[1 Mastermind 2](#_Toc372890155)

[1.1 Lösungsansatz 2](#_Toc372890156)

[1.1.1 Aufteilung Unit 2](#_Toc372890157)

[1.1.2 Aufteilung Hauptprogramm 3](#_Toc372890158)

[1.2 Source 4](#_Toc372890159)

[1.2.1 MasterMindUnit 4](#_Toc372890160)

[1.2.2 MasterMind 7](#_Toc372890161)

[1.3 Tests 11](#_Toc372890162)

[1.3.1 Fehlerhafte Eingaben Start 11](#_Toc372890163)

[1.3.2 Korrekte Eingaben Start 11](#_Toc372890164)

[1.3.3 Ungültige Farbwahl 11](#_Toc372890165)

[1.3.4 Gültige Farbwahl 12](#_Toc372890166)

[1.3.5 Normale Programmablauf (Stop/Restart) 12](#_Toc372890167)

# Mastermind

Folgend ist die Dokumentation der Übung 4 beschrieben, die die Entwicklung des Spiels Mastermind fordert. Hierbei sollen Farbkombinationen per Zufall vom Programm erstellt und vom Benutzer erraten werden. Die Ergebnisse sollen über die Konsole ausgegeben werden, wobei bereits erratenen Farben gekennzeichnet und dem Benutzer mitgeteilt werden sollen. Hierbei ist auf eine schöne Formatierung der Ausgabe zu achten, sowie eine korrekte Fehlerbehandlung, was miteinschließt, dass ungültige Eingaben des Benutzers nicht zu einen Programmabsturz oder einem unerwarteten Verhalten des Programms führen.

Die Beendigung des Spiels ist nur möglich am Beginn des ersten Spiels oder nachdem alle Farben erraten wurden. Eine außerordentliche Beendigung während des Spiels ist nur über die Tastenkombination STRG + C und anschließend Y möglich, was einem Programmabbruch gleichkommt.

## Lösungsansatz

Folgend ist der Lösungsansatz für die Entwicklung des Spiels Mastermind angeführt.

Es soll ein Datentyp spezifiziert werden, der die zu erratene Farbkombination oder auch einen Rateversuch aufnehmen kann.

Das Spiel Mastermind soll maximal 6 Farbkombinationen unterstützen, daher soll ein Datentyp spezifiziert werden, der diese Beschränkung spezifiziert. Diesen Datentyp soll einen eigenen Datentyp eines Array verwenden um nicht Arrays im Programm zu verwenden, die größere Dimensionen aufweisen als die unterstütze Anzahl der gespielten Farben. Ebenso soll eine Konstante Array of String eingeführt werden, welcher die unterstützten Farben spezifiziert.

Für die Repräsentation des Ergebnisses soll ein eigener Datentyp spezifiziert werden, welcher den vorherige beschriebenen Datentyp für die Farbkombination wiederverwendet (Die gewählten Farben des Benutzers) und welcher die Position der bereits erratenen Farben beinhaltet sowie die Anzahl der erratenen Farben.

Die Implementierungen und Schnittstellen des Spiels sollen in einer Unit zusammengeführt werden, da dieses Programm auch ein Gedächtnis besitzen soll (generierte Farben, Anzahl der zu generierenden Farben). Hierbei soll darauf geachtet werden, dass der Zugriff auf das Gedächtnis nur als Lesezugriff erlaubt wird. Hilfsfunktionen und –Prozeduren sollen nur innerhalb der Unit sichtbar sein und nicht von außen aufrufbar sein.

### Aufteilung Unit

Es sollen nur Funktionen und Prozeduren nach außen sichtbar gemacht werden, welche für die Implementierung des Spiels in einem Hauptprogramm absolut erforderlich sind, Utility Funktionen und Prozeduren sollen nur innerhalb der Unit gehalten werden, sowie das Gedächtnis, welches nur mittels Lesezugriff nach außen sichtbar gemacht werden soll.

Es sollen Initialisierung Funktionen für die verwendeten Datentypen implementiert werden, damit undefinierte Zustände bei den verwendeten Instanzen nicht auftreten können.

