INDEX

AIM:

Write a Python program to control a snake to move and collect food

ABSTRACT

Snake game is a simple console application without graphics. In this project, you can play the popular “Snake Game” just like you played it elsewhere. You must use the up, down, right or left arrows to move the snake. Food is provided at the several co-ordinates of the screen for the snake to eat.

Every time the snake eats the food, its length will be increased by one element along with the score. I have used python as Programming language for writing the code for the project and VsCode for writing the programs. Operating system used Windows 10.

INTRODUCTION

Playing games is fun and exciting, it gives us relief from stress and helps us unwind from our stressful work. Many of us spend our vacant time or others use most of their time playing and exploring new games. Today with the rapid development of technology we have, games that are rising up together with it. Nowadays with technology we have many games that are being developed for computers, most specifically for windows. With the high technology equipped with these computers, computer games become robust and attract many people to buy or have this gadget for them to experience what's inside it which makes it a trend for the new generation of gadget.

Snake game is a computer action game; whose goal is to control a snake to move and collect food on a map. It has been around since the earliest days of home computing and has re-emerged in recent years on mobile phones. It isn't the world's greatest game, but it does give you an idea of what you can achieve with a simple python program, and perhaps the basis by which to extend the principles and create more interesting games on your own

To move the snake, use 'W' for up, 'S' for down, 'A' for left and 'd' for right.

The aim of the game is to collect the dots (food) and avoid the obstacles (crosses borders). As you collect the food, the snake gets longer. The score also increases. There is no concept of life. Once You hit an obstacle, that's it, game over.

LITERATURE SURVEY

The history of the Snake game goes back to the 1970's, the concept originated in the 1976 arcade game Blockade, and its simplicity has led to many implementations. However, it was the1980'swhen the game took on the look that we will be using. It was sold under numerous names and on many platforms but probably gained widespread recognition when it was shipped as standard on Nokia mobile phones in the late 1990's. The first published Nokia, for monochrome phones. It was programmed in1997 by Taneli Armanto of Nokia and introduced on the Nokia6110.The game involves controlling a single block or snakehead by turning only left or right by ninety degrees until you manage to eat a block. When you get the block, the Snake grows an extra block or body segment. If, or rather when, he snakes bumps into the edge of the screen or accidentally eats himself the game is over. The more blocks the snake eats the higher the score.

SYSTEM DESIGN

To create a Snake game that allows users to control the movement of a snake on a screen, to get points for eating food and avoiding running into the walls or the growing tail of the snake itself. In this problem, we want to write a game where a graphical representation of a snake moves across the screen.

When it encounters a piece of food, the snake grows longer, and we gain a point. If it hits the wall we die. To write this program we are going to need:

● A way of representing the snake

● A way of representing the food

● A way to display the score,

● a way for our instructions to reach the snake,

● and a way to know when we've run into something and died

Our system is going to involve working with both hardware and software, and so we will need to understand what we have available in hardware that can assist us. If we build our software so that the snake is controlled by directional arrows on the keyboard.

Now that we understand how our hardware will work in the design of our system, let's move on to starting the design of our software system.

SYSTEM AND HARDWARE REQUIREMENTS

SYSTEM REQUIREMENTS

OPERATING SYSTEM: WINDOWS XP or Higher

IDE: VisualStudio.NET 2005/2008

FRONT END: WINDOWS

LANGUAGE: Python

HARDWARE REQUIREMENTS

Intel P4 1.5GHz or above

512MB RAM

80GB HDD Minimum

FUNCTIONAL REQUIREMENTS

Here are the requirements (functional requirements) for how the snake moves.

1. The snake must appear to move around the screen.
2. The snake must turn in response to user input.
3. The snake will increase in length if it eats food.
4. The snake will die if it runs into the walls.
5. The snake never stops moving.

NON-FUNCTIONAL REQUIREMENTS

The primary features of IT projects involve implementing an application, a piece of infrastructure, a patch. In this specific context functional requirements tells us about what project does when interact, whereas nonfunctional requirements describe about the bounds of performance should be

I. Robustness:

Robustness is nothing but its ability to tolerate the effects of a system functional body. And it can also be defined as its systems ability that it can withstand to change without transforming its initial stable configuration

II. Reliability:

The system is trustworthy, and it is consistently good in performance. It can also be stated as the system performs the function without any failure under certain conditions and specified period of time.

III. Availability:

The system is available 24*7. Availability and Reliability are directly proportional as reliability increase availability also increases. The user can have access to the system all the time.

