1806ICT Assignment – Cascade

Himath Ratnayake
s5209861

Table of Contents

Problem Statement

This task involves producing a program in the ‘C’ language that allows a user to play a single player game on an n x n gameboard of a specified input size ‘n’ populated with random symbols from a pre-defined list.

Through user inputs, symbols must be swapped to create 3 or more of the same symbols in a row or column, at which point that row or column of >3 symbols disappear, and the user’s score is incremented. Then, symbols above the disappeared match must ‘cascade’ down to fill the spaces left behind.

Each game must also be able to get terminated, or saved and then loaded later to play again, preserving the board layout and time elapsed at time of last save.

1. Program Inputs, Outputs, and Related User Interactions/Requirements

PHASE 1 – START UP		
Input Type	**User Input Commands**	**Interaction Outcomes (Output)**
Load Previous Save	**Prompt:** "Would you like to load a previous save?" **Input:** 'y' for yes or 'n' for no	**'Yes' Input:** User will be prompted, *"Enter name of save file you would like to load".* Upon entering name of save file, the file's gameboard will be displayed and be ready to play with. If an invalid save file name is entered, they will be re-prompted, "*Invalid name! Please try again"* **'No' Input**: Advance to next prompt
Board Size	**Prompt:** "Enter Gameboard Size:" **Input:** Value must be >3 This value will be stored as &size_of_board	**Invalid Input:** If input number is less than 3, return with "*Board is too small!"* and return to prompt asking for board size. A board size greater than 3 is necessary otherwise, so that matches can be created properly. **Valid Input:** If input number is >3, generate gameboard of given value
Number of available symbols, from the following: • ! • ? • & • $ • # • @	**Prompt:** "Enter Number of Symbols:" **Input:** [Number between 3 and 6] This value will be stored as &symbol_num	**Invalid Input:** If input number is <3 or >6, return with *"Invalid Value"* **Valid Input:** If input is between 1 and 5, advance to next prompt

PHASE 2 – GAMEPLAY		
Input Type	**Input Command Syntax**	**Interaction Outcomes (Output)**
Swap Command	Enter ‘s’, then the coordinates of the symbols needing to be swapped into terminal. E.g. – ‘s 7 5 8 5’	**Invalid Input:** If the swap inputted does not pass any of the following validity checks: • does not create 3 of the same symbol in a row or column • is not between two adjacent symbols (diagonals do *not* count) – note that in the corners of the board, number of valid adjacent symbols is reduced from 4 to 2 as well • contains invalid numbers for given board size (i.e. beyond board bounds) Then return “*Invalid Swap!”* **Valid Input:** Else if input creates 3 values in a row and passes all validity checks, the values will disappear and the other gameboard values related to the disappeared ones will ‘cascade’ down, and the current score value will be incremented by 10 for each symbol matched. Then next input can be entered.

PHASE 3 – TERMINATION		
Input Type	**Input Command Syntax**	**Interaction Outcomes (Output)**
Finish Command	Enter ‘q’ into terminal	Final score scaled against time and time taken will be displayed: *“__ Points, In ___ Seconds”* The user will be asked whether they wish to save *“Would you like to save?”* • If yes, user will be prompted to enter name of file, and then will be returned to start screen. *“Please enter name for this save:”* • If no or the user has saved the file, they will be sent to the next prompt. Program will prompt, “*Would you like to play again?”* • If yes, user will be taken back to start-up phase. • If no, program will end, with a message saying, “Thanks For Playing!”

Other Software Requirements

- When a gameboard is generated, any randomly generated symbols that are placed 3 in a row should immediately be cleared, but not added to the starting score.
- Likewise, if 3 or more symbols in a row or column disappear while playing the game, which then causes another 3 or more symbols to fall into place in a row or column (a chain reaction), these extra symbols will also disappear and cause another cascade of characters falling down. In this case, this extra cascade(s) will also be incremented to the score; it is ensured that the user will not see the same three adjacent symbols in a row or column.
- File names can be up to 30 characters long and are stored in the program's current directory.

2. Program Data

This program contains multiple variables that are utilized by many different functions within the program, so they are set as global variables for brevity in the code.

The integers 'i' and 'j' are also listed as global variables, as they are used to index through the board in each function. By assigning them as global variables, they do not need to be reintroduced as a new variable in each function as "int i" or "int j".

