Technical Report for the Computer Game – Snake

Uruj Sarwar

Table of Contents

1 Introduction

This report highlights the different aspects that were associated in developing the game 'Snake' in Python version 2.7. The game is basically based on the functionality of the game 'Nibbles' that was presented in the 1970s and further being implemented for PC. With respect to the logic of Nibbles, the following game Snake is also directed towards the functions of the snake eating food items and the snake body increases at every food item. Apple has been selected as the food item in this context. The logic also involves the movement of the snake in every direction and it touches the walls and its own body, the game is terminated.

2 Snake – Purpose of the Project and Game Requirements

The game Snake comprises of certain requirements that should be analysed in order to process the Python program. The GUI platform of Python has been introduced in this project with respect to the outlook of the game, snake movement, and snake collision with the food item and walls, and the game terminating by displaying the score and names of the players in the leaderboard. The GUI aspect involves the Canvas platform that allows Python to implement graphics functions.

Apart from the GUI platform defined above, program has been centered towards different factors. This involves the directions in which the snake can move, creating objects on the Tkinter Canvas, and locating the apple randomly on the Canvas. The major purpose of the program involves how the keyboard functions can be applied in order for the game to process by the end user.

2.1 Identifying Game Objects on the Canvas

The requirements for the game initially involve the game objects to be present on the Canvas. This refers to the appearance of the developed images on the user screen when the game is being processed by the user. This involved creating the objects on the Canvas.

2.2 Snake Movement

The movement of the snake had been an essential requirement as well in terms of being controlled by the user keyboard. This factor is related to the four snake movements, left, right, up and down being controlled by the direction keys present on the user keyboard.

2.3 Randomized Apple Location

Along with the operation of creating objects on the Canvas, the operation of locating the apple in a randomized manner had been an important requirement for the game. The main aim of this aspect was simply set the apple image at a random point on the Canvas simultaneously when the apple is eaten by the snake and disappears from the Canvas.

2.4 Finding Objects on the Canvas

This requirement involves the head encountering the head of the snake and resulting in creating a new connection, thus increasing the snake body. For this, the collision of the coordinates for the two images is necessary. This would be in line with the randomized apple location because once the apple encounters the head, the existing apple should disappear and should appear somewhere else on the Canvas.

2.5 Snake Colliding

This function of the game requires the program checking that whether the snake has either collided with its own body or with the walls. This in reference to the operation of game being over. In this condition if the snake collides with its own body or with the wall, the game is over.

2.6 Timer and Delay

The timer and delay requirements are based on controlling the speed at which the snake would move in the game. The timer function should be controlling the actions that should be processed after a specific amount of time has been passed. This means that the timer should keep on going with respect to having a delay in between.

2.7 Game Over

As mentioned in the snake colliding requirement that when the snake collides with its own body or the walls, the game should be over. However, when the game is over it needs to depict the scores of all the players and must ask for the player name for the current game score. The game over function refers to ending the all the operations and focusing on the showing the users the scores on a leaderboard. This requirement involves in deleting all the objects present on the Canvas and showing the scores in the end.

2.8 Game Scoring and Player Name Input

The game scoring should be based on the counts of how many apples have been eaten by the snake. The game should ask for the name of the player once the game is over and the names and scores of the previous players should be listed. The list should be in a descending order from highest score to the lowest score.

2.9 Locating Bad Apple and Game Termination

Another function of bad apple should be introduced that when the snake interacts with it, the game should be over and player names of scores should be displayed in a sorted manner from highest to lowest as mentioned above. When the snake encounters the bad apple, the bad apple should be deleted. During the game, the bad apple would appear on the Canvas in a randomized manner.

3 Methodology for the Game – Python Functions for Game Requirements

The game Snake has the following outlook. It can be observed that as the snake eats the apple, the body increases. Initially, three images had been developed using the Adobe Photoshop CC for the appearance of the snake and food item. This includes the head, the body and the apple. The size of every part of the snake is 10 pixels. The bad apple image has also been introduced in the game.