### Aufteilung Hauptprogramm

Es sollen die einzelnen Teile des Spielablaufs in einzelnen Funktionen oder Prozeduren aufgeteilt werden, um die einzelnenn Teilfunktionen des Spiels klar voneinander zu trennen damit diese in einer Prozedur namens Run zusammenzufassen werden können. In dieser Prozedur werden dann die einzelnen Abläufe des Spiels klar ersichtlich.

Mittels einer Prozedur namens **Run** soll das Spiel solange ausgeführt werden, bis der Benutzer das Spiel beendet.

Über eine Funktion namens **Start** soll das Programm gestartet werden, wobei hier der Benutzer gefragt werden soll ob er das Programm starten oder beenden will. Fehlerhafte Eingaben sollen abgefangen werden, und diese Funktion soll nicht beendet werden, bis der Benutzer eine gültige Eingabe getätigt hat.

Mittels einer Funktion **Configure** soll der Benutzer das Programm konfigurieren können, wobei der Benutzer die Anzahl der zu spielenden Farben definieren kann. Auch hier soll diese Funktion solange nicht verlassen werden, bis der Benutzer eine gültige Wahl getroffen hat.

Über eine Funktion **ReadUserInput** sollen die Benutzereingaben bezüglich der Rateversuche zusammengefasst und über eine ColorCombination Instanz zurückgegeben werden. Hierbei soll der Benutzer solange Farben eingeben müssen, bis er alle verlangten Farben gültig angegeben hat.

Mittels einer Funktion **ValidateUserInput** sollen die Benutzereingaben validiert und die Ergebnisse ausgegeben werden.

Für jede weiter Dokumentation sei auf den Source verweisen.

## Source

Folgend ist der Source des Programms Mastermind angeführt. Hierbei gibt es zwei Teile

1. MasterMindUnit (Enthält Spielfunktionalitäten)
2. MasterMind (Hauptprogramm, realisiert Spiel über die Konsole)

### MasterMindUnit

Folgend ist der Source der MasterMindUnit aufgeführt, welche die Hauptfunktionalitäten des Spiels enthält sowie ein Gedächtnis, welches die Anzahl der zu spielenden Farben, wie auch die generierten Farben hält, solange das Gedächtnis nicht von außen erneut initialisiert wird.

{

Unit for the master mind application, which has a memory to hold the

randomly generated color combination and intended difficulty specified by the

color count.

By default this unit is initialized with a color combination of the depth Low(Range).

The Range type specifies the range of supported colors.

Call Init to reset or reinitialize the master mind memory

}

**UNIT** MasterMindUnit**;**

{ ##### Interface part of the master mind unit ##### }

**INTERFACE**

**TYPE**

{ Array which specifies the supported colors }

SupportedColors **=** **ARRAY[**1**..**7**]** **OF** **STRING;**

{ Specifies the range of the supported color count for the generated color combination }

Range **=** 1**..**6**;**

{ The combound which holds a color combination }

ColorCombination **=** **RECORD**

colors**:** **ARRAY[**Range**]** **OF** **STRING;**

**END;**

{ The combound which holds the result of the user guess }

Result **=** **RECORD**

guess**:** ColorCombination**;**

matchedIdx**:** **ARRAY[**Range**]** **OF** Range**;**

matchCount**:** INTEGER

**END;**

{ ###### Function and procedure visible to the main program ###### }

{

Initializes the mastermind memory by generation of a new color combination

with the specified count.

@param

generatedCount: the count of to generated colors contained in the ColorCombination instance

}

**PROCEDURE** Init**(**generatedCount**:** Range**);**

{

Function which validates the user guess and returns the result of the validation via a Result instance.

The case of the defined colors in the ColorCombination instance is ignored.

@param

guess: the ColorCombination which holds the user guess

@result

the result represented by a Result instance which contains the user guess and an array with

the matched indexes and also the count of the matches.

}

**FUNCTION** Validate**(**guess**:** ColorCombination**):** Result**;**

{

Gets the current count of the generated colors.

Allows on read access to the memory count.

