Project #1

Title:

Mastermind

(ver 1.0)

Course:

CSC-5

(40718)

Release Date:

02 February 2015

Author:

Taylor Yost

Table of Contents

1 Introduction \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_01

2 Game Rules/Description\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_03

3 Development/Summary \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_04

3.1 Code Breakdown\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_04

3.2 Development Goals Achieved\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_04

3.3 Development Goals Planned \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_05

4 Concept Locations \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_05

5 Flowcharts \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_07

6 Pseudocode \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_10

7 Sample IO \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_13

**1|** Introduction

Welcome to the game of Mastermind! This program was created as part of a midterm project under the supervision of Dr. Mark E. Lehr, Riverside City College. Code was developed and written by Taylor Yost. This documentation serves as a record of the development process, and more directly explains the practices used within this program.

**2|** Game Rules/Description

The game of Mastermind is played as follows:

The game of Mastermind is a two player game. One player is the Code-Maker, while the other is the Code-Breaker. The goal of the Code-Breaker is to guess the code made by the Code-Maker. Should the Code-Breaker fail to deduce the code correctly within ten turns; the game is lost and may be played again.

Job of Code-Breaker: The Code-Breaker will have to guess both the correct color and order of the code. Using feedback from the Code-Maker, the Code-Breaker can use logic to reveal the correct code. Valid colors are as follows: Red(R), Orange(O), Yellow(Y), Green(G), Blue(B), and Purple(P). Colors may be repeated, and are random. If a color other than the ones listed above is entered, then that turn is wasted; only colors from the above list will be chosen by the Code-Maker.

Feedback: The Code-Maker gives feedback to the Code-Breaker with two colors. Black(B) means that the Code-Breaker has guessed a correct color, and that it is in the right position. White(W) means that the Code-Breaker has guessed a correct color, but it is in the wrong position. Since duplicates are allowed within the code, both a Black and White color may be given for the same color guess.

**3|** Development/Summary

Project code was fairly simple; however there are a lot of logical statements that had to be checked. The verification process of these lines is what took up the majority of the developmental process. In total, excluding the documentation, about 32 hours were spent in development.

The decision to split all feedback decision logic into separate functions was made early on in the process. While it made the code longer, it greatly improved readability by allowing me to free main from what would seem like an endless stream of if-else statements. Once the decisions were correctly assigned to their respective output block variables, the project became much simpler.

Additionally, understanding the difference between pass-by-value and pass-by-reference were pivotal in getting the final version running properly.

**3.1|** Code Breakdown

Comment Lines: 48

Miscellaneous Lines: 364 (Bracket lines, skipped lines, formatting)

Total Lines: 1105

Unique Lines: 693

Functions: 42

**3.2|** Development Goals Achieved

- Game functional

- Actual “playing field” instead of just a boring linear i/o stream

- Computer makes code with duplications

- Plays true to the style of real Mastermind

**3.3|** Development Goals Planned

- Reduce bulk

- Utilize arrays

- Streamline output process

- Standardize coding style

- Improve feedback by randomizing or standardizing B/W positions into one box per turn

- More user-friendly interface

**4|**Concept Locations

Data Types:

global constants----------------------------------------------------------line 21

float –---------------------------------------------------------------------line 370

int ---------------------------------------------------------------------------line 79

string -----------------------------------------------------------------------line 80

unsigned int-------------------------------------------------------------line 119

char-----------------------------------------------------------------------line 120

arrays---------------------------------------------------------------------line 122

boolean-------------------------------------------------------------------line 149

Loops:

if else----------------------------------------------------------------line 356-364

else if----------------------------------------------------------------line 186-354

if----------------------------------------------------------------------------line 200

while----------------------------------------------------------------line 332-346

do while------------------------------------------------------------line 85-108

switch---------------------------------------------------------------line 91-105

for-------------------------------------------------------------------line 161-355

Functions:

function declaration----------------------------------------------------line 24

function call---------------------------------------------------------------line 94

function definition-----------------------------------------------line 113-365

Other:

time-----------------------------------------------------------------------line 151

rand-----------------------------------------------------------------line 152-155

