter
class GenerateBoard:
    def __init__(self):
        pass


    def create_board(self, difficulty):
        # create an empty starting board
        self.starting_board = [[0 for _ in range(9)] for _ in range(9)]

        # Fill in 13 random squares with a random number according to the rules of Sudoku
        for i in range(13):
            num = random.randint(1,9)
            r = random.randint(0,8)
            c = random.randint(0,8)

            while not self.valid(num, r, c): # If the current selection isn't valid, choose a new random square and number
                num = random.randint(1,9)
                r = random.randint(0,8)
                c = random.randint(0,8)



            self.starting_board[r][c] = num # update the board with the number selected in the row and column selected

        self.solve() # Solve the board using the solver


        # The two numbers of the difficulty range
        a, b = DICT_NUMS_TO_BE_REMOVED[difficulty][0], DICT_NUMS_TO_BE_REMOVED[difficulty][1]
        nums_to_be_removed = random.randint(a,b) # Generate a random number of squares to be removed based on the difficulty


        # Generate a random list of cells
        # This is every square on the 9x9 board.
        cells = [(r, c) for r in range(9) for c in range(9)]
        random.shuffle(cells)   # Shuffle all the moves around


        # Clears the number of cells specified
        for i in range(nums_to_be_removed):
            self.remove_square(cells[i])


        return self.starting_board # Return the board


    # Function for printing the board for debugging purposes
    def print_board(self):
        for i in range(len(self.starting_board)):
            if i % 3 == 0 and i != 0:
                print("- - - - - - - - - - - - ")

            for j in range(len(self.starting_board[0])):
                if j % 3 == 0 and j != 0:
                    print(" | ", end="")

                if j == 8:
                    print(self.starting_board[i][j])
                else:
                    print(str(self.starting_board[i][j]) + " ", end="")


    def find_empty(self): # find empty space on board
        for i in range(len(self.starting_board)):
            for j in range(len(self.starting_board[0])):
                if self.starting_board[i][j] == 0:
                    return (i, j) # row, column

        return False # There are no empty squares
```

```
    def valid(self, num, r, c): # Checks if the current selection is valid
        # Check row
        for i in range(len(self.starting_board[0])):
            if self.starting_board[r][i] == num:
                return False

        # Check column
        for i in range(len(self.starting_board)):
            if self.starting_board[i][c] == num:
                return False

        # Check 3x3 cube
        box_y = r // 3
        box_x = c // 3

        for i in range(box_y * 3, box_y*3 + 3):
            for j in range(box_x * 3, box_x*3 + 3):
                if self.starting_board[i][j] == num:
                    return False

        return True


    # Solve function
    def solve(self):
        find = self.find_empty()
        if find == False:
            return True
        else:
            row, col = find[0], find[1]

        for i in range(1, 10):
            if self.valid(i, row, col):
                self.starting_board[row][col] = i

                if self.solve():
                    return True
                else:
                    self.starting_board[row][col] = 0

        return False


    # Clear the cell specified
    def remove_square(self, tuple):
        r, c = tuple[0], tuple[1]
        self.starting_board[r][c] = 0

```

new_game.py

new_game.py > ...

```python
from tkinter import *
from tkinter import messagebox
from copy import deepcopy
import json
import windows
import game_template
import generate_board
import accounts
import save_and_load
import sql_commands


generateBoardClass = generate_board.GenerateBoard() # Create an instance of the GeneratedBoard class
db = sql_commands.Sql()



# Select Difficulty window
class SelectDifficuly:
    def __init__(self):
        self.window = Toplevel() # Create a new Tkinter window

        self.FONT = ("Arial", 15, "bold") # Set the window font size

        self.window.title("Select difficulty") # Set the window title

        # Difficulty buttons which call the set_difficulty() function
        self.easy_but = Button(self.window, text="Easy", font=self.FONT, width=8, command=lambda: self.set_difficulty("easy")).place(x=45, y=5)
        self.medium_but = Button(self.window, text="Medium", font=self.FONT, width=8, command=lambda: self.set_difficulty("medium")).place(x=45, y=55)
        self.hard_but = Button(self.window, text="Hard", font=self.FONT, width=8, command=lambda: self.set_difficulty("hard")).place(x=45, y=105)

        # Return button
        self.return_but = Button(self.window, text="Return", font=("Arial", 12, "bold"), width=8, command=self.welcome_page).place(x=52, y=155)


    # Close the window
    def close(self):
        self.window.destroy()

    # Return to the welcome page
    def welcome_page(self):
        self.close()
        windows.Welcome()

    # Call the NewGame class with the desired difficulty
    def set_difficulty(self, difficulty):
        self.close()
        NewGame(difficulty)
```

NewGame class

```
49
50
51
52  # NewGame window
53  class NewGame(game_template.GameTemplate):
54      def __init__(self, difficulty):
55          super().__init__() # Inherits from the GameTemplate class
56
57          self.username = "" # When signed in, username is stored
58
59          self.is_signed_in = False # Variable to store whether the user is signed in or not
60          self.lives = 5 # Number of lives the user has
61          self.INCORRECT_ICON = "❌"
62
63
64
65          # A constant which is the starting board generated from the GenerateBoard class in generate_board.py
66          self.STARTING_BOARD = generateBoardClass.create_board(difficulty)
67
68          db_starting_board = json.dumps(deepcopy(self.STARTING_BOARD))
69          self.puzzle_id = db.get_latest_puzzleid() + 1
70
71
72          # Insert the starting board and difficulty into the database
73          db.insert_into_puzzle(self.puzzle_id, db_starting_board, difficulty)
74
75
76          self.window.title(f"New Game - {difficulty.capitalize()} difficulty. Puzzle {self.puzzle_id}") # Sets the window title
77
78
79          #  Changes the game_board variable in the GameBoard class in the "board_class_file.py"
80          self.board_class.game_board = deepcopy(self.STARTING_BOARD)
81
82          self.populate_board() # Populate the board with numbers
83
84
85          # Buttons
86          self.account_label = Label(self.window, text="", font=("Arial", 10)) # Account label for when user is signed in
87          self.account_label.grid(row=0, column=12)
88
89          self.create_sign_in_buttons() # creates sign in buttons
90
91
92          self.game_options_label = Label(self.window, text="Game options", font=self.FONT)
93          self.game_options_label.grid(row=1, column=11, columnspan=3)
94
95          self.clear_but = Button(self.window, text="Clear", font=self.FONT, padx=10, command=self.clear)
96          self.clear_but.grid(row=2, column=11)
```

```
97
98          self.undo_but = Button(self.window, text="Undo", font=self.FONT, padx=10, command=self.undo)
99          self.undo_but.grid(row=2, column=12, columnspan=2, padx=10)
100
101         self.save_but = Button(self.window, text="Save", font=self.FONT, padx=10, command=self.save_puzzle)
102         self.save_but.grid(row=3, column=11, padx=10)
103
104         self.load_but = Button(self.window, text="Load", font=self.FONT, padx=10, command=self.load_puzzle)
105         self.load_but.grid(row=3, column=12, columnspan=2, padx=10)
106
107
108         self.load_previous_but = Button(self.window, text="Load pre-generated puzzles", font=("Arial", 10, "bold"), command=self.load_starting_board)
109         self.load_previous_but.grid(row=4, column=11, columnspan=3, padx=10)
110
111
112         self.return_label = Label(self.window, text="Return options", font=self.FONT)
113         self.return_label.grid(row=5, column=11, columnspan=3)
114
115         self.quit_but = Button(self.window, text="Quit", font=self.FONT, padx=10, command=self.quit)
116         self.quit_but.grid(row=6, column=11, padx=10)
117
118
119         self.return_but = Button(self.window, text="Return", font=self.FONT, padx=10, command=self.select_diffficulty)
120         self.return_but.grid(row=6, column=12, columnspan=2)
121
122         # "Incorrect guesses" text
123         self.return_label = Label(self.window, text="Incorrect guesses", font=self.FONT)
124         self.return_label.grid(row=7, column=11, columnspan=3)
125
126         self.incorrect_guesses_label = Label(self.window, text=f"", fg="red", font=self.FONT)
127         self.incorrect_guesses_label.grid(row=8, column=11, columnspan=3)
128
129
130     # Save puzzle function
131     def save_puzzle(self):
132         if not self.is_signed_in: # If the person isn't signed in
133             messagebox.showerror(title="Error", message="You must be signed into your account to save puzzles")
134         else:
135             edited_board_copy = deepcopy(self.board_class.game_board)
136             save_and_load.SavePuzzle(self.username, self.puzzle_id, self.lives, edited_board_copy) # Call SavePuzzle class
137
```

```
138
139     # Load puzzle function
140     def load_puzzle(self):
141         if not self.is_signed_in: # Person must be signed in
142             messagebox.showerror(title="Error", message="You must be signed into your account to load puzzles")
143         else:
144             save_and_load.LoadPuzzle(self.username, self) # Call LoadPuzzle class (pass through "self", so functions in this class can be called from the LoadPuzzle class)
145
146
147     # Repopulate a loaded puzzle (with user's edits)
148     def repopulate_loaded_puzzle(self, lives, start_board, edited_board, saveid):
149         self.STARTING_BOARD = json.loads(start_board)
150         self.board_class.game_board = json.loads(edited_board).copy() # Json.loads converts a json string back into the original 2D list as it must be stored as a string in the DB
151         self.lives = lives # Changes the number of lives to the number on the previous save
152
153         self.populate_board() # Add the starting board numbers onto the board
154
155
156         for row in range(9):
157             for column in range(9):
158                 num = self.board_class.game_board[row][column] # grab the current number from the board
159
160                 if num != 0 and num != self.STARTING_BOARD[row][column]: # If the number is not 0 or not equal to a game square then place it as the user has edited it
161                     if self.board_class.num_valid(num, row, column): # If the number is actually valid
162                         self.cells_dict[(row, column)].config(text=num, foreground="blue", bg=self.BUTTON_BG_COLOUR, font=self.FONT)
163
164                     else: # If the old inputted numbers were incorrect according to the rules of Sudoku
165                         self.cells_dict[(row, column)].config(text=num, foreground="white", bg="red", font=self.FONT)
166
167
168         # Change title to say loaded puzzle
169         self.window.title(f"Loaded game - {saveid}")
170
171         # Add the number of lives lost
172         self.incorrect_guesses_label.config(text=self.INCORRECT_ICON*(5-lives)) # Put the number of incorrect guesses onto the screen
173
174
175     # Load a starting board
176     def load_starting_board(self):
177         if not self.is_signed_in: # user must be signed in
178             messagebox.showerror(title="Error", message="You must be signed into your account to load puzzles")
179         else:
180             save_and_load.LoadStartingBoard(self) # Call the LoadStartingBoard class and pass in "self"
181
182
```

```
183     # Repopulate the starting board
184     def repopulate_starting_board(self, start_board, puzzleid):
185         self.STARTING_BOARD = json.loads(start_board) # Json.loads() to convert back to a 2D list
186         self.board_class.game_board = deepcopy(self.STARTING_BOARD) # Make the game board equal to the starting board
187
188         self.populate_board() # Wipes the board, clears stack, adds numbers to the board
189
190         self.window.title(f"Playing loaded puzzle {puzzleid}")
191
192         # Reset lives and incorrect guesses icon as user is starting a fresh game
193         self.lives = 5
194         self.incorrect_guesses_label.config(text="")
195
196
197
198     # Shows the window
199     def show_window(self):
200         self.window.deiconify()
201
202
203     # Create Tkinter sign in buttons
204     def create_sign_in_buttons(self):
205         self.sign_in_button = Button(self.window, text="Sign in", font=self.FONT, command=self.sign_in)
206         self.sign_in_button.grid(row=0, column=11)
207
208         self.create_account_button = Button(self.window, text="Create account", font=self.FONT, command=self.create_account)
209         self.create_account_button.grid(row=0, column=12, columnspan=2, padx=10)
210
211
212     def signed_in(self, username): # The user is now signed in
213         self.username = username # Set the username variable to be the username of whoever is currently signed in
214
215         self.is_signed_in = True
216         self.create_account_button.destroy() # Remove the sign in and create account buttons, as user is signed in
217         self.sign_in_button.destroy()
218
219         self.sign_out_button = Button(self.window, text="Sign out", font=self.FONT, command=self.sign_out) # Create a sign out button
220         self.sign_out_button.grid(row=0, column=11)
221
222
223         self.account_label.config(text=f"Signed in as: {username}")
224
```

```
224
225
226     def sign_out(self):
227         self.sign_out_button.destroy() # Destroy the sign out button
228
229         self.create_sign_in_buttons() # Re place the sign in buttons
230
231         self.account_label.config(text="")
232
233         self.is_signed_in = False
234
235
236     def sign_in(self):
237         self.window.withdraw() # Hide the window
238
239         accounts.SignIn(self) # Passing in this instance into the sign in class so functions can be called from that class
240
241
242     def create_account(self):
243         self.window.withdraw() # Hide the window
244         accounts.CreateAccount(self) # Pass in self
245
246
247     # Function to close the window and return to the select difficulty page
248     def select_diffficulty(self):
249         self.close()
250         SelectDifficuly()
251
252
253     # Clears only the user's inputs
254     def clear(self):
255         for row in range(9):
256             for column in range(9):
257                 if (row, column) not in self.generated_buttons_dict:
258                     self.cells_dict[(row, column)].config(text="", bg=self.BUTTON_BG_COLOUR) # Clears the GUI board's numbers
259
260         self.board_class.clear_user_inputs(self.STARTING_BOARD) # Uses the board_class "clear user inputs" function to clear the backend board of the user's numbers
261
262         # Clears stack when the board is cleared
263         self.numbers_stack.clear_stack()
264
265
```

```
264
265
266     # Wipes entire board
267     # Polymorphism because overwriting the wipe() method in the game_template file
268     def wipe(self):
269         self.generated_buttons_dict = {} # Reset the buttons dictionary
270
271         self.numbers_stack.clear_stack() # Clear the stack
272
273         for button in self.cells_dict:
274             self.cells_dict[button].config(text="", bg=self.BUTTON_BG_COLOUR) # Make all the buttons have no text
275             self.cells_dict[button]["state"] = NORMAL # Make the buttons able to be pressed
276
277
278     # Function to populate the board with the numbers from the generated sudoku
279     def populate_board(self):
280         self.wipe()
281         for row in range(9):
282             for col in range(9):
283                 num = self.STARTING_BOARD[row][col]
284                 if num != 0:
285                     self.generated_buttons_dict[(row, col)] = self.cells_dict[(row, col)]
286                     self.generated_buttons_dict[(row, col)].config(text=num) # Put the number on the board
287                     self.generated_buttons_dict[(row, col)]["state"] = DISABLED # Disable the starting numbers buttons so the user cannot overwrite them
288
```

```python

    # A function that is called when the users trys to update a button on the board
    def player_update_num(self, row, col):
        # Checks if there is a selected number and throws an error if user has not selected a number
        try:
            num = self.selected_num
        except:
            messagebox.showerror(title="Number Error", message="Please select a number before trying to place a number")
        else:
            if num != 0: # If the button is not the "clear" button
                if self.board_class.num_valid(num, row, col): # If the number is actually valid
                    self.cells_dict[(row, col)].config(text=num, foreground="blue", bg=self.BUTTON_BG_COLOUR, font=self.FONT)

                else: # If the user inputted number is incorrect according to the rules of Sudoku
                    self.cells_dict[(row, col)].config(text=num, foreground="white", bg="red", font=self.FONT)

                    new_text = self.incorrect_guesses_label.cget("text") + "❌" # Add another "X" to the incorrect guesses label
                    self.incorrect_guesses_label.config(text=new_text)

                    self.lives -= 1 # Decrease the user's lives each time they place a wrong value

                    if self.lives == 0:
                        messagebox.showerror(title="You lose!", message="You have used all 5 of your lives, you lose!")
                        self.close()
                        SelectDifficuly() # Return to the select difficulty page if user has used all their lives

                existing_num = self.board_class.return_num(row, col) # What number is in the selected spot?

                if existing_num != 0: # There is already a number in that spot on the board
                    if num != existing_num: # If the user has overwritten the already placed number with a new number
                        self.board_class.update(num, row, col) # Update the backend board (not the GUI)
                        self.numbers_stack.remove_element((row, col)) # Remove the old number from the stack
                        self.numbers_stack.push([(row,col), num])
                else: # There is no current number on the board (it is 0)
                    self.board_class.update(num, row, col)
                    self.numbers_stack.push([(row,col), num]) # push to stack


                if self.board_class.is_board_full() and self.board_class.whole_board_valid(): # If the board is completed and fully valid
                    messagebox.showinfo(title="Congratulations!", message="Congratulations, you have completed the puzzle!")


            else: # If the button is the "clear" button, put empty text on the game grid
                self.cells_dict[(row, col)].config(text="", bg=self.BUTTON_BG_COLOUR, foreground="blue", disabledforeground="blue", font=self.FONT)
                self.numbers_stack.remove_element((row, col)) # Remove the element that is being cleared from the stack
```

accounts.py

accounts.py > ...