There is also the gameboard – a 2D array with the data type of 'char' *(char **gameboard)* as it displays characters that are to be swapped by the user to play the game. Due to its presence in a majority of the program's functions, it is defined as a global variable so that it's values can be indexed by functions such as 'row_seq' which checks to see if more than 3 of the same character are presented, and can also be modified by functions such as 'remove_rows' to replace these matches that are made with new symbols.

There are 4 other global variables pertaining to user inputs.

Two of these are defined as a set number by the user at the start of the game:

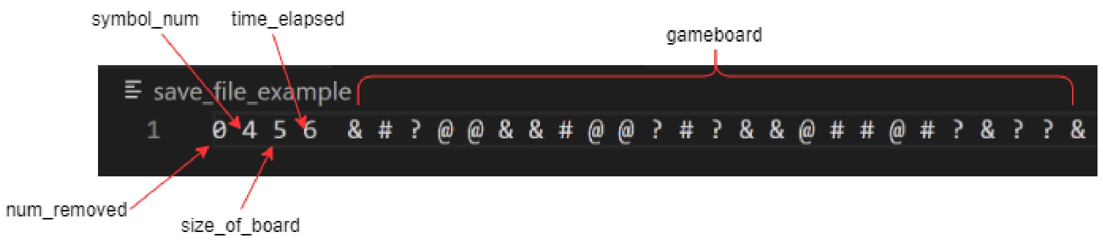
- **The size of the board:** *size_of_board* (integer)
- **The number of different symbols:** *symbol_num* (integer)

The other two variables change throughout the game depending on the user's interactions, and ultimately contribute to the final score achieved by the user:

- **The number of symbols removed:** *num_removed* (integer)
- **The time taken from start to finish:** *time_elapsed* (integer)

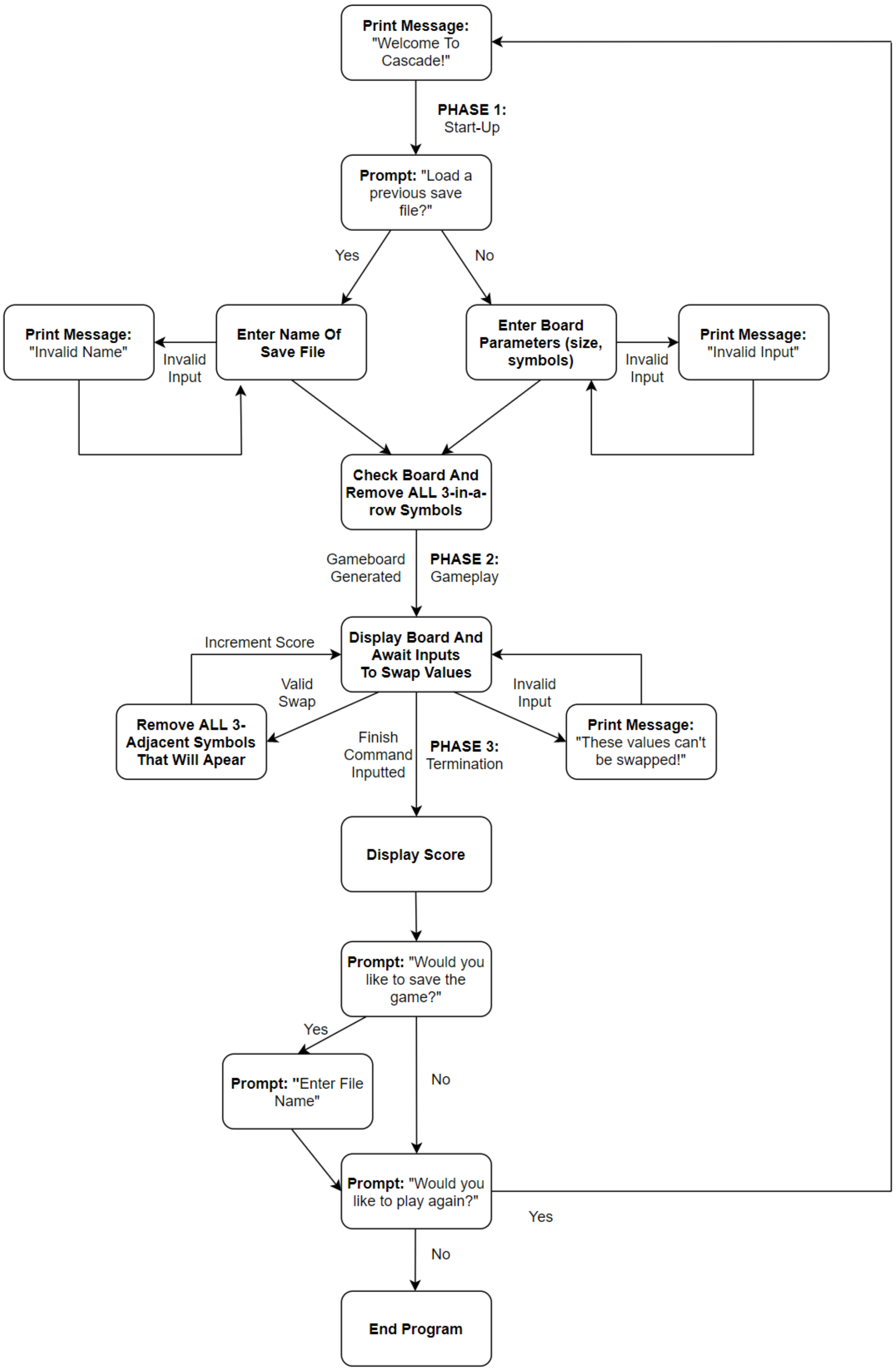
These 4 global variables as well the gameboard are also stored within the save file for each game.

