JANUARY 2024

JANUARY 2024

TABLE OF CONTENTS

FIGURES

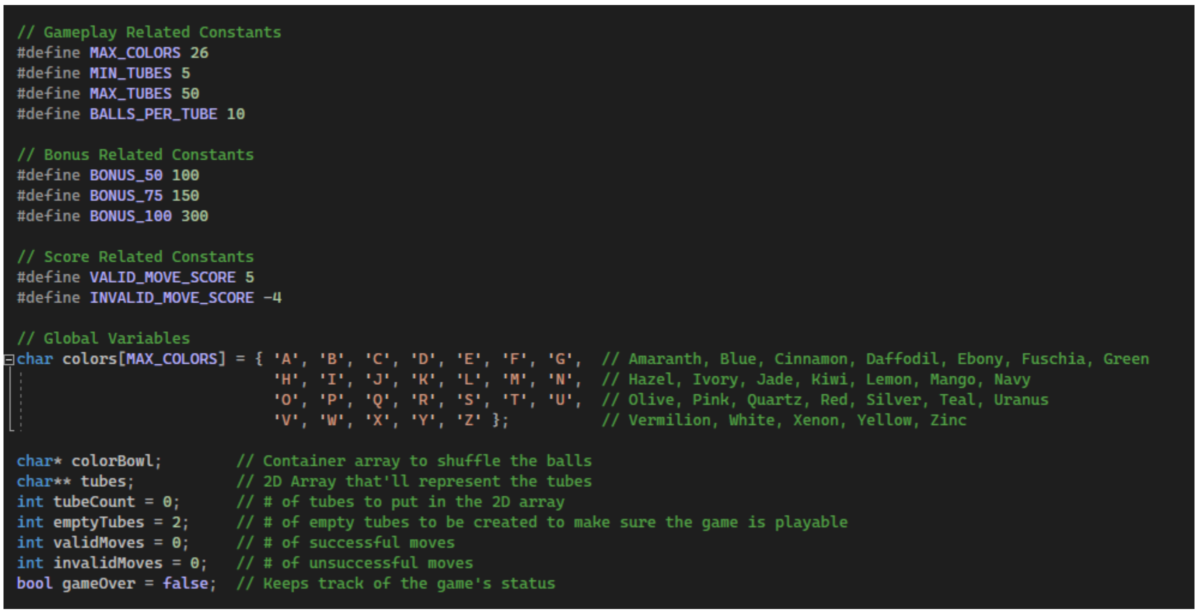
TABLES

1. DEFINITION OF THE PROBLEM

This program was developed to create a strategy game where the player sorts out the colored balls in tubes. Program must be scalable depending on how many tubes the player wishes to play with. The rules of the game were defined as below:

1. Minimum number of tubes is 5. Player can't play with less than 5 tubes.
2. Since the program is a console-based application, the colors of the balls inside the tubes will be represented with letters.
3. Tubes can't contain more than a specific number of balls.
4. The balls can't be moved to a tube that is full.
5. Only the ball on top can be moved.
6. The balls can't be moved if there is a different colored ball underneath.
7. There should be a counter that keeps the number of moves made by the player.
8. The game ends when each tube only contains balls which have the same color.

2. HOW THE PROGRAM HANDLES THE PROBLEM



```c
// Gameplay Related Constants
#define MAX_COLORS 26
#define MIN_TUBES 5
#define MAX_TUBES 50
#define BALLS_PER_TUBE 10

// Bonus Related Constants
#define BONUS_50 100
#define BONUS_75 150
#define BONUS_100 300

// Score Related Constants
#define VALID_MOVE_SCORE 5
#define INVALID_MOVE_SCORE -4

// Global Variables
char colors[MAX_COLORS] = { 'A', 'B', 'C', 'D', 'E', 'F', 'G',  // Amaranth, Blue, Cinnamon, Daffodil, Ebony, Fuschia, Green
                            'H', 'I', 'J', 'K', 'L', 'M', 'N',  // Hazel, Ivory, Jade, Kiwi, Lemon, Mango, Navy
                            'O', 'P', 'Q', 'R', 'S', 'T', 'U',  // Olive, Pink, Quartz, Red, Silver, Teal, Uranus
                            'V', 'W', 'X', 'Y', 'Z' };          // Vermilion, White, Xenon, Yellow, Zinc

char* colorBowl;        // Container array to shuffle the balls
char** tubes;           // 2D Array that'll represent the tubes
int tubeCount = 0;      // # of tubes to put in the 2D array
int emptyTubes = 2;     // # of empty tubes to be created to make sure the game is playable
int validMoves = 0;     // # of successful moves
int invalidMoves = 0;   // # of unsuccessful moves
bool gameOver = false;  // Keeps track of the game's status
```