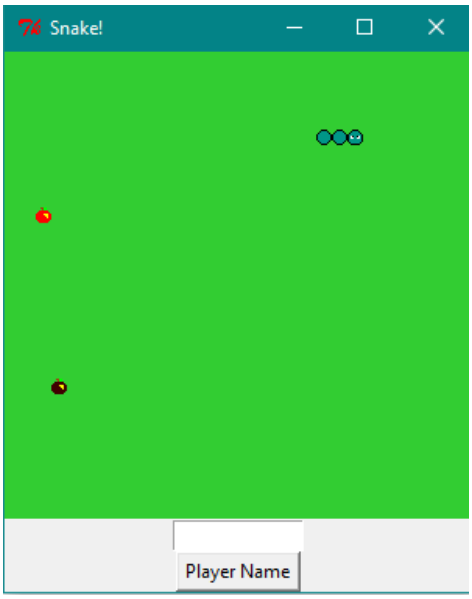

Figure 1 The Snake Game Outlook

The program has utilised the platform of GUI along with the normal functionality of the game involving the snake movement, eating, entering player name and showing the highest scores. Python comprises of its own GUI package known as Tkinter. Along with this, other graphics functions have been listed below, which had been used in developing the program,

- Tkinter
- PIL
- Clmage.py
- Canvas

3.1 Tkinter

Tkinter is one the most commonly used GUI package based on its easiness to use. The platform is also properly documented. It is based on the Tk GUI toolkit interface. This contributes in presenting Tk widgets within Python. Within the program, initially the Tkinter had been imported along with the other modules. Tkinter can first be checked within Python that whether the program comprises of the setup or not. If not, the Tkinter should be then installed.

3.2 PIL

PIL refers to the Python Imaging Library that presents Python with graphics options. This involves in locating, opening, saving, and processing several types of image file formats. PIL contributes in providing the ability of image processing to be carried out by Python. The PIL had been imported initially in the program for the game Snake. In order for the game to process the developed images on Photoshop CC as identified above, PIL helps in adding a library that could support image file formats that can be easily processed by the Python interpreter. The library has to be first checked by the user that whether Python has it or not. If not, then the PIL setup should be installed.

3.3 cImage.py

The image library for Python is simply based on the Python modules involving PIL. However, in order to install the image library of Python, it is necessary to firstly install the module cImage.py. it is the main image processing module, after which user friendly functions in terms of image processing are added by PIL.

3.4 Tkinter Canvas

The Canvas in Tkinter is a widget that allows different graphics operations to be implemented in Python. This involves making graphs and depicting plots. This platform has contributed in providing the X and Y coordinates for the program. This includes giving the background for the game and the body of the snake through the aspect of image processing from cImage and PIL. In the Python program for snake, after when all the defined modules had been identified at the beginning of the program, the X and Y coordinates were then introduced. When the Python program is run, the game is initiated in an instant manner. At the point when the game ends, 'Ooops, game is over' appears on the screen along with the score board.

3.5 Snake Program Structure

Within the above defined modules, independent subprograms had been further created. These were based on the requirements of the game. The program structure has been defined below in terms of how independent subprograms and procedures had been used,

1. Importing required modules
2. Game variables
3. X and Y coordinates for Snake Dots
4. Class Board(Canvas):
    - def __init__(self, parent)
    - def initGame(self)
    - def createObjects(self)
    - def checkApple(self)
    - def doMove(self)
    - def checkCollisions(self)
    - def locateApple(self)
    - def locate_bad_apple(self)
    - def delete_bad_apple(self)
    - def onKeyPressed(self, e)
    - def onTimer(self)
    - def gameOver(self)
5. class Snake(Frame)
    - def __init__(self, parent)
6. def main()
    - def retrieve_input()

4 Subprograms for Snake

4.1 Importing required Modules

Initially in the main program, several modules have been imported in Python.

