



**NEW HORIZON  
COLLEGE OF ENGINEERING**

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC  
Accredited by NAAC with 'A' Grade, Accredited by NBA

## **MINI PROJECT REPORT**

*for*

**Problem Solving Using Python (21CSE23A) on**

**Snake Game**

*Submitted by*

**Kalyan Ram P**

USN: 1NH21AI042

**Sumanth Raj P**

USN:1NH21AI073

**Dhiran Kumar Reddy**

USN:1NH21AI026

*In partial fulfilment for the award of the degree of*

**BACHELOR OF ENGINEERING**

*in*

**COMPUTER SCIENCE AND ENGINEERING**

**Academic Year: 2021-22(EVEN SEM)**



**NEW HORIZON  
COLLEGE OF ENGINEERING**

Autonomous College Permanently Affiliated to VTU, Approved by AICTE & UGC  
Accredited by NAAC with 'A' Grade, Accredited by NBA

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

## **CERTIFICATE**

This is to certify that the mini project work titled

**Snake Game**

submitted in partial fulfilment of the degree of Bachelor of Engineering in  
Computer Science and Engineering by

**Kalyan Ram P  
USN:1NH21AI042**

**Sumanth Raj P  
USN:1NH21AI073**

**Dhiran Kumar Reddy  
USN:1NH21AI026**

*DURING*

*EVEN SEMESTER 2021-22*

*for*

*Course: 21CSE23A - Problem Solving using Python*

-----  
Signature of Guide  
(Ms. Nayana)

-----  
Signature of HOD

# **ABSTRACT**

Python is a widely used general-purpose, high level programming language. It was initially designed by Guido Van Rossum in 1991 and developed by Python Software Foundation. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of codes. Python is a programming language that lets you work quickly and integrate systems more efficiently. Reason for increasing popularity: 1. Emphasis on code readability, shorter codes, ease of writing. 2. Programmers can express logical concepts in fewer lines of code in comparison to languages such as C++ or Java. 3. Python supports multiple programming paradigms, like object-oriented, imperative and functional programming or procedural.

## ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned my efforts with success.

I have great pleasure in expressing gratitude to **Dr. Mohan Manghnani**, Chairman, New Horizon Educational Institutions, for providing necessary infrastructure and creating good environment.

I take this opportunity to express my profound gratitude to **Dr. Manjunatha**, Principal, New Horizon College of Engineering, for his constant support and encouragement.

I would like to thank **Dr. Anandhi R**, Professor and Dean-Academics, NHCE, for her valuable guidance.

I would also like to thank **Dr. Anusuya**, Professor and HOD, Department of Chemistry for Applied Sciences, for her constant support.

I also express our gratitude to **Ms. Nayana**, my guide, for constantly monitoring the development of the project and setting up precise deadlines. Her valuable suggestions were the motivating factors in completing the work.

Finally, a note of thanks to all the teaching and non-teaching staff of Dept of Computer Science and Engineering, for their cooperation extended to me, and my parents and friends, who helped me directly or indirectly in the course of the project work.

**Kalyan Ram P**

USN: 1NH21AI042

**Sumanth Raj P**

USN:1NH21AI073

**Dhiran Kumar Reddy**

USN:1NH21AI026

# CONTENTS

**ABSTRACT**

**ACKNOWLEDGEMENT**

<b>1. INTRODUCTION</b>	<b>I</b>
<b>2. SYSTEM REQUIREMENTS</b>	<b>IV</b>
• HARDWARE REQUIREMENTS	
• SOFTWARE REQUIREMENTS	
<b>3. DESIGN</b>	<b>V</b>
<b>4. IMPLEMENTATION</b>	<b>VII</b>
<b>5. OUTPUT SNAPSHOTS</b>	<b>XI</b>
<b>6. CONCLUSION</b>	<b>XII</b>
<b>REFERENCES</b>	<b>XIII</b>

---

# CHAPTER 1

## INTRODUCTION

### 1.1 Snake Game

Snake is the common name for a video game concept where the player maneuvers a line which grows in length, with the line itself being a primary obstacle. The concept originated in the 1976 arcade game Blockade, and the ease of implementing Snake has led to hundreds of versions (some of which have the word snake or worm in the title) for many platforms. After a variant was preloaded on Nokia mobile phones in 1998, there was a resurgence of interest in the snake concept as it found a larger audience.

