CS11 Machine Problem 2 Documentation

Abcede, Ma. Pauline Heffron, Joaquin
Reyes, Victor Edwin

January 10, 2022

Chapter 1

Introduction

This document is made as external documentation of the algorithms and logic underpinning our implementation of the Mastermind Game. The User Manual for lay users explaining how to play the game is also located in this document.

In case this project was distributed by other means, this project also has a GitHub repository here:
https://github.com/VeeIsForVanana/MP2

1.1 Project Requirements

This project's GUI implementation requires the tcod python package. Install it using 'pip install tcod' or refer to this link
https://python-tcod.readthedocs.io/en/latest/installation.html.

1.2 Project Structure

The directory containing this project will contain two scripts of interest:

- 'mastermind.py' Contains the base implementation of the game, with minimum features as detailed in the project specifications.
- 'gui.py' Contains an implementation of the game that makes use of a graphical user interface courtesy of the tcod package. This implementation makes use of concepts not taught in the CS11 curriculum such

as classes, inheritance, and enums. These concepts were used out of sheer necessity due to the complex demands of the GUI implementation. This script has three dependencies contained in the project:

- ‘handler.py‘ Contains an implementation of event handler classes inherited from the tcod package. Said handlers also handle most of the game’s logic.
- ‘constants.py‘ Contains vital game data detailing color data and window size.
- ‘mastermind.py‘ Along with being the gui-less game implementation, some functions in this script are borrowed by the gui implementation for the game logic

Both implementations will be documented in separate sections of this document.

Chapter 2

User Manual

2.1 User Manual 1: mastermind.py—

This script makes use primarily of text inputs and a textual interface. The program solely implements Mastermind with some added features. Guiding prompts are in all parts of the program, but this manual will still provide a detailed guide into the game.

The game opens with prompts asking for required parameters
(code length, and whether colors will repeat) for the game setup. Input is validated and the request
will loop until it receives what is considered valid input.

By default, length is expected to be within 4 and 8, inclusive. Meanwhile, repeat will always be either "Yes" or "No".

Afterward, the game proper will start.

To summarize, in Mastermind, the player is expected to make guesses what the computer-generated code is. Said code will consist of "colors" (numbers from 1 to 8 by default). The player is expected to match this code with their guess to win the game.

Otherwise, if they fail to do so within ten turns, they lose. When the player inputs a guess, the computer prints out feedback in the form of "Red" and "White" numbers, based on which they may alter their input. They may also make use of lifelines, as shall be detailed below.

At the start of every turn, the computer will prompt the user to input their valid guess. The player has the option of inputting a guess for the computer's code or to ask for a lifeline by inputting "lifeline#1" or "lifeline#2".

The game will loop until a valid input is entered.

When a valid guess is entered, it will be checked against the computer's code. Feedback will be given as follows:

- 'Red: x' when the player's input matches the computer's code exactly at x positions
- 'White: y' when the player's input has elements in the computer's code that are not in the right place at y positions

If a lifeline is asked for, feedback will be as follows:

- 'lifeline#1' gives the player the value of a random element, but not the position. As a penalty, the player's turns increment by 1
- 'lifeline#2' gives the player the value and position of a random element. It then changes the hint code to reflect this. As a penalty, the player's turns increment by 2

When the player's turns prevent them from taking a certain lifeline without immediately losing the game afterward, the game blocks use of that lifeline.

2.2 User Manual 2: gui.py

This script makes use of a graphical user interface to allow the user to interact with the game. The program implements Mastermind with some extra features that are not necessarily the same as those in 'mastermind.py'. The game requires the player to use only the arrow and enter keys to navigate.

On launching, the player is met with the main menu screen. Here, they may navigate between the two options "Play" and "Quit" using the arrow keys, and select using the `ENTER` key.

As one might expect, "Quit" allows the player to exit the game. "Play" starts a new game of Mastermind.

When the game starts, the player will encounter a screen with three sections. The section to the upper left is the code input screen. To the lower left is a panel to record the player's past guesses. The rightmost and largest panel is for the computer to communicate textually with the player. It should be noted that at any stage of play, the player may press `ESC` to try

to leave the game, at which point the computer will confirm if they want to leave by pressing `ESC` again.

At the very start of the round, the computer will ask the player to provide a length for the code (from 4 to 8) and then whether the code will repeat or not. At each of these stages, the computer will provide the player with choices from which to choose using the keyboard controls. The code will then be generated in the background and then the game will begin.