The size of the board, number of removed symbols, number of different symbols, and the time elapsed are all saved to a text file. Next to it, the symbols in the board are also saved in a singular row which can be read back in by the program when the user wants to load that specific file in.



```
0 4 5 6  & # ? @ @ & & # @ @ ? # ? & & @ # # @ # ? & ? ? &
```

3. High Level Design Flow Chart



Print Message: "Welcome To Cascade!"
PHASE 1: Start-Up
Prompt: "Load a previous save file?"
Yes
No
Print Message: "Invalid Name"
Invalid Input
Enter Name Of Save File
Enter Board Parameters (size, symbols)
Invalid Input
Print Message: "Invalid Input"
Check Board And Remove ALL 3-in-a-row Symbols
Gameboard Generated
PHASE 2: Gameplay
Increment Score
Display Board And Await Inputs To Swap Values
Valid Swap
Invalid Input
Remove ALL 3-Adjacent Symbols That Will Apear
Finish Command Inputted
PHASE 3: Termination
Print Message: "These values can't be swapped!"
Display Score
Prompt: "Would you like to save the game?"
Yes
Prompt: "Enter File Name"
No
Prompt: "Would you like to play again?"
Yes
No
End Program

4. Functions

reset_color	
Description	This is a function to reset the color of colored text within the game, using a simple print statement that contains the "ANSI" color code for resetting a color ("[0m"].
Data Types	The data type for this function is void.

get_parameters	
Description	This is the initial function that is called at the start of the game, which presents a welcome message, as well as prompts for the input of two int variables: 'size_of_board' and 'symbol_num'.
Input Parameters & How They're Used	The integer size_of_board is entered by the user and is used later in the program to determine the number of columns and rows the gameboard of the program will have. The integer symbol_num denotes the number of symbols that the player would like to use within the board. There are no local declarations in this function.
Data Types	The return type of this function is 'void', as all relevant parameters (size_of_board and symbol_num – both of which are integers) are global variables, meaning they can be accessed inside any of the functions defined in the program. This holds true for all functions using global variables with the data type void in this program.
Notes	This function also calls upon the 'resetColor' function, to set the color of the words in the terminal back to white, in contrast to the colored prompts. There is also a check in place, such that if the size_of_board or 'symbol_num' variables are not in the specified valid range, the program terminates with a warning message to enter valid parameters, as specified by prompts.

allocate_memory	
Description	This function is used dynamically allocate memory through the standard library function 'malloc' which reserves a black of memory of a specific number of bytes for the gameboard.
Input Parameters & How They're Used	The global variable '**gameboard' is a 2D array of data type 'char'. It uses a pointer to a pointer char (**char – same type as gameboard), and the 'size_of_board' – a global variable – is multiplied by the size of the row pointers for the board. Then each row of the board (gameboard[i]) is allocated for, through an iteration through the board.
Data Types	The data type for this function is 'void' as the gameboard is a global variable
Notes	To ensure that allocation of memory was successful, an if statement ensures that if the gameboard's value is NULL, a warning is issued.