Gameplay: The player controls a dot, square, or object on a bordered plane. As it moves forward, it leaves a trail behind, resembling a moving snake. In some games, the end of the trail is in a fixed position, so the snake continually gets longer as it moves. In another common scheme, the snake has a specific length, so there is a moving tail a fixed number of units away from the head. The player loses when the snake runs into the screen border, a trail or other obstacle, or itself. The Snake concept comes in two major variants:

1. In the first, which is most often a two-player game, there are multiple snakes on the playfield. Each player attempts to block the other so the opponent runs into an existing trail and loses. The Light Cycles segment of the Tron arcade game is a single-player version where the other "snakes" are AI controlled.
2. In the second variant, a sole player attempts to eat items by running into them with the head of the snake. Each item eaten makes the snake longer, so avoiding collision with the snake becomes progressively more difficult. Examples: Nibbler, Snake Byte.

---

## EXTERNAL LIBRARY FUNCTIONS

- Pygame:
  - `pygame.display.set_mode()`: controls the display and window screen.
  - `pygame.mouse.get_pos()`: obtains the position of the mouse.
  - `pygame.draw()`: draws shapes in the given position and colour.
  - `pygame.event.get()`: interacts with events such as button clicks.
  - `pygame.display.set_caption()`: displays the specified caption on the top left corner of the screen.

## FUNCTIONS

- `game_over()`: This function is responsible for terminating the snake game either if the player collides the snake into its own tail or hits the boundaries of the game window.

After terminating the game it shows a black screen saying 'Game Over' and it also shows the players final score.

- `show_score()`: This function is responsible for showing the players score at the top left corner of the game window. It keeps incrementing the players score by one, every time the snake engulfs the food. The following are the variables that have been used in the snake game:

1. `snake_x` Controls the length of the snake.
2. `snake_y` Controls the breadth of the snake.
3. `food_x` Controls the length of the food.
4. `food_y` Controls the breadth of the food.
5. `frame_x` This variable has been used to adjust the length of the game window.
6. `frame_y` This variable has been used to adjust the breadth of the game window

---

## Pygame

Pygame is a cross-platform set of Python modules designed for writing video games. It includes computer graphics and sound libraries designed to be used with the Python programming language. Pygame was originally written by Pete Shinnars to replace PySDL after its development stalled. It has been a community project since 2000 and is released under the open-source free software GNU Lesser General Public License.

**Pygame** is a set of Python modules designed for writing video games. Pygame adds functionality on top of the excellent SDL library. This allows you to create fully featured games and multimedia programs in the python language.

Pygame is highly portable and runs on nearly every platform and operating system.

Pygame itself has been downloaded millions of times.

Pygame is free. Released under the LGPL licence, you can create open source, freeware, shareware, and commercial games with it. See the licence for full details.

You call pygame functions, they don't call your functions. This gives you greater control when using other libraries, and for different types of programs.



---

## **CHAPTER 2**

### **SYSTEM REQUIREMENTS**

#### **HARDWARE REQUIREMENTS**

- Processor : Any Processor above 500 MHz
- RAM : 512Mb
- Hard Disk : 10 GB
- Input device : Standard Keyboard and Mouse
- Output device : VGA and High resolution Monitor