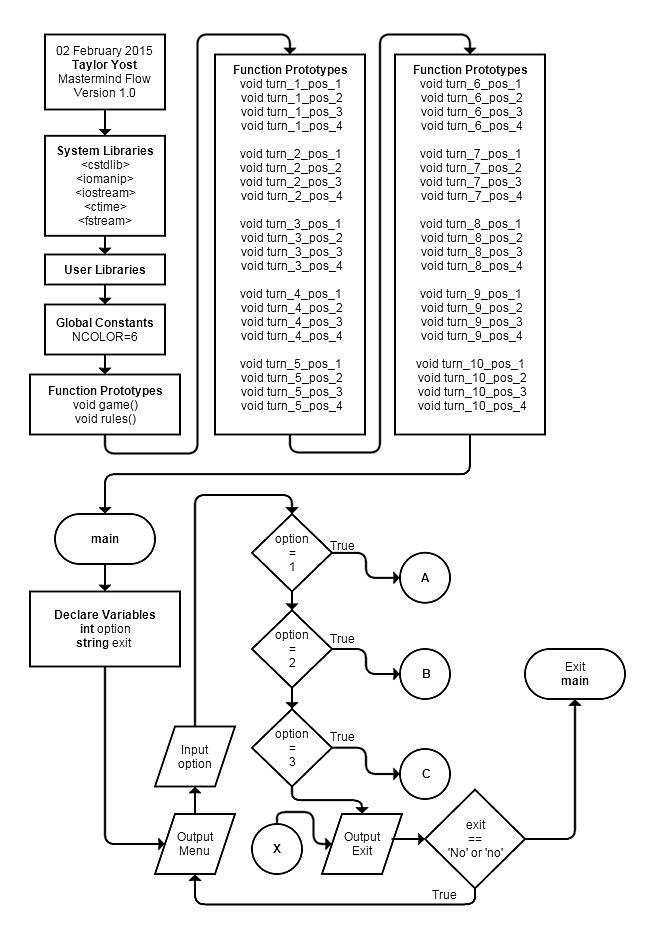
modulus------------------------------------------------------------line 152-155

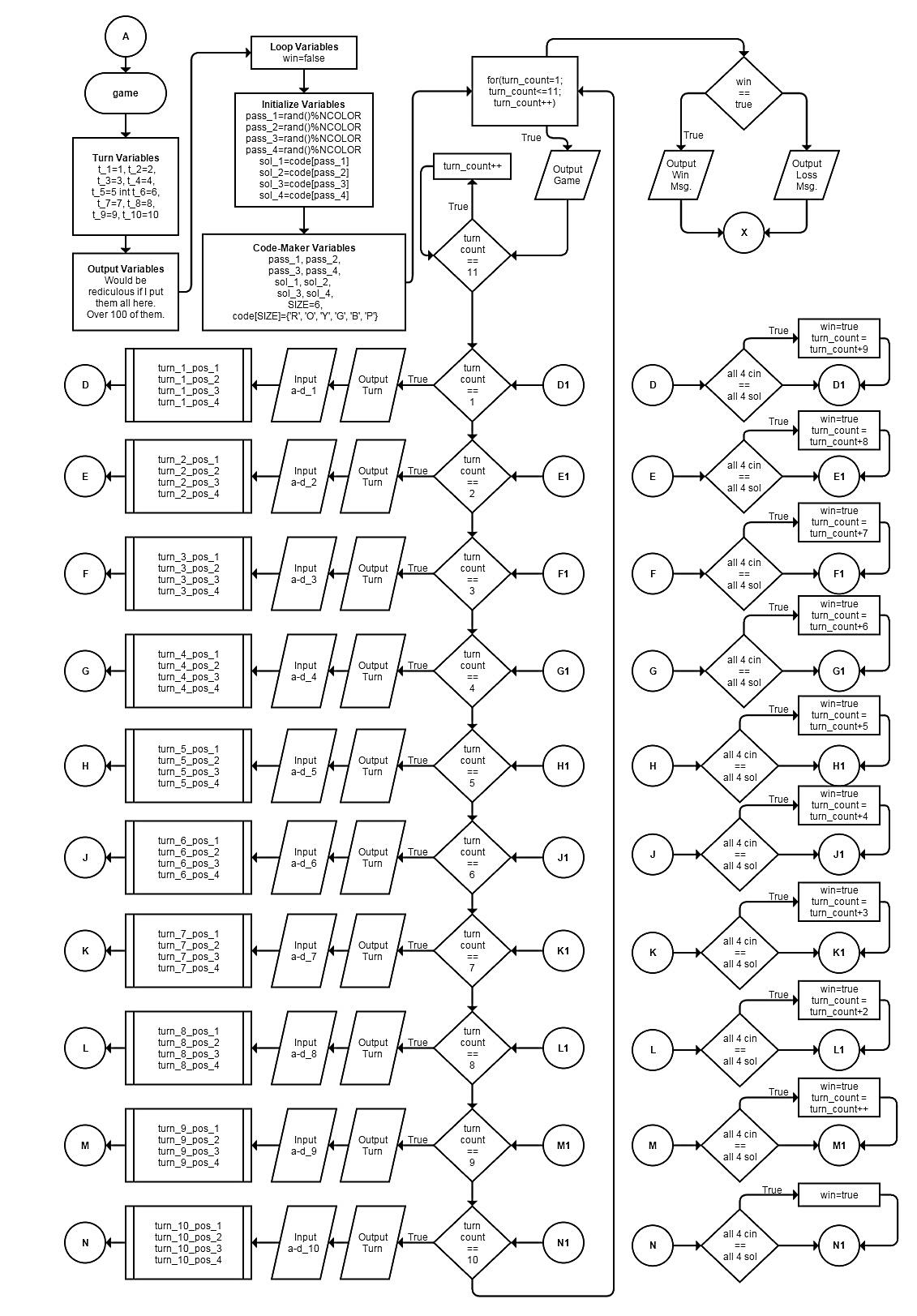
increment/decrement------------------------------------------------line 161