IV. Reusability:

The system can be used any number of times by the specific user. And the reusability is consistent, adaptable and stable.

V. Effectiveness:

The algorithm is capable of producing desired result or it has the ability to provide better results.

WORKING ALGORITHM

Let's look at how a program to run the whole game might look:

1. Draw the playing area with bounding rectangle, set the counter to zero and display it.
2. Draw the snake in a starting position.
3. Draw the food in the starting location.
4. On user input, change snake direction.
5. Move the snake one move
6. If the snake is over food, eat it, increase the score, grow, move the food,
7. else if the snake is over in a wall, die.
8. Go back to 4.
9. Until the snake dies.

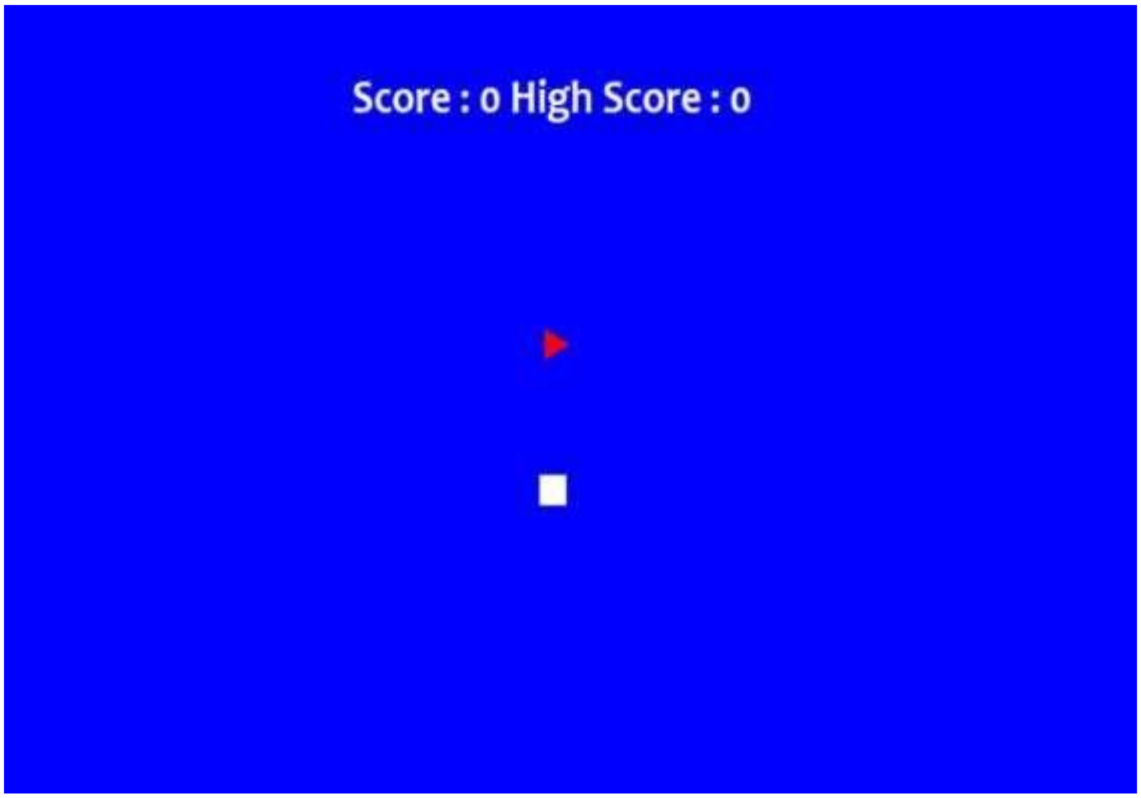
SOURCE CODE

```
import turtle
import time
import random

delay = 0.1
score = 0
high_score = 0
# Creating a window screen
wn = turtle.Screen()
wn.title("Snake Game")
wn.bgcolor("blue")

# the width and height can be put as user's choice
wn.setup(width=600, height=600)
wn.tracer(0)


# head of the snake
head = turtle.Turtle()
head.shape("square")
head.color("white")
head.penup()
head.goto(0, 0)
head.direction = "Stop"


# food in the game
food = turtle.Turtle()
colors = random.choice(['red', 'green', 'black'])
shapes = random.choice(['square', 'triangle', 'circle'])
food.speed(0)
food.shape(shapes)
food.color(colors)
food.penup()
food.goto(0, 100)


pen = turtle.Turtle()
```

```
pen.speed(0)
pen.shape("square")
pen.color("white")
pen.penup()
pen.hideturtle()
pen.goto(0, 250)
pen.write("Score : 0 High Score : 0", align="center",
        font=("candara", 24, "bold"))
# assigning key directions
def group():
if head.direction != "down":
        head.direction = "up"


def godown():
if head.direction != "up":
        head.direction = "down"


def goleft():
if head.direction != "right":
        head.direction = "left"


def goright():
if head.direction != "left":
        head.direction = "right"


def move():
if head.direction == "up":
        y = head.ycor()
        head.sety(y+20)
if head.direction == "down":
        y = head.ycor()
        head.sety(y-20)
if head.direction == "left":
        x = head.xcor()
        head.setx(x-20)
if head.direction == "right":
```

```
x = head.xcor()
head.setx(x+20)


wn.listen()
wn.onkeypress(group, "w")
wn.onkeypress(godown, "s")
wn.onkeypress(goleft, "a")
wn.onkeypress(goright, "d")
segments = []

# Main Gameplay
while True:
wn.update()
if head.xcor() > 290 or head.xcor() < -290 or head.ycor() > 290 or head.ycor()
< -290:
        time.sleep(1)
        head.goto(0, 0)
        head.direction = "Stop"
        colors = random.choice(['red', 'blue', 'green'])
        shapes = random.choice(['square', 'circle'])
        for segment in segments:
                segment.goto(1000, 1000)
        segments.clear()
        score = 0
        delay = 0.1
        pen.clear()
        pen.write("Score : {} High Score : {} ".format(
                score, high_score), align="center", font=("candara", 24, "bold"))
if head.distance(food) < 20:
        x = random.randint(-270, 270)
        y = random.randint(-270, 270)
        food.goto(x, y)

        # Adding segment
        new_segment = turtle.Turtle()
        new_segment.speed(0)
        new_segment.shape("square")
        new_segment.color("orange") # tail colour
        new_segment.penup()
        segments.append(new_segment)
        delay -= 0.001
        score += 10
        if score > high_score:
                high_score = score
        pen.clear()
        pen.write("Score : {} High Score : {} ".format(
                score, high_score), align="center", font=("candara", 24, "bold"))
# Checking for head collisions with body segments
for index in range(len(segments)-1, 0, -1):
        x = segments[index-1].xcor()
        y = segments[index-1].ycor()
        segments[index].goto(x, y)
if len(segments) > 0:
        x = head.xcor()
        y = head.ycor()
        segments[0].goto(x, y)
move()
for segment in segments:
        if segment.distance(head) < 20:
                time.sleep(1)
                head.goto(0, 0)
                head.direction = "stop"
                colors = random.choice(['red', 'blue', 'green'])
                shapes = random.choice(['square', 'circle'])
                for segment in segments:
                        segment.goto(1000, 1000)
                segments.clear()

                score = 0
                delay = 0.1
                pen.clear()
                pen.write("Score : {} High Score : {} ".format(
                        score,  high_score),  align="center",  font=("candara",  24,
"bold"))
time.sleep(delay)

wn.mainloop()
```