#### **SOFTWARE REQUIREMENTS**

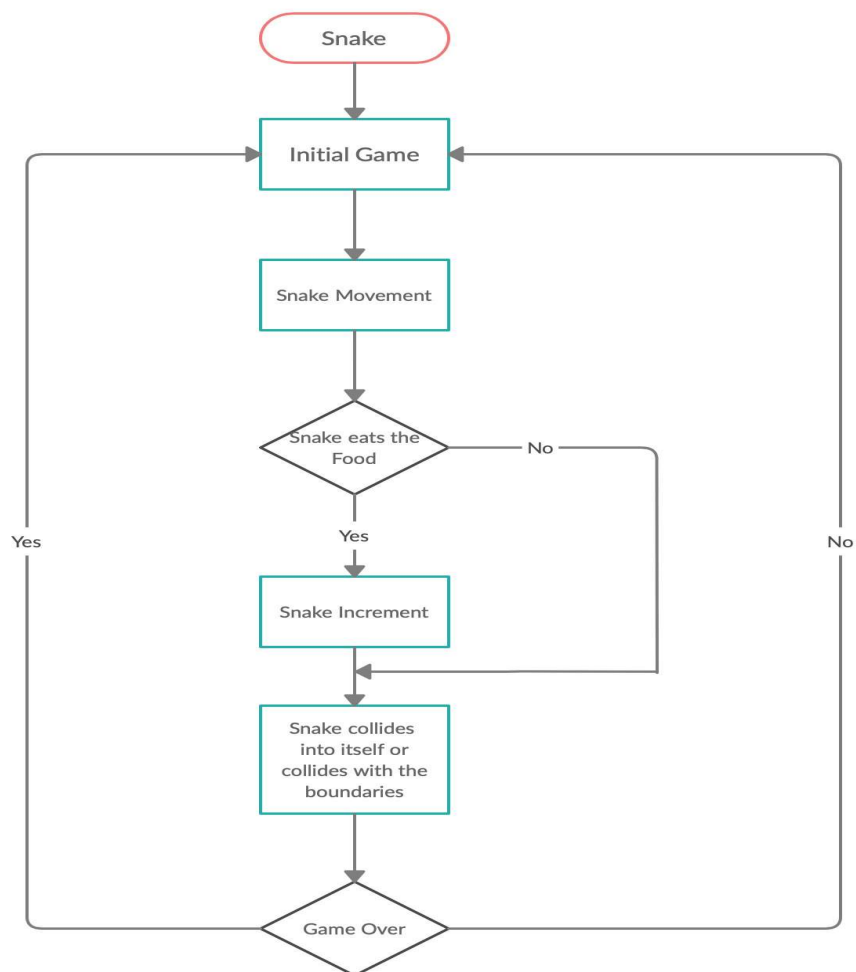
- Operating system : Windows XP
- Front End : ASP.Net 2.0
- IDE : Visual Studio 2008
- Data Base : SQL Server Management Studio 2005
- Server : Internet Information Services
- Database Connectivity: ODBC Sources (with SQL Server)

---

## CHAPTER 3

### DESIGN

#### 3.1 FLOWCHART



---

## 3.2 ALOGRITHM

Step 1. Start

Step 2. Initialize the game window

Step 3. Input the movement of the snake using arrow keys

Step 4. If snake position is same as the food, then the length of the snake is increased

Step 5. Elif the snake hits itself or the boundary of the window, then go to step 6

Step 6. Show the score of the player

Step 7. Stop

---

## CHAPTER 4

### IMPLEMENTATION

#### SOURCE CODE

```
import pygame
import sys
import time
import random

# Difficulty settings
# Easy      - 15
# Medium    - 30
# Hard      - 60

difficulty = 15

# Snake size
snake_x = 10
snake_y = 10

# Food size
food_x = 10
food_y = 10

# Window size
frame_size_x = 900
frame_size_y = 600

# Checks for errors encountered
check_errors = pygame.init()
if check_errors[1] > 0:
    print(f'[] Had {check_errors[1]} errors when initialising the game, exiting...')
    sys.exit(-1)
else:
    print('Game successfully initialised')

# Initialising the game window
pygame.display.set_caption('Snake Game')
game_window = pygame.display.set_mode((frame_size_x, frame_size_y))

# Colors
black = pygame.Color(0, 0, 0)
```