@return

the current set color count

}

**FUNCTION** GetCurrentCount**:** INTEGER**;**

{

Returns the supported colors.

Allows only read access.

@return

the supported colors

}

**FUNCTION** GetSupportedColors**:** SupportedColors**;**

{

Function which answers the question if the given string represents a valid color.

The case is ignored.

@param

color: the color to check if its is valid, which means it is supported

@return

true if thegiven color is a valid one, false otherwise

}

**FUNCTION** IsValidColor**(**color**:** **STRING):** BOOLEAN**;**

{

Initializes the a ColorCombination instance with default values.

All colors will be set to an empty string

@return

the initialized ColorCombination instance

}

**FUNCTION** InitColorCombination**:** ColorCombination**;**

(\*{ ## Just for testing ## }

FUNCTION GetGeneratedColors: ColorCombination;\*)

{ ##### Implementation part of the master mind unit ##### }

**IMPLEMENTATION**

**CONST**

{ Array which specifies the supported colors }

Colors**:** SupportedColors **=** **(**'red'**,** 'green'**,** 'blue'**,** 'yellow'**,** 'orange'**,** 'white'**,** 'black'**);**

{

the memory of this unit

}

**VAR**

generated**:** ColorCombination**;**

count**:** Range**;**

{ ###### Private function and procedures ###### }

{

Inits the a Result instance.

1. mathCount will be set to 0

2. matchedIdx to Low(Range) (Access shall be depending on the set mathCount)