SAMPLE OUTPUT



Score : 0 High Score : 0

CONCLUSION

The project in python programming of Snake Game is a simple console application with very simple graphics. In this project, you can play the popular “Snake Game” just like you played it elsewhere. You must use the up, down, right, or left arrows to move the snake. Foods are provided at the several coordinates of the screen for the snake to eat. Every time the snake eats the food, its length will increase by one element along with the score. It isn’t the world’s greatest game, but it does give you an idea of what you can achieve with a relatively simple python programming, and perhaps the basis by which to extend the principles and create more interesting games on your own

FUTURE SCOPE

In this project, I have used a simple application. This project will be able to be implemented in future after making some changes and modifications as I made this project at a low level. The modifications that can be made in this project are1. It can be made with good graphics.2. We can add more options like Top scores and Player Profile.3. We can add a multiplayer option.

REFERENCE

1) https://data-flair.training/blogs/snake-game-python- program/

REFERENCES:

✓ Class Lecture.

✓https://www.youtube.com/watch?v=rFFVyNn9 8A

✓ http://www.c-sharpcorner.com/UploadFile/udeshikah/snake-gameapplication-in-C-Sharp/

✓ http://www.dreamincode.net/forums/topic/243537-control-issue-withsnake-game-in-c%23/

✓http://esharp.net-informations.com/communications/csharp- chatclient.htm