Figure 1 - Definitions of the constants and variables inside the source code.

The program could be written using a stack data structure, but since the C programming language doesn't have a built-in stack structure, char arrays were used.

An array of chars called Colors, which has all the letters in English alphabet was defined to represent the colors of the balls. Hence, the game can be played with 26 different colors out of the box.

Constants MIN_TUBES 5, MAX_TUBES 50, and BALLS_PER_TUBE 10, were defined globally. And, to make the game a bit more fun, two additional constants called VALID_MOVE_SCORE 5, and INVALID_MOVE_SCORE -4, were defined, so at the end of the game program can show the user a score based on the accuracy of the moves.

The logic followed to actualize the program as follows:

1. Depending on the number of tubes specified by the user, the game creates 10 same-colored balls for each tube starting from the first color and pushes them into a dynamically expanding array called color bowl. The reasoning behind this approach will be clarified in the following paragraph.
2. If the number of the tubes exceeds the number of available colors, the program will go back in the color array and start over from the first color.
3. Then, using the current time as the randomization seed, the game randomly distributes the balls into tubes.
4. Since the game starts with each tube filled, to make sure the player will be able to make a move at least 2 empty additional tubes are created. At the beginning of the game, program asks the player how many empty tubes to create. 2 empty tubes are generally enough to complete the game when dealing with small numbers of tubes. However, to prevent the player from exploiting the ability to determine the number of empty tubes, program restricts the creation of maximum number of empty tubes as the double of the generated tubes (taking the higher numbers of tubes in consideration, because small numbers of empty tubes such as 2 won't suffice when dealing with more tubes).

The reason why the balls were put together in a different array first is to make sure the game is playable. Because I couldn't figure out a way to generate the balls randomly in equal amounts. If the number of each color doesn't match, the player won't be able to play the game as it may result in insufficient number of tubes to sort out every color.

To prevent that from happening, I come up with a solution inspired by lottery. Just like they do it in lottery, I put the specific number of balls in a bowl, so I can keep track of the numbers of the colors generated, and then I can distribute them randomly into tubes while being sure that the game is playable.

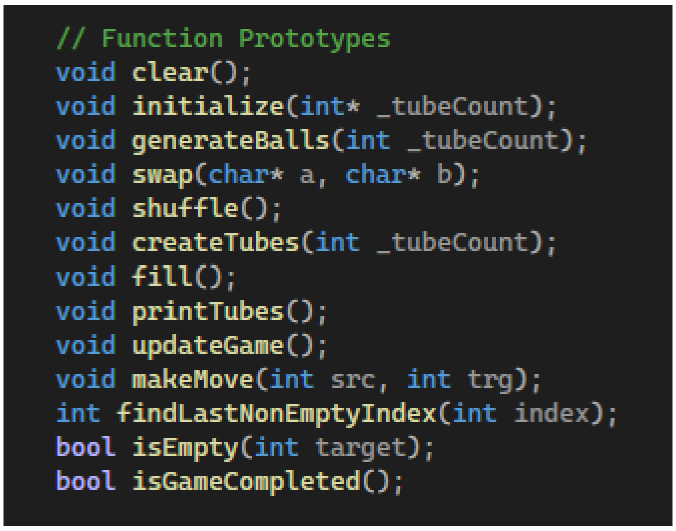
After setting up the game following the logic explained above, the program asks the user to enter moves. After each move, the program controls whether the move was valid or not, and updates two counters called valid moves and invalid moves.