3. guess.colors will be set to empty string

@return

the initialized Result instance

}

**FUNCTION** InitResult**:**Result**;**

**VAR**

i**:** RANGE**;**

**BEGIN**

InitResult**.**matchCount **:=** 0**;**

InitResult**.**guess **:=** InitColorCombination**;** {.colors[i] := '';}

**FOR** i **:=** Low**(**Range**)** **TO** High**(**Range**)** **DO** **BEGIN**

InitResult**.**matchedIdx**[**i**]** **:=** Low**(**Range**);**

**END;**

**END;**

{ ###### Public function and procedures implementations ###### }

{ Init color combination instance }

**FUNCTION** InitColorCombination**:** ColorCombination**;**

**VAR**

i**:** Range**;**

**BEGIN**

**FOR** i**:=** Low**(**Range**)** **TO** High**(**Range**)** **DO** **BEGIN**

InitColorCombination**.**colors**[**i**]** **:=** ''**;**

**END;**

**END;**

{ Init the memeory }

**PROCEDURE** Init**(**generatedCount**:** Range**);**

**VAR**

i**:** Range**;**

idx**:** INTEGER**;**

**BEGIN**

{ Aet memeory with default initialized state }

count **:=** generatedCount**;**

generated **:=** InitColorCombination**;**

{ generate random colors }

Randomize**;**

**FOR** i **:=** Low**(**Range**)** **TO** count **DO** **BEGIN**

idx **:=** Random**(**High**(**Colors**));**

{ avoid 0 idx }

**IF** idx **=** 0 **THEN** **BEGIN**

idx **:=** idx **+** 1**;**

**END;**

generated**.**colors**[**i**]** **:=** Colors**[**idx**];**

**END;**

**END;**

{ Validates the user guess }

**FUNCTION** Validate**(**guess**:** ColorCombination**):** Result**;**

**VAR**

i**,** k**:** Range**;**

res**:** RESULT**;**

**BEGIN**

res **:=** InitResult**;**

k **:=** Low**(**Range**);**

**FOR** i **:=** Low**(**Range**)** **TO** count **DO** **BEGIN**

res**.**guess**.**colors**[**i**]** **:=** guess**.**colors**[**i**];**

**IF** generated**.**colors**[**i**]** **=** LowerCase**(**guess**.**colors**[**i**])** **THEN** **BEGIN**

res**.**matchCount **:=** res**.**matchCount **+** 1**;**

res**.**matchedIdx**[**k**]** **:=** i**;**

**IF** k **<** High**(**Range**)** **THEN** **BEGIN**

Inc**(**k**)**

**END**

**END**

**END;**

Validate **:=** res

**END;**

{ checks for valid color string }

**FUNCTION** IsValidColor**(**color**:** **STRING):** BOOLEAN**;**

**VAR**

i**:** INTEGER**;**

valid**:** BOOLEAN**;**

**BEGIN**

valid **:=** false**;**

i **:=** Low**(**Colors**);**

**WHILE** **(NOT** valid**)** **AND** **(**i **<=** High**(**Colors**))** **DO** **BEGIN**

**IF** Colors**[**i**]** **=** LowerCase**(**color**)** **THEN** **BEGIN**

valid **:=** true**;**

**END;**

Inc**(**i**)**

**END;**

IsValidColor **:=** valid**;**

**END;**

{ ###### Public getter functions, realizes only read access ###### }

{ Gets the supported colors }

**FUNCTION** GetSupportedColors**:** SupportedColors**;**

**BEGIN**

GetSupportedColors **:=** Colors**;**

**END;**

{ read access tot eh current set count }

**FUNCTION** GetCurrentCount**:** INTEGER**;**

**BEGIN**

GetCurrentCount **:=** count**;**

**END;**

(\* { ## Just for testing ## }

FUNCTION GetGeneratedColors: ColorCombination;

BEGIN

GetGeneratedColors := generated;

END; \*)

**BEGIN**

{ Init this unit with lowest valud, so that memory has no invalid state }

Init**(**Low**(**Range**));**

**END.**

### MasterMind

Folgend ist der Source des Hauptprogams MasterMind aufgeführt, welches das Spiel über die Konsole realisiert.

{

Program which implements a game called master mind, with colors.

It uses the MasterMindUnit because it provides all necessary functionality

needed to realize a implementation of this game.

The game works via the console, and will ask the user as long as has guessed the colors

correctly.

The user is able to configure the game by entering the count of colors.

On start and after the user has guessed the colors correctly, the user is

asked if he wants to proceed.

If he says no then the program exists, otherwise the program will restart.

}

**PROGRAM** MasterMind**;**

{

The MasterMindUnit to be used in this main program.

}

**USES** MasterMindUnit**;**

{

Start the application by asking the user if he wants to start.

If the user won't start then this function will return false, otherwise true

@return

true if the user wnats to start the application, false otherwise

}

**FUNCTION** Start**:** BOOLEAN**;**

**VAR**

input**:** **STRING;**

initialized**:** BOOLEAN**;**

**BEGIN**

initialized **:=** false**;**

**WHILE** **NOT** initialized **DO** **BEGIN**

Write**(**'Do you want to start the game [y, n] ?: '**);**

ReadLn**(**input**);**

{ Validate the user input }

**IF** **(**'y' **=** LowerCase**(**input**))** **OR** **(**'n' **=** LowerCase**(**input**))** **THEN** **BEGIN**

**IF** 'y' **=** LowerCase**(**input**)** **THEN** **BEGIN**

initialized **:=** true**;**

Start **:=** true**;**

WriteLn**(**'---------------------------------------------------------'**);**

**END**

{ return false on no option }

**ELSE** **IF** 'n' **=** LowerCase**(**input**)** **THEN** **BEGIN**

initialized **:=** true**;**

Start **:=** false**;**

**END;**

**END**

{ Handle invalid input }

**ELSE** **BEGIN**

Writeln**(**'Invalid input ['**,** input**,** ']'**);**

**END**

**END;**

**END;**

{

Procedure which configures the application.

The user can enter the color count which defines the difficulty.