Like in 'mastermind.py' the player must guess the computer's code within 10 turns. The player will input this using the input panel. The input panel will display the player's current unsubmitted guess, each digit of which the player may change using the arrow keys. The player can move the digit selector using the left and right arrows, and change the digit using the up (to increment by 1) and down (to decrement by 1) keys. The player may select from eight color-digits. A ninth option is reserved to input "LIFELINE", when the player selects this ninth digit for the length of their allowed guess, the program will complete "LIFELINE" for the rest of the guess (including unallowed digits). Do note that the LIFELINE digit is not compatible with other guess digits and if used in a guess, the player's guess will not be recognized and they will be given a chance to reenter a valid guess.

When the player presses `ENTER`, they submit their guess for evaluation by the computer. If their guess is an attempt at guessing the code, their guess will be evaluated and feedback will be returned (see User Manual for 'mastermind.py'). If their guess was asking for a lifeline, and they still have available lifelines, they will be allowed to specify their lifeline request by entering 1 or 2 in the modified input panel. They will then receive feedback corresponding to the lifeline they selected as well as receiving a corresponding turn malus (also see User Manual for 'mastermind.py').

When the player wins, or passes ten turns without winning, the corresponding congratulatory or consolatory is displayed After this, the player may return to the main menu using `ESC`, from which they may start another game or quit.

Chapter 3

Implementations

3.1 Implementation 1: mastermind.py

This script implements the algorithm below. Do note that there is not an exact one-to-one correspondence between the code and this algorithm.

```
Define a function to validate input taking a list of accepted
    inputs and the player input as parameters:
  The function returns the player input if it is in the list of
      accepted values. Otherwise it outputs nothing.

Define a function to validate a color code input taking the desired
     code length, a list of accepted characters, and the
 player input as parameters:
  The function returns the player input if it is of the desired
      length and if all its characters are accepted else
  The function returns a request for a lifeline if the player input
      is such else
  The function returns nothing

Define a function to request a valid player code length with a
    maximum and minimum value as parameters:
  The function prints a request to the player for a valid length.
  The function validates the player's input against a list of
      integers between the minimum and maximum length.
  The function only returns the input when valid.
```

```
Define a function to request a valid player input for whether the
    code will repeat colors or not:
  The function prints a request for player input of "Yes" or "No".
  The function validates the player's lowercase input against "Yes"
      and "No".
  The function only returns the input when valid.

Define a function to randomize a code, taking a length and whether
    the code repeats colors or not as input:
  The function returns either a random sequence of color codes if it
       may repeat codes or permutates the string of all
   codes and cuts down the length if not.

Define a function to check the computer's code against the player's
     guess, taking both as input:
  The function creates a dictionary with the colors as keys and the
      number of their occurences in the computer's code.
  The function traverses both the code and guess and stores the
      number of exact matches as "red". For each exact match
  of a color, 1 is subtracted from its corresponding entry in the
      dictionary.
  The function traverses the player's guess only and stores the
      number of times the colors in the player's guess appears
   in the code (excepting the already-present exact matches) as long
        as the dictionary entry for that color is not zero
   or less. Then it subtracts this count from the corresponding
       dictionary entry. The count is also stored as "white".
  The function returns red and white as results.

Define a function to handle lifeline 1, taking only the computer's
    code as input:
  The function picks a random element of the code and prints it out
      without location data.

Define a function to handle lifeline 2, taking only the computer's
    code as input:
  The function picks a random position in the code and prints it out
       with element data.
  The function returns the position.
```

```
~~MASTERMIND IMPLEMENTATION BEGINS HERE~~
START

Define the 'usable_colors' (by default integers 1 to 8)
Define boolean values to track, all start as False:
    Whether the player has won, 'win'
    Whether the player has used lifeline1, 'used_lifeline1'
    Whether the player has used lifeline2, 'used_lifeline2'
Set integer values:
    For how much a call of lifeline1 deducts from turns, '
       lifeline1_loss' (by default 1)
    For how much a call of lifeline2 deducts from turns, '
       lifeline2_loss' (by default 2)

Set 'length' to be None (an value)
While 'length' is None (while 'length' is an invalid value):
    Set 'length' to be the validated player length as in the
       function above.
Set 'repeat' to be None (an value)
While repeat is None (while 'repeat' is an invalid value):
    Set 'repeat' to be the validated player repeat choice as in the
       function above.
Randomize 'code' using the function defined above.
Set 'turns' to be 1
Set 'visible_code' to be a string of only '*' with length equal to
    code length

While turns <= 10 and the player has not won:
    Set 'player_input' to be None (an invalid value)
    While 'player_input' is None (while 'player_input' is invalid):
        Print a prompt detailing turn count and a code hint (
           visible_code)
        Set 'player_input' to be the validated code as in the
           function above
        If the player asks for a lifeline x and the lifeline x has
           already been used ('used_lifelinex'):
           Invalidate player_input by setting to None
        Else if player input is still invalid (None):
           Print a corrective message
```

```
    If the player requested for lifeline 1:
        Execute function lifeline1 with argument 'code' as defined
            above
        Add the specified 'lifeline1_loss' to 'turns'
    Else if the player requested for lifeline 2:
        Execute function lifeline2 with argument 'code' as defined
            above. Take its return value as 'position'
        Change the value of the character at 'position' of '
            visible_code' to the element at 'position' in 'code'
        Add the specified 'lifeline2_loss' to 'turns'
    Else (if the player entered a validated guess):
        Let 'red', 'white' be the variable outputs of the code
            checking between 'player_input' and 'code'
         as defined above
        If 'red' is the same as the length of the code (i.e. if
            there is a one-to-one match across the full length):
            The player has won. Set 'win' to True
        Else:
            Print out 'red' and 'white' in player-readable format.
    Increment 'turn' by 1

If win (if the player has won):
    Print congratulatory message
Else (if turns >= 10 and the player has not yet won):
    Print loss message

Set player_input to None

While player_input is None:
  Ask the player if they want to play again, validate lowercase of
      input against "yes" or "no"

If the player asks to play again:
  Execute the whole game loop again.
Else:
  Exit

END
```