---

```

white = pygame.Color(255, 255, 255)
red = pygame.Color(255, 0, 0)
green = pygame.Color(0, 255, 0)
blue = pygame.Color(0, 0, 255)

# FPS (frames per second) controller
fps_controller = pygame.time.Clock()

# Game variables
snake_pos = [100, 50]
snake_body = [[100, 50], [100-10, 50], [100-(2*10), 50]]
food_pos = [random.randrange(1, (frame_size_x//10)) * 10, random.randrange(1, (frame_size_y//10)) * 10]
food_spawn = True
direction = 'RIGHT'
change_to = direction
score = 0

# Game Over function
def game_over():
    my_font = pygame.font.SysFont('times new roman', 90)
    game_over_surface = my_font.render('Game Over', True, red)
    game_over_rect = game_over_surface.get_rect()
    game_over_rect.midtop = (frame_size_x/2, frame_size_y/4)
    game_window.fill(black)
    game_window.blit(game_over_surface, game_over_rect)
    show_score(0, white, 'times', 60)
    pygame.display.flip()
    time.sleep(5)
    pygame.quit()
    sys.exit()

# Score function
def show_score(choice, color, font, size):
    score_font = pygame.font.SysFont(font, size)
    score_surface = score_font.render('Score : ' + str(score), True, color)
    score_rect = score_surface.get_rect()
    if choice == 1:
        score_rect.midtop = (frame_size_x/8, 15)
    else:
        score_rect.midtop = (frame_size_x/2, frame_size_y/1.25)
    game_window.blit(score_surface, score_rect)

# Main logic
while True:

```

---

---

```

for event in pygame.event.get():
    if event.type == pygame.QUIT:
        pygame.quit()
        sys.exit()
    elif event.type == pygame.KEYDOWN:
        if event.key == pygame.K_UP or event.key == ord('w'):
            change_to = 'UP'
        if event.key == pygame.K_DOWN or event.key == ord('s'):
            change_to = 'DOWN'
        if event.key == pygame.K_LEFT or event.key == ord('a'):
            change_to = 'LEFT'
        if event.key == pygame.K_RIGHT or event.key == ord('d'):
            change_to = 'RIGHT'
        if event.key == pygame.K_ESCAPE:
            pygame.event.post(pygame.event.Event(pygame.QUIT))

    # Making sure the snake cannot move in the opposite direction instantly
    if change_to == 'UP' and direction != 'DOWN':
        direction = 'UP'
    if change_to == 'DOWN' and direction != 'UP':
        direction = 'DOWN'
    if change_to == 'LEFT' and direction != 'RIGHT':
        direction = 'LEFT'
    if change_to == 'RIGHT' and direction != 'LEFT':
        direction = 'RIGHT'

    # Movement of the snake
    if direction == 'UP':
        snake_pos[1] -= 10
    if direction == 'DOWN':
        snake_pos[1] += 10
    if direction == 'LEFT':
        snake_pos[0] -= 10
    if direction == 'RIGHT':
        snake_pos[0] += 10

    # Code for increasing the length of the Snake when it eats the food
    snake_body.insert(0, list(snake_pos))
    if snake_pos[0] == food_pos[0] and snake_pos[1] == food_pos[1]:
        score += 1
        food_spawn = False
    else:
        snake_body.pop()

