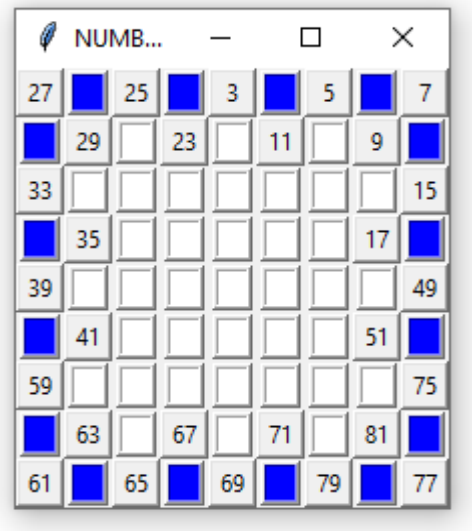


NUMBRIX PUZZLE

(REPORT)

GROUP MEMBERS:

AYESHA AAMIR (CS-19026)
IZMA AZIZ (CS-19020)

COURSE CODE: CS-218

COURSE TITLE: DATA STRUCTURES AND ALGORITHMS

SUMBITTED TO : Ms.UROOJ AINUDDIN

NUMBRIX PUZZLE

WHAT IS A NUMBRIX PUZZLE?

Numbrix is a puzzle game that rely on a simple strategy. Unlike other number-placement puzzles, you do not need math skills or lots of guessing to complete this puzzle. The puzzle requires you to make an unbreakable path, that allows you to count from 1 to 81 in a 9×9 grid. First look at the numbers provided within the grid. Select a number from 1 to 81 that does not already appear on the grid. Insert the number you selected horizontally or vertically and in sequence to one of the provided numbers. Insert additional numbers into the grid and fill all the empty spaces. The numbers you insert must be consecutive and form a horizontal or vertical path.

LIBRARY USED: tkinter

DATA STRUCTURE USED:

To implement the Numbrix Puzzle, we have used a 2-dimension list where each sublist represents a row having 9 values. A 2 dimensional list is a list that requires two indices to identify an element so we used this data structure to access and manipulate the rows and columns of our Numbrix puzzle grid.

CLASSES:

Our game program has one class:

***cell* Class:** The Numbrix puzzle has 9 boxes in each row and column and all of these boxes are surrounded by other boxes in the puzzle. So to be able to manipulate these boxes for our game implementation we decided to convert them into cell objects. Each cell object has certain attributes to itself namely, value, up_cell_value, down_cell_value, right_cell value and left_cell_value. Using these attributes, we have implemented our Numbrix Puzzle.

FUNCTIONS:

The functions that we use in our main program are:

1. ***create_cell**:* this function uses a 2d puzzle list to return another 2d list containing *cell* class objects.
2. ***start_time**:* This function starts a timer as soon as the user presses START so as to keep track of the time taken by the player to complete the puzzle.
3. ***is_valid**:* whatever the user enters in the entry widget is passed to this function and what this function does is check whether that entered value is an integer or not. If it is an integer the function returns TRUE else, it returns FALSE.
4. ***is_surrounded**:* a cell object is passed to this function and what it does is that it checks whether all of the adjacent values of the cell object i.e. up_cell_value, down_cell_value, right_cell value and left_cell_value are equal to 0 or not. If all the adjacent values are not

equal to 0. The function returns TRUE, meaning the cell is surrounded from all sides. If even one of the adjacent values is equal to 0, the function returns FALSE.

5. ***warning****:* The warning function takes two parameters namely, the entry from the entry widget and a string of the kind of warning we need to show. It creates a pop up window to display this warning depending on the type of warning that has been passed to it.
6. ***is_complete****:* When the user presses STOP we use this function to check whether the all the entry widgets of the grid have been filled or not. The function returns TRUE if all of them have been filled in.
7. ***check_numbrix****:* We use this puzzle to check whether all of the entries that the player has entered match the answer key (lst) of the numbrix puzzle.
8. ***win****:* This function is used to display a pop up window showing the player that they've successfully completed the puzzle and also to display a timer to show them how long it took for them to complete the puzzle.
9. ***lost****:* This function does the opposite of the win function meaning it shows the player that they've lost. If the player has finished the puzzle completely but the sequence doesn't match the answer key (lst) then the message displays the timer and a losing message. However, if the player has clicked the STOP button but the puzzle isn't complete it shows that the player has resigned and also displays the time it took for them to resign.
10. ***hint****:* We've created this function to help the player incase they've gotten stuck somewhere and can't figure out how to continue the sequence.
11. ***in_range****:* This function checks whether the integer that player has entered in the entry widget is between the ranges of 0 to 81.
12. ***stop****:* This function is called when the player presses the STOP button. It gets the exact time when the played pressed STOP then subtracts it from the time when the player pressed START and displays the total time. It also calls the lost and win functions depending on how the player has completed the puzzle.
13. ***entry****:* This function is called when the player presses the ENTER key after entering a value in the entry widget. It checks whether the entered value is_valid() and in_range() and if both of these functions return TRUE this function sets the entry value to the value attribute of the cell object. It also updates the adjacent cells of that particular cell and changes their colour if they are an entry widget and now surrounded due to the entry in that cell.
14. ***grid****:* We use the grid function to implement our numbrix puzzle window. This numbrix puzzle window has three different types of cells, a label widget box, 1 blue entry widget and 1 white entry widget. All of these cells are set using three conditions.