For each valid move, player gets 5 points, and for each invalid move 4 points are taken back from the player. At the end of the game, program calculates the accuracy of the player's moves and gives bonus points.

- 50%+ accuracy yields +100 points,
- 75%+ accuracy yields +150 points,
- 100% accuracy yield 300 points.

The bonus points are not cumulative. At the end of the game player is informed about the accuracy of the moves and how many bonus points were granted if granted any.

3. ALGORITHM FLOW DIAGRAM



```
// Function Prototypes
void clear();
void initialize(int* _tubeCount);
void generateBalls(int _tubeCount);
void swap(char* a, char* b);
void shuffle();
void createTubes(int _tubeCount);
void fill();
void printTubes();
void updateGame();
void makeMove(int src, int trg);
int findLastNonEmptyIndex(int index);
bool isEmpty(int target);
bool isGameCompleted();
```

Figure 2 - Prototypes of the functions used throughout the program.

As seen in the Figure 2, program utilizes a variety of functions. To keep the code cleaner, function prototypes were created above the main function, and they were implemented under the main function. Below is the algorithm flow diagram:

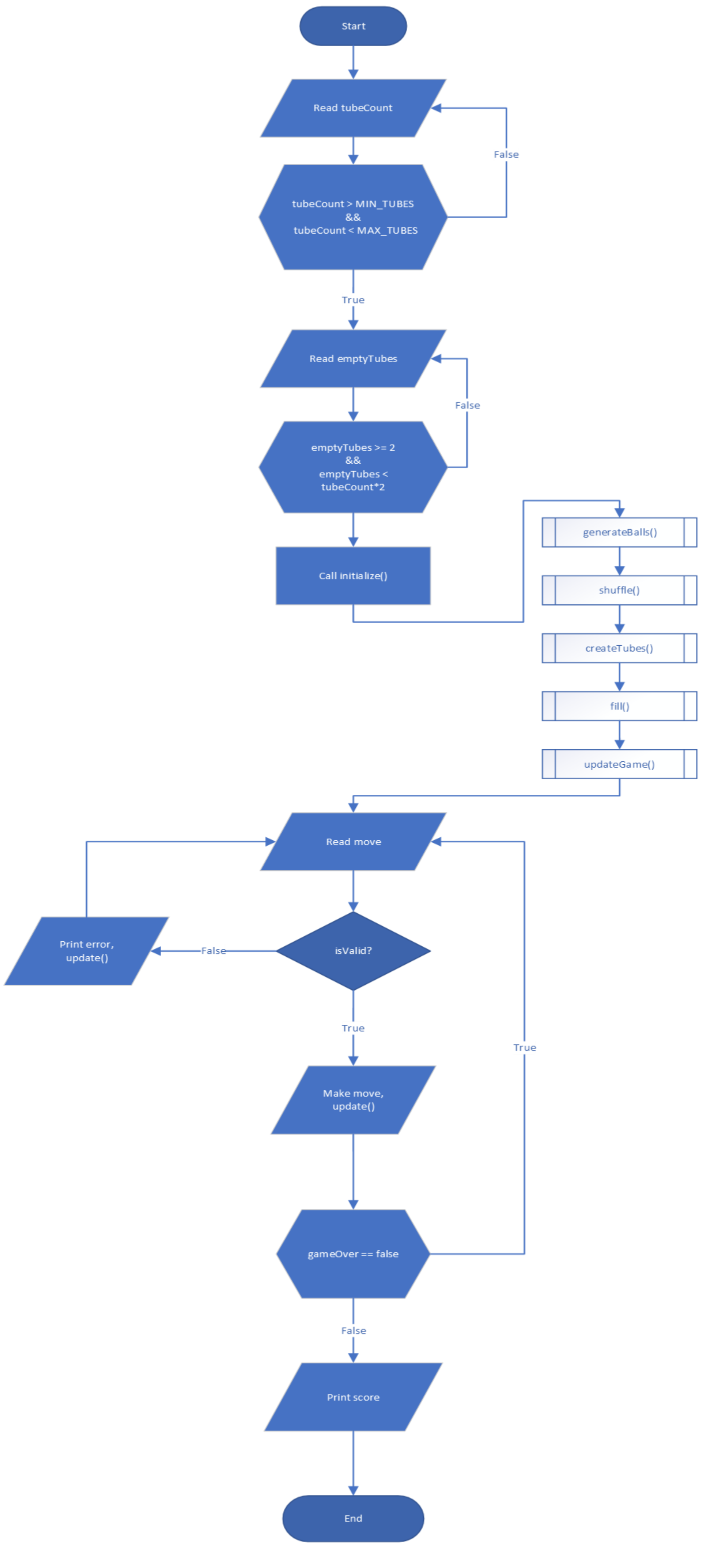
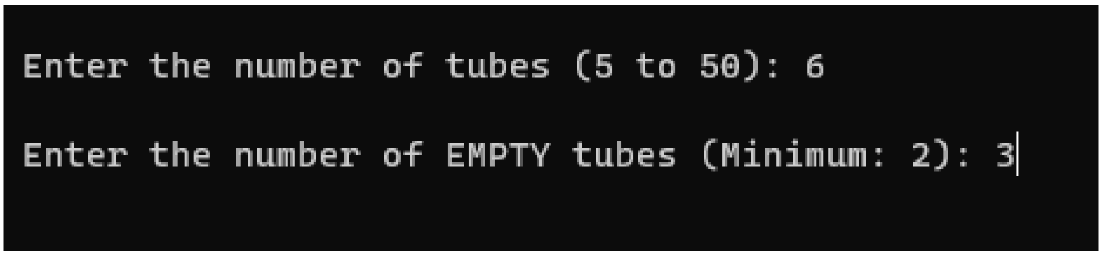