@return

the defined configuration which would be the color count

}

**FUNCTION** Configure**:** Range**;**

**VAR**

input**:** **STRING;**

i**,** count**,** code**:** INTEGER**;**

valid**:** BOOLEAN**;**

**BEGIN**

valid **:=** false**;**

count **:=** 0**;**

code **:=** 0**;**

{ run as long as the user has not entered a valid configuration }

**WHILE** **NOT** valid **DO** **BEGIN**

Write**(**'How many colors you would like to play ['**,** Low**(**Range**),** '-'**,** High**(**Range**),** ']?: '**);**

ReadLn**(**input**);**

Val**(**input**,** count**,** code**);**

{ Validate color range }

**IF** **(**code **=** 0**)** **AND** **(**count **>=** Low**(**Range**))** **AND** **(**count **<=** High**(**Range**))** **THEN** **BEGIN**

Writeln**(**'You have chosen '**,** count**,** ' color(s)'**);**

WriteLn**(**''**);**

Writeln**(**'Supported colors: '**);**

{ Print the supported colors }

**FOR** i **:=** Low**(**getSupportedColors**)** **TO** High**(**GetSupportedColors**)** **DO** **BEGIN**

Write**(**getSupportedColors**[**i**]);**

**IF** i **<** High**(**GetSupportedColors**)** **THEN** **BEGIN**

Write**(**', '**);**

**END**

**END;**

WriteLn**(**''**);**

valid **:=** true

**END**

{ Handle invalid input }

**ELSE**

Writeln**(**'Invalid input ['**,** input**,** ']'**);**

**END;**

Configure **:=** count**;**

**END;**

{

Reads the user input as long as all inputs have been done correctly.

@return

the ColorCombination isntance which holds a valid user input

}

**FUNCTION** ReadUserInput**:** ColorCombination**;**

**VAR**

input**:** **STRING;**

i**:** Range**;**

count**:** INTEGER**;**

**BEGIN**

ReadUserInput **:=** MasterMindUnit**.**InitColorCombination**;**

count **:=** 1**;**

i **:=** Low**(**Range**);**

Writeln**(**''**);**

Writeln**(**'Make your guess !'**);**

Writeln**(**''**);**

{ Get all color guesses which are necessary }

**WHILE** count **<=** MasterMindUnit**.**GetCurrentCount **DO** **BEGIN**

Write**(**count**,** ' Color: '**);**

ReadLn**(**input**);**

{ Validate for correct input }

**IF** MasterMindUnit**.**isValidColor**(**input**)** **THEN** **BEGIN**

ReadUserInput**.**colors**[**i**]** **:=** input**;**

count **:=** count **+** 1**;**

**IF** i **<** High**(**Range**)** **THEN** **BEGIN**

Inc**(**i**);**

**END**

**END**

{ Handle invalid input }

**ELSE** **BEGIN**

Writeln**(**''**);**

WriteLn**(**'Color not supported ['**,** input**,** ']'**);**

Writeln**(**''**)**

**END;**

**END**

**END;**

{

Validates the user input against the generated colors and prints the result to

the console.

@param

guess: the user made guess

@return

true if the user has matched the generated colors

}

**FUNCTION** ValidateUserInput**(**guess**:** ColorCombination**):** BOOLEAN**;**

**VAR**

res**:** Result**;**

i**:** Range**;**

**BEGIN**

{ Init result and get validation }

ValidateUserInput **:=** false**;**

res **:=** MasterMindUnit**.**validate**(**guess**);**

Writeln**(**''**);**

{ If at least one but not all colors have been gueesed correctly }

**IF** **(**res**.**matchCount **>** 0**)** **AND** **(**res**.**matchCount **<** MasterMindUnit**.**GetCurrentCount**)** **THEN** **BEGIN**

WriteLn**(**'The following '**,** res**.**matchCount**,** ' colors are correct: '**);**

**FOR** i **:=** Low**(**Range**)** **TO** **(**Low**(**Range**)** **+** res**.**matchCount **-**1**)** **DO** **BEGIN**

WriteLn**(**res**.**matchedIdx**[**i**],** ' Color: '**,** res**.**guess**.**colors**[**res**.**matchedIdx**[**i**]]);**

**END;**

**END**

{ All colors have been guessed ccorectlly }

**ELSE** **IF** res**.**matchCount **=** MasterMindUnit**.**GetCurrentCount **THEN** **BEGIN**

WriteLn**(**'Congratulations, all colors have been guessed correctly !'**);**

ValidateUserInput **:=** true**;**

**END**

{ If no color has been guessed correctly }

**ELSE** **BEGIN**

WriteLn**(**'Sorry no color have been guessed correctly ! '**);**

**END;**

Writeln**(**''**);**

**END;**

{

Procedure which represents the running game.

