COMP 1011 Project Documentation

Name: SHOAIB Muhammad  
Student ID: 18079999D  
Date: 18th April, 2019

**AIM:**To develop a slightly user-unfriendly Candy Crush type game.  
To the best of my knowledge, I have tried to make the game as playable as it can be.

**INPUTS:**

* Grid size (*N* x *M*) at the beginning.
* Index positions to swap (when asked).

**OUTPUTS:**

* Score update (if score is incremented).
* Grid is printed with corresponding indexes from 0 to *N* and from 0 to *M*.
* Certain messages/game status updates provided by the program.

**FUNCTIONS:**

I have used a total of 6 functions/modules and a main program to call them.

The function signatures are as follows:

**bool** **check**(**int** arr[NUM][NUM], **int** r, **int** c, **int** \* score);

**void** **PrintArray**(**int** arr[NUM][NUM], **int** r, **int** c);

**void** **refill**(**int** arr[NUM][NUM], **int** r, **int** c);

**void** **swap**(**int** arr[NUM][NUM], **int** r1, **int** c1, **int** r2, **int** c2);

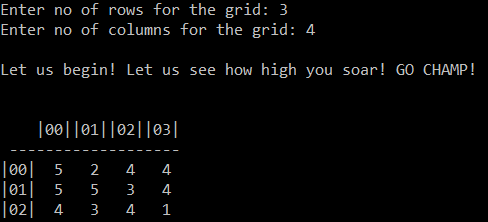
**bool** **swapFeasible**(**int** arr[NUM][NUM], **int** r, **int** c);

**bool** **provideHint**(**int** arr[NUM][NUM], **int** r, **int** c);

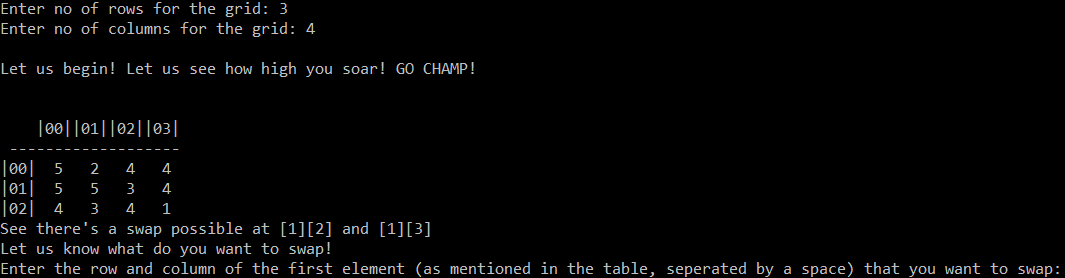
* The **check** function checks consecutive similar candies in the array and replaces the matches with -1, and it also helps to update the score of player.
* The **PrintArray** function is used to display/print the contents of the array at the instant it is called.
* The **refill** function is used to refill the array with values described in the rules (bring down elements if required).
* The **swap** function helps to swap the index positions told by the user to make the game playable.
* The **swapFeasible** and **provideHint** are used to provide the user with a hint, and if no more swaps are possible, then the main program uses them to tell the user that no more swaps are possible and the user should enter any key to regenerate the grid.