```python
import re
import bcrypt
from tkinter import *
from tkinter import messagebox
import sql_commands
import forgotton_password


# So I can call SQL functions from the Sql class
db = sql_commands.Sql()


# Used so I do not have to keep repeating code such as creating a Toplevel, and the font etc.
class UserWindowsTemplate:
    def __init__(self):
        self.window = Toplevel()
        self.FONT = ("Arial", 12, "bold")


    # Close the window
    def close(self):
        self.window.destroy()


    # This function is used in both classes and they both have a "game_window" variable.
    # This is used to return to the game without signing in / creating an account
    def go_back(self):
        self.game_window.show_window()
        self.close()
```

```python
class SignIn(UserWindowsTemplate):
    def __init__(self, game_window): # Receives the current gamewindow so this class can show it when the user is done with the sign in
        # When called in the other class, it passes in "self" so this class can call functions in the other class when it is done.
        super().__init__()
        self.window.title("Sign in")
        self.game_window = game_window


        # Tkinter UI variables
        self.username_label = Label(self.window, text="Username:", font=self.FONT)
        self.username_label.grid(row=0, column=0, padx=5)

        self.username_input = Entry(self.window, font=("Arial", 12))
        self.username_input.grid(row=0, column=1, padx=10, pady=10, columnspan=2)


        self.password_label = Label(self.window, text="Password:", font=self.FONT)
        self.password_label.grid(row=1, column=0)

        self.password_entry = Entry(self.window, show="*", font=self.FONT)
        self.password_entry.grid(row=1, column=1, columnspan=2)


        self.sign_in_but = Button(self.window, text="Sign in", font=self.FONT, command=self.check_inputs)
        self.sign_in_but.grid(row=2, column=0)

        self.create_account_but = Button(self.window, text="Create account", font=self.FONT, command=self.create_account)
        self.create_account_but.grid(row=2, column=1, pady=10)


        self.return_but = Button(self.window, text="Return", font=("Arial", 10), command=self.go_back)
        self.return_but.grid(row=3, column=0, pady=10)

        self.forgot_pw = Button(self.window, text="Forgot Password", font=("Arial", 10), command=self.forgotton_pw)
        self.forgot_pw.grid(row=3, column=1)
```

```
    # Check that both entries have text
    def check_inputs(self):
        if self.username_input.get() and self.password_entry.get(): # If the user actually inputted text
            self.sign_in()
        else:
            messagebox.showerror(title="Error", message="Please enter text in all fields before continuing")



    def sign_in(self):
        # Sign in logic
        self.inputted_username = self.username_input.get()
        self.inputted_password = self.password_entry.get().encode() # Have to convert the password from "string", to "bytes" format for the Bcrypt algorithm

        try: # Check if there is a user account with the username supplied
            pre_existing_hash = (db.return_password(self.inputted_username)).encode()
        except:
            messagebox.showerror(title="Error", message="There is no user with that Username.")
        else:
            if bcrypt.checkpw(self.inputted_password, pre_existing_hash): # If the password's match
                self.game_window.show_window() # Shows the hidden window
                self.game_window.signed_in(self.inputted_username) # Calls the function in the "New Game" file
                self.close() # Close this window
            else:
                messagebox.showerror(title="Error", message="Username or Password is not valid.") # Otherwise, user has entered something wrong


    # Calls the ForgottonPassword class
    def forgotton_pw(self):
        forgotton_password.ForgottonPassword()


    # Calls the CreateAccount class
    def create_account(self):
        CreateAccount(self.game_window)
        self.close()
```

```python
# CreateAccount class
class CreateAccount(UserWindowsTemplate):
    def __init__(self, game_window): # Takes in the "self" from the NewGame class, so it can call functions in that file to go back
        super().__init__()
        self.window.title("Create an account")
        self.game_window = game_window


        # Tkinter UI variables
        self.email_label = Label(self.window, text="Email:", font=self.FONT)
        self.email_label.grid(row=0, column=0, padx=5)

        self.email_input = Entry(self.window, font=("Arial", 10), width=40)
        self.email_input.grid(row=0, column=1, columnspan=2, padx=10)


        self.username_label = Label(self.window, text="Username:", font=self.FONT)
        self.username_label.grid(row=1, column=0, padx=5)

        self.username_input = Entry(self.window, font=("Arial", 12), width=31)
        self.username_input.grid(row=1, column=1, padx=10, pady=10, columnspan=2)


        self.password_label = Label(self.window, text="Password:", font=self.FONT)
        self.password_label.grid(row=2, column=0)

        self.password_entry = Entry(self.window, show="*", font=self.FONT, width=31)
        self.password_entry.grid(row=2, column=1, columnspan=2)


        self.second_pw_label = Label(self.window, text="Re-enter Password:", font=self.FONT)
        self.second_pw_label.grid(row=3, column=0, padx=10, pady=10)

        self.password_reentry = Entry(self.window, show="*", font=self.FONT, width=31) # show="*" makes the password show up as ** whatver the user types
        self.password_reentry.grid(row=3, column=1, columnspan=2)


        self.password_info_button = Button(self.window, text="Password requirements", font=("Arial", 10), command=self.password_requirements_popup)
        self.password_info_button.grid(row=4, column=0)


        self.create_password_button = Button(self.window, text="Create!", font=("Arial", 13, "bold"), command=self.check_inputs, padx=20)
        self.create_password_button.grid(row=4, column=1, pady=10)

        self.return_but = Button(self.window, text="Return", font=("Arial", 10), command=self.go_back)
        self.return_but.grid(row=4, column=2, pady=10)

```

```
157
158     # Pop-up to show the user the password requirements
159     def password_requirements_popup(self):
160         messagebox.showinfo(title="Password requirements", message=("Please create a strong password, which is at least "
161                                                                     "8 characters long, using a combination of capital "
162                                                                     "and lowercase letters, numbers, and symbols.\n"
163                                                                     "Symbols accepted: #?!@$%^&*-\n"
164                                                                     "You must use at least one of everything mentioned."))
165
166
167     # Return True if all the inputs are filled in and False if not
168     def all_inputs_filled(self):
169         return self.email_input.get() and self.username_input.get() and self.password_entry.get() and self.password_reentry.get()
170
171
172     # Validates the entered email against the regular expression, to see if it is a valid email address
173     def validate_email(self):
174         user_email = self.email_input.get()
175
176         regex = re.compile(r'^[a-zA-Z0-9_.+-]+[@][a-zA-Z0-9-]+[.][a-zA-Z0-9-.]+$')
177         if regex.fullmatch(user_email):
178             return True
179         else:
180             messagebox.showerror(title="Error", message="Please enter a valid email address")
181
182
183     # Validate the username against regex checking that there are only letters, numbers, and '_.-'
184     def validate_username(self):
185         user_username = self.username_input.get()
186
187
188         if db.check_username(user_username) == 1: # If the username already exists
189             messagebox.showerror(title="Error", message="Username already taken")
190         else:
191             regex = re.compile(r'^[a-zA-Z0-9_.-]+$')
192             if regex.fullmatch(user_username):
193                 return True
194             else:
195                 messagebox.showerror(title="Error", message=("Please enter a username only containing upper and "
196                                                              "lowercase letters, numbers, and the characters: _.-"))
197
```

```
199     # Validate the inputted password against the regex.
200     # This regex is different, because the prefix (?=.*?) means that there must be one or more of the following characters
201     # This is needed, as you need one or more of capital letter, lowecase letters, numbers, and symbols for a strong password
202     # so this was the regex that I came up with so the user needs one or more of everything, and at least 8 characters long
203     def validate_password(self):
204         user_password = self.password_entry.get()
205         user_password_reentry = self.password_reentry.get()
206
207         regex = re.compile(r'^(?=.*?[0-9])(?=.*?[a-z])(?=.*?[A-Z])(?=.*?[#?!@$%^&*_-]).{8,}$')
208         if regex.fullmatch(user_password):
209             if user_password == user_password_reentry: # checks both passwords entered are the same
210                 return True
211             else:
212                 messagebox.showerror(title="Error", message="Passwords do not match.")
213         else:
214             messagebox.showerror(title="Error", message="Please enter a valid password at least 8 characters long")
215
216
217     # Make sure every input is valid
218     def check_inputs(self):
219         if self.all_inputs_filled():
220             if self.validate_email() and self.validate_username() and self.validate_password():
221                 self.create_account()
222         else:
223             messagebox.showerror(title="Error", message="Please enter text in all fields before continuing")
224
225
226     # The user has now created an account
227     def create_account(self):
228         messagebox.showinfo(title="Congratulations!", message="You have successfully created an account!")
229
230         # Create account
231         self.email = self.email_input.get()
232         self.username = self.username_input.get()
233         self.password = self.password_entry.get().encode() # make it "bytes" format for hashing
234
235         salt = bcrypt.gensalt() # Create the salt
236
237         hashed_pw = bcrypt.hashpw(self.password, salt) # Hash the password
238         hashed_pw = hashed_pw.decode() # Turn it back into a string to put it in the database
239
240         db.add_user(self.username, self.email, hashed_pw) # Insert username, email, hashed password into the database
241
242
243         # Call the SignIn class so the user can sign into their account now they've created one
244         SignIn(self.game_window)
245         self.close() # Close this window
```

forgotten_password.py

forgotten_password.py > ...

```python
from tkinter import *
from tkinter import messagebox
import smtplib
import random
import re
import bcrypt
import sql_commands



with open("email.txt") as f: # Get the email password from the text file that it is stored in. No point storing one key in a whole database table
    email_pw = f.read()


my_email = "theopythontesting@gmail.com" # Email address used for sending forgotton password emails
db = sql_commands.Sql() # creating an instance of the Sql class so I can execute queries using the classes' functions



# Template window so other windows can inherit reused code such as close function, and the self.FONT constant
class PasswordTemplateWindows:
    def __init__(self):
        self.window = Toplevel()
        self.FONT = ("Arial", 12, "bold")


    def close(self):
        self.window.destroy()




```

```python
# Forgotton password class
class ForgottonPassword(PasswordTemplateWindows):
    def __init__(self):
        super().__init__()
        self.window.title("Forgotton password")


        self.email_label = Label(self.window, text="Enter your email", font=("Arial", 15, "bold"))
        self.email_label.grid(row=0, column=0, columnspan=3)

        self.email_entry = Entry(self.window, width=40)
        self.email_entry.grid(row=1, column=0, columnspan=3, padx=20, pady=10)


        self.return_but = Button(self.window, text="Return", command=self.close)
        self.return_but.grid(row=3, column=0, padx=10, pady=10)

        self.forgotton_password_button = Button(self.window, text="Send Code", font=self.FONT, command=self.send_email)
        self.forgotton_password_button.grid(row=3, column=1, padx=10, pady=10)


    def validate_email(self):
        self.entered_email = self.email_entry.get()

        return db.check_email(self.entered_email) == 1 # Make sure there is already this email address in the database


    def send_email_code(self, code): # The actual emailing, takes code as input so the user can request the code again
        try:
            with smtplib.SMTP("smtp.gmail.com") as connection: # saves having to do connection.close()
                connection.starttls()
                connection.login(user=my_email, password=email_pw)
                connection.sendmail(from_addr=my_email,
                    to_addrs=self.entered_email,
                    msg=f"Subject: Password code\n\nHello there. Your code is: {code}."
                )
        except:
            return False # If the email cannot be sent for any reason
        else:
            return True


```

```
75
76
77      def send_email(self): # Function to generate the code and call the emailing function
78          self.random_code = random.randint(100000, 999999) # Random code to be emailed to the user
79
80          if self.validate_email(): # If the email is already in the DB
81              if self.send_email_code(self.random_code): # Store the email sending code in another function so I can call it to resend email
82                  self.close()
83                  CheckCode(self, self.random_code, self.entered_email) # if email sent successfully, close this window and open the CheckCode class
84              else:
85                  messagebox.showerror(title="Error", message="An issue prevented the email being sent, either the email cannot be reached or there is a connection issue.")
86          else:
87              messagebox.showerror(title="Error", message="Email does not exist.")
88
89
90
91
92
```

```
93  # CheckCode class
94  class CheckCode(PasswordTemplateWindows):
95      def __init__(self, window, code, entered_email):
96          super().__init__()
97          self.forgotton_pw_window = window # Takes in the old window's "self" so functions in the ForgottonPassword class can be called
98          self.emailed_code = code
99          self.entered_email = entered_email
100
101         self.window.title("Verify code")
102
103
104         # Tkinter UI variables
105         self.enter_code_label = Label(self.window, text="Enter the code sent to your email", font=self.FONT)
106         self.enter_code_label.grid(row=0, column=0, padx=10, pady=10, columnspan=3)
107
108         self.code_input = Entry(self.window, font=("Arial", 20, "bold"), width=7, justify="center")
109         self.code_input.grid(row=1, column=0, columnspan=2, padx=10, pady=10)
110
111         self.create_new_password_but = Button(self.window, text="Verify", font=self.FONT, command=self.verify_code)
112         self.create_new_password_but.grid(row=2, column=1, pady=10)
113
114         self.resend_but = Button(self.window, text="Resend email", font=("Arial", 10, "bold"), command=self.resend_email)
115         self.resend_but.grid(row=2, column=0)
116
117
118
119     # If the user wants to resend the code
120     def resend_email(self):
121         self.forgotton_pw_window.send_email_code(self.emailed_code)
122
123
124     def verify_code(self):
125         # check if code is correct
126         try:
127             inputted_code = int(self.code_input.get())
128         except:
129             messagebox.showerror(title="Error", message="Something went wrong. Please enter the numerical code that you were emailed")
130         else:
131             if inputted_code == self.emailed_code:
132                 messagebox.showinfo(title="Congratulations", message="Code is correct. Please enter a new password when prompted.")
133                 self.close()
134                 NewPassword(self.entered_email) # Open the NewPassword class so user can create a new password
135             else:
136                 messagebox.showerror(title="Error", message="The codes do not match.")
137
138
139
```

```python
# New password class
# Once the user has successully received the email and typed in the correct code
class NewPassword(PasswordTemplateWindows):
    def __init__(self, entered_email):
        super().__init__()

        self.entered_email = entered_email

        self.window.title("Create new password")


        # Tkinter UI variables
        self.create_new_pw_label = Label(self.window, text="Enter a new password", font=self.FONT)
        self.create_new_pw_label.grid(row=0, column=0, columnspan=2)

        self.password_entry = Entry(self.window, show="*", font=self.FONT, width=20) # Show="*" makes every character when typing in appear as an * so people cannot see what you're typing
        self.password_entry.grid(row=1, column=0, columnspan=2, padx=20)


        self.second_pw_label = Label(self.window, text="Re-enter Password", font=self.FONT)
        self.second_pw_label.grid(row=2, column=0, columnspan=2, pady=10)

        self.password_reentry = Entry(self.window, show="*", font=self.FONT, width=20)
        self.password_reentry.grid(row=3, column=0, columnspan=2, padx=20)


        self.finish_but = Button(self.window, text="Submit", font=("Arial", 12, "bold"), command=self.update_db)
        self.finish_but.grid(row=4, column=0, columnspan=2, padx=10, pady=10)
```

```python
    # Validate the password against the regex
    # This regex is different, because the prefix (?=.*?) means that there must be one or more of the following characters
    # This is needed, as you need one or more of capital letter, lowecase letters, numbers, and symbols for a strong password
    # so this was the regex that I came up with so the user needs one or more of everything, and at least 8 characters long
    def validate_password(self):
        self.user_password = self.password_entry.get()
        self.user_password_reentry = self.password_reentry.get()

        regex = re.compile(r'^(?=.*?[0-9])(?=.*?[a-z])(?=.*?[A-Z])(?=.*?[#?!@$%^&*_-]).{8,}$')
        if regex.fullmatch(self.user_password):
            if self.user_password == self.user_password_reentry:
                return True
            else:
                messagebox.showerror(title="Error", message="Passwords do not match.")
        else:
            messagebox.showerror(title="Error", message="Please enter a valid password at least 8 characters long")


    # Hash the password
    # .encode() converts a string to "Bytes" format which is the data type needed for the hashing function to work
    def hash_pw(self):
        salt = bcrypt.gensalt() # Create the salt

        # hash the password
        hashed_pw = bcrypt.hashpw(self.user_password.encode(), salt)

        self.hashed_pw = hashed_pw.decode() # Turn it back into a string


    def update_db(self):
        if self.validate_password():
            # Hash PW
            self.hash_pw()

            # update DB
            db.update_password(self.entered_email, self.hashed_pw)

            messagebox.showinfo(title="Success", message="Congratulations, new password has been saved.")

            # Return to main program
            self.close()
```

save_and_load.py

save_and_load.py > ...

```python
from tkinter import *
from tkinter import messagebox
import json
import random
import sql_commands



db = sql_commands.Sql() # Create an instance of the Database class



# Class SavePuzzle
class SavePuzzle:
    def __init__(self, username, puzzleid, lives, edited_board):
        self.window = Toplevel()

        # Generate SaveID
        self.save_id = random.randint(1000, 9999) # Random 4 digit number
        while db.check_save_id(self.save_id) == 1: # While there is an already existing saveID the same as the just generated one
            self.save_id = random.randint(1000, 9999)

        self.username = username # The username of whoever is signed in
        self.puzzle_id = puzzleid
        self.lives = lives
        self.edited_board = json.dumps(edited_board) # json.dumps() converts a list into a string, so it can be put into the database

        self.window.title("Save Puzzle")
        self.FONT = ("Arial", 12, "bold")

        self.info_label = Label(self.window, text=f"Your Save ID is: {self.save_id}. Please note it down or remember it", font=self.FONT, wraplength=200)
        self.info_label.grid(row=0, column=0, columnspan=3, padx=10, pady=10)

        self.submit_but = Button(self.window, text="Submit", font=self.FONT, command=self.submit)
        self.submit_but.grid(row=1, column=1, pady=5)



    def close(self):
        self.window.destroy()


    def submit(self):
        # Get UserID of whoever is signed in
        user_id = db.get_user_id(self.username)

        # DB insert into save
        db.insert_into_save(self.save_id, user_id, self.puzzle_id, self.lives, self.edited_board)

        messagebox.showinfo(title="Success", message="Board saved.")

        self.close()
```

```python
# LoadPuzzle class
class LoadPuzzle:
    def __init__(self, username, game_window):
        self.window = Toplevel()

        self.username = username
        self.game_window = game_window # The instance of the class which called this class so functions from that class can be called from this class

        self.window.title("Load puzzle")
        self.FONT = ("Arial", 12, "bold")

        self.info_label = Label(self.window, text="Enter your unique Save ID:", font=self.FONT)
        self.info_label.grid(row=0, column=0, columnspan=3)

        self.load_entry = Entry(self.window, font=self.FONT)
        self.load_entry.grid(row=1, column=0, columnspan=3, padx=10, pady=10)

        self.load_but = Button(self.window, text="Load", font=self.FONT, command=self.load)
        self.load_but.grid(row=2, column=1, pady=5)



    def close(self):
        self.window.destroy()


    # Loads the board
    def load(self):
        saveid = self.load_entry.get()
        user_id = db.get_user_id(self.username) # Grab the userID from the person's username

        try:
            edited_puzzle = db.get_edited_board(user_id, saveid) # See if there is a puzzle with that SaveID
            starting_puzzle = db.get_starting_board_with_saveid(saveid)
        except:
            messagebox.showerror(title="Error", message="We could not find a puzzle with that Save ID")
        else:
            # get lives
            lives = db.get_lives(user_id, saveid)

            self.game_window.repopulate_loaded_puzzle(lives, starting_puzzle, edited_puzzle, saveid) # Call the function in the "New Game" file

            self.close()
```

```python


# Class LoadStartingBoard
class LoadStartingBoard:
    def __init__(self, game_window):
        self.window = Toplevel()

        self.game_window = game_window # The instance of the class which called this class so functions from that class can be called from this class

        self.window.title("Load Starting Board")
        self.FONT = ("Arial", 12, "bold")


        self.info_label = Label(self.window, text="Enter puzzleID:", font=self.FONT)
        self.info_label.grid(row=0, column=0, columnspan=3, padx=10, pady=10)

        self.puzzleid_entry = Entry(self.window, font=("Arial", 20, "bold"), width=7, justify="center")
        self.puzzleid_entry.grid(row=1, column=0, columnspan=3, padx=10, pady=10)


        self.load_but = Button(self.window, text="Load", font=self.FONT, command=self.load)
        self.load_but.grid(row=2, column=1, padx=10, pady=10)



    def close(self):
        self.window.destroy()


    # loads the starting board
    def load(self):
        inputted_puzzleid = self.puzzleid_entry.get() # puzzle ID entered by user
        try:
            start_board = db.get_starting_board_with_puzzleid(inputted_puzzleid) # Get the starting board from the puzzle ID
        except:
            messagebox.showerror(title="Error", message="No starting board could be found with that Puzzle ID")
        else:
            self.game_window.repopulate_starting_board(start_board, inputted_puzzleid) # Call the function in the "New Game" file

            self.close()

```

sql_commands.py

sql_commands.py > ...