ALGORITHM OF THE GAME:

1. As soon as we run the program the first thing that happens is that a window is created using the tkinter library. On this window a few instructions have been displayed for the player. Under these instructions 3 buttons are show: START, STOP, HINT.

2. START:

2.1. As soon as the player presses the START button the function *grid()* is called. This function calls another function *create_cell()* which returns a 2D list of cell objects and this list is then assigned to the cells variable. Then this function calls the *start_time()* function to get the time when the player pressed START. Now, to implement the Numbrix Puzzle grid this function uses the 2D list stored in the *cells* variable to make three different types of widgets depending on the cell values.

2.1.1. If the cell's value attribute is equal to 0 and the cell is surrounded from all four sides (we check this using the *is_surrounded()* function): The program creates a blue entry widget.

2.1.2. If the cell's value attribute is equal to 0 but it is not surrounded from all four sides: The program creates a white entry box. We do this to inform the player that these blue entry widgets should be their priority while solving the puzzle.

2.1.3. If the cell's value attribute is not equal to 0: The program creates a label widget instead of an entry widget.

2.2. Now as soon as the player enters a value in the entry widget and presses the ENTER key. Another function *entry()* function is called. This function checks the entry for certain conditions:

2.2.1. If the entered value is not an integer (check using the *is_valid()* function): we call *warning(entered value, 'valid'*) to display a warning window.

2.2.2. If the entered value is not between the ranges of 0 to 81 (check using *in_range()* function): we call *warning(entered value, 'range').*

2.2.3. Else: we set the entered value to the value attribute of our cell object and also update the up_cell_value, down_cell_value, right_cell value and left_cell_value. When updating these adjacent cells, we also check if these adjacent entry cells are now surrounded. If they are we change their colour to blue.

3. HINT:

If the player presses the HINT button, we call a function *hint()* which converts the first entry widget it could find into a label so as to help the player if they're stuck while solving the sequence.

4. STOP:

4.1. Now when the player has completed their puzzle or they feel like they can't continue any longer they will press the STOP button. As soon as this button is pressed the function *stop()* is called. This function first gets the time when the player has pressed the button STOP. Then it checks a few conditions:

4.1.1. If the player has completed the puzzle (we check this using the *is_complete()* function) and if the completed puzzle is correct (we checks this using the *check_numbrix()* function): Then we call the *win()* function to display a pop

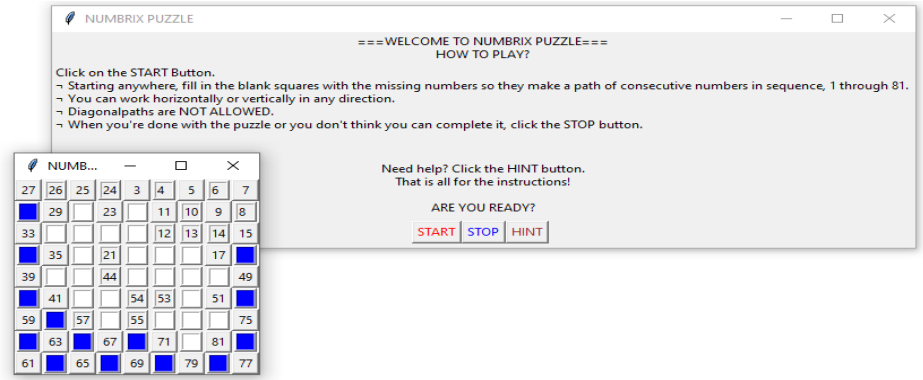
up window letting the player know that they've filled the puzzle correctly in _ amount of time.

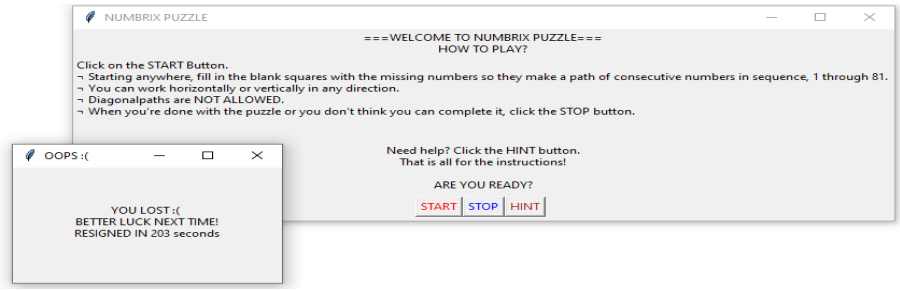
4.1.2. If the completed puzzle is not correct: Then we call the *lost()* function to display a window letting the player know that they've filled the puzzle incorrectly in _ amount of time.

4.1.3. If the player has not completed the puzzle: then we call the *lost()* function again but this time to display a pop up window showing that the player had resigned in _ amount of time.

TEST RUN #1

(RESIGNING IN BETWEEN)





TEST RUN #2

(WINNING SCENERIO)