row_seq, col_seq	
Description	This function is used to detect any row or column sequences where there are 3 of the same character present, using the standard bool library for Boolean expressions.
Input Parameters & How They're Used	The local variable integers 'row' and 'col' are defined. In the row_seq command, 3 of the same symbols can only appear when the column size is greater than 2, but less than size of the board (ensuring that the user is indexing within an appropriate range). If the gameboard's value for 'col' at a specified point in a board is equal to col – 1 and col – 2, then this means a match of 3 is found, at which point the bool returns a value of 'true'. This same logic is applied to the col_seq command, but for this instance, the value for rows must be greater than two. Row – 1 (variable in the row directly above) and Row - 2 (variable 2 rows directly above) must also equal the original row for this command to return 'true'.
Data Types	This function is a bool, and has a return type of either true (when the parameters mentioned above are met) or false.

create_board	
Description	This function is used to fill the 2D gameboard array with randomly generated symbols.
Input Parameters & How They're Used	A local variable, known as rand_index (an integer) is created using the rand() function for randomly generated values, but limited to the specific number of symbols requested by the user, as denoted by the global variable symbol_num. Each row and column value generated through the iteration is populated by a randomly generated char, but at each iteration, a 'while' function is used, which calls the row_seq and col_seq functions mentioned above and passes in the int 'i' correspond to rows and 'j' corresponding to columns, for that specific iteration. The row_seq and col_seq booleans then detect any instances where 3 of the same character in a row are generated, by returning a value of 'true'. If the scenario exists where either row_seq or col_seq return true, the characters at i and j for that iteration are replaced by new symbols. This ultimately modifies the gameboard (global variable) until there is no longer any matches of 3 left.
Data Types	The data type for this function is void.

move_down	
Description	This function is used to 'cascade' the symbols in the board to fill where spots in the blank space were formed.
Input Parameters & How They're Used	Once the symbols have been cascaded, the remaining blank spaces which will now be at the top of the board are replaced by new symbols. Then, the row_seq and col_seq functions are called to remove any further matches and increment the global variable num_removed. Once there are no longer any row or column sequences, the newly modified gameboard is printed again.
Data Types	The data type for this function is void.

print_board	
Description	This function prints both the numbers and randomly generated characters for the gameboard.
Input Parameters & How They're Used	Iterations are first done across the rows. An if statement is used, so that in the case where the size of the board is greater than or equal to 10, the increase in place values is accounted for in how it offsets the board; for iterations of i less than 9, an extra space is added, but when i > 9, one less space is added as numbers printed are now in double digits. This process is repeated in another for loop for the numbers going down vertically. The gameboard that was previously generated in the create_board function is also printed.
Data Types	The data type for this function is void.

check_range	
Description	This function is used to check if the range of coordinates inputted by the user is within the confines of the gameboard
Input Parameters & How They're Used	The two integer coordinates (x1,y1) and (x2,y2) are passed into the function, and if both coordinate pairs are greater than the number 1 and less than the size of the board, they are within the board's range and the function returns true. Otherwise, this function returns false, meaning the swap is deemed to be invalid.
Data Types	The data type for this function is a bool. It returns true if the both coordinates are in the confines of the board.

check_adjacent	
Description	This function is used to check if the two coordinates inputted are next to each other.
Input Parameters & How They're Used	The two integer coordinates (x1,y1) and (x2,y2) are passed into the function as input parameters for this function from the main function. The absolute value of the two y coordinates or two x coordinates is computed – if this result is equal to one for either, the distance between the two symbols is one. Therefore, they are adjacent, and the swap is valid.
Data Types	The data type for this function is a bool. It returns true if coordinate swaps are adjacent to each other for the x or y values in the coordinates

remove_rows, remove_cols, remove_crosses	
Description	These functions are used to remove horizontal and vertical matches greater than 3 for the game.
Input Parameters & How They're Used	For each each sequence found within the board, it is replaced with a blank space. Any extra sequences left or right for the remove_rows function, or up or down for the remove_cols function is also accounted for. Each time a match is found, the num_removed global variable is incremented for each symbol in a match. For the remove_crosses function to be triggered, a match must be created where there is both a row and column created when a symbol is swapped, at which point both of these matches are removed.
Data Types	This function's data type is void.