```python
import sqlite3



# All of my SQL code is executed inside this class
class Sql:
    def __init__(self):
        self.con = sqlite3.connect("data.db") # Connect to the DB

        self.cur = self.con.cursor() # Cursor


    def commit(self):
        self.con.commit() # commit function


    # Close the DB
    def close(self):
        self.con.close()


    # Create the "user" table
    def create_user_table(self):
        command1 = ("CREATE TABLE user("
                    "UserID INTEGER PRIMARY KEY,"
                    "username TEXT,"
                    "email TEXT,"
                    "password TEXT)")

        self.cur.execute(command1)



    # Add a new user
    def add_user(self, username, email, password):
        command = ("SELECT * FROM user ORDER BY UserID DESC LIMIT 1") # Fetch the latest item to see the newest user ID
        # So this user inserted into the DB has a userID one more than the most recent user

        self.cur.execute(command)
        results = self.cur.fetchall()
        latest_user_id = results[0][0]


        command2 = (f"INSERT INTO user VALUES ({latest_user_id+1}, '{username}', '{email}', '{password}')")
        self.cur.execute(command2)

        self.commit()

```

```
49
50      # Return password
51      def return_password(self, username):
52          command = (f"SELECT password FROM user WHERE username='{username}'")
53          self.cur.execute(command)
54
55          result = self.cur.fetchall()
56          return result[0][0]
57
58
59      # Check if the inputted email is in the DB
60      def check_email(self, inputted_email):
61          command = (f"SELECT EXISTS(SELECT 1 FROM user WHERE email='{inputted_email}')")
62          self.cur.execute(command)
63
64          result = self.cur.fetchall()
65          return result[0][0]
66
67
68      # Check if the inputted username is already taken
69      def check_username(self, username):
70          command = (f"SELECT EXISTS(SELECT 1 FROM user WHERE username='{username}')")
71          self.cur.execute(command)
72
73          result = self.cur.fetchall()
74          return result[0][0]
75
76
77      # Update the password (user has changed their password)
78      def update_password(self, email, new_password):
79          command = (f"UPDATE user SET password='{new_password}' WHERE email='{email}'")
80          self.cur.execute(command)
81
82          self.commit()
83
84
85      # Get the user ID from the username provided
86      def get_user_id(self, username):
87          command = (f"SELECT UserID from user WHERE username='{username}'")
88          self.cur.execute(command)
89
90          result = self.cur.fetchall()
91          try:
92              return result[0][0] # Return the ID if it exits. It should always return the UserID as this is only called when the person is signed in
93                                  # so they will always have a valid username
94          except:
95              return False # Else, return False
96
```

```
97
98          # Delete values - used for testing
99          def delete_user_values(self, table):
100             command = (f"DELETE FROM {table} WHERE UserID = 4")
101
102             self.cur.execute(command)
103
104             self.commit()
105
106
107
108     # Save table
109
110         def create_save_table(self):
111             command = ("CREATE TABLE save("
112                        "SaveID INTEGER PRIMARY KEY,"
113                        "UserID INTEGER,"
114                        "PuzzleID INTEGER,"
115                        "lives INTEGER,"
116                        "editedBoard TEXT)")
117
118             self.cur.execute(command)
119
120
121         # Check if the inputted save ID exists in the save table
122         def check_save_id(self, inputted_saveid):
123             command = (f"SELECT EXISTS(SELECT 1 FROM save WHERE SaveID='{inputted_saveid}')")
124             self.cur.execute(command)
125
126             result = self.cur.fetchall()
127             return result[0][0]
128
129
130         # Insert into the save table
131         def insert_into_save(self, saveid, userid, puzzleid, lives, editedBoard):
132             command = (f"INSERT INTO save VALUES ({saveid}, {userid}, {puzzleid}, {lives}, '{editedBoard}')")
133
134             self.cur.execute(command)
135
136             self.commit()
137
```

```
138
139     # Get the puzzle ID from the save ID
140     def get_puzzle_id(self, saveid):
141         command = (f"SELECT PuzzleID FROM save WHERE SaveID={saveid}")
142         self.cur.execute(command)
143
144         result = self.cur.fetchall()
145
146         return result[0][0]
147
148
149     # Get the edited board from the user ID and the save ID to make sure the correct user is accessing the edited board, as they are locked to the account that saved them
150     def get_edited_board(self, userid, saveid):
151         command = (f"SELECT editedBoard FROM user, save WHERE user.UserID = save.UserID AND user.UserID={userid} AND save.SaveID={saveid}")
152         self.cur.execute(command)
153
154         result = self.cur.fetchall()
155
156         return result[0][0]
157
158
159     # Get the number of lives from the "save" table
160     def get_lives(self, userid, saveid):
161         command = (f"SELECT lives FROM user, save WHERE user.UserID = save.UserID AND user.UserID={userid} AND save.SaveID={saveid}")
162         self.cur.execute(command)
163
164         result = self.cur.fetchall()
165
166         return result[0][0]
167
168
169 # Puzzle tables
170
171     def create_puzzle_table(self):
172         command = ("CREATE TABLE puzzle("
173                    "PuzzleID INTEGER PRIMARY KEY,"
174                    "startingBoard TEXT,"
175                    "difficulty TEXT)")
176
177         self.cur.execute(command)
```

```python

    # Get the latest puzzleID
    def get_latest_puzzleid(self):
        command = ("SELECT * FROM puzzle ORDER BY PuzzleID DESC LIMIT 1") # Fetch the latest item to see the newest user ID

        self.cur.execute(command)
        results = self.cur.fetchall()
        latest_puzzle_id = results[0][0]
        return latest_puzzle_id


    # Insert into the "puzzle" table
    def insert_into_puzzle(self, puzzle_id, starting_board, difficulty):
        command2 = (f"INSERT INTO puzzle VALUES ('{puzzle_id}', '{starting_board}', '{difficulty}')")
        self.cur.execute(command2)

        self.commit()


    # Get the starting board from the save ID
    def get_starting_board_with_saveid(self, saveid):
        command = (f"SELECT startingBoard FROM save, puzzle WHERE save.PuzzleID = puzzle.PuzzleID AND save.SaveID={saveid}")
        self.cur.execute(command)

        result = self.cur.fetchall()
        return result[0][0]


    # Get the starting board from the puzzle ID
    def get_starting_board_with_puzzleid(self, puzzleid):
        command = (f"SELECT startingBoard FROM puzzle WHERE PuzzleID={puzzleid}")
        self.cur.execute(command)

        result = self.cur.fetchall()
        return result[0][0]


```

free_play.py

free_play.py > ...

```python
from tkinter import *
from tkinter import messagebox
import time
import game_template
import windows

# The Class for the "Free Play" mode which inherits from the GameTemplte class
class FreePlayWindow(game_template.GameTemplate):
    def __init__(self):
        super().__init__()

        self.window.title("Free Play") # Sets the windows title

        self.solving_speed = 0 # Default value

        # Don't need if "clear" button is there
        # self.grid_10_blank_space = Label(self.window, text="      ")
        # self.grid_10_blank_space.grid(row=0, column=10)

        # Text which says "Solve options"
        self.solve_label = Label(self.window, text="Solve options", font=self.FONT)
        self.solve_label.grid(row=0, column=11, columnspan=3)

        # The "Solve" button which is then added to the "utilities_dict"
        self.solve_but = Button(self.window, text="Solve", font=self.FONT, command=self.solve)
        self.solve_but.grid(row=1, column=11)
        self.utilities_dict["solve"] = self.solve_but

        # The "Instant Solve" button which is added to the "utilities_dict"
        self.instant_solve_but = Button(self.window, text="Instant Solve", font=self.FONT, command=self.instant_solve)
        self.instant_solve_but.grid(row=1, column=12, padx=20, columnspan=2)
        self.utilities_dict["instant-solve"] = self.instant_solve_but

        # The Delay Slider text
        self.slider_text = Label(self.window, text="Delay Slider:", font=self.FONT)
        self.slider_text.grid(row=2, column=11, columnspan=2)

        # The Delay Slider
        self.slider_val = DoubleVar()
        self.speed_slider = Scale(self.window, from_=0, to=100, showvalue=0, orient="horizontal", command=self.slider_changed, variable=self.slider_val)
        self.speed_slider.grid(row=2, column=13, columnspan=2)
        self.disable_slider()  # Disables the slider when you open this window


        # Game options text
        self.game_options_label = Label(self.window, text="Game options", font=self.FONT)
        self.game_options_label.grid(row=4, column=11, columnspan=3)

        # Wipe button which is added to the "utilities_dict"
        self.wipe_but = Button(self.window, text="Wipe", font=self.FONT, padx=10, command=self.wipe)
        self.wipe_but.grid(row=5, column=11)
        self.utilities_dict["wipe"] = self.wipe_but

        # Undo button which is added to the "utilities_dict"
        self.undo_but = Button(self.window, text="Undo", font=self.FONT, padx=10, command=self.undo)
        self.undo_but.grid(row=5, column=12, columnspan=2)
        self.utilities_dict["undo"] = self.undo_but


        # "Return options" text
        self.return_label = Label(self.window, text="Return options", font=self.FONT)
        self.return_label.grid(row=7, column=11, columnspan=3)

        # Button used to quit the whole program which is added to the "utilities_dict"
        self.quit_but = Button(self.window, text="Quit", font=self.FONT, padx=10, command=self.quit)
        self.quit_but.grid(row=8, column=11, padx=10)
        self.utilities_dict["quit"] = self.quit_but

        # Button used to return to the Welcome Page which is added to the "utilities_dict"
        self.return_but = Button(self.window, text="Return", font=self.FONT, padx=10, command=self.welcome_page)
        self.return_but.grid(row=8, column=12, columnspan=2)
        self.utilities_dict["return"] = self.return_but
```

```
    # Close current window and call the "Welcome" class from the "windows.py"
    def welcome_page(self):
        self.close()
        windows.Welcome()


    # This is called if the slider value has changed, so "self.solving_speed" is updated with the new speed
    def slider_changed(self, event):
        self.solving_speed = self.slider_val.get() / 1000
        # Scale goes from 0-100 so divide by 1000 to get the accurate time.sleep sleep time


    # Updates the board cell buttons with the solver numbers
    def solver_update_num(self, num, row, col, colour):
        self.cells_dict[(row, col)].config(text=num, disabledforeground=colour, fg=colour, font=self.FONT)
        # Change the certain button to the new solver's updated button


    # Function to disable the slider
    def disable_slider(self):
        self.speed_slider["state"] = DISABLED


    # Function to enable the slider
    def enable_slider(self):
        self.speed_slider["state"] = NORMAL

```

```

    # A function that is called when the users trys to update a button on the board
    def player_update_num(self, row, col):
        # Checks if there is a selected number and throws an error if user has not selected a number
        try:
            num = self.selected_num
        except:
            messagebox.showerror(title="Number Error", message="Please select a number before trying to place a number")
        else:
            if num != 0: # If the button is not the "clear" button
                if self.board_class.num_valid(num, row, col): # If the number is actually valid
                    self.cells_dict[(row, col)].config(text=num, foreground="blue", disabledforeground="blue", bg=self.BUTTON_BG_COLOUR, font=self.FONT)
                else:
                    self.cells_dict[(row, col)].config(text=num, foreground="white", bg="red", font=self.FONT)


                existing_num = self.board_class.return_num(row, col)

                if existing_num != 0: # There is already a number in that spot on the board
                    if num != existing_num: # If the user has overwritten the already placed number with a new number
                        self.board_class.update(num, row, col)
                        self.numbers_stack.remove_element((row, col)) # Remove the old number from the stack
                        self.numbers_stack.push([(row,col), num])
                else:
                    self.board_class.update(num, row, col)
                    self.numbers_stack.push([(row,col), num]) # Push the number onto the stack

            else: # If the button is the "clear" button, put empty text on the game grid
                self.cells_dict[(row, col)].config(text="", foreground="blue", disabledforeground="blue", bg=self.BUTTON_BG_COLOUR)
                self.board_class.update(num, row, col)
                self.numbers_stack.remove_element((row, col))


    # Instant solve function
    def instant_solve(self):
        if self.board_class.whole_board_valid(): # checks if the board is valid first
            self.numbers_stack.clear_stack() # Clears the stack
            self.board_class.instant_solve() # Calls the instant solve in the "board_class_file.py"

            for row in range(9): # Updates every number with the solved numbers
                for col in range(9):
                    if self.cells_dict[(row, col)]["text"] == "":
                        num = self.board_class.return_num(row, col)
                        self.solver_update_num(num, row, col, "green")

        else:
            messagebox.showerror(title="Error", message="Sorry, the current board is unsolvable")
            # If board is unsolvable, throw an error to the user
```