- **import sys:** this refers to the system specific parameters and functions. The sys module involves the availability of variables that are utilised by the Python interpreter and the operations, which perform with the interpreter.
- **import random:** in order for the program to function random numbers within a specified range, or for picking random items from a list, the random module is used. For the Snake game, this refers to the scoring process, displaying the scores and names of players, and the randomized arrangement of the apple in the Canvas.
- **from PIL import Image, ImageTk:** this helps in importing the Pillow module that contributes in the loading of the images and reading them. This presents a platform for graphics, so that the concerned images could be used and arranged in the required manners. The function of ImageTK refers to the Tkinter processing the images.
- **from Tkinter import Tk, Frame, Canvas, ALL, NW, Text, Button:** along with the importing of the module Tkinter, these functions are the specified operations that have been imported from Tkinter in order to operate the program. The Frame refers to the Tkinter frame widget that allows a rectangular region as an output from the program on the screen of the user. The Tkinter Canvas is used for presenting graphics operations for Tkinter. Tkinter ALL is based on having most of the Tkinter modules in Python. The text option imports text writing module and NW is the direction Northwest where the text is placed. For Tkinter Button, it refers to text or images for the program.

4.2 Game variables

After importing all the modules, the game variables involving the measurements for the frame of the rectangular region for the game as an output, the time delay for the movement of the snake, the pixel size for the appearance of the snake had been defined through different variables.

4.3 X and Y coordinates for Snake Dots

The pixel size has been referred by the variable DOT_SIZE for which the x and y coordinates have been identified as well, in order for the snake dots to connect on the Canvas of the program.

4.4 Class Board(Canvas):

The above defined variables and x and y coordinates have been addressed in this subprogram. This initially creates the outlook window of the game by utilizing the above depicted variables. Within this subprogram, several procedures have been developed based on different operations.

- **def __init__(self, parent):** within this procedure, the Tkinter class fame is being called, which means that frame instances could be utilised as frames and depict the frame output. This restricts to a single frame for the game. The width and height variables have been addressed here, along with the colour of the game background. self.parent = parent calls the parent class. This is in reference to the Canvas method.
- **def initGame(self):** this procedure contributes in initializing the variables, loading the depicted images for the game, the head, body, and apple, and runs the timer function. It has assigned values of x and y for the apple and the bad apple, further assigning an initial value of 0 to score of the apples eaten through the following variables

```
self.apple_score = 0
self.apple_x = 100
self.apple_y = 190
self.bad_apple_x = 100
self.bad_apple_y = 190
```

The following lines read the three images within the Tkinter

```
self.idot = Image.open("dot.png")
self.dot = ImageTk.PhotoImage(self.idot)
self.ihead = Image.open("head.png")
self.head = ImageTk.PhotoImage(self.ihead)
self.iapple = Image.open("apple.png")
self.apple = ImageTk.PhotoImage(self.iapple)
```

```
self.ibad_apple = Image.open("badapple.png")
self.bad_apple = ImageTk.PhotoImage(self.ibad_apple)
```

In order for the snake to move when the program has been ran, the following syntax within the procedure of def initGame(self) has been set with True and False conditions that the user can use the direction keys on the keyboard for moving the snake on the Canvas.

```
self.left = False

self.right = True

self.up = False

self.down = False

self.inGame = True
```

The procedures for creating the targeted objects, locating the apple and the bad apple on the Canvas randomly, and associating the keyboard keys functionality to the game have been called under def initGame(self).

- **def createObjects(self):** this procedure has been defined in order to create the objects for program. Based on the functions imported from Tkinter in the program, the objects are defined by utisling the x and y values that were initially defined, along with selection of the image and its position being NW. This goes for all the four images of the apple, bad apple, head and body that is referred by ‘do’ in the program as identified below,

```
self.create_image(self.apple_x, self.apple_y, image=self.apple, anchor=NW, tag="apple")
self.create_image(self.bad_apple_x, self.bad_apple_y, image=self.bad_apple, anchor=NW, tag="bad_apple")
self.create_image(50, 50, image=self.head, anchor=NW, tag="head")
self.create_image(30, 50, image=self.dot, anchor=NW, tag="dot")
self.create_image(40, 50, image=self.dot, anchor=NW, tag="dot")
```

The structure of the above commands comprises of x and y coordinates in the beginning, the image option for the appearance of the image on the Canvas, and the anchor parameter contributes in highlighting the position of the image. In this case, NW refers to North West that is at the upper left point of the Canvas. The tag parameter is used to identify the objects on the Canvas. One tag can be used for several objects on the Canvas.