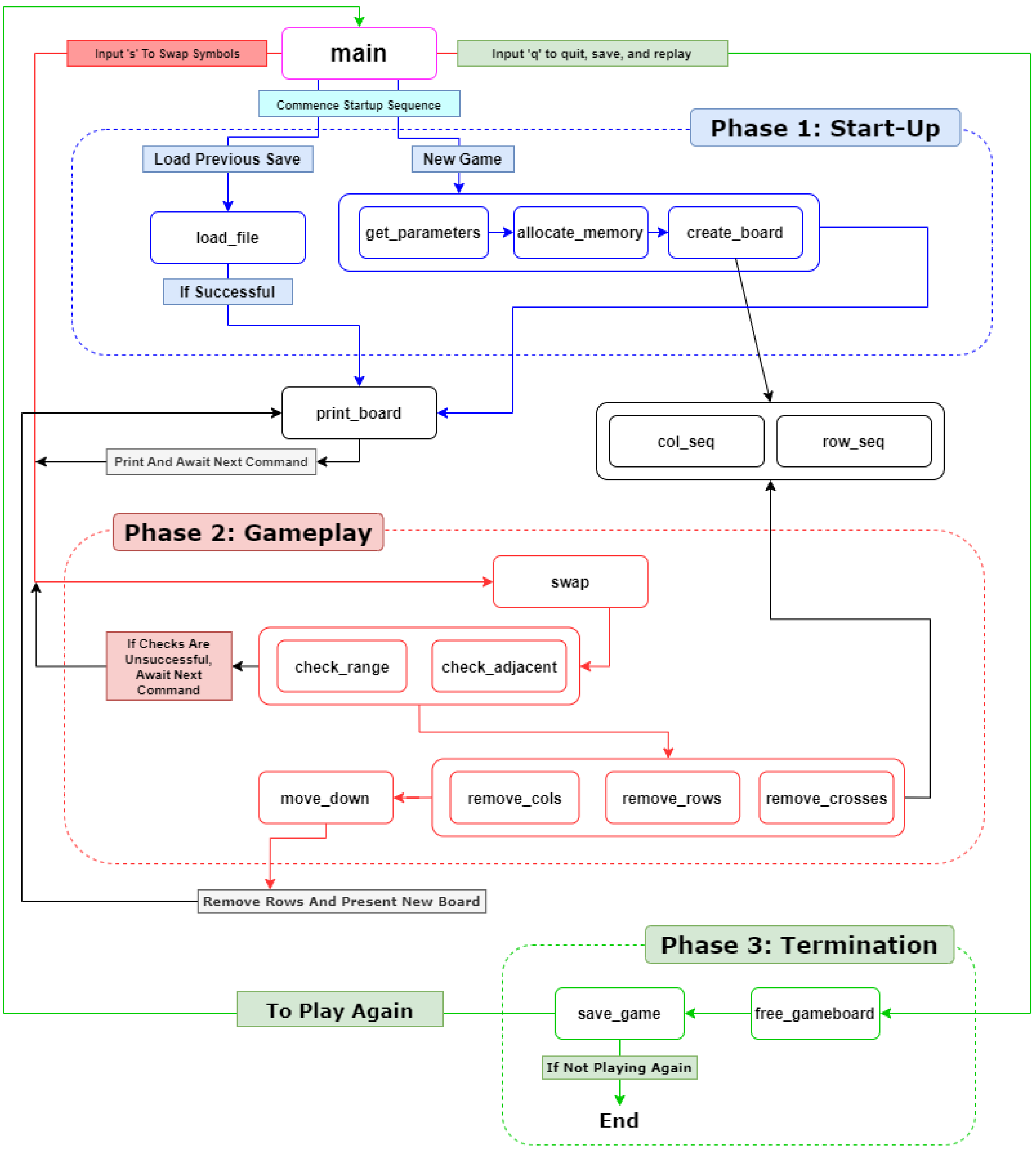
swap	
Description	This is the main function for swapping two symbols.
Input Parameters & How They're Used	Coordinates x1, y1, x2, and y2 are passed in from the main function. These values are then passed into both the check_range and check_adjacent function. Both functions must be true for the swap to be valid. Furthermore, a match of 3 must be created for the symbols to successfully swap. A temporary local integer is created, and 1 is subtracted from each coordinate variable, as the 2d array gameboard starts from position 0, but users will input coordinates starting from 1. Then a local integer known as originalnum is equated to the current number of removed symbols (num_removed). Then the functions to remove matches and cascade symbols is called. If the originalnum variable defined is equal to the current num_removed value, then this means that no matches were found, so the two symbols are swapped back and an error message is sent. However, if the num_removed integer has increased from its original value, that means matches have been created, so board is printed again by calling the print_board function and the next command is awaited.
Data Types	The data type for this function is void.