3.2 Implementation 2: gui.py

In the setup, the script sets up the tools used for rendering the game visually along with handling input

```
Import window height and window width from an outside list of
    constant values
Load the image "dejavu10x10_gs_tc.png" as the tilesheet

Set up a new terminal with window height and window width from the
    aforementioned variables and the loaded tileset and
use the title "Mastermind MP2"

Create a "console" object (functionally similar to a canvas)
    tailored for the terminal we are using.
Create a "Menu Handler" object (a custom class) to handle input and
     the game logic and give it the
console as its property

Enter an infinite loop ...
```

To elaborate on what just happened, the three most important concepts above are the terminal, which is the window itself; the console, which contains the graphics drawn onto the terminal; and the handler, which handles input and the game.

The terminal will rarely be referenced hereafter except when it is instructed to render the console.

The console is an object similar to a canvas with methods for drawing on itself (printing characters, tiles, etc.). It will be passed to the handler, which will use these methods to alter it and draw on it.

The handler is an object (of the Menu Handler class) for now that has methods to handle and process key and mouse events, and that contains the logic of the game. These functionalities will especially be focused on later in this document.

For now, let us continue and examine the contents of the game's infinite loop.

```
Clear the console in preparation for what is to be drawn next.
Let the console be modified by the Manu Handler's rendering
    function.
```

```
For all events (keypresses, mouse movement on the window)
    recognized by the terminal:
  Pass the event to the Menu Handler and expect the Menu Handler to
      change into possibly a different object based on
  what it returns
  Draw the console onto the terminal
```

The game loop can be separated into two parts: the graphics rendering (the first two lines plus drawing onto the terminal) and the event handling (the last two lines)

The graphics rendering first clears the function (take note that the infinite loop passes very quickly). Then it asks the current Handler for what it wants to draw onto the console.

The event handling then takes all events happening on the terminal. When a key is pressed, or the mouse is moved on the terminal, an "event" containing data about key identity, mouse location, etc. is passed to the handler's method for handling events.

Depending on what happens due to the event the handler type may change (e.g. moving from the Menu Handler to the Game Handler after the player presses enter.).

Note: The handler's function for handling events passes the event to another method more appropriate for handling the event (e.g. ev_keydown(event) for a keypress vs ev_mousemotion(event) for a mouse movement).

3.3 Menu Handler

```
Set a variable inherent to the handler object "cursor_location" set
     to 0

Define Render as a function taking no inputs:
  Render a game screen consisting of a title, and a helpful tips box
       underneath.
  Render two options "Play" and "Quit" in the middle of the screen.
      Whenever the cursor_location is 0, highlight the
  Play option by changing its background color and its foreground
      color. Similarly, when the cursor_location is 1,
  highlight the Quit option.
  Return this handler (Do not change the current handler)
```

```
Define Keydown Event as a function taking an event as a parameter
    when event is a keydown event:
  When the keydown pressed is up:
    Decrement cursor_location if its value is not 0 or less
  When the keydown pressed is down:
    Increment cursor_location if its value is not 1 or more
  When they keydown pressed is ENTER:
    If the cursor_location is 0:
      Return an instance of the handler type "MainGameHandler" with
          this handler's console as a property.
      (Change the current handler)
    If the cursor_location is 1:
      Quit the game
  Return this handler (Do not change the current handler)
```