```
150
151
152     # Solve function
153     def solve(self):
154         if self.board_class.whole_board_valid(): # Checks if the board is valid first
155             self.numbers_stack.clear_stack() # Clears the stack
156             self.disable_game_cells() # Makes the cells disabled so user cannot press them
157             self.disable_utilities() # Disables the utility buttons so user cannot press them
158             self.enable_slider() # Enables the delay slider
159 
160             find = self.board_class.find_empty() # Find an empty square
161 
162             if find == False:
163                 # Solver has finished
164                 self.enable_utilities() # Enables utilities
165                 self.disable_slider() # Disables slider
166                 return True
167             else:
168                 row, col = find[0], find[1]
169 
170             for i in range(1, 10):
171                 if self.board_class.num_valid(i, row, col): # Checks if the number is valid in that spot
172                     self.board_class.update(i, row, col)
173 
174                     self.window.update() # Have to do this with Tkinter when using the "time" module
175                     if self.solving_speed != 0:
176                         time.sleep(self.solving_speed) # Put in a delay
177                     self.solver_update_num(i, row, col, "green")
178 
179 
180                     if self.solve(): # Recursive call of this function
181                         return True
182                     else:
183                         # Otherwise, "backtrack" by placing a 0
184                         self.board_class.update(0, row, col)
185 
186                         self.solver_update_num("-", row, col, "red") # Update the button with red to show backtracking
187         else:
188             messagebox.showerror(title="Error", message="Sorry, the current board is unsolvable")
189             # Show the user an error if the board is unsolvable
190 
191         return False
```

Testing

Test Table

Number	Test description	Requir-ement	Input data	Expected output	Actual output	Pass / Fail?	Evidence
1	Is user greeted with Tkinter window?	1	None	Window which welcomes them and has the options: new game, free play, how to play and a quit button	Window which welcomes them and has the options: new game, free play, how to play and a quit button	Pass	E1
2	If “New game” is selected, does a new window show up?	2	None	Select difficulty window which has 3 difficulty options: easy, medium, hard	Select difficulty window which has 3 difficulty options: easy, medium, hard	Pass	E2
3	Does the game window open?	3.1	User’s selected difficulty	New window with 9x9 grid and numbers on the board (unclickable by user) correspondin g to that difficulty	New window with 9x9 grid and numbers on the board (unclickable by user) correspondi ng to that difficulty	Pass	E3
4	Can the user enter a number onto the board?	3.2 / 3.3	User’s selected number	The game window with user’s number on it (in blue if valid, in red if not valid)	The game window with user’s number on it (in blue if valid, in red if not valid)	Pass	E4
5	Error if user has not selected a number	3.2.1	None	Error should show up prompting user to select a number before attempting to	Error should show up prompting user to select a number before attempting	Pass	E5

				add it to the board	to add it to the board		
6	Sign in window	3.4.1	None	New window opens which prompts a username and password input	New window opens which prompts a username and password input	Pass	E6
7	Forgotten password window	3.5.1	None	New window prompting user to enter their email address	New window prompting user to enter their email address	Pass	E7
8	Validating email for forgotten password	3.5.2	User enters incorrect email (“NotValidEmail”)	Error if user enters invalid email not found tied to an account	Error as user entered invalid email not found tied to an account	Pass	E8
9	Sending email to user with code	3.5.3	Correct email entered	Upon entering valid email, user receives code	Upon entering valid email, user receives code	Pass	E9
10	User prompted to enter code received	3.5.4	Code emailed to the user (incorrect)	If code is not valid, error.	Error as code entered was not valid.	Pass	E10
11	User prompted to create new password once valid code has been entered	3.5.4	Code emailed to the user (correct)	New window opens, prompting user to create a new password	New window opens, prompting user to create a new password	Pass	E11
12	User then prompted to sign in again	3.5.5	None	New window opens asking for username and password input	New window opens asking for username and password input	Pass	E12

13	Create account window	3.6	None	New window asking for email, username, password, and password again	New window asking for email, username, password, and password again	Pass	E13
14	Must be text in every space	3.7	None	An error if there is no text in any input field	An error as there is missing text in input fields	Pass	E14
15	Usernames must be unique	3.8.1	Username (“Testing”), as it is already taken	An error saying it is already taken	An error saying it is already taken	Pass	E15
16	Usernames must be valid	3.8.3	Username “Not ValidUsername”	Error saying not valid username	Error saying not valid username	Pass	E16