save_board	
Description	This function is used to save the current game for the board.
Input Parameters & How They're Used	A string named filename is created with a maximum length of 30 characters that a user can name. If the name of the file is invalid, the user will be re-prompted. A file pointer named save_input is used to open the file with the chosen file name to be written into. Then, using the fprintf command, the global variables num_removed, symbol_num, size_of_board, and time_elapsed are written into the game file. The gameboard symbols are also written in, and the file is closed.
Data Types	The data type for this function is void.

load_file	
Description	This function is used to load a previous save file for the board.
Input Parameters & How They're Used	A string named savename is created with a maximum length of 30 characters that a user can name. If the name of the file is invalid, the user will be re-prompted. A file pointer named readfrom is used to open the file with the chosen file name to be read from. Using the command fscanf, the global variables num_removed, symbol_num, size_of_board, and time_elapsed are read into the current game. The gameboard symbols are also written in, ensuring that adequate memory is allocated for the 2d array, and the file is closed.
Data Types	The data type for this function is void.

free_gameboard	
Description	This command is used to free dynamically allocated memory for gameboard array's rows and columns.
Data Types	The data type for this function is void.

5. Function Structure Chart



6. Algorithms

The main algorithm present in this program is in the cascading process when a match is detected in the board. It is as follows:

Logic for removing rows/columns (remove_cols, remove_rows functions)

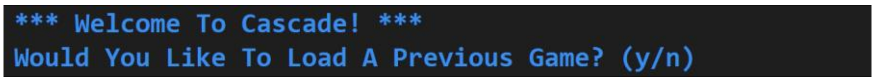
```
for gameboard in remove row or column function
for i, where i is less than size of board, increment i
    for j, where j is less than size of board, increment j
        if there is a row or column sequence of 3,
        replace that sequence with a blank space
        and increment number of removed by 3


        then, index left/right for rows, or
        up/down for columns to ensure extra
        symbols are accounted for if sequence is
        >3. If there is an extra symbol, increment
        num_removed by 1 for each extra symbol and replace
        that symbol with a blank space ' '
```

Logic for cascading symbols after matches (move_down functions)

```
for x, where x is less than size of board, increment x
    for i, where i is less than size of board - 1, increment i
        for j, where j is less than size of board, increment j
            if the spot below the current gameboard space is empty
            move that empty spot to the current gameboard space.
            continue until the top of gameboard is reached
check through the gameboard again for blank spaces using for loops
    if there is a blank space, replace that space with a new symbol
    while there are still row or column sequences being create,
    increment num_removed for each new match and replace symbols
    until no more matches are created
```

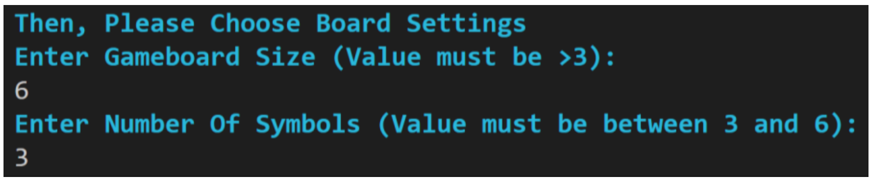
7. User Instructions

1. Copy the game program file into a C language compiler
2. Run the program and await prompt to enter gameboard size



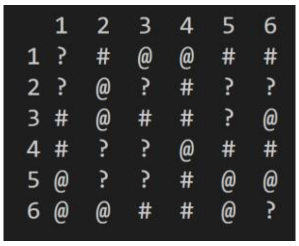
```
*** Welcome To Cascade! ***
Would You Like To Load A Previous Game? (y/n)
```

3. If first time playing, respond ‘n’ for no to the query about loading a previous game
4. Enter valid gameboard size and number of symbols for gameplay

