PICO SNAKE (A FUN GAME)

A PROJECT REPORT

Submitted

In the partial fulfillment of the requirements for the Mid-Term Project Assignment Evaluation of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND ENGINEERING

by

Kirti Shree [1941012382]

Pallav Raj [1941012877]

UNDER THE GUIDANCE OF

MR. TUHINANSU PRADHAN

Associate Professor

MR. SHAKTIJEET MAHAPATRA

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH

SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY) BHUBANESWAR-751003, ODISHA, INDIA.

DECEMBER-2022

DECLARATION

I hereby declare that the work described in this thesis "**Pico Snake – A Fun Game**" which is being submitted in partial fulfillment for the award of **Bachelor Of Technology** in the Department of **Computer Science and Engineering** affiliated to Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar (Odisha) is the result of investigations carried out by me under the Guidance of **MR. TUHINANSU PRADHAN and** of **MR. SHAKTIJEET MAHAPATRA Centre for IoT**, **Institute of Technical Education and Research (ITER), Bhubaneswar .**

The work is original and has not been submitted for any Degree/Diploma of this or any other university.

Place: Bhubaneswar

Date: 24.12.22

Pallav Raj

Kirti Shree

CERTIFICATE

This is to certify that the project report entitled "**Pico Snake**" being submitted by **Pallav Raj (1941012877)** and **Kirti Shree (1941012382)** in partial fulfillment for Mid-Term Project Assignment Evaluation of **Bachelor of Technology in** the Department of **Computer Science and Engineering** affiliated to Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar (Odisha)**,** is a record of bonafide work carried out by them during the academic year 2022-2023 under our guidance and supervision.

The results embodied in the report have not been submitted to any other University or Institution for the award of any degree or diploma.

MR. **TUHINANSU PRADHAN**

Project Guide

MR. **SHAKTIJEET MAHAPATRA**

Project Guide

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to Dr. Debahuti Mishra, Professor, Head of the department, Computer Science and Engineering, Institute of Technical Education and Research(ITER), **Siksha 'O' Anusandhan (Deemed to be University)**, Bhubaneswar for his continued support and valuable guidance and encouragement extended to me during my project work. I thank her for his painstaking efforts to guide me throughout my work. I thank Dr. Biswaranjan Swain, Associate Professor of Center for Internet of Things (CIoT), Institute of Technical Education and Research (ITER), Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar for his valuable advice, support and help during my project work. I thank Mr **TUHINANSU PRADHAN** and MR. **SHAKTIJEET MAHAPATRA** Center for Internet of Things (CIoT), Institute of Technical Education and Research (ITER), Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar for her valuable comments and suggestions that greatly helped in improving quality of report. I thank Mr. Avinav Rath , Mr. Swastik Chodhary , Mrs Shradhanjali Nayak, Mr. Upendra Kumar Prusty, Institute of Technical Education and Research . (ITER), Siksha 'O' Anusandhan (Deemed to be University), Bhubaneswar for their valuable comments and suggestions that greatly helped during circuit analysis and housing of the project. I thank to all my teachers and professors for their valuable comments after reviewing my project reports. I wish to extend my special thanks to all my colleagues and friends who helped directly or indirectly to complete my project assignment work. I extend my thanks to my parents and all my family members for their unceasing encouragement and support who wished me a lot to complete this work.

Pallav Raj & Kirti Shree

Signature of Students

ABSTRACT

The main purpose of this project is to have some fun along with applying the idea of IOT. The main objective of the project is to provide users with a fun and engaging gaming experience that utilizes IoT technology to allow for real-time communication and interaction with the game. The game also incorporates sensors and actuators that enable it to respond to changes in the physical environment, adding an additional layer of complexity to the gameplay. Overall, Pico Snake serves as a useful tool for exploring and demonstrating the capabilities of IoT technology, as well as for educating users about its potential applications. In this project we are making a SNAKE GAME that we used to play earlier in our childhood. In this game the snake is hunting the food and while doing so if the snake gets touched to itself or the boundary of the game , the game will get over. To again start the game there is a reset button and O-led will display ARE YOU READY and in the end when the game will get over the OLED will display GAME OVER along with our registration number and LED's glowing in the END.

TABLE OF CONTENTS

TOPICS

LIST OF FIGURES

TOPICS

CHAPTER 1: INTRODUCTION

1.1 Introduction:

Pico Snake is an Internet of Things (IoT) based project that involves the use of connected devices to build a snake-like game. The game is designed to be played on a grid-like board, with the aim of the