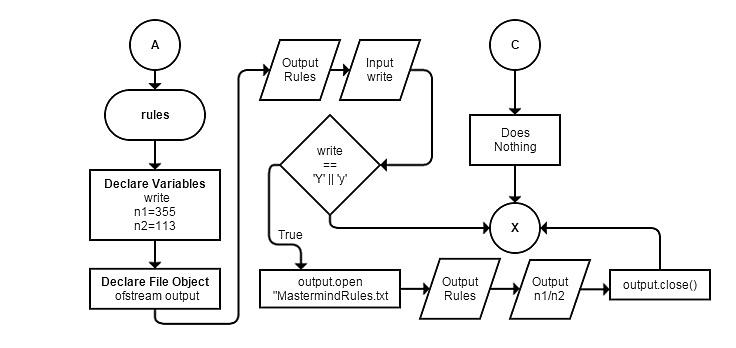
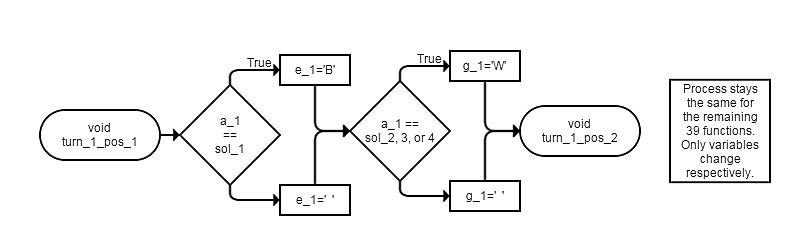
writing to a file----------------------------------------------------line 400-423

I/O manipulation-------------------------------------------------------line 421

pass by reference/pass by value------------------------------------line 427

**5|** Flowcharts





**6|** Pseudocode

int main()

prompt choices to player:

[1] Play Game

[2] Game Rules

[3] Exit Game

input:

If input = 1:

play game

go to game() function

If input = 2:

game rules

go to rules() function

If input = 3:

exit program

If input !=1, 2, or 3

invalid input...exit

ask if player wants to exit

if player does not want to exit game, loop ^

void game()

declare all turn number variables using:

- t\_1, t\_1, etc...

declare all Code-Maker solution variables using:

- sol\_1, sol\_2, etc...

- pass\_1, pass\_2, etc...