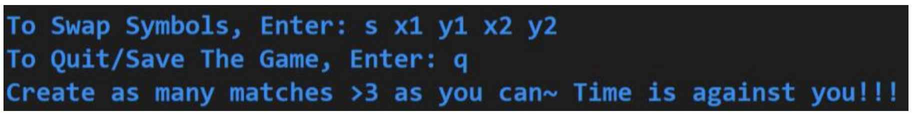


```
Then, Please Choose Board Settings
Enter Gameboard Size (Value must be >3):
6
Enter Number Of Symbols (Value must be between 3 and 6):
3
```

5. Once gameboard is displayed, press ‘s’ and enter two coordinates to swap symbols, under the following conditions
   a. Symbols must be adjacent to each other
   b. Coordinates entered must be within the confines of the gameboard’s size
   c. Any swap must create a match of 3 or more characters

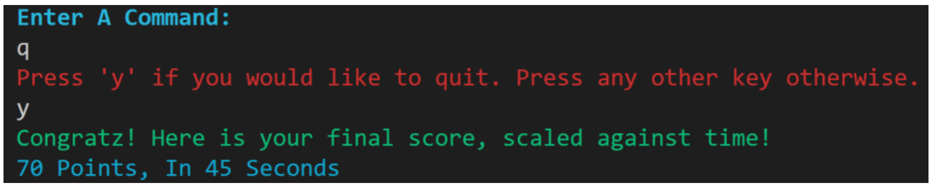


```
   1  2  3  4  5  6
1 ?  #  @  @  #  #
2 ?  @  ?  #  ?  ?
3 #  @  #  #  ?  @
4 #  ?  ?  @  #  #
5 @  ?  ?  #  @  @
6 @  @  #  #  @  ?
```



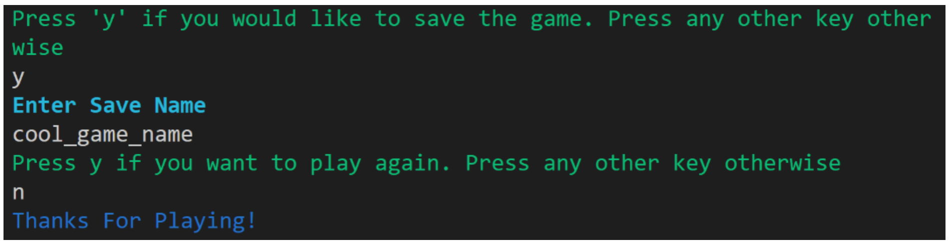
```
To Swap Symbols, Enter: s x1 y1 x2 y2
To Quit/Save The Game, Enter: q
Create as many matches >3 as you can~ Time is against you!!!
```

6. Continue swapping numbers as fast as possible, as the current score will be scaled against time at the end.
7. When wanting to end the game, enter ‘q’ and follow prompts to be presented with final score



```
Enter A Command:
q
Press 'y' if you would like to quit. Press any other key otherwise.
y
Congratz! Here is your final score, scaled against time!
70 Points, In 45 Seconds
```

8. When prompted to save the game, if answer is yes, enter a name for the game file that is less than 30 characters long – this file will be saved in the current directory with the current time elapsed, gameboard, and parameters
9. If wanting to play again, respond ‘y’ to the prompt – any previous game files can now be replayed with or a new game can be created
10. Otherwise, the program will close



```
Press 'y' if you would like to save the game. Press any other key other
wise
y
Enter Save Name
cool_game_name
Press y if you want to play again. Press any other key otherwise
n
Thanks For Playing!
```

8. Requirement Acceptance Tests

Software Requirements #	Test	Implemented (Full / Partial / None)	Test Results (Pass / Fail)	Comments
1	Previous game files can be loaded when given the correct file name. Display warning message if a file does not exist or if a file name is invalid	**Full**	**Pass**	
2	Previous game files read in the board, number of symbols, time elapsed in the last game, and size of board correctly.	**Full**	**Pass**	
3	If board size is less than 3, display appropriate warning message and re-prompt	**Full**	**Pass**	
4	If symbol number is less than 2 or greater than 6, appropriate warning message displayed and user is re-prompted	**Full**	**Pass**	
5	Memory is allocated for the board successfully. If not, a warning is given	**Full**	**Pass**	
6	Gameboard will not display any matches of symbols when printed	**Full**	**Pass**	
7	Swap command can only be successful if 4 coordinates are entered	**Full**	**Pass**	
8	Symbols can only be swapped if match is created, symbols are adjacent to each other, and are in an appropropriate range	**Full**	**Pass**	
9	Any matches of 3 disappear and current score is incremented correctly	**Full**	**Pass**	"L" shaped matches are accounted for correctly, where there is a horizontal *and* vertical match