It will run as long as the user intends to.

@see

Start

Configure

ReadUserInput

ValidateUserInput

}

**PROCEDURE** Run**;**

**VAR**

count**:** Range**;**

run**,** valid**:** BOOLEAN**;**

guess**:** ColorCombination**;**

**BEGIN**

run **:=** true**;**

{ Run game as long as the user wants to

If a run has started then it must be completed and cannot

be aborted

}

WriteLn**(**'#################### Mastermind #########################'**);**

**WHILE** run **DO** **BEGIN**

{ 1. Start the game }

run **:=** Start**;**

**IF** run **THEN** **BEGIN**

{ 2. Configure the pplication }

count **:=** Configure**;**

{ 2. Init the game }

MasterMindUnit**.**Init**(**count**);**

{ Read user input and validate it as long as the user has guessed the correct colors }

valid **:=** false**;**

**WHILE** **NOT** valid **DO** **BEGIN**

WriteLn**(**'---------------------------------------------------------'**);**

{ 3. Read user input }

guess **:=** ReadUserInput**;**

{ 3. Validate user input }

valid **:=** validateUserInput**(**guess**);**

WriteLn**(**'---------------------------------------------------------'**);**

**END;**

**END;**

**END;**

WriteLn**(**'#################### Mastermind #########################'**);**

**END;**

(\*PROCEDURE TestMasterMindUnit;

VAR

i, j: INTEGER;

color: STRING;

res: RESULT;

guess: ColorCombination;

BEGIN

{ Test isValidColor }

FOR i:= Low(GetSupportedColors) TO High(GetSupportedColors) DO BEGIN

color := GetSupportedColors[i];

WriteLn(i, '-Color: ', color:8, ' / valid: ', MasterMindUnit.isValidColor(color));

END;

WriteLn('----------------------------------------------------------------------------');

{ Test Init }

FOR i := Low(Range) TO High(Range) DO BEGIN

Init(i);

WriteLn('Generated Colors: ', i);

FOR j := Low(Range) TO High(Range) DO BEGIN

WriteLn(j, '-color: ', GetGeneratedColors.colors[j]);

END;

WriteLn('----------------------------------------------------------------------------');

END;

{ Test Validate }

FOR i := Low(Range) TO High(Range) DO BEGIN

Init(i);

guess := InitColorCombination;

FOR j := Low(Range) TO i DO BEGIN

guess.colors[j] := GetGeneratedColors.colors[j];

END;

res := Validate(guess);

FOR j := Low(Range) TO High(Range) DO BEGIN

WriteLn(j, '-generated-color: ', GetGeneratedColors.colors[j]:8, ' / ', j, '-guessed-color: ', res.guess.colors[j]:6, ' / match count: ', res.matchCount);

END;

WriteLn('----------------------------------------------------------------------------');

END;

END;\*)

**BEGIN**

(\*TestMasterMindUnit;\*)

{ Runs the game as long as the user stops it, after he has guessed the colors correctly }

Run**;**

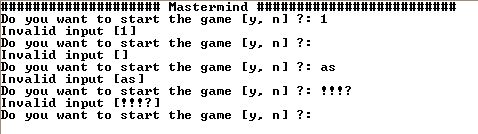
**END.**

## Tests

Folgend ist die Dokumentation der Tests des Programms Mastermind angeführt.

### Fehlerhafte Eingaben Start

Dieser Test testet das Verhalten des Programms, wenn der Benutzer beim Start des Spiels, wenn er gefragt wird ob er das Spiel starten will, ungültige Eingaben tätigt.



Es ist zu sehen, dass diese Fehler abgefangen, dem Benutzer mittgeteilt werden und das die Eingaben solange verlangt werden solange der Benutzer ungültige Eingaben tätigt.