17	Email addresses must be valid	3.9	Not valid email “NotValidEmail”	Error saying not valid email	Error saying not valid email	Pass	E17
18	Passwords must meet the password standards	3.10	Not valid password “password1”	Error saying not valid password	Error saying not valid password	Pass	E18
19	Passwords must match	3.10.1	“Password1!” and “Password2!”	Error saying passwords do not match	Error saying passwords do not match	Pass	E19
20	If everything is valid, passwords are hashed and stored in the database.	3.11	“validemail@gmail.com” “ValidUsername” “Password1!” “Password1!”	Successful page popping up congratulating user on creating account	Successful page popping up congratulating user on creating account	Pass	E20
21	On sign in screen, if username and password match the database	3.12 / 3.14	“Theo” “Password1!”	They are signed in. Main game will reopen, and username displayed at	They are signed in. Main game reopened and their username is	Pass	E21

	records, user is signed in			the top of the screen	at the top of the screen		
22	Sign in validation	3.13	Both inputs empty And then, Invalid username and password (“notvalid” and “notvalid”) Then, valid username (“Theo”), but incorrect password “notvalid”	If sign in credentials are not filled in or wrong, error message	Error message as not valid username and passwords	Pass	E22
23	Saving puzzles error prevention	3.16.1 / 3.16.2	Not signed in user pressing the “save” button	Error saying that you must be signed in to save puzzles	Error saying that you must be signed in to save puzzles	Pass	E23
24	Gives user their Save ID	3.16.3 / 3.16.4	Pressing the “save” button while signed in	New window with a random 4 digit Save ID	New window with a random 4 digit Save ID	Pass	E24
25	Load button error prevention	3.17.1 / 3.17.2	Pressing the “load” button while not being signed in	Error saying that you must be signed in to load puzzles	Error saying that you must be signed in to load puzzles	Pass	E25
26	Enter Save ID to load puzzle	3.17.3	Pressing the “load” button while signed in	Window shows up prompting user to enter their Save ID	Window shows up prompting user to enter their Save ID	Pass	E26
27	Old edits will be placed on the board	3.17.4	Entering correct Save ID while logged into the correct account which saved that puzzle	Game window updated to have the progress left off when the user last saved it (their edits in blue + red)	Game window updated to have the progress left off when the user last saved it (their edits in blue + red)	Pass	E27
28	Load any previously	3.18	Entering a Puzzle ID into	Window to prompt user	Window to prompt user	Pass	E28

	generated starting puzzle		the window, after pressing the “load pre-generated puzzles”	to enter a puzzle ID to load, and if it is valid, update the game window to show the starting board of that puzzle id	to enter a puzzle ID to load, and it was valid, so the game window was updated to show the starting board of that puzzle id		
29	Clear button	3.19	Numbers inputted by the user onto the board (1, 2, 3, 4, 5)	Clears the board of any numbers inputted by the user, but keeps pre-generated puzzle board numbers	Cleared the board of any numbers inputted by the user, but keeps pre-generated puzzle board numbers	Pass	E29
30	Undo button	3.20.1 – 3.20.4	3 numbers inputted by user, undo button pressed 3 times.	Every time undo button pressed, last placed number should disappear off the board	Every time undo button was pressed, last placed number disappeared off the board	Pass	E30
31	Undo button error preventio n	3.20.5	User pressing “undo” while none of their placed numbers are on the board	Error saying you have not placed anything that you can undo	Error saying you have not placed anything that you can undo	Pass	E31
32	Lose all 5 of their lives	3.21	User inputs 5 incorrect numbers (9, 9, 9, 9, 9)	Message saying “you have lost”, and return to the select difficulty screen	Message saying “you have lost”, and return to the select difficulty screen	Pass	E32
33	Return button	3.22	User presses “return” button	Takes user back to select difficulty window	Takes user back to select difficulty window	Pass	E33
34	Free play board	4.1	User presses “Free Play” on the welcome page	Game window is generated, full of empty	Game window is generated, full of empty	Pass	E34

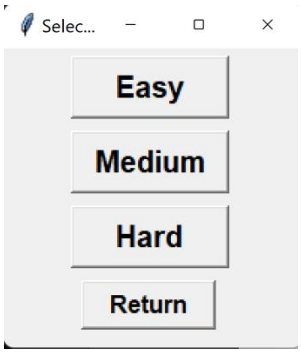
				squares and the numbers 1-9 at the bottom, for the user to input numbers onto the board	squares and the numbers 1-9 at the bottom, for the user to input numbers onto the board		
35	Solve button	4.2.1 – 4.2.3	User's inputs on the board (8, 9, 1, 2) in that order	User sees the algorithm solving the puzzle, with green numbers show what the solver is doing, and red showing where it backtracks. Finished when every number in the finished board is green (user's inputs remain blue)	Algorithm solved the puzzle, with green numbers show what the solver is doing, and red showing where it backtracks. Finished when every number in the finished board is green (user's inputs remain blue)	Pass	E35
36	Solve button error prevention	4.2.4	User enters numbers that are invalid for the rules of Sudoku (1, 1)	Error saying the solver cannot start because the board is invalid at the moment	Error saying the solver cannot start because the board is invalid at the moment	Pass	E36