10	Symbols cascade down correctly once a gap is created from a match	**Full**	**Pass**	
11	Symbols can only be swapped if match is created, symbols are adjacent to each other, and are in an appropropriate range	**Full**	**Pass**	
13	Symbols cascade down correctly once a gap is created from a match	**Full**	**Pass**	
14	The final score is correctly calculated, scaled against time	**Full**	**Pass**	Any time from previous game sessions if that file has been loaded from a save has been accounted for
15	When saving, the user can successfully enter a file name. If file name is greater than 30 characters, a warning message is issued.	**Full**	**Pass**	
16	All memory allocated dynamically has been freed successfully.	**Full**	**Pass**	Program has been run through Valgrind to check for memory leaks multiple times.
17	No warnings issued when compiling	**Full**	**Pass**	

9. Detailed Software Testing

#	Test	Expected Result	Actual Result
1.0	**Start Up**		
1.10	**Loading Saves**		
1.11	Input other than ‘n’ or ‘y’ entered when asked if user wants to load a previous game	Warn and re-prompt user	
1.12	Non-existent file name inputted		**All as expected**
1.13	File name greater than 30 characters inputted		
1.14	Save file is read back into a program	Gameboard and all variables working correctly	
1.20	**Gameboard**		
1.21	Gameboard size as value less than 3	Warn and re-prompt user	**All as expected**
1.22	Gameboard size is 1000	All game functions and memory allocation are working as normal	
1.22	Symbol number as less than 2, greater than 6, or not a number	Warn and re-prompt user	
1.23	Gameboard dynamically allocates memory properly	Warning message not given; Valgrind tests display no memory leaks	
2.00	**Gameplay**		
2.10	**Swapping Symbols**		
2.11	Two symbols not adjacent to each other	Invalid swap. Warn and re-prompt user	**All as expected**
2.12	One or more coordinates beyond board domain		
2.13	Swap that does not result in a match >3		
2.14	Swap command without all coordinates, or non-integer coordinates		
2.15	Input successful swap that results in a match of 3	Results in score incremented and cascaded board printed without any match-3s	
2.16	Input successful swap that results in a match greater than 3	Results in score incremented correctly, accounting for extra symbols, and cascaded board printed without any match-3s	**All as expected**
2.17	Swap results in an ‘L’ shape horizontal AND vertical match	Results in score incremented correctly, accounting for correct number of extra symbols being removed from	

		both the horizontal and vertical matches. Cascaded board also printed without any match-3s	
2.18	A successful swap is completed	Board is reprinted with no matches ≥3	
2.20	**Entering Termination Phase**		
2.21	Pressing ‘q’	Prompts user to confirm going into termination phase	**All as expected**
2.22	Pressing any key other than ‘q’ or ‘s’	Warn user and re-prompt for a new command	
3.00	**Termination**		
3.10	**Score Calculation**		
3.11	Exiting from a brand-new game (not a loaded save)	Score and time elapsed displayed correctly	**All as expected**
3.12	Exiting from a game loaded in that has been saved once before	Score and time elapsed displayed correctly, accounting for time elapsed in previous session	
3.13	Exiting from a game loaded in that has been saved three times before	Score and time elapsed displayed correctly, accounting for and adding all previous sessions’ time elapsed correctly	
3.20	**Game Saving**		
3.21	Gameboard and all relevant variables successfully saved	No warnings issued	**All as expected**
3.22	User enters file name less than 30 characters	No warnings issued and game saved successfully	
3.23	User enters file name >30 characters	Warning issue and user is re-prompted	
3.30	**Replaying**		
3.31	Check dynamically allocated memory has been freed	Valgrind reports no memory being leaks	**All as expected**
3.32	User answers yes (‘y’) when asked if they want to replay	User can play again successfully	
3.33	User inputs any other integer than ‘y’ when asked if they want to replay	Program exits with a message saying, “Thanks For Playing!”	