### Korrekte Eingaben Start

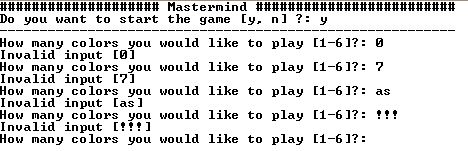
Dieser Test zeigt, dass sobald eine gültige Eingabe erfolgt hat, zum nächsten Schritt weitergegangen wird.



In diesem Fall ist der nächste Schritt die Beendigung des Programms, da hier die Option n (n = No) gewählt wurde.

### Ungültige Farbwahl

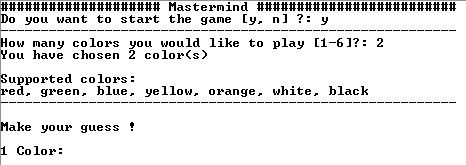
Dieser Test zeigt das Verhalten des Programms wenn der Benutzer ungültige Eingaben bei der Anzahl der Farben tätigt.



Hier ist zu sehen, dass nicht nur ungültige Eingaben bezüglich ungültiger Zeichenfolgen, sondern auch ungültige Wertebereiche abgefangen werden. Auch hier wird der Benutzer solange gezwungen Eingaben zu tätigen solange er ungültige Eingaben tätigt.

### Gültige Farbwahl

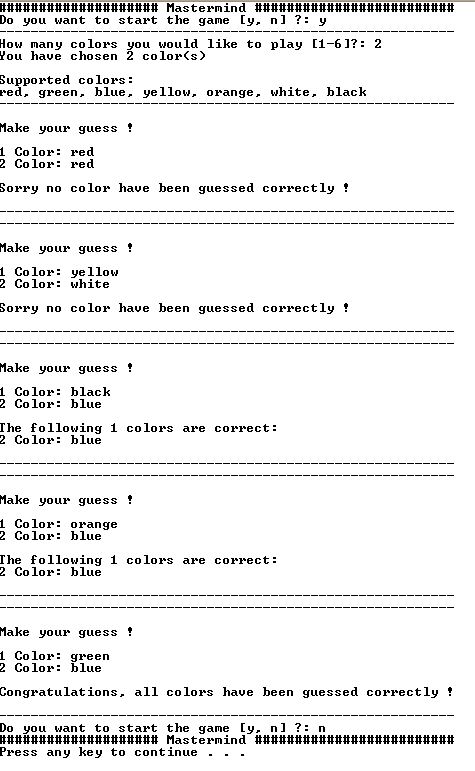
Dieser Test zeigt das Verhalten des Programms wenn der Benutzer eine gültige Angabe bei der Farbenanzahl getätigt hat.



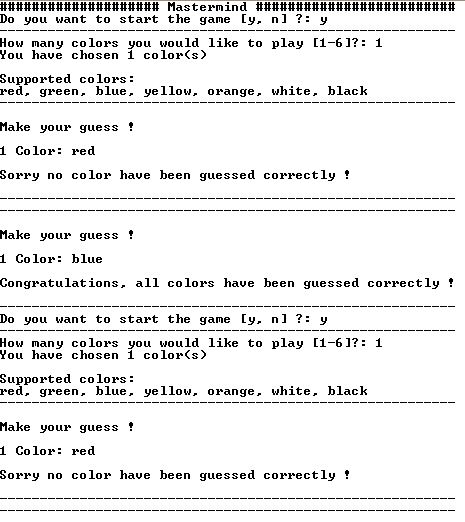
Es ist zu sehen, dass nach einer gültigen Werteeingabe zum nächsten Schritt im Spiel gegangen wird.

### Normale Programmablauf (Stop/Restart)

Dieser Test zeigt zwei Beispiele für einen normalen Programmablauf, einmal mit Stoppen und einmal mit einen erneuten Starten des Spiels nach dem der Benutzer alle Farben erfolgreichen erraten hat.



Dieses Beispiel zeigt, dass das Programm ordnungsgemäß beendet wird, nachdem der Benutzer sich nach dem Spielende dafür entschieden hat.



Dieses Beispiel zeigt, dass das Spiel ordnungsgemäß gestartet wird, nachdem der Benutzer sich nach Speilende dafür entschieden hat.