SOFE 3980U
Software Quality
Winter 2022

Assignment-2: Static and Dynamic Analysis (12.5 points)

Owais Quadri
100697281

Software Quality Assignment 2

Explanation of the static analysis technique used with snapshots of the code how it is done with the results

Static analysis is when code is analysed preceding the execution of the aforementioned code. Program slicing is performed by isolating the changes made on a certain variable in code to make testing and debugging easier. Static slicing considers all of the possible executions and will therefore produce larger slices. Slices can be considered subsets of the program that relate to a certain variable.
Static slicing for the variable called “color_order”:

```python
#static slice for the variable 'color_order' :
MAX_GUESSES = 6
WORD_LEN = 6
class Wordify:
    word = "wordle"
    def __init__(self):
        pass
    def render(self, wordify):
        # portray words wrt correct word
        color_order=""
        for i in range(WORD_LEN):
            if wordify[i] == self.word[i]:
                # correct
                color_order+="green"
            elif wordify[i] in self.word:
                # incorrect spot
                color_order+="yellow"
            else:
                # incorrect letter
                color_order+="red"
        return (color_order)
    def guess(self, num, guessed_word=None):
        if num != MAX_GUESSES:
            if guessed_word is None:
                print("Please guess a 6 letter word: " +
                      str(num+1)+"/"+str(MAX_GUESSES))
                guessed_word = input()
            guessed_word = guessed_word.lower()
```

```python
            len_guessed = len(guessed_word)
            if len_guessed != WORD_LEN:
                return self.guess(num)
            else:
                print(self.render(guessed_word))
                if guessed_word in self.word:
                    # win
                    return (guessed_word)
                else:
                    return self.guess(num+1)
    def start(self):
        print(self.guess(0))
if __name__ == '__main__':
    game = Wordify()
    game.start()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
Please guess a 6 letter word: 1/6
asdasd
redredyellowredredyellow
Please guess a 6 letter word: 2/6
worlde
greengreengreenyellowyellowgreen
Please guess a 6 letter word: 3/6
wordle
greengreengreengreengreengreen
wordle
```

While creating this slice, I considered each possibility where the ‘color_order’ variable could change and I reduced the code by ~30 lines because I could remove lines of code where this variable could not be changed in any execution.

Explanation of the dynamic analysis technique used with snapshots of the code how it is done with the results

The reason that this technique is considered ‘dynamic’ is because the values that are considered are only during the execution of the program. This way, we can narrow down the slice into a more precise selection for a particular execution of the code. Dynamic slices can be considered subsets of the static slices. In static, we consider every execution and in dynamic we only consider one.

Dynamic slicing for “color_order” when the word is 'wordle' and it is solved in 1 guess:

```python
#dynamic slice for the variable 'color_order' : when the word is 'wordle' and it is solved in 1 guess
class Wordify:
    word = "wordle"
    def __init__(self):
        pass
    def render(self, wordify):
        # portray words wrt correct word
        color_order=""
        for i in range(6):
            if wordify[i] == self.word[i]:
                # correct
                color_order+="green"
        return (color_order)
    def guess(self, guessed_word=None):
        print(self.render(guessed_word))
        if guessed_word in self.word:
            # win
            return (guessed_word)
    def start(self):
        print(self.guess("wordle"))
if __name__ == '__main__':
    game = Wordify()
    game.start()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
owais@DESKTOP-1MICDPL MINGW64 ~/Documents/QUAL_A1 (main)
$ python wordify_dynamicSlice.py
greengreengreengreengreengreen
wordle
```

This slice of code is considerably shorter due to the fact that only one execution was considered and simulated. The input checks were not needed since the input was already defined and other inclusions such as other colours being added to the color_order could be omitted. In the end, the program would go from 79 lines in total, to 23 lines in a dynamic slice.

Challenges and lessons learned

The challenges that I faced was refactoring some of the code to not include the tuple in the return statement, as well as removing useless if statements. This was all corrected after reading more about how slicing works and how tracing the code can help in this regard. One lesson I learned was that creating a program that is well broken down into functions easily allowed me to omit most of what I needed to isolate the variable in the slice.