- **def checkApple(self):** this method allows us to find out if the snake ate an apple object. If so, add one snake connection and run locateApple. With the defined tags in this method, it allows to find an object on the Canvas through the following syntax,

```
apple = self.find_withtag("apple")
bad_apple = self.find_withtag("bad_apple")
head = self.find_withtag("head")
```

The procedure further returns the points of the boundaries of the object and with the use of the find_overlapping () method, collision of objects with these coordinates is depicted as per the following syntax,

```
x1, y1, x2, y2 = self.bbox(head)

overlap = self.find_overlapping(x1, y1, x2, y2)
```

If an apple encounters a head, a new connection is created for the snake on the coordinates of the apple. The following for loop has been implemented in order to locate the apple object and add +1 to the score per apple object. This contributes in add +1 to the score at the end when the leaderboard is displayed when the game is over. Along with the scoring, for creating the apple object and bad apple object randomly, the range has been defined. Within this method, the locateApple() is run that removes the old apple from the Canvas and creates a new one at a random point.

```
for ovr in overlap:
    if apple[0] == ovr:
      x, y = self.coords(apple)
      self.create_image(x, y, image=self.dot, anchor=NW, tag="dot")
      self.locateApple()
      self.apple_score += 1
```

```
            for index in xrange(5,1005,5):
                if self.apple_score == index:
                    self.locate_bad_apple()


        for bad in xrange(0,len(bad_apple)):
            if bad_apple[bad] == ovr:
                self.inGame = False


        for second_index in xrange(5,1005,5):
            if self.apple_score == second_index:
                for bad in xrange(0,len(bad_apple)):
                    if apple[0] == ovr:
                        self.delete_bad_apple()
                        self.locate_bad_apple()
```

- **def doMove(self):** this method turns on the game key algorithms and all body connections of the snake move behind the head as one single chain. This can be seen in the simple addition operation in the following syntax after finding the dot and head,

```
dots = self.find_withtag("dot")

head = self.find_withtag("head")

items = dots + head
```

For the following syntax, it allows the connections to move in the same chain, as it compares to the length of the above function of items and that the body, the head and the added body objects after eating the apple object are in the same line.

```
z = 0

while z < len(items) - 1:

        c1 = self.coords(items[z])

        c2 = self.coords(items[z + 1])
```

```
                self.move(items[z], c2[0] - c1[0], c2[1] - c1[1])
                z += 1
```

The following syntax is for the movement of the snake's head to the left.

```
            if self.left:
                self.move(head, -DOT_SIZE, 0)
```

The following syntax is for the movement of the snake's head to the right.

```
            if self.right:
                self.move(head, DOT_SIZE, 0)
```

The following syntax is for the movement of the snake's head up.

```
            if self.up:
                self.move(head, 0, -DOT_SIZE)
```

The following syntax is for the movement of the snake's head down.

```
            if self.down:
                self.move(head, 0, DOT_SIZE)
```

- **def checkCollisions(self):** this method determines when the snake strikes itself or one of the walls. Initially it reads the two images and then variables have been assigned regarding to the position of the snake and the overlapping phenomenon. Self in the entire program refers to the new variables that are created and its methods are called.

```
            dots = self.find_withtag("dot")
            head = self.find_withtag("head")
            x1, y1, x2, y2 = self.bbox(head)
            overlap = self.find_overlapping(x1, y1, x2, y2)
```

The method further defines the positions in which the snake could hit itself and hit the walls. The procedure refers to defining the variables that were addfressed in the inGame procedure above.