```

---

---

```
# Spawning food on the screen
if not food_spawn:
    food_pos = [random.randrange(1, (frame_size_x//10)) * 10, random.randrange(1, (frame_size_y//10)) * 10]
    food_spawn = True

# Game Graphics
game_window.fill(black)
for pos in snake_body:
    pygame.draw.rect(game_window, green, pygame.Rect(pos[0], pos[1], snake_x, snake_y))

# Snake food
pygame.draw.rect(game_window, red, pygame.Rect(food_pos[0], food_pos[1], food_x, food_y))

# Game Over conditions
if snake_pos[0] < 0 or snake_pos[0] > frame_size_x-10:
    game_over()
if snake_pos[1] < 0 or snake_pos[1] > frame_size_y-10:
    game_over()
for block in snake_body[1:]:
    if snake_pos[0] == block[0] and snake_pos[1] == block[1]:
        game_over()

show_score(1, white, 'consolas', 40)

# Refresh game screen
pygame.display.update()

# Refresh rate
fps_controller.tick(difficulty)
```

---

## CHAPTER 5

### DESIGN

#### OUTPUT SNAPSHOTS



Fig.5.1

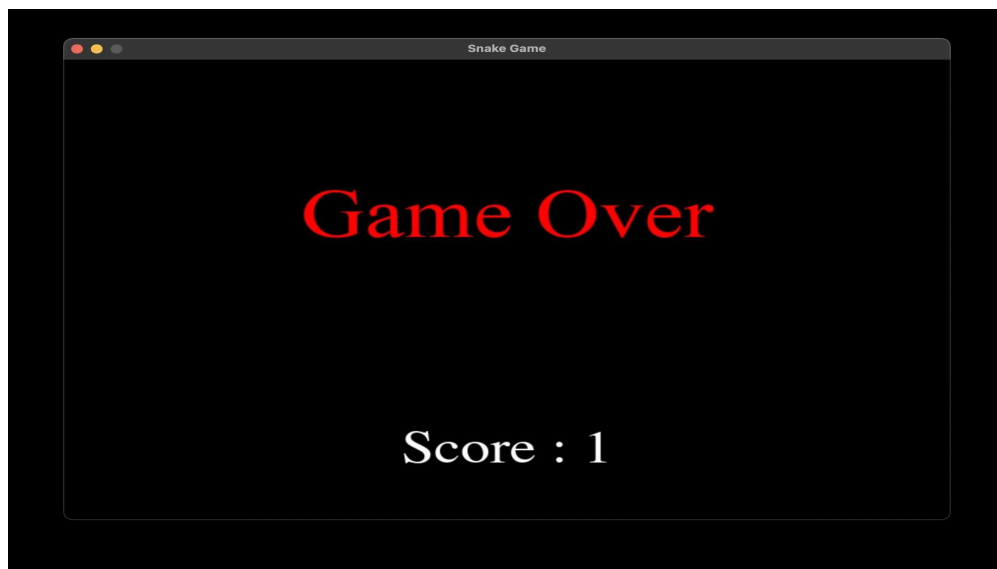


Fig. 5.2



---

## CHAPTER 6

### CONCLUSION

The coding of Snake was extremely difficult with many errors arising. Many systems had to be written numerous ways before a final working solution was found. For example, two different movement methods were used prior to final version; however, even the final version is flawed as vertical movement causes the snake to change scale. There were also issues with the food – snake collision detection. While the final version resulted in a snake that could eat food, the movement glitch caused the food to cause further size issues.

For these reasons, it is recommended that anyone who wishes to recreate this game starts simply when writing the code. It is advisable that they first perfect the snake's movement controls before messing with the food generation. By taking the code in small sections, it is easier to get individual features to work. Building off this, use functions to contain each aspect of the game. Using functions made it easier to determine where errors were occurring when debugging the code. It also kept the code more organized.

At the very end, we could fix as many glitches and bugs as possible and the game is now running smoothly but we couldn't add any user-friendly menu at the start of the game apart from that, the game is totally functional and ready for the consumer to be played.

---

## References

- [1] Pygame Tutorials:  
<https://www.youtube.com/watch?v=i6xMBig-pP4&list=PLzMCGfZo4-lp3jAExUCewBfMx3UZFkh5>
- [2] Troubleshooting: <https://stackoverflow.com>
- [3] Computer Science with Python - Sumita Aroras
- [4] Documentation: <https://www.w3schools.com>