37	Instant solve button error prevention	4.3.1	User enters numbers that are invalid for the rules of Sudoku (1, 1)	Error saying the solver cannot start because the board is invalid at the moment	Error saying the solver cannot start because the board is invalid at the moment	Pass	E36 (as the same error message both times)
38	Instant solve button	4.3.2	User's inputs on the board (8, 9, 1, 2) in that order	User just sees the GUI updated to have the full solved board in green, and their inputs	User just sees the GUI updated to have the full solved board in green, and their inputs	Pass	E37

				remain in blue	remain in blue		
39	Undo button	4.4	Same exact functionality as before	Refer to number 30 and 31	Refer to number 30 and 31	Pass	E30 and E31
40	Wipe button	4.5	User inputs numbers (9,5,1,2,3) and then presses wipe	Wipes all of the user's inputs, and clears the stack	Wipes all of the user's inputs, and clears the stack (pressing undo after this leads to the error shown in E31)	Pass	E38
41	Return button	4.6	User presses "return" button	Takes the user back to the Welcome Window	Takes the user back to the Welcome Window	Pass	E39
42	How to play button	5	On the "Welcome" window, user presses "How to play"	Opens window with a new page which has the rules of Sudoku on it	Opens window with a new page which has the rules of Sudoku on it. User can then return back to the "Welcome page"	Pass	E40
43	"Quit" button	6	"Quit" button is pressed at any time in the program	Program is terminated	Program is terminated	Pass	No evidence. (shown in the video right at the very end)

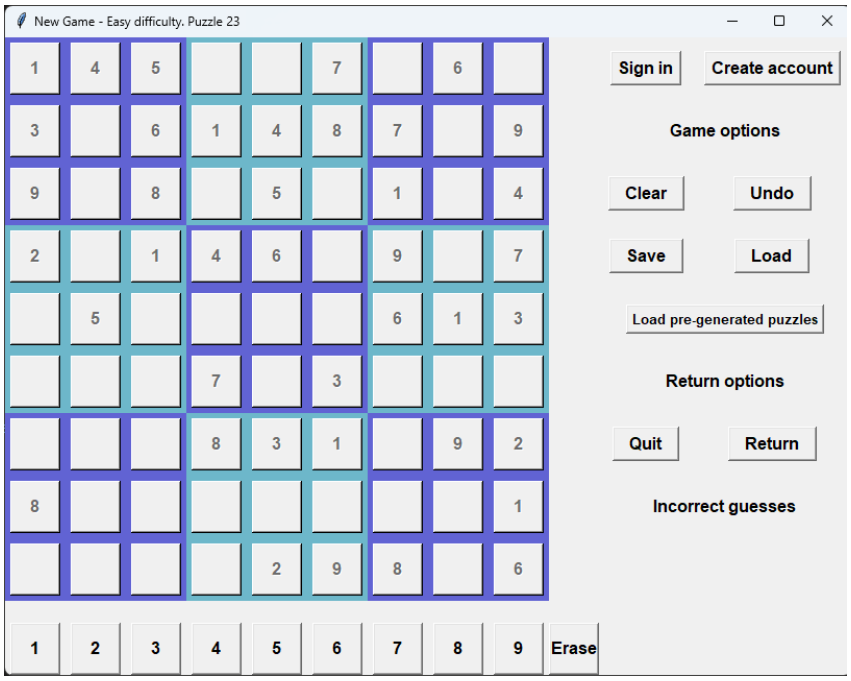
Testing Evidence

E1

E2



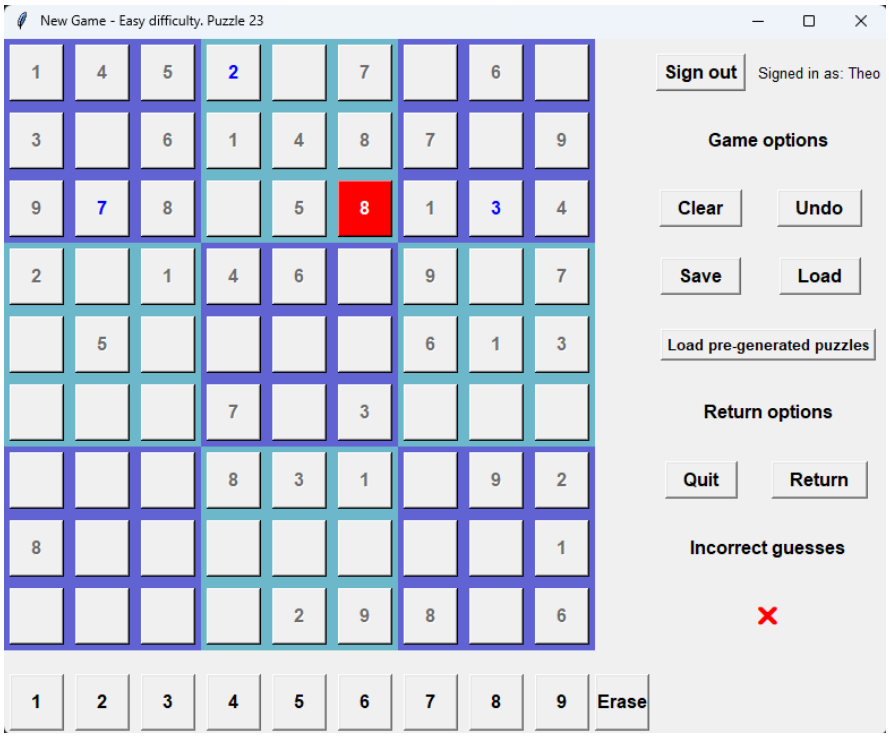
E3



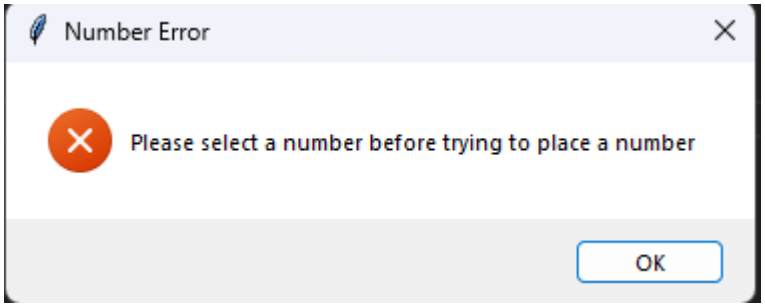
E4

User has placed numbers that are correct (in blue), and numbers that are incorrect (in red)

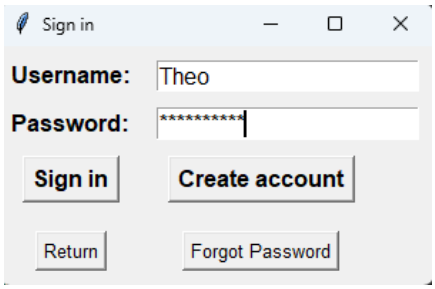
This time the user is signed in, but this works whether the user is signed in or not.



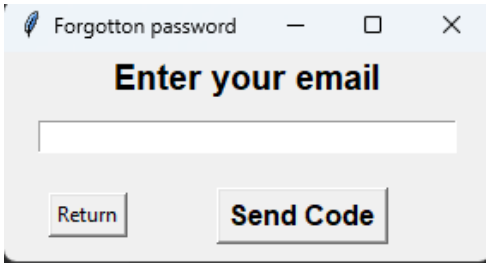
E5



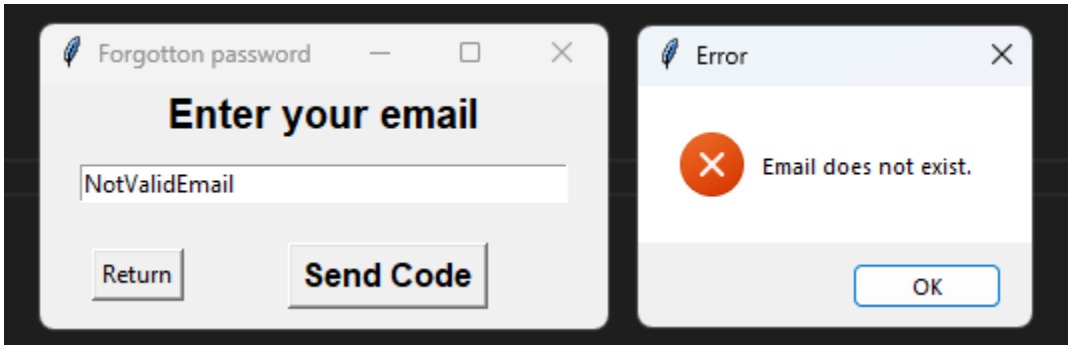
E6



E7

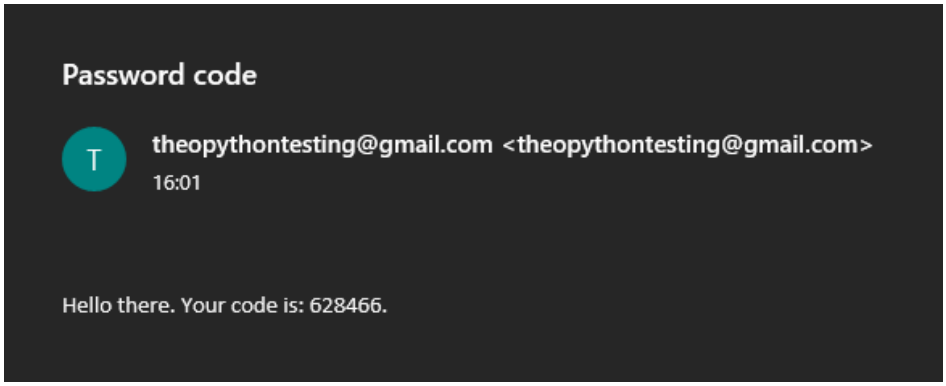


E8

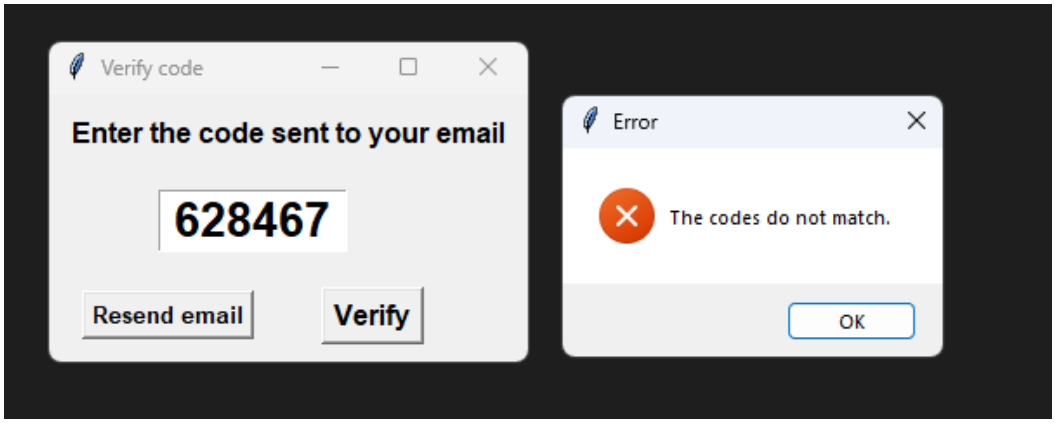


E9

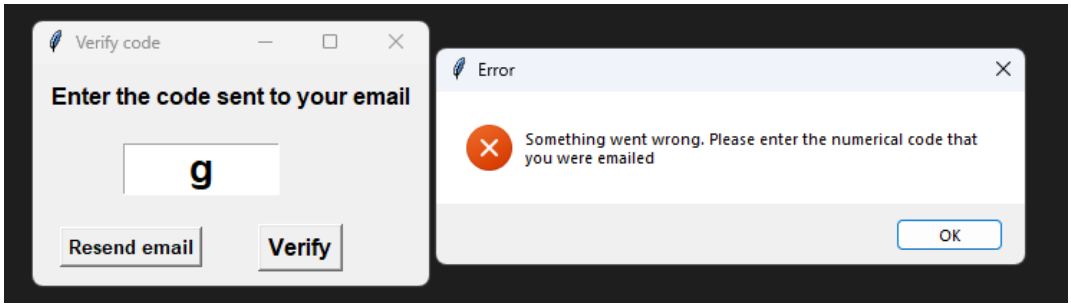
This is from the user's actual email account



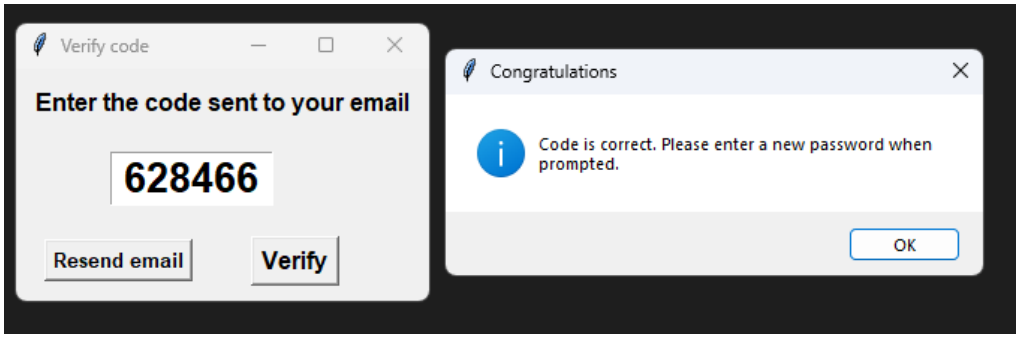
E10

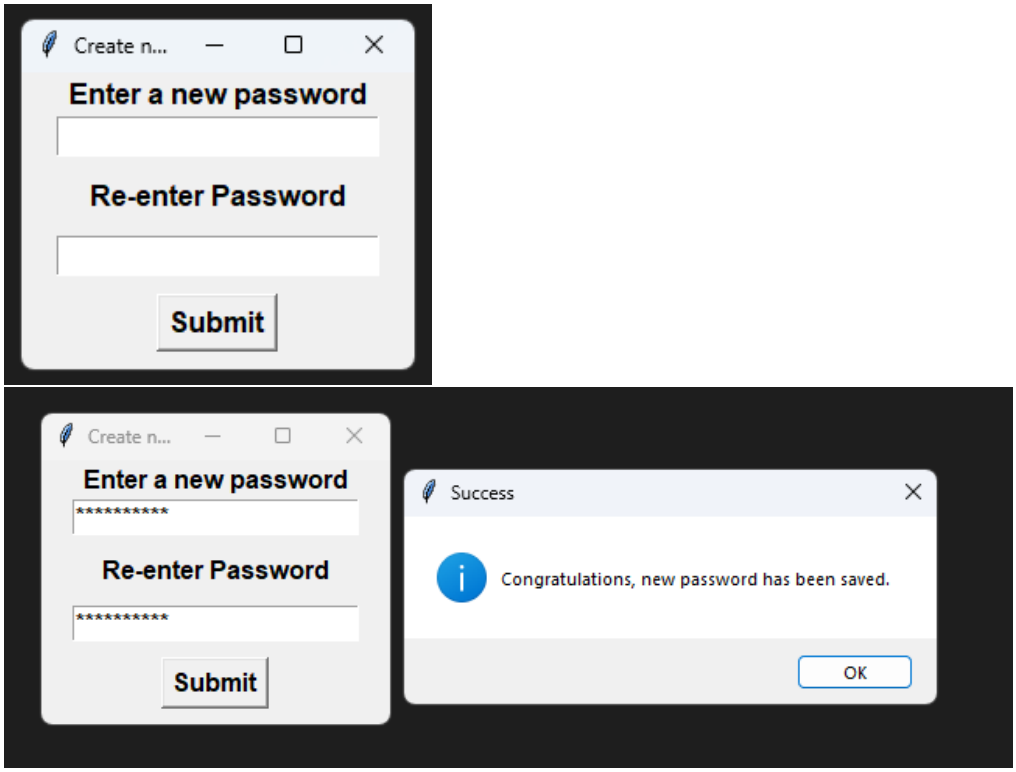


Any other issues (e.g. if non-numerical sequence is inputted)



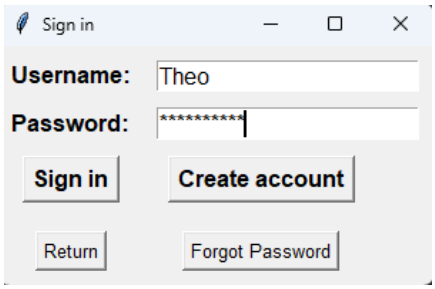
E11



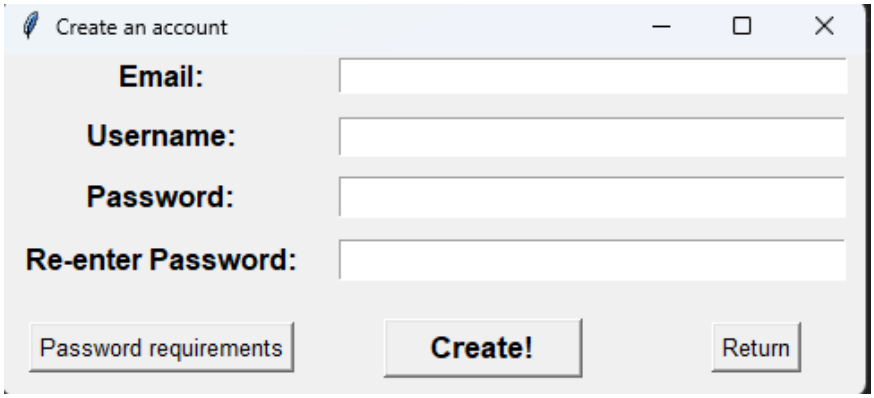


If user creates a new, valid password and they both match.

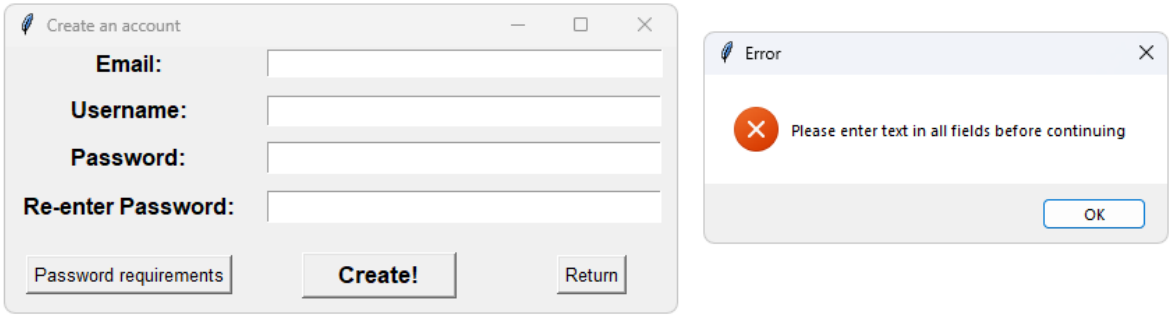
E12



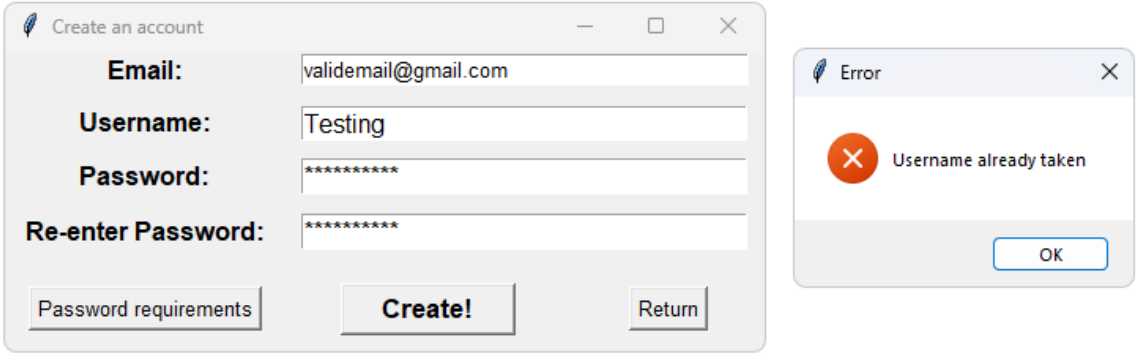
E13



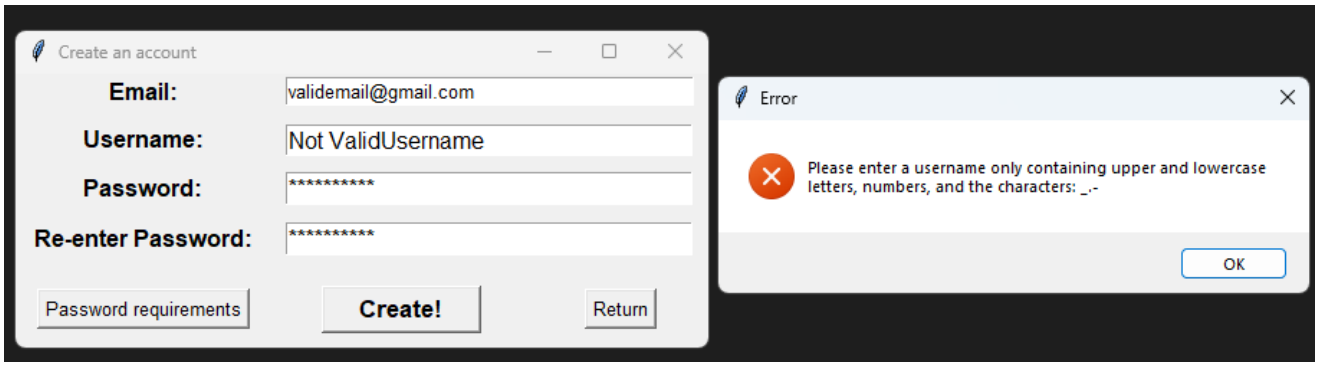
E14


Create an account
Email:
Username:
Password:
Re-enter Password:
Password requirements
Create!
Return
Error
Please enter text in all fields before continuing
OK

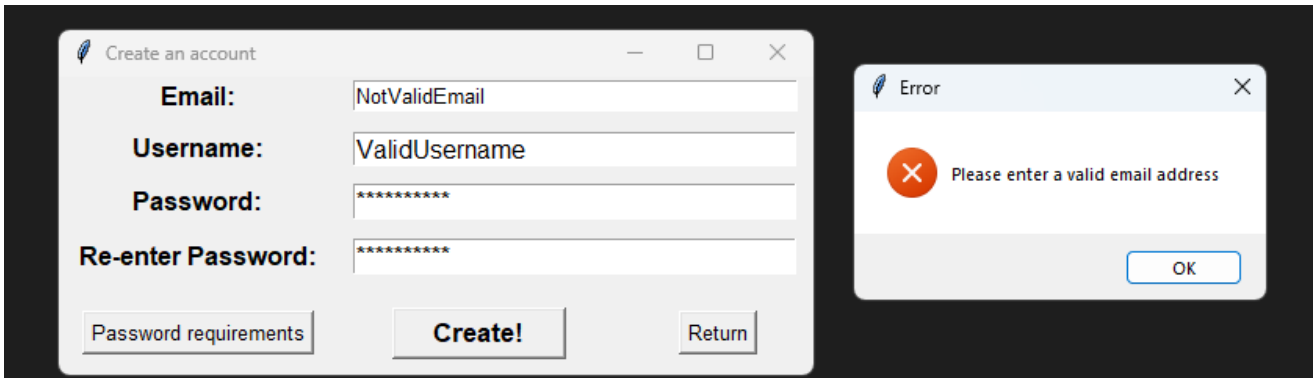
E15


Create an account
Email:
validemail@gmail.com
Username:
Testing
Password:
****
Re-enter Password:
****
Password requirements
Create!
Return
Error
Username already taken
OK

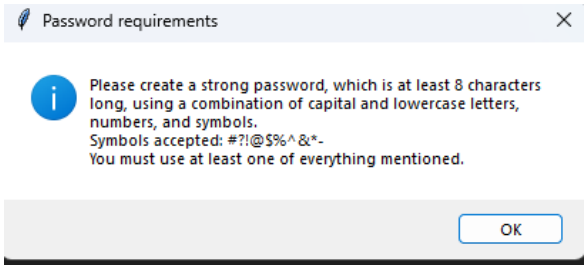
E16

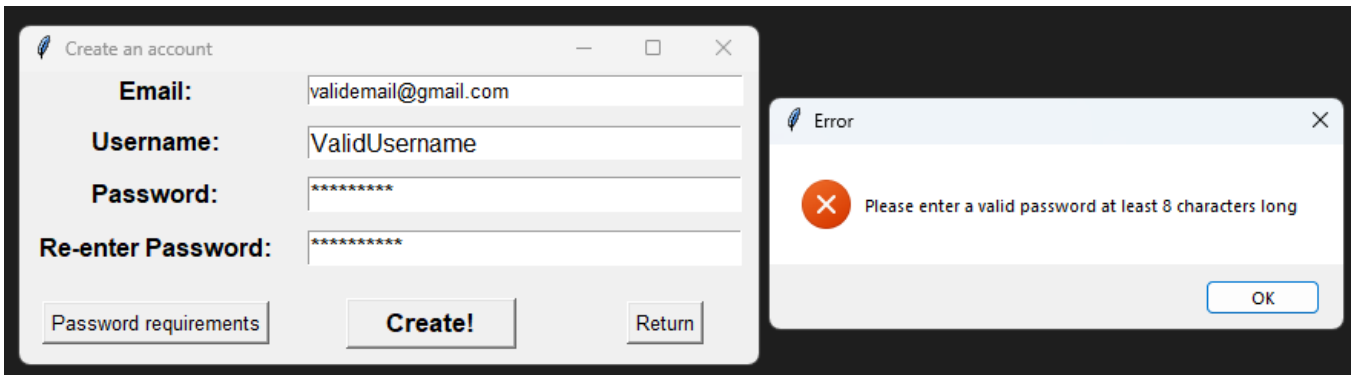

Create an account
Email:
validemail@gmail.com
Username:
Not ValidUsername
Password:
****
Re-enter Password:
****
Password requirements
Create!
Return
Error
Please enter a username only containing upper and lowercase letters, numbers, and the characters: _.-
OK

E17

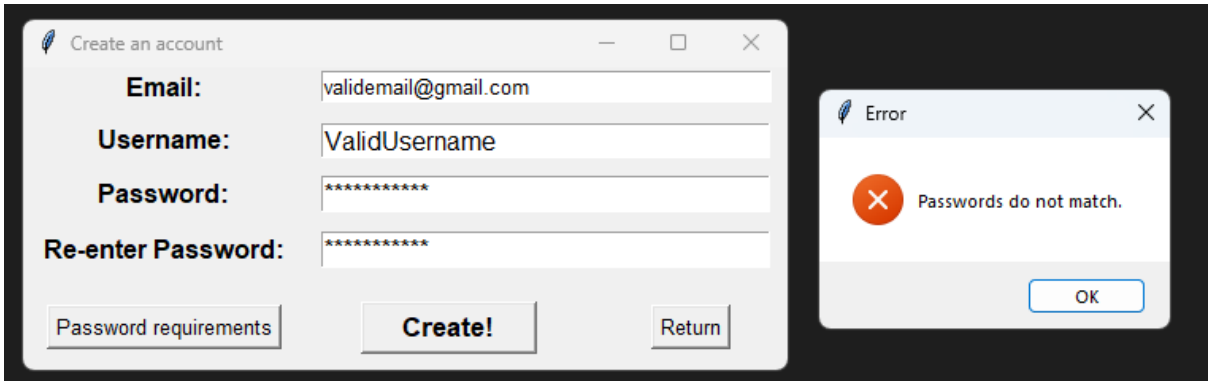

Create an account
Email:
NotValidEmail
Username:
ValidUsername
Password:
****
Re-enter Password:
****
Password requirements
Create!
Return
Error
Please enter a valid email address
OK

E18

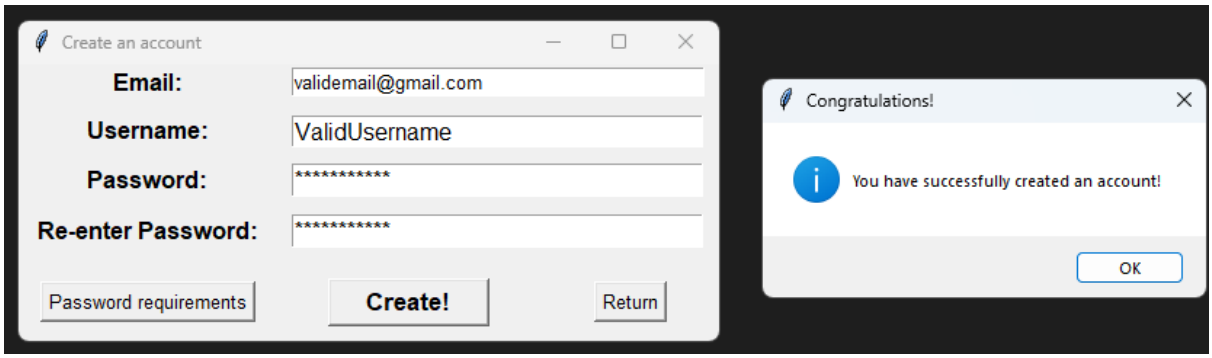

Password requirements
Please create a strong password, which is at least 8 characters long, using a combination of capital and lowercase letters, numbers, and symbols.
Symbols accepted: #?!@$%^&*-
You must use at least one of everything mentioned.
OK


Create an account
Email:
validemail@gmail.com
Username:
ValidUsername
Password:
****
Re-enter Password:
****
Password requirements
Create!
Return
Error
Please enter a valid password at least 8 characters long
OK

E19


Create an account
Email:
validemail@gmail.com
Username:
ValidUsername
Password:
****
Re-enter Password:
****
Password requirements
Create!
Return
Error
Passwords do not match.
OK

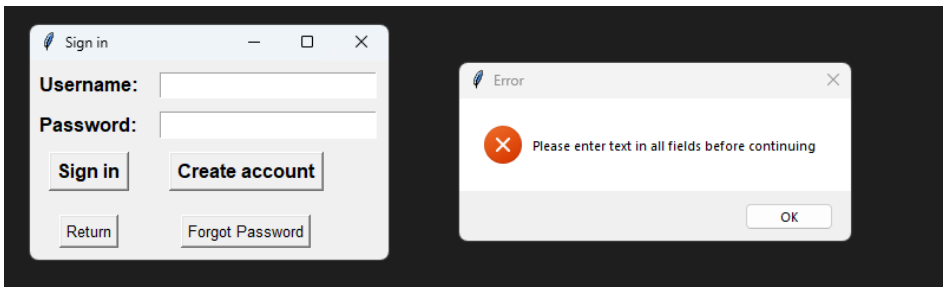
E20

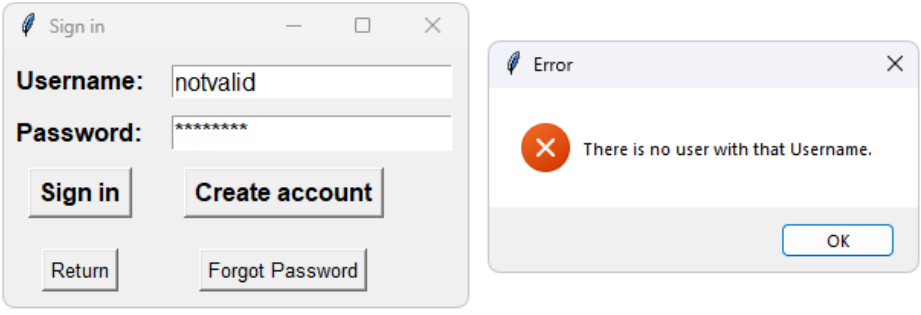

Create an account
Email:
validemail@gmail.com
Username:
ValidUsername
Password:
****
Re-enter Password:
****
Password requirements
Create!
Return
Congratulations!
You have successfully created an account!
OK

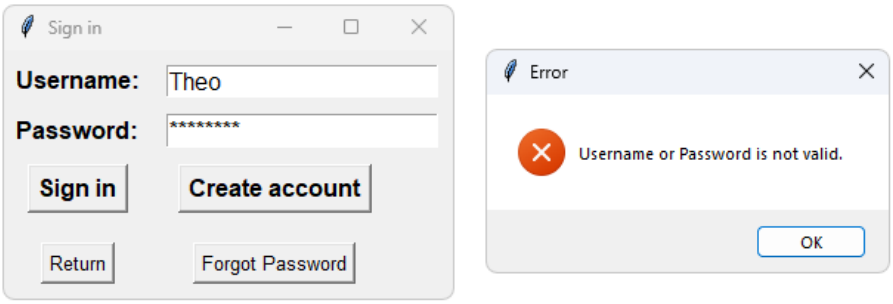
E21


Sign out
Signed in as: Theo
Game options

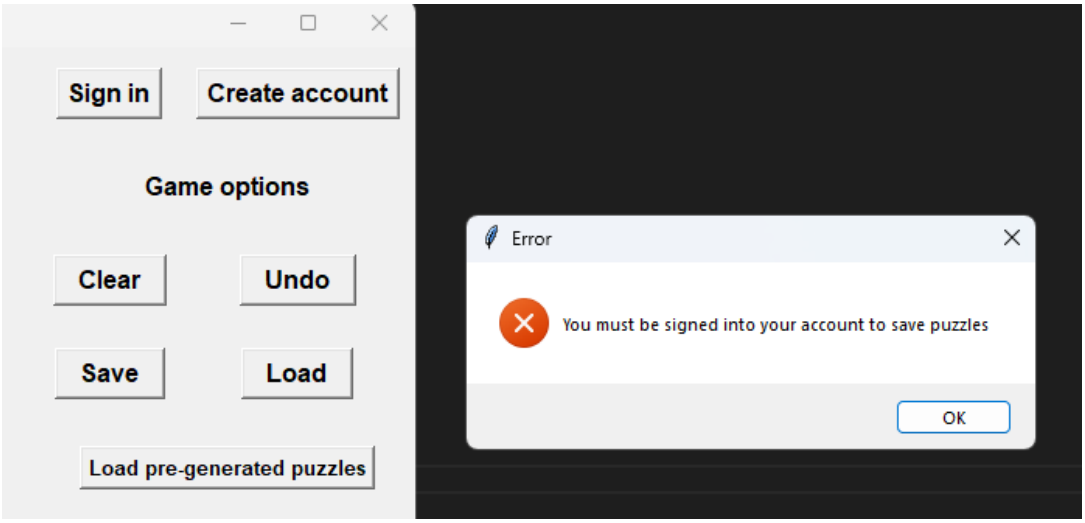
E22


Sign in
Username:
Password:
Sign in
Create account
Return
Forgot Password
Error
Please enter text in all fields before continuing
OK

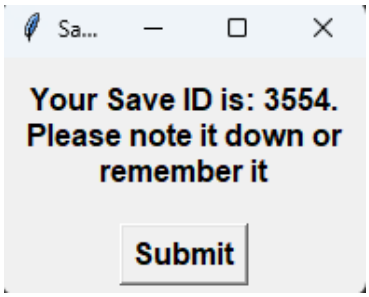

Sign in
Username:
notvalid
Password:
****
Sign in
Create account
Return
Forgot Password
Error
There is no user with that Username.
OK


Sign in
Username:
Theo
Password:
****
Sign in
Create account
Return
Forgot Password
Error
Username or Password is not valid.
OK

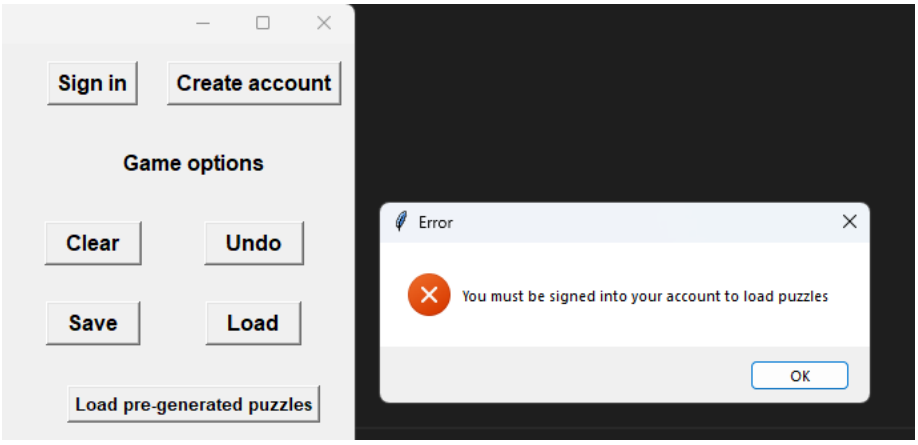
E23


Sign in
Create account
Game options
Clear
Undo
Save
Load
Load pre-generated puzzles
Error