Figure 3 - Algorithm flow diagram.

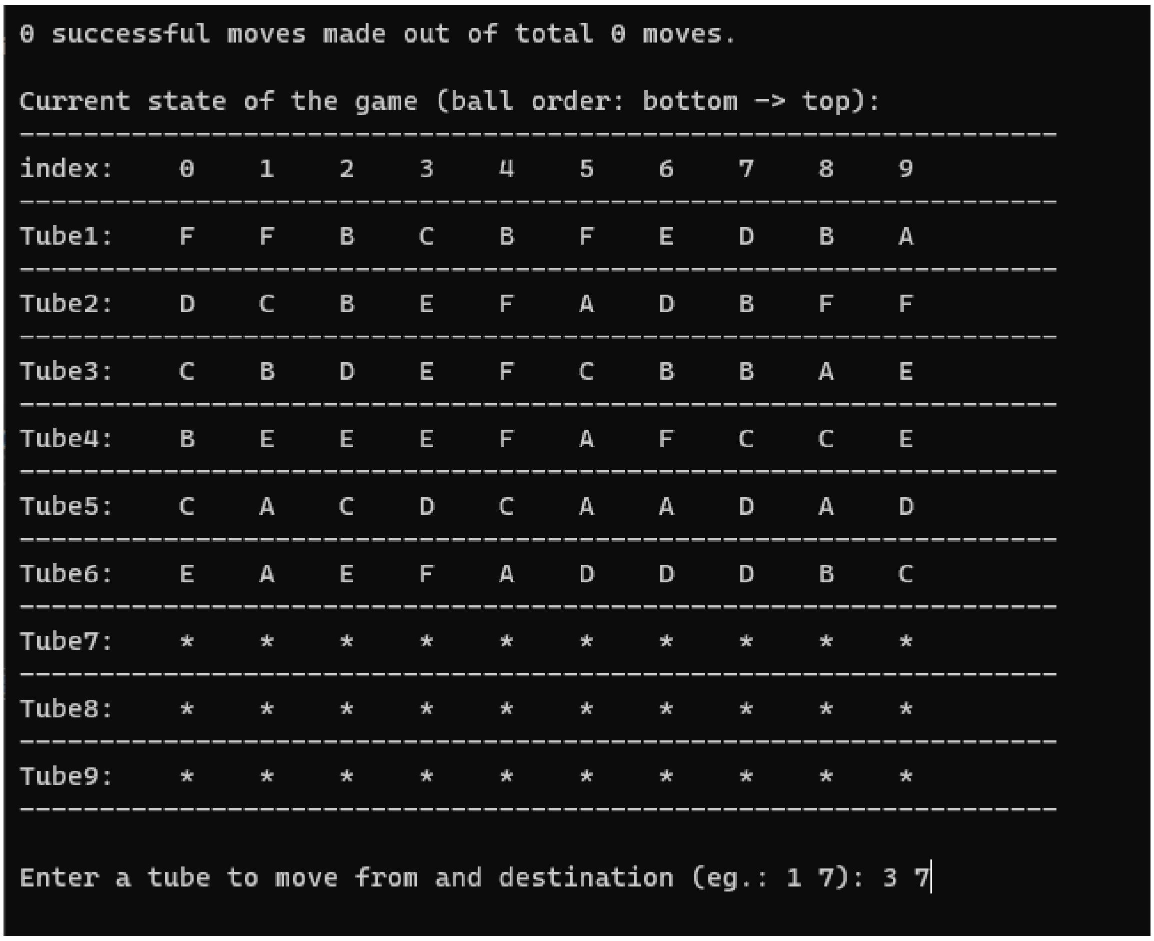
4. IN-GAME SCREENSHOTS



```
Enter the number of tubes (5 to 50): 6

Enter the number of EMPTY tubes (Minimum: 2): 3
```