**HOW DOES THE GAME WORK?**

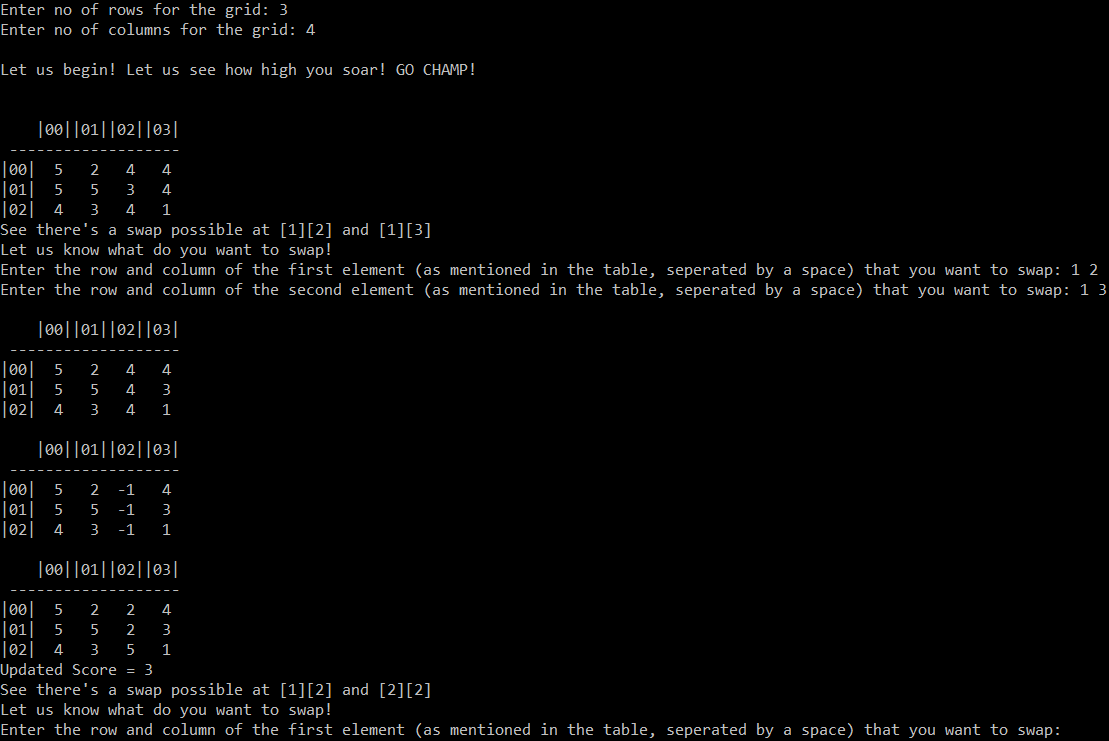
* The program generates a grid comprising of *r* rows and *c* columns, as entered by the user.
* The program then randomly initializes the N x M candy field with 6 candy types and prints it out. See this example below:



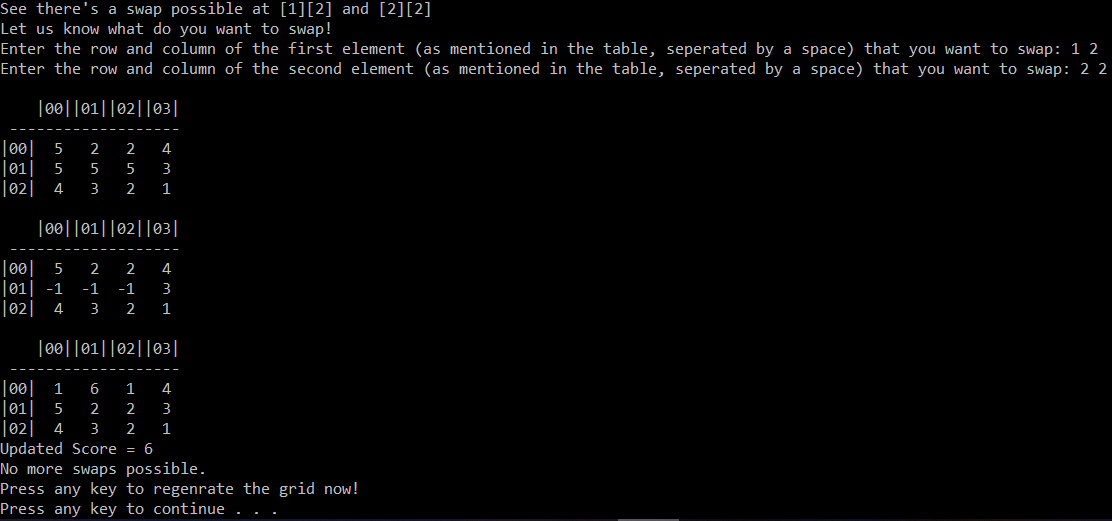
* The program then checks the existence of any valid (horizontal and vertical) candy chain and prints the corresponding “-1” labelled candy array.
* It clears the candy chain(s), if they exist, and refill the candy fields accordingly.
* The program then checks if there are any possible swaps and provides the user with a hint to recognize the swap. The example is continued:



