Janahan Ravichandran (100701645)
February 11, 2021
SOFE 3980 – Software Quality

Assignment 2 – Static & Dynamic Analysis

a.) To perform static analysis of the code, forward manual slicing was used. Using this method, a variable X, was chosen and all statements involving, or utilizing X are sliced from the program. To perform this a python file was used to read the python code as a text file line by line. For each line it is parsed to look for the specific variable name being sliced. The variable name is passed as a command line argument to the when the program is running from the terminal.

Figure 1: Code used to perform static analysis.

```
#!/usr/bin/python

import sys


def parse(statement, cur_line):
    if statement in cur_line:
        return True

    return False


file = open("Source.txt", "r")
line_number = 1

for line in file:
    if parse(sys.argv[1], line):
        print('line ', line_number, ':', line.strip())

    line_number += 1

file.close()
```

The code reads the text file line by line, and calls parse() function using the current line of the file, as well as the variable name the slicing is according to. If the variable name is found in that line the line is then printed to the terminal, along with the line number.

Figure 2: Static analysis for snack_x variable.

```
nt 2\Static Slicing>python main.py snack_x
line  29 : snack_x, snack_y = random.randrange(SAFE_WIDTH_MIN, SAFE_WIDTH_MAX), \
line  32 : if snack_x == player_pos[0] and snack_y == player_pos[1]:
line  37 : self.pos = (snack_x, snack_y)
```

Figure 3: Static analysis for Snake object.

```
nt 2\Static Slicing>python main.py Snake
line  73 : class Snake(object):
line  222 : player = Snake(WIDTH // 2, HEIGHT // 2)
```

Figure 4: Static analysis for screen variable.

```
nt 2\Static Slicing>python main.py screen
line  14 : screen = pygame.display.set_mode((WIDTH, HEIGHT))
line  40 : pygame.draw.rect(screen, LIME_GREEN, (self.pos[0], self.pos[1], 40, 40))
line  70 : pygame.draw.rect(screen, RED, (obs[0], obs[1], 40, 40))
line  131 : pygame.draw.rect(screen, WHITE, (self.x, self.y, 20, 20))
line  134 : def update_screen(snake, snack, obstacle, score):
line  135 : screen.fill(BLACK)
line  144 : screen.fill(BLACK)
line  147 : screen.blit(game_over_text, (235, 200))
line  148 : screen.blit(score_text, (230, 300))
line  163 : screen.blit(text, (30, 30))
line  214 : update_screen(snake, snack, obstacle, score)
```

Figure 5: Static analysis for clock variable.

```
nt 2\Static Slicing>python main.py clock
line  169 : clock = pygame.time.Clock()
line  215 : clock.tick(30)
```

Figure 6: Static analysis for HEIGHT variable.

```
c Slicing>python main.py HEIGHT
line  6 : HEIGHT, WIDTH = 500, 700
line  7 : SAFE_HEIGHT_MIN, SAFE_WIDTH_MIN = 45, 45
line  8 : SAFE_HEIGHT_MAX, SAFE_WIDTH_MAX = HEIGHT - SAFE_HEIGHT_MIN, WIDTH - SAFE_WIDTH_MIN
line  14 : screen = pygame.display.set_mode((WIDTH, HEIGHT))
line  30 : random.randrange(SAFE_HEIGHT_MIN, SAFE_HEIGHT_MAX)
line  57 : random.randrange(SAFE_HEIGHT_MIN, SAFE_HEIGHT_MAX)
line  115 : if self.y < SAFE_HEIGHT_MIN:
line  116 : self.y = HEIGHT
line  117 : if self.y > HEIGHT:
line  118 : self.y = SAFE_HEIGHT_MIN
line  125 : if self.y < SAFE_HEIGHT_MIN:
line  126 : self.y = HEIGHT
line  127 : if self.y > HEIGHT:
line  128 : self.y = SAFE_HEIGHT_MIN
line  222 : player = Snake(WIDTH // 2, HEIGHT // 2)
```

b.) In order to perform dynamic analysis on the code an instrumentation method was used on the source code. With this method measurements probes such as time modules are used to examine how the code operates at runtime. To perform this the timeit module was used to keep record of time spent per method call. Along with the module, multiple lists were initialized to keep track and organize the information. After the game is finished executing all data collected from the analysis is outputted via console.

Figure 7: List for holding data collected from instrumentation.

```
# list for counting number of calls
num_calls = [0] * 5
# list to keep track of time elasped per function
time_list = [0] * 5
# name of each function
names = ['generate_snack', 'detect_touch', 'update_screen', 'end_game', 'generate_obstacle']
headings = ['Method Name', 'Number of Calls', 'Time Per Call']
```

Figure 8: Code to count number of calls and start the timer to for the function call.

```
num_calls[1] += 1
t_start = timeit.default_timer()
```

Figure 9: Code to end timer and update the amount of time taken by that specific method.

```
t_end = timeit.default_timer()
time_list[1] = t_end - t_start
```

Figure 10: Code to output data collected from analysis.

```
print('\n' + '\t\t\t'.join(headings))
for i in range(5):
    print(str(names[i]) + '\t\t\t' + str(num_calls[i]) + '\t\t\t\t' + str(time_list[i]))
```

Figure 11: Output from analysis

```
Method Name         Number of Calls         Time Per Call
generate_snack          6                       9.599999998499698e-06
detect_touch            483                     0
update_screen           482                     0.000764000000000209
end_game                1                       2.500949300000002
generate_obstacle       10                      3.7999999999982492e-06
```