Figure 4 - Initial prompts to player once the program launched.

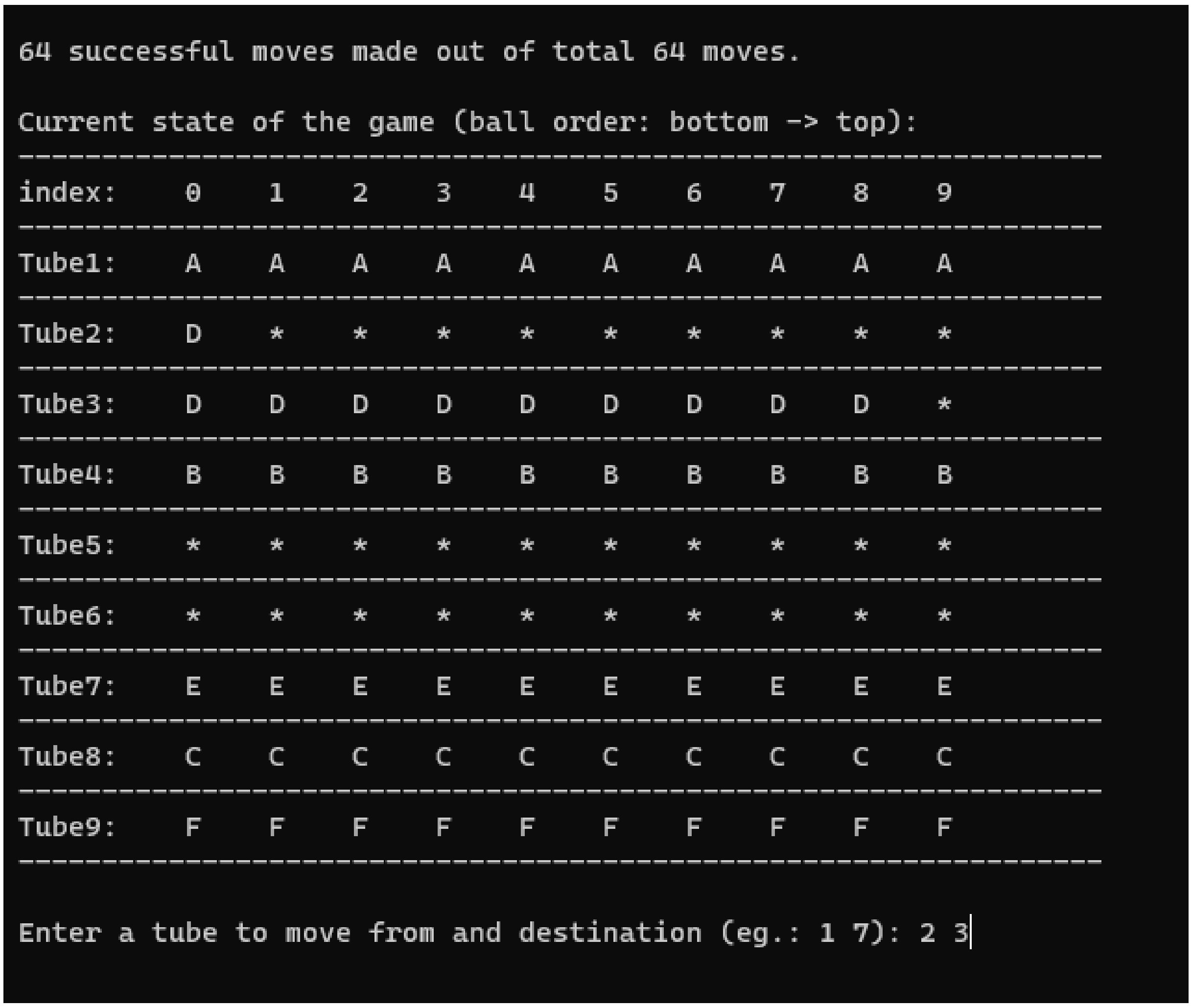


```
0 successful moves made out of total 0 moves.

Current state of the game (ball order: bottom -> top):
-----------------------------------------------------------------
index:    0    1    2    3    4    5    6    7    8    9
-----------------------------------------------------------------
Tube1:    F    F    B    C    B    F    E    D    B    A
-----------------------------------------------------------------
Tube2:    D    C    B    E    F    A    D    B    F    F
-----------------------------------------------------------------
Tube3:    C    B    D    E    F    C    B    B    A    E
-----------------------------------------------------------------
Tube4:    B    E    E    E    F    A    F    C    C    E
-----------------------------------------------------------------
Tube5:    C    A    C    D    C    A    A    D    A    D
-----------------------------------------------------------------
Tube6:    E    A    E    F    A    D    D    D    B    C
-----------------------------------------------------------------
Tube7:    *    *    *    *    *    *    *    *    *    *
-----------------------------------------------------------------
Tube8:    *    *    *    *    *    *    *    *    *    *
-----------------------------------------------------------------
Tube9:    *    *    *    *    *    *    *    *    *    *
-----------------------------------------------------------------

Enter a tube to move from and destination (eg.: 1 7): 3 7
```