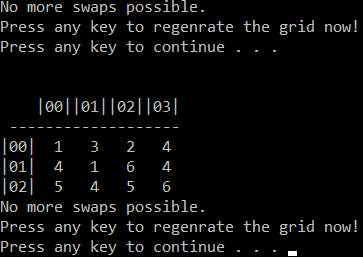
* Now, the user must enter the indexes of the elements he needs to swap. They could either be from the hint, or if possible, any other indexes that the user wants to swap. The example follows:



* The program displays the replaced values and updated scores.
* The program then goes on to tell the user about any more swaps possible. The user continues to play the game. Example follows:



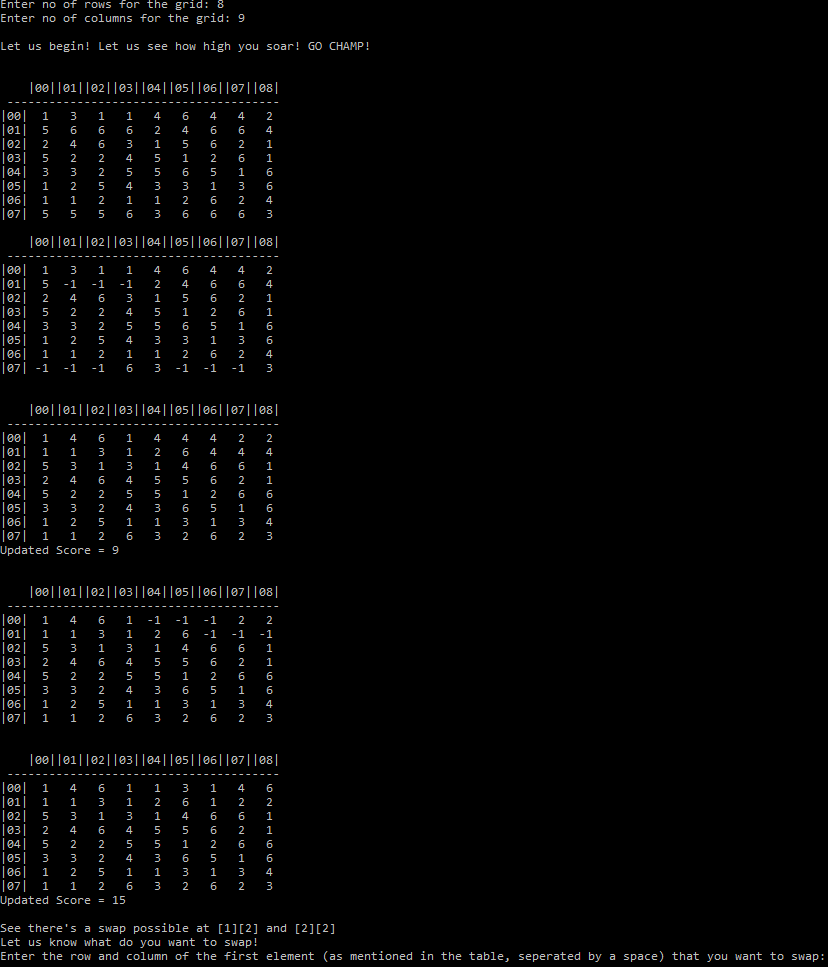
* When there are no possible swaps in the grid (like in the example shown above), the program asks user to press any key to regenerate the grid.
* The grid is then regenerated. Example follows:



* Since the regenerated grid doesn’t contain a match again, the user is asked again to press a key to regenerate a grid.
* The program goes on and keeps regenerating the grid.

**PLEASE TURN TO THE NEXT PAGE**

* Let us look at another example containing a large grid. Example follows:



* The user keeps swapping and the game keeps running 😊.

**PROS OF THE PROGRAM:**

* It is not that user unfriendly.
* Using a modular design, breaking the problem into smaller chunks makes it easier to code, understand the program and re-use the modules/functions again and again.
* Every time, the grid generated is totally random, so it is a very fair game.

I have tried my best to implement a playable game.  
I hope that you understand my code and like my game.  
Thank you!