You must be signed into your account to save puzzles
OK

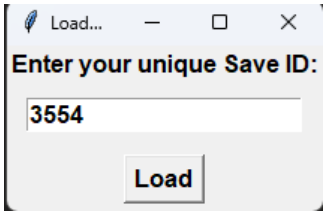
E24


Sa...
Your Save ID is: 3554. Please note it down or remember it
Submit

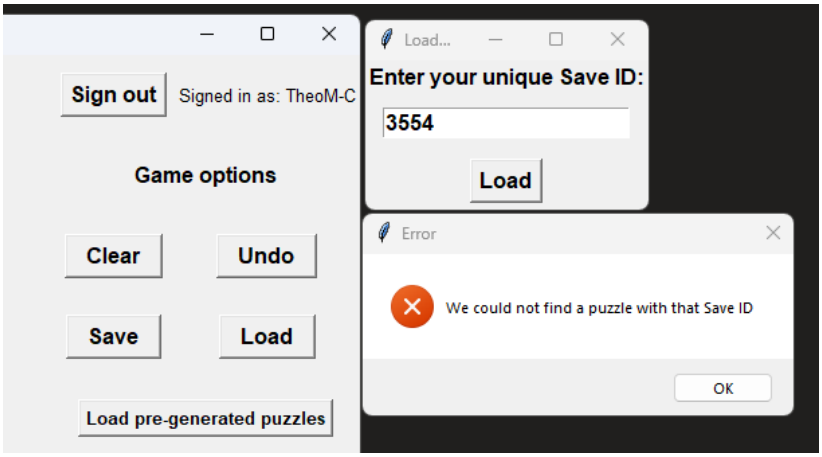
E25



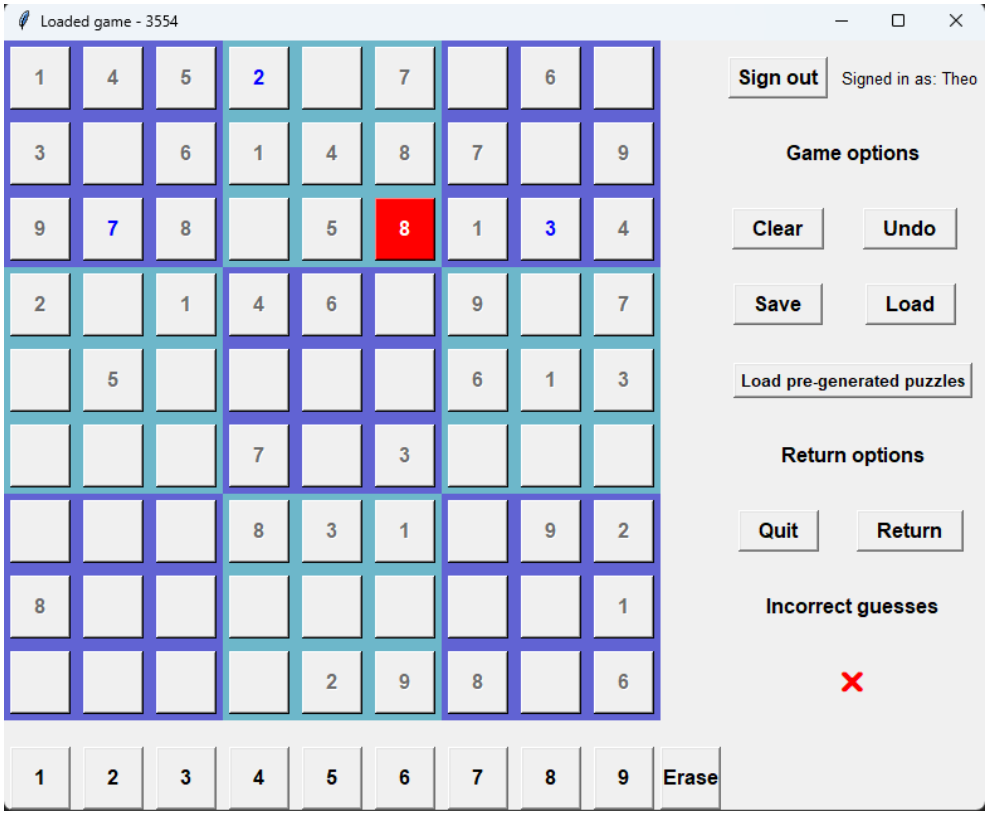
E26



Error message if user "TheoM-C" attempts to load user "Theo" 's puzzle with the same SaveID, as puzzle progress is locked to a single account (in this case, user "Theo")

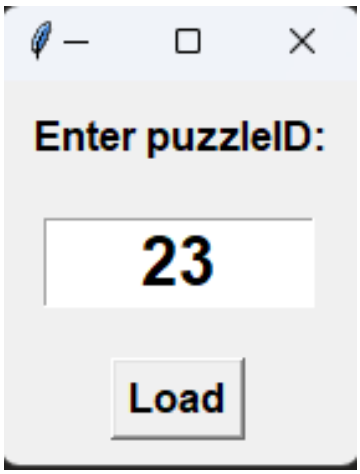


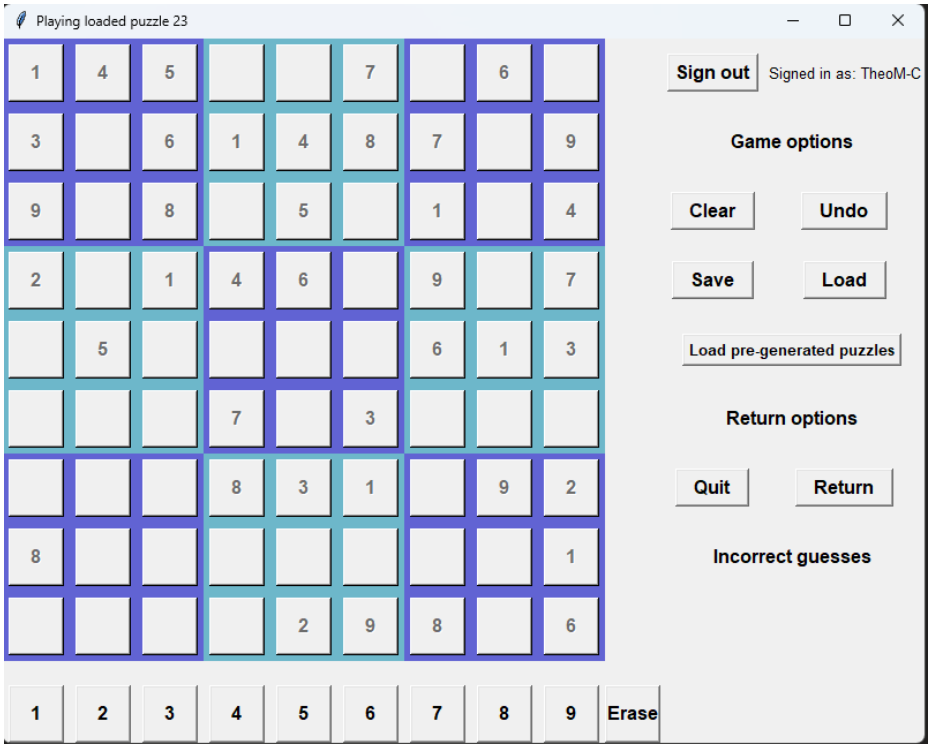
E27



User "Theo" has loaded their save which they previously edited - the incorrect guesses and lives are changed to be what they were at the time the puzzle was saved

E28

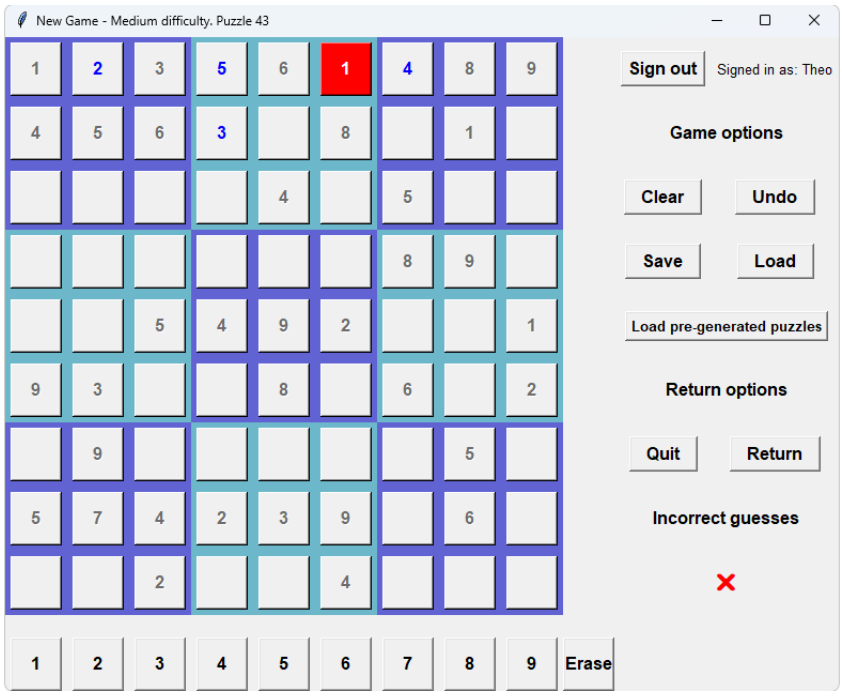




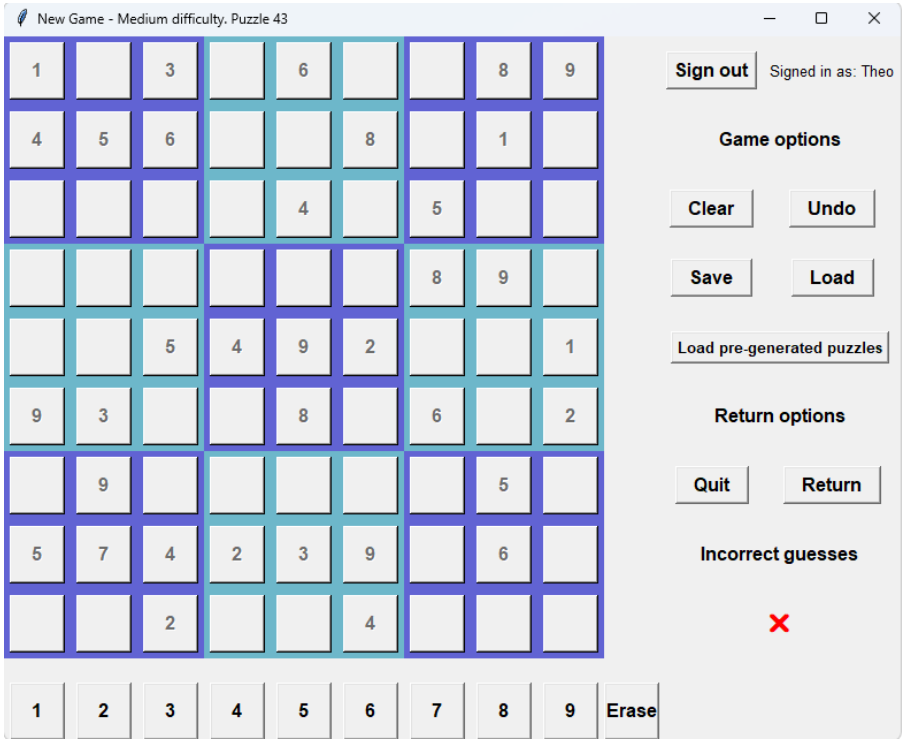
As you can see, a different user has loaded the same starting board. Therefore, just by sharing the Puzzle ID, any users can access the same starting puzzle and have a go but saves and progress are tied to only a single account, so in this case user "TheoM-C", and anyone, can access the same starting board as user "Theo", but they cannot access user "Theo's" edits and save progress.

E29

User's modifications before pressing the "clear" button (they inputted 1, 2, 3, 4, 5)

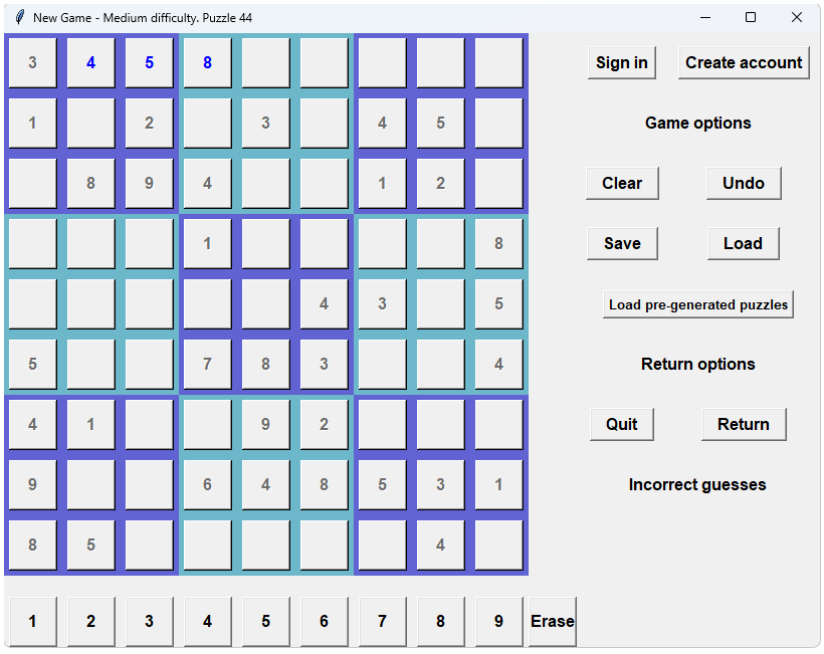


User's board after pressing the "clear" button (their incorrect guesses stays there, as they have lost a life for putting the "1" in the incorrect place)

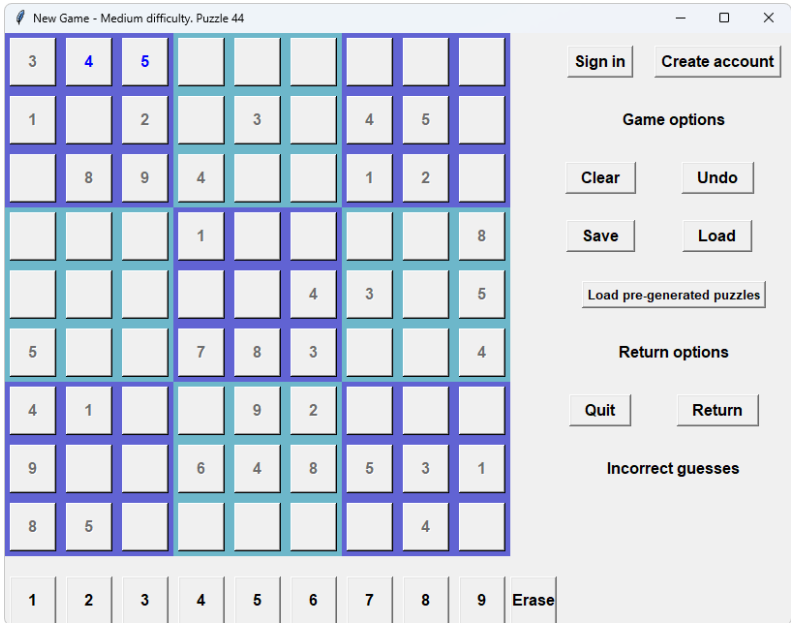


E30

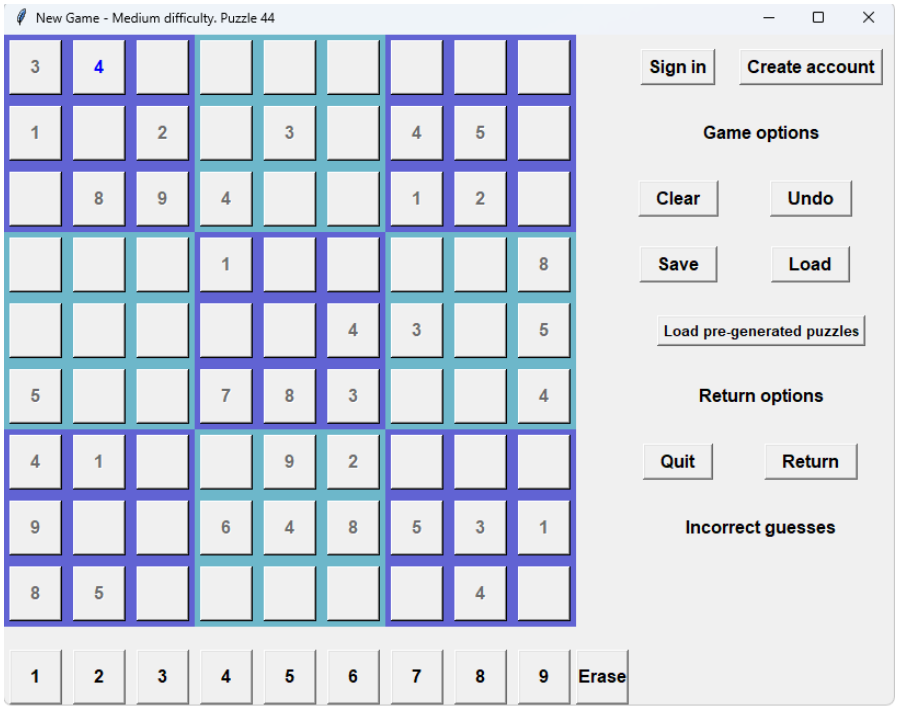
User pressed 4, 5, 8 in that order.



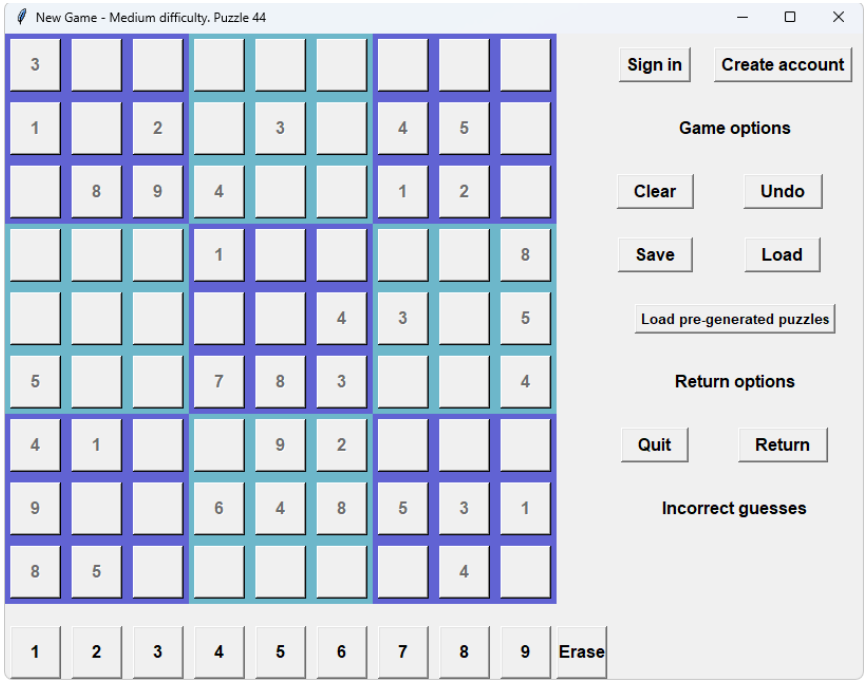
Undo button pressed once



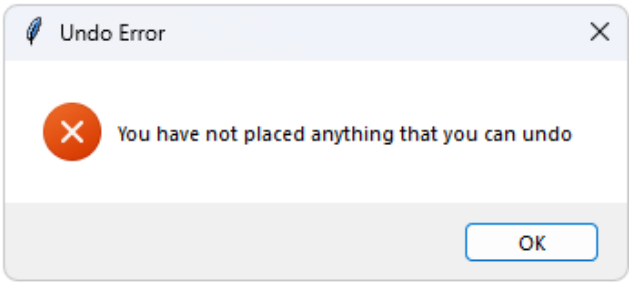
Undo button pressed again



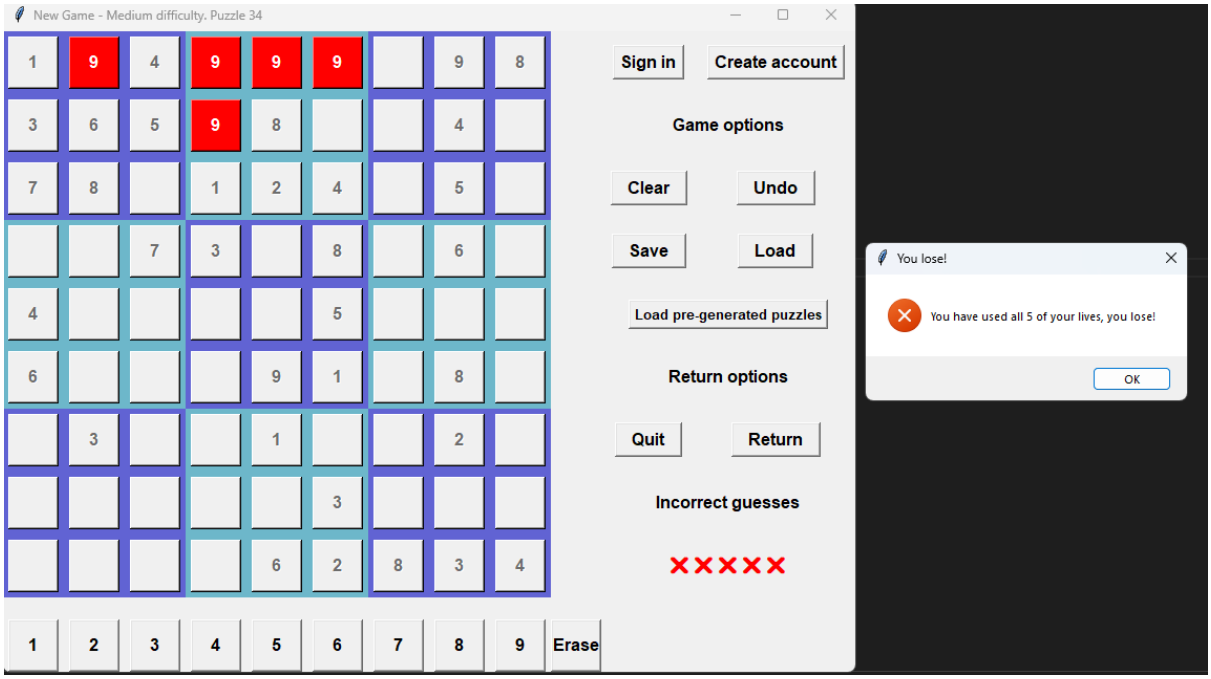
Undo button pressed a 3rd time



E31

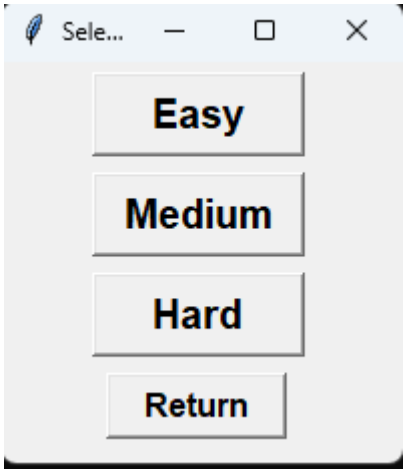


E32

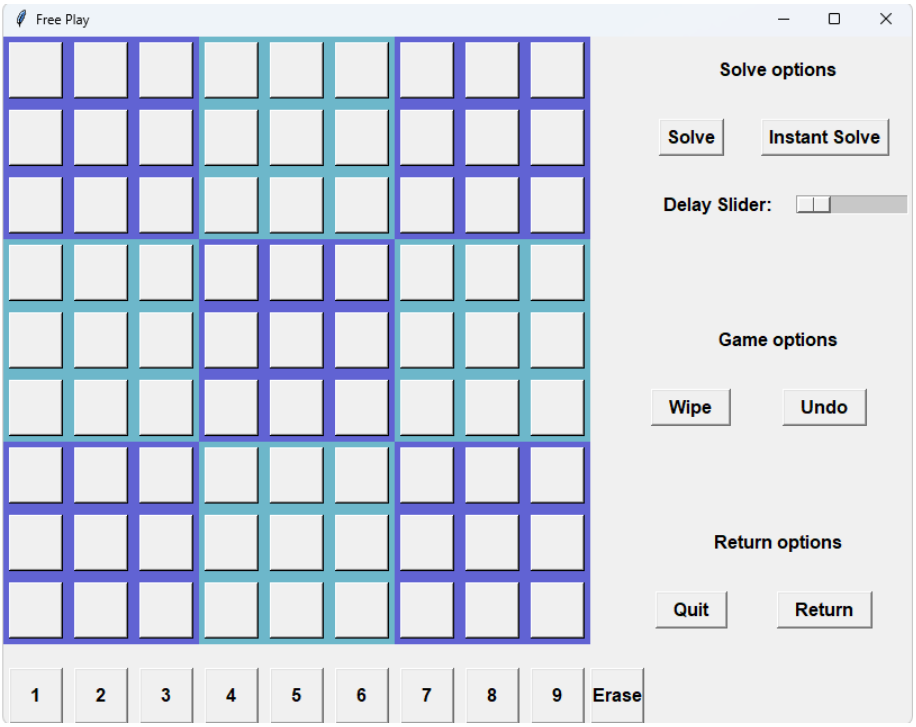


User is then returned to select difficulty screen, after pressing “Ok” on the error message.

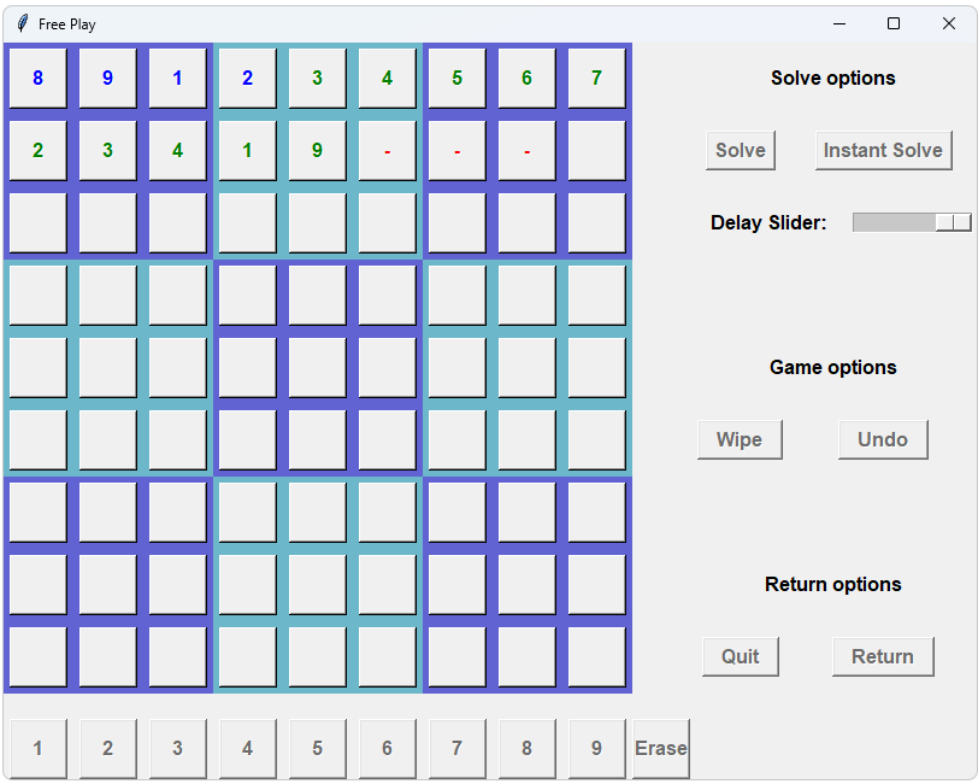
E33



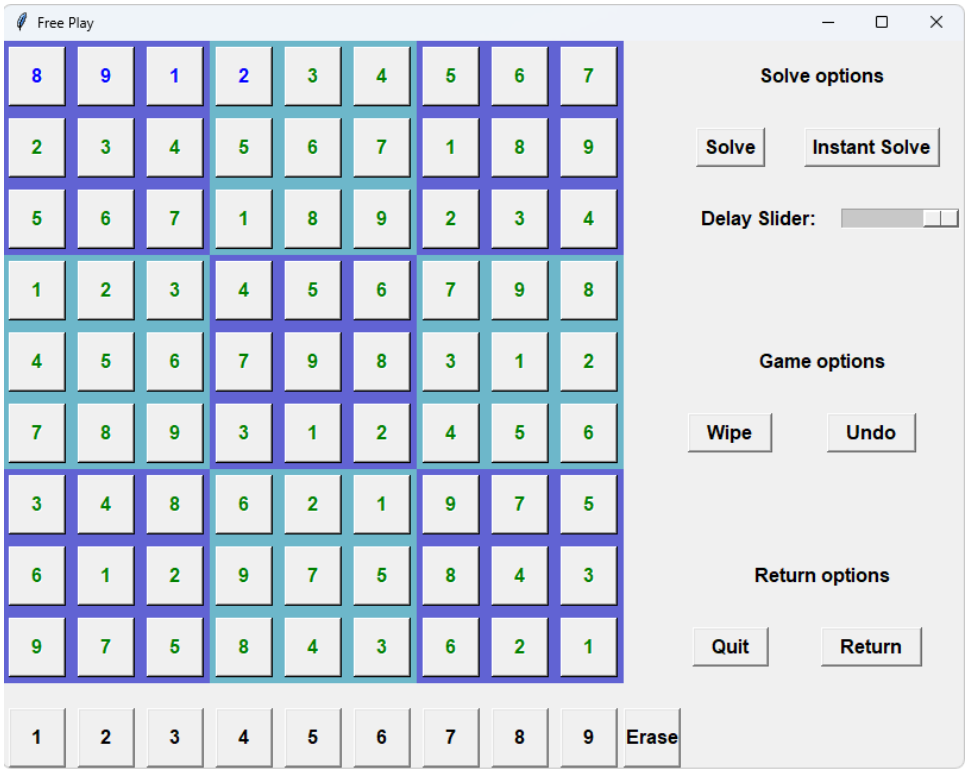
E34



E35

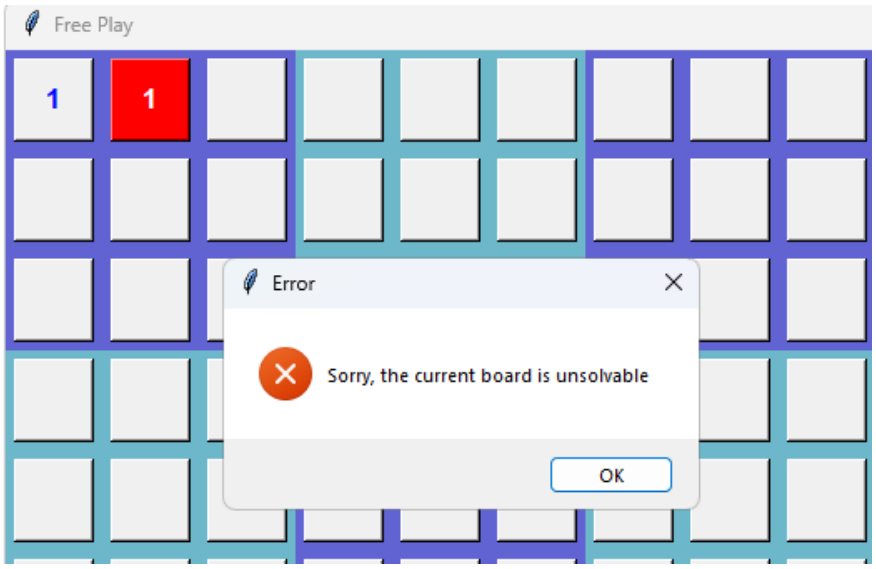


Midway through solving (with delay turned all the way up), red showing where the solver has got to but at this moment it was backtracking, and green showing correct numbers placed by the solver. Blue were the user's inputs

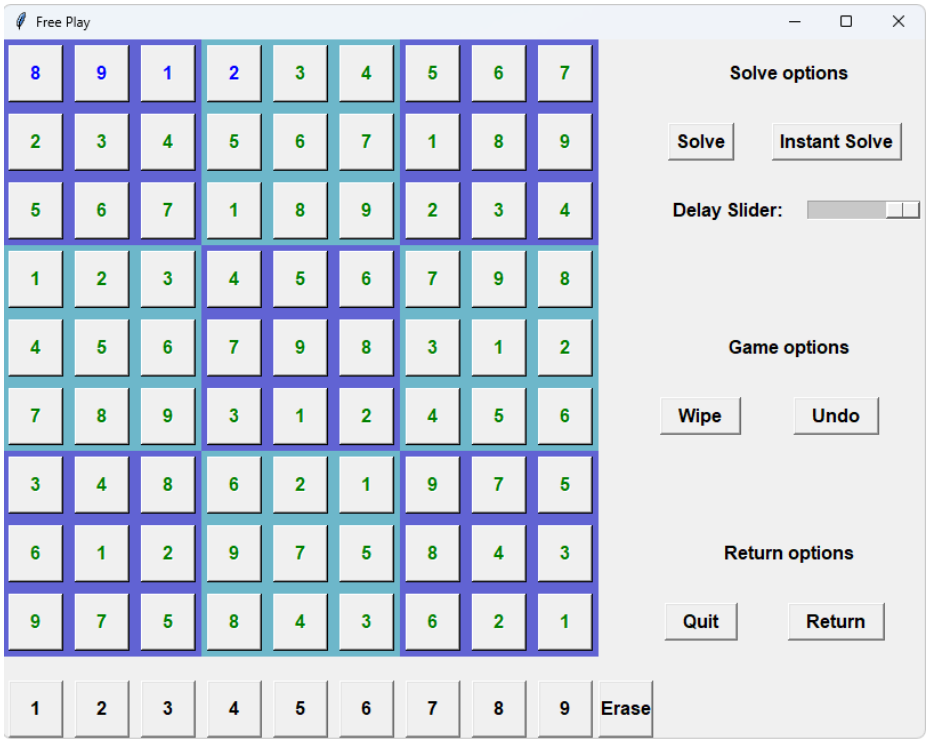


Fully solved board after the solver has completed. (User's inputs still in blue)

E36

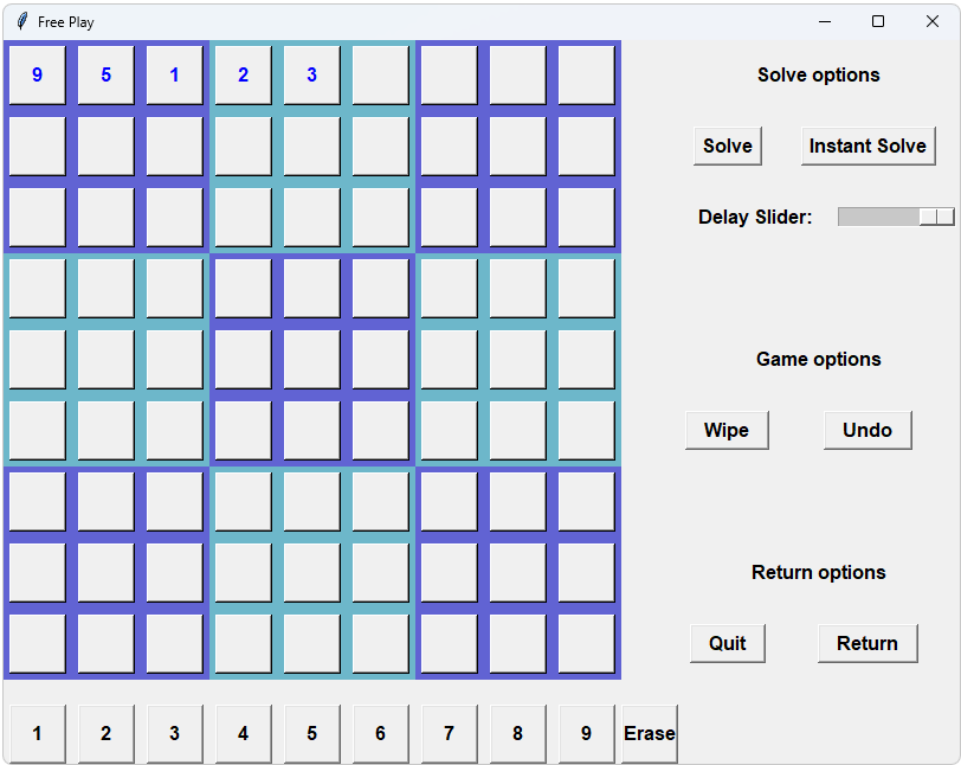


E37

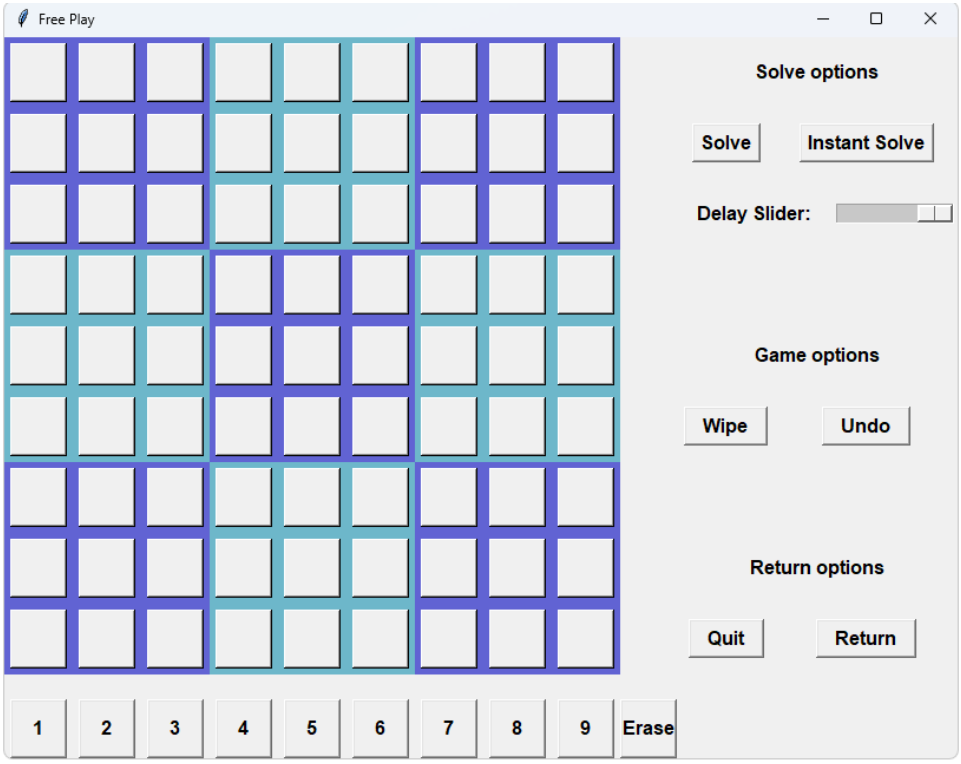


User entered 8, 9, 1, 2 and pressed “Instant Solve”

E38



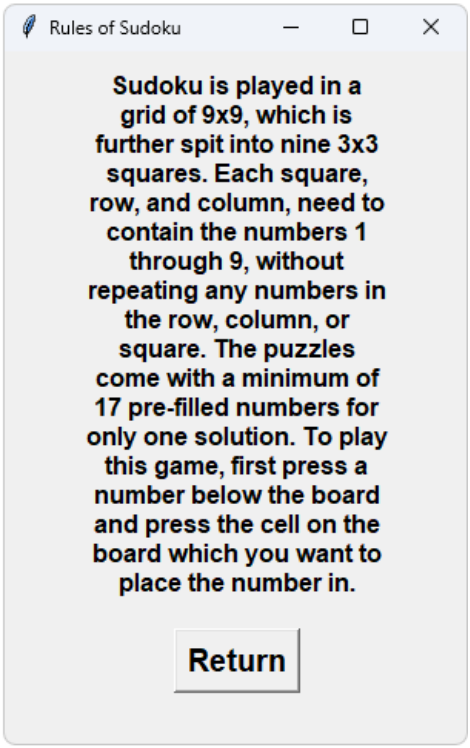
The user has entered some numbers onto the board



Board after the “Wipe” button has been pressed

E39

E40


Rules of Sudoku
Sudoku is played in a grid of 9x9, which is further spit into nine 3x3 squares. Each square, row, and column, need to contain the numbers 1 through 9, without repeating any numbers in the row, column, or square. The puzzles come with a minimum of 17 pre-filled numbers for only one solution. To play this game, first press a number below the board and press the cell on the board which you want to place the number in.
Return

Evaluation

I will revisit my main objectives, and the responses with my end user.

I showed the program to my sister, my end user. She really enjoyed navigating through the menus, refreshing herself on the rules of Sudoku. She said it was very easy to understand how to do what, and she enjoyed viewing the solving algorithm solve her inputted numbers as a puzzle, and she liked the delay slider so she could make the solver go as fast or slow as she liked. Often she used the instant solve button to generate a solution instantly. I explained the accounts, which she understood quickly as it is a common occurrence in modern programs. She was a bit confused about the concept of saving progress but when I showed her how to save and load previous puzzles, she seemed confident that she could replicate it.

She was worried that anyone could access her save progress, but I showed her that the progress was tied to her account, and no one else, even with the Save ID, could access her save data, and she seemed reassured.

She created her account easily enough, although the password creation took a few attempts as her password was too simplistic, but she liked the fact that she could see what the requirements were by pressing the button to see the window with the information, and she really liked how the password was blurred out so I couldn't see it! I also showed her how her password was hashed in the database so no one could ever find out what it was, but she could reset it if she wants. She was happy by this and noted how all modern programs need this feature as users often forget passwords.

She liked how we could both load the same starting puzzles and save our respective progress.

Her only improvement would be to add a timer when the user plays the game, so people can compare times and you can save your time with your progress in the game. I thought that was a really good suggestion, and one which I would definitely add in the future, or at the start if I was doing this project again.

Overall, Alyssa was really impressed with the program and said she enjoyed how easy it was to use and was shocked at how fast and effective the solver was.