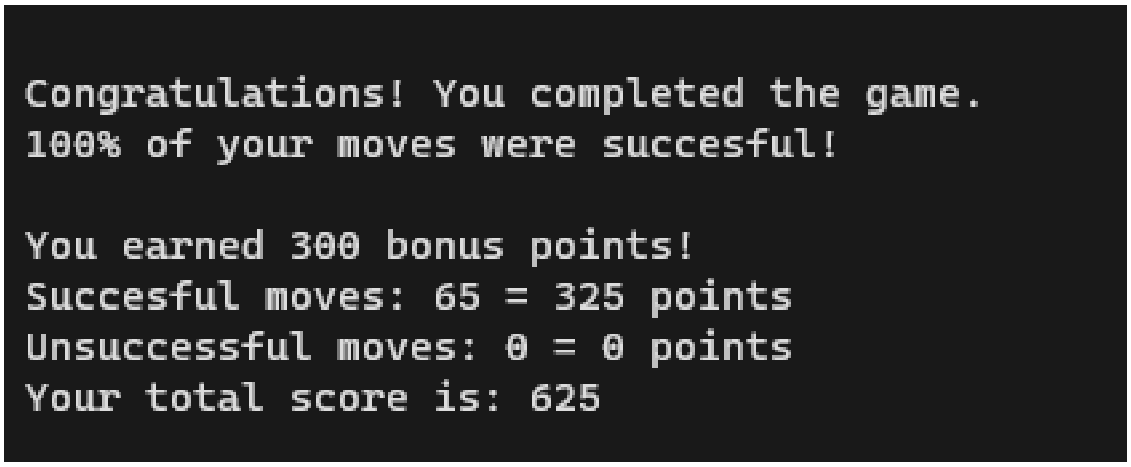
Figure 5 - Game is ready and waiting for the player's input.


```
64 successful moves made out of total 64 moves.

Current state of the game (ball order: bottom -> top):
-----------------------------------------------------------------
index:    0    1    2    3    4    5    6    7    8    9
-----------------------------------------------------------------
Tube1:    A    A    A    A    A    A    A    A    A    A
-----------------------------------------------------------------
Tube2:    D    *    *    *    *    *    *    *    *    *
-----------------------------------------------------------------
Tube3:    D    D    D    D    D    D    D    D    D    *
-----------------------------------------------------------------
Tube4:    B    B    B    B    B    B    B    B    B    B
-----------------------------------------------------------------
Tube5:    *    *    *    *    *    *    *    *    *    *
-----------------------------------------------------------------
Tube6:    *    *    *    *    *    *    *    *    *    *
-----------------------------------------------------------------
Tube7:    E    E    E    E    E    E    E    E    E    E
-----------------------------------------------------------------
Tube8:    C    C    C    C    C    C    C    C    C    C
-----------------------------------------------------------------
Tube9:    F    F    F    F    F    F    F    F    F    F
-----------------------------------------------------------------

Enter a tube to move from and destination (eg.: 1 7): 2 3
```

Figure 6 - Game is about to end with the upcoming move.


```
Congratulations! You completed the game.
100% of your moves were succesful!

You earned 300 bonus points!
Succesful moves: 65 = 325 points
Unsuccessful moves: 0 = 0 points
Your total score is: 625
```

Figure 7 - End-game.

5. TABLE OF THE SOURCE CODE

Type	**Name**	**Description**
Constant	MAX_COLORS	Used to define colors array
Constant	MIN_TUBES	Minimum number of tubes possible
Constant	MAX_TUBES	Maximum number of tubes possible
Constant	BALLS_PER_TUBE	Number of balls each tube can hold
Constant	BONUS_50	Bonus points for 50% accuracy
Constant	BONUS_75	Bonus points for 75% accuracy
Constant	BONUS_100	Bonus points for 100% accuracy
Constant	VALID_MOVE_SCORE	Points to give for each valid move
Constant	INVALID_MOVE_SCORE	Points to erase for each invalid move
Variable	char colors[MAX_COLORS]	Holds the chars to represent colors
Variable	char* colorBowl	Used to mix up colors
Variable	char** tubes	Tubes that hold the balls
Variable	int tubeCount	Number of tubes
Variable	int emptyTubes	Number of empty tubes
Variable	int validMoves	Valid move counter
Variable	int invalidMoves	Invalid move counter
Variable	bool gameOver	Follows the game state
Function	void clear()	Clears screen
Function	void initialize(int*)	Sets up the game
Function	void generateBalls(int)	Generates colored balls
Function	void swap(char*, char*)	Axillary function for shuffling
Function	void shuffle()	Shuffles the color bowl array
Function	void createTubes(int)	Creates necessary number of tubes
Function	void fill()	Fills the shuffled balls into the tubes
Function	void printTubes()	Prints the tubes
Function	void updateGame()	Updates the game state
Function	void makeMove()	Controls ball movement
Function	int findLastNonEmptyIndex(int)	Finds the ball on top
Function	bool isEmpty()	Checks if the tube empty
Function	bool isGameCompleted()	Checks if the game is over

Table 1 - Shows the constants, variables and functions used in the source code and their basic functions.