COMPSCI 130: Assignment - Creative Extension

Leo Duncan
UPI: ldun202
ID: 657579290

14th October 2023

Important: This has to be run in in an IDE with an integrated terminal (e.g. VS Code, PyCharm), or straight in the terminal/command prompt. I've talked to Burkhard about this and he's said this should be fine.

Overview

For the creative extension of the "simplified version of solitaire" game, four improvements have been made. Namely, improved colours, improved text printing, the addition of a welcome message, and default shuffled cards. These changes improve the aesthetics, user friendliness, and playability of the game, overall improving the experience of the game.

1 Improved Colours

The first improvement to the game was to make it more visually appealing, specifically, by improving the colours outputted to the terminal.

A colour scheme of blue, pink, black was used for terminals with a light theme, and a scheme of blue, pink, white was used for terminals with a dark theme. An example game with both themes can be seen below.

(a) Light theme.

(b) Dark theme.

Figure 1: Improved colours.

The main colours of blue and pink were chosen as they work well together and are contrasting, significantly improving the aesthetics of the game. The more subtle blue colour is used for general informational text (i.e. the text where the game is communicating with the player), while the more alerting, contrasting pink colour is used to highlight key words. This helps the player to very quickly see what is happening and what they need to know.

Blue and pink shows up well on both light and dark themes. This allows the game to be played with on most IDEs/terminals, and means that the core character and identity of the game is maintained regardless of what terminal theme is used (as opposed to the confusion that may come with radically changing colours depending on the theme).

The colour used to dictate the information the player has more control over is black (for light themes) or white (for dark themes), which are generally the default terminal colours for the respective themes. Users of terminals are generally used to typing and controlling their terminal using the default colours, and so the decision to keep these colours consistant with what players are used to was a very intentional one. This consistency allows the player to immediately recognise what they have control over in the game

Overall, these improvements to the colours outputted by the terminal has significant positive effects. It gives the game its own visual character and identity (which is consistant across light and dark themes), it makes the game much more aesthetically pleasing, and it makes the game more playable by making it easier for players to immediately identify what information they need to know, and what they have control over.

This was difficult to achieve, as generally non-core libraries are required to make this work. However, by using ANSI escape codes, we are able to get the terminal to output the coloured text.[1]

The specific escape code used is `\033[38;5;{colour_ID}m{text}`. This uses the escape sequence `\033` (this is the escapce sequence for Octal but through experimentation it seems that the escape sequence for hexadecimal also works), followed by the code sequence `[38;5;{colour_ID}m{text}` (which is the ANSI escape code sequence for 256 text foreground color).

In essence, this tells the terminal to output our text with this applied formatting.

To make this easier to implement, the following function was created.

```
# A function that makes it much easier to print in whatever
# colour is desired.
def colours(a_colour, text):
    return(f"\033[38;5;{a_colour}m{text}")
    # Applies the new colour using an ANSI escape code
```

[1]More information about ANSI escape code sequences here:
https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

It takes a number from 0 - 255 (which corresponds to a colour), and the text that you want converted to that colour. It returns the text formatted to the desired colour. Through experimentation, the best blue and pink colours were 74 and 199 respectively.

As ANSI escape codes are specific for formatting in terminals, this unfortunately does not work with the IDLE Shell (as it does not emulate a terminal), hence the game does need to be run in some sort of terminal.

2 Improved Text Printing

The second improvement to the game was to improve the way the game printed its text. Specifically, to make it more reminiscent of classic text-based games such as *The Hitchhiker's Guide to the Galaxy* or *Zork*.

This is done by printing one character at a time, emulating typing. This makes it feel much more natural and easy to read, which is a stark contrast to the unnatural and confronting way the original code printed line-by-line. This is why these classic text-based games choose this method of character-by-character typing.

Additionally, to further increase readability, the program briefly pauses typing before and after the key words coloured in pink (discussed in section 1). This brings more attention to these key words, making it even easier for the player to read and understand what they need to know.

In order to achieve this much more aesthetic and readable printing style, a new function, `print_slowly()`, was created.

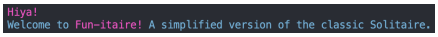
```
import time

# A function that makes it much easier to perform the aesthetic
# character-by-character printing.
def print_slowly(text, the_end='\n'):
    for char in text[:-1]:
        print(char, end='')
        time.sleep(0.03)
    print(text[-1], end=the_end)
    # similar to "end" in the print() function.
```

This function uses the built in library `time` to wait a certain amound of time between each character. After some experimentation, 0.03 seemed to be the best interval between characters. This function was used instead of `print()` whenever the program printed something to the screen.

3 Welcome Message

The third improvement to the game is the addition of a welcome message. This is a nice little touch that makes the game feel much more polished and friendly to a player.

(a) Welcome message (light theme).



(b) Welcome message (dark theme).

Figure 2: Welcome message.

As a part of this, the game was also given a name. Previously the game was simply "simplified version of solitaire" which feels very clinical and assessment-y. Giving it a much more inviting name like `Fun-itaire!` makes it feel much more like a fun finished game.

4 Shuffled Cards

The final improvement to the game is starting the player with shuffled cards. In the original game, the player had to manually enter a list of their cards into the code. Now there is a default set, `[1,2,3,4,5]` which is then randomly shuffled using `random.shuffle()`. Through testing, a game with 5 cards felt like the right length to be both a challenge while not taking too long. This set can be easily changed in the code if so desired. Randomly shuffling the cards whenever the code is run means that every game is unique.

Overall, this improvement improves user-friendliness (i.e. not being required to manually change the code) and playability.