3.4 Main Game Handler

When the Main Game Handler is handling the game, it is expected that the game is thus active. This handler is extremely difficult to describe efficiently, and thus the algorithm described below has less accuracy corresponding to the actual code.

```
Set a variable inherent to the handler object "cursor_location" set
     to 0

Define a function to update the game without any inputs:
  If the game is in setup (no code generated):
    Clear the message log
    If the code length is not yet set:
      Add a prompt to the message log for the length.
    Else if the code repeat is not yet set:
      Add a prompt to the message log for the length.
    Else:
      Generate the code and end the setup.
      Add a helpful tip to the message log about navigation.
  Else:
    If there is no submitted guess:
```

```
  Keep the cursor location (the selected digit index) within the
       bounds of the code length
  Return nothing from the function (Ends the function
      prematurely)
Else: (When there is a submitted guess)
  If the guess has an invalid character:
    Add a message to the message log telling the user about the
        issue and asking for valid input
    Do nothing else
  Else if the guess is asking for a lifeline and lifelines can
      still be used:
    Tell the program the player is asking for a lifeline (
        telling rendering function to change input system, etc.)
  Else if the guess is asking for a lifeline and lifeliens are
      unavailabe:
    Add a message to the message log telling the user about the
        issue and asking for valid input
  Else if the player is asking for a lifeline (and the player
      has chosen a lifeline):
    Run the respective lifeline if the player can use the
        lifelines, else add a message to the log about the issue
      and do not continue
    Iterate the turn counter for the required number of times.
    Add "LIFELINE" to the list of past guesses for the lost
        turns.
    Restore normal functionality to the input
    Tell the program the player is no longer asking for a
        lifeline and normal behavior may resume
  Else if the guess is equal to the code:
    Tell the program the game is over.
    Tell the program the player has won.
    Add the guess to the list of past guesses
    Congratulate the player
  Else if ten turns have already passed and the player has not
      won:
    Tell the program the game is over.
    Add consolatory message to log.
    Add the guess to the list of past guesses.
  Else:
```

```
        Check the guess and code using the function described in the
             mastermind.py implementation
        Display red and white in their respective colors via the
            message log
        Add the guess to the list of past guesses
        Iterate turn once
      Set the current guess to nothing
      Set the guess being edited to its default ("1111...")

Define a Rendering function without inputs:
  Use a function to update the game
  Render the background (boxes with titles and past guesses as "0"
      and the up and down arrows for the input panel)
  Render the input panel by highlighting the selected digit in gray
      and coloring the digits in use white and displaying
    the appropriate chosen digits with proper background color (
        taken from constants.py)
  If for the allowed length of the guess, the ninth choice has been
      selected, overwrite the guess and display the
    characters of "LIFELINE"
  Render the past guesses by taking the list of past guesses and
      displaying them digit by digit from the top down
  Render the message log, line by line up to a certain range. When
      the length of the message log exceeds what can be
    shown by the window, display from the latest up to the furthest
        back the window can display.

Define Keydown Event as a function taking an event as a parameter
    when event is a keydown event:
  If the key pressed is ESC:
    If the player has already been warned about quitting or the game
         is over:
      Return the Menu Handler (Returns to menu)
    Warn the player via message log
  If the game is over:
    Prematurely end the function by returning self
  If the key pressed is RIGHT ARROW:
    Increment the cursor location by 1, keep it bound depending on
        what is being inputted (4 for code length, 1 for
      repeat, code length for guess)
```

```
If the key pressed is LEFT ARROW:
  Decrement the cursor location by 1, keep it bound depending on
      what is being inputted (0 for all)
If the key pressed is UP ARROW and the player is inputting a guess
    :
  Increment the digit of the guess at the cursor location by 1,
      keep the digit value bound from 1 to 9.
If the key pressed is DOWN ARROW and the player is inputting a
    guess:
  Decrement the digit of the guess at the cursor location by 1,
      keep the digit value bound from 1 to 9.
If the key pressed is ENTER:
  If the game is requesting for a code length:
    Add 4 to cursor location and use this value as code length
  If the game is requesting for repeat:
    If cursor location is 0, repeat is "yes"
    Else repeat is "no"
  If the game is requesting for a code:
    Submit the guess in the input panel and parse it as a string.
        Let the submitted guess be the elements of
    this from 0 to the designated code length.
  If the game is expecting a lifeline:
    Submit the single digit being edited as the guess string.
  Set cursor location to 0
Return self
```

Chapter 4

Resources

This script made use of the tcod python package and used development techniques described in Roguelike Tutorial Revised 2020.

Welcome to the python-tcod documentation! - python-tcod 13.3.0 documentation. (2009). Python-Tcod. Retrieved January 10, 2022, from https://python-tcod.readthedocs.io/en/latest/index.html

Welcome to the Roguelike Tutorial Revised. (2020, July 7). Roguelike Tutorials. http://rogueliketutorials.com/