I have gone back to my main requirements, and requests by my end user, and check if they have been met

Requirement	**Has it been met?**
Is the user presented with a welcoming window asking them what they want to do?	Yes
Does the new game option open a game window where the user can play a pre-generated puzzle of their difficulty choosing?	Yes
Can the user sign in and create account to save and load puzzles?	Yes
Can the user reset their password?	Yes
Can the user experiment with the Sudoku solver and use the Free play mode to explore and see the solving algorithm work and solve their custom puzzle?	Yes
Can the user see the rules of Sudoku in its own window?	Yes
Can the user navigate through all of the UI, and use the “return” buttons to return to the previous window?	Yes
Can the user quit the game at any time?	Yes
Did I create a tool that works on computers?	Yes (as discussed with end user)
Can the user input their own numbers and solve their challenges using the program?	Yes (as discussed with end user)
Does the program generate puzzles so the user can play the game?	Yes (as discussed with end user)
Is there different difficulties to select from?	Yes (as discussed with end user)
Can you save and load progress through your account?	Yes (as discussed with end user)
Is the UI simple enough to understand?	Yes (as discussed with end user)

My reflections

Overall, I am really impressed by this program and how it turned out. I identified and spoke to a member of my target audience, and selected and modified my requirements to reflect that conversation.

I designed my program using multiple flowcharts, and the algorithms used I carefully thought through and documented, carefully planning as I went along. This helped shape the overall code as I could really stay focused on what I had to do code-wise for the algorithm to be implemented.

I used previous prototypes to shape my final UI design, taking features I liked and continued them throughout the project, and used parts I didn't like to make me rethink how to do certain aspects, and eventually refine it into features that I am proud of. This helped me craft a project that I am really proud of, and is shaped by parts which didn't go so well in the past that I had since rethought.

I tested my program throughout development, making sure every function and class worked as expected before being integrated back into the full program. This is shown during my testing stage of development, as my project passed every requirement that I set out to achieve, and I have taken feedback from my end user on board throughout, as I always had the users in mind, especially designing the UI, where simplicity was a priority, and I think that my project reflects that well.

Regarding the program, I am really pleased with its outcome. It was my first time using so many concepts, such as hashing, linked list maintenance, stacks, and more. I enjoyed the object-oriented approach to developing programs, and my project's loosely coupled modules allowed changes not affecting other modules too much. This approach allowed easy testing and maintenance, as each class and function can be assessed one at a time.

I stuck to a naming convention, using underscores to represent spaces, using capitals for constants, and capitals to signify a class. I used meaningful identifiers for variables, functions, and classes, to allow for ease of understanding. Plenty of comments were used to explain what parts of my code were doing.

All of these combined made my coding easier, as I spent less time confused about previous code and what it does, and more time implementing my algorithms and making the program user-friendly.

What I would change in the future / if I was doing it again

After speaking to my end user again, a timing system would definitely be a priority for future upgrades, or if I was doing the project again. This would allow user's to see their improvements in times and compare times against each other.

Also from this conversation, I picked up that more information should be provided to the users, for example explaining the saving and loading system and how their password's are safe, just to make the experience even more rewarding and informative.

I noticed myself, that instead of a user having to remember their Save ID's, a new window could be created which allows a user to see all of their saves, and they can select one, or possibly from a drop-down menu. This way they can see every puzzle they have saved, and can choose one accordingly.

Similarly to that idea, I imagined how a window / dropdown could be implemented to allow users to see every starting puzzle generated, and perhaps filter by difficulty instead of having to type in the Puzzle ID, and they again could select a starting puzzle to input into their program.

All of these changes would increase the user's satisfaction with the program and are definitely points which I will take into my next project.

This has all been an incredible learning experience for me, and I am deeply proud of my progress and the final outcome of the project.

Despite all the highs and the lows, I have learnt a huge amount from completing this project, and I will hopefully carry that throughout my programming journey.