Snake hits own body.

```
for dot in dots:
    for over in overlap:
        if over == dot:
            self.inGame = False
```

Snake hits the walls.

```
if x1 < 0:
    self.inGame = False
if x1 > WIDTH - DOT_SIZE:
    self.inGame = False
if y1 < 0:
    self.inGame = False
if y1 > HEIGHT - DOT_SIZE:
    self.inGame = False
```

- **def locateApple(self):** this method puts a new apple at a random point on the canvas and removes the old one. The following syntax locates and delete the eaten apple,

```
apple = self.find_withtag("apple")
self.delete(apple[0])
```

The following syntax sets the random X and Y coordinates for new apple,

```
r = random.randint(0, RAND_POS)
self.apple_x = r * DOT_SIZE
r = random.randint(0, RAND_POS)
self.apple_y = r * DOT_SIZE
```

```
self.create_image(self.apple_x, self.apple_y, anchor=NW, image=self.apple, tag="apple")
```

- **def locate_bad_apple(self):** in a similar manner, this method locates the eaten apple. The following syntax sets the random X and Y coordinates for new apple,

```
r = random.randint(0, RAND_POS)
self.bad_apple_x = r * DOT_SIZE
r = random.randint(0, RAND_POS)
self.bad_apple_y = r * DOT_SIZE
self.create_image(self.bad_apple_x, self.bad_apple_y, anchor=NW, image=self.bad_apple, tag="bad_apple")
```

- **def delete_bad_apple(self):** this method deletes the eaten apple.

```
bad_apple = self.find_withtag("bad_apple")
self.delete(bad_apple[0])
```

- **def onKeyPressed(self, e):** this procedure detects the keys that are pressed on the keyboard of the player. It defines all the fours positions of the snake, right, left, up, and down. This method assigns the directions keys on the keyboard functions by true and false conditions. In return, when the user presses the direction keys, the snake follows in the directed direction. The keysym command here refers to usual functions of the direction keys on a keyboard for the user. This indicates how a user would himself understand the directions just by looking at the four direction keys on his keyboard.
- **def onTimer(self):** this method involves the starting of the timer method after every delay. The timer is based on the after() method, which only starts the method after DELAY once. To restart the timer, the onTimer() method is run. This involves analysing the previous procedures and their current statuses. However, it that is not the case, then the program goes towards depicting the leaderboard.

```
if self.inGame:
    self.checkCollisions()
    self.checkApple()
```

```
            self.doMove()
            self.after(DELAY, self.onTimer)
        else:
            self.sorted_leaderboard = open("leaderboard.txt","r")
            self.content = self.sorted_leaderboard.readlines()
            self.sorted_leaderboard.close()
            for index in xrange(len(self.content)):
                self.content[index] = self.content[index].replace("\n","")
                self.content[index]= self.content[index].split(":")
```

Within the same method, the scores are sorted and the notepad file for the scoring is updated every time so that the program could read the names and scores at the end. This can be observed in the def sort_int(x) procedure that has been called under def onTimer(self).

- **def gameOver(self):** this deletes all objects on the canvas and shows the final notice of 'Ooops, game is over!'. It further specifies that when the game is over, the output screen should identify the leaderboard and indicate current scores with names.
- **class Snake(Frame):** this defines the text in the header of the window and the header itself.

```
def __init__(self, parent):
    Frame.__init__(self, parent)
    parent.title('Snake!')
    self.board = Board(parent)
    self.pack()
```

- def main(): this call the Tkinter program and comprises of a an input procedure within def retrieve_input(). This refers to creating a text box when the game is over that initially reads the leaderboard.txt and then writes the input into it and saves it, so that the player names could appear every time the game is being played. It depicts the structure and position of text box as well.

```
def retrieve_input():
    inputValue=textBox.get("1.0","end-1c")
    name_leaderboard = open("leaderboard.txt","a")
    name_leaderboard.write(",Player: " + inputValue)
    name_leaderboard.close()
textBox=Text(root, height=1, width=10)
textBox.pack()
buttonCommit=Button(root, height=1, width=10, text="Player Name",
command=lambda: retrieve_input())
buttonCommit.pack()
```

Further, root being the object is identified that comprises of the mainloop method.

```
root.mainloop()
```

The following syntax defines the main entry block of the program,

```
if __name__ == '__main__':
    main()
```

5 Conclusion

This report has been focused on identifying the requirements and the methods that have been utilised in the development of the game Snake. Initially the report has introduced background on the game and its platform, which is then followed by the use of GUI in this context. The report has indicated the parameters that have been used in developing the game and how they were applied in program on Python within its graphics environment.