- code[amt of colors]={All possible colors in game that rand can choose from in this array}

declare all output variables using:

- a through f, 1 through 10 or 20 (depending on which boxes)

- ex: a\_1, b\_1, ..., f\_1 (for turn 1)

declare win/loss boolean value using:

- win=false

srand(static\_cast<unsigned int>(time(0)));

do all 4 passes like so:

ex: pass\_1=rand()%6;

then, use the random value of pass\_1 to pull a random index

of code[]

ex: sol\_1=code[pass\_1];

for(int turn\_count=1; turn\_count<=11; turn\_count++){

output game field here

if(last turn=true)

turn\_count++ to make the game jump to end

else if(turn\_count is 1)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_1\_pos\_1(send needed values)

turn\_1\_pos\_2(send needed values)

turn\_1\_pos\_3(send needed values)

turn\_1\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 2)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_2\_pos\_1(send needed values)

turn\_2\_pos\_2(send needed values)

turn\_2\_pos\_3(send needed values)

turn\_2\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 3)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_3\_pos\_1(send needed values)

turn\_3\_pos\_2(send needed values)

turn\_3\_pos\_3(send needed values)

turn\_3\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 4)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_4\_pos\_1(send needed values)

turn\_4\_pos\_2(send needed values)

turn\_4\_pos\_3(send needed values)

turn\_4\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 5)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_5\_pos\_1(send needed values)

turn\_5\_pos\_2(send needed values)

turn\_5\_pos\_3(send needed values)

turn\_5\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 6)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_6\_pos\_1(send needed values)

turn\_6\_pos\_2(send needed values)

turn\_6\_pos\_3(send needed values)

turn\_6\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 7)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_7\_pos\_1(send needed values)

turn\_7\_pos\_2(send needed values)

turn\_7\_pos\_3(send needed values)

turn\_7\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 8)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_8\_pos\_1(send needed values)

turn\_8\_pos\_2(send needed values)

turn\_8\_pos\_3(send needed values)

turn\_8\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 9)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_9\_pos\_1(send needed values)

turn\_9\_pos\_2(send needed values)

turn\_9\_pos\_3(send needed values)

turn\_9\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

else if(turn\_count is 10)

prompt for turn one

send cin values of turn one to turn one position functions:

turn\_10\_pos\_1(send needed values)

turn\_10\_pos\_2(send needed values)

turn\_10\_pos\_3(send needed values)

turn\_10\_pos\_4(send needed values)

check for game win, if win is true then jump out of loop

if(win is true)

output “You win!”

else

output “You lose!”

}

void rules()

output all game rules here

void turn\_1\_pos\_1(bring in values)

if(first position = first solution)

assign a black peg

else

do nothing

if(first position = any other solution)

assign a white peg

else

do nothing

This continues for the other 39 functions, and variables change accordingly. At this point I started work on the actual code. Some features were added after this stage; however, they are not difficult to comprehend their changes.

**7|**Sample IO

Please refer to the examples below that demonstrate some features of the game:

Example 1: What it looks like to win!

Example 2: What it looks like to lose!

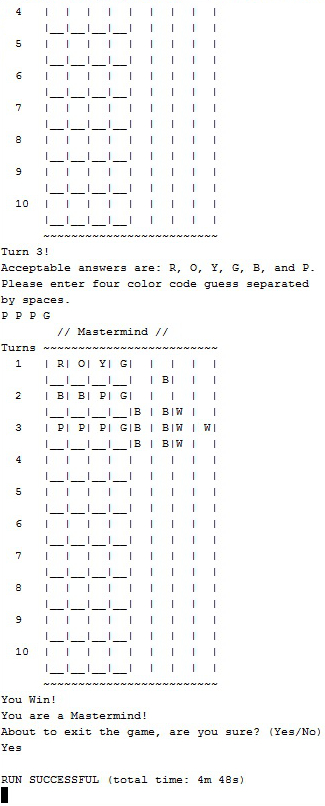
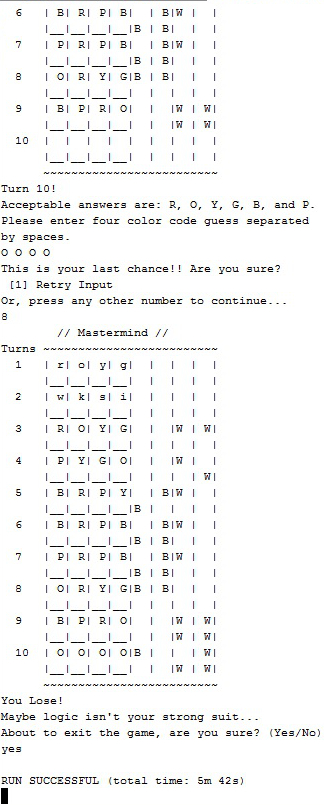
Example 3: How new game starts

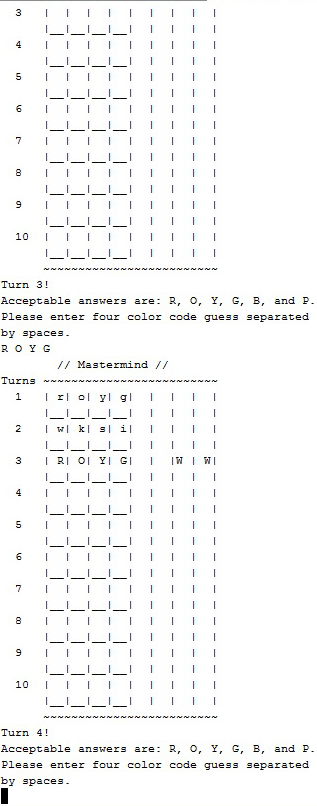
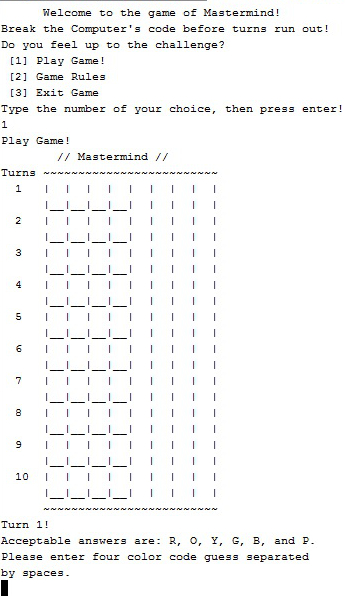
Example 4: How invalid inputs are treated in game

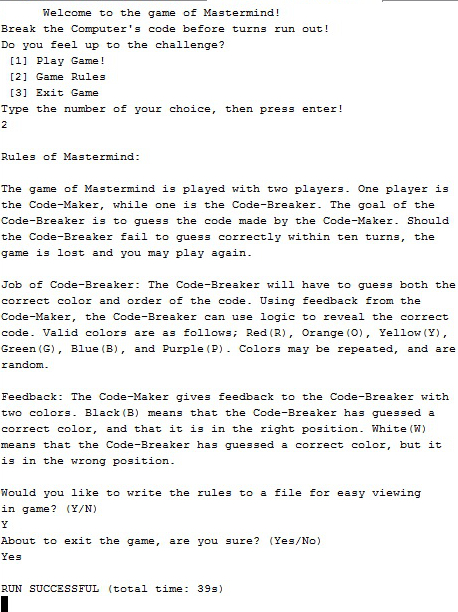
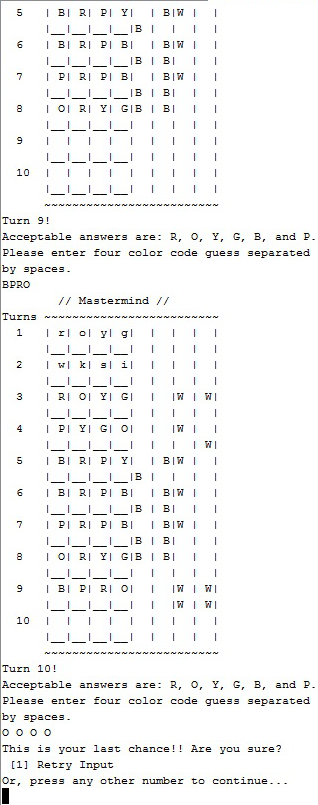
Example 5: Last chance mechanic

Example 6: Viewing the rules

Example 1: Example 2:



Example 3: Example 4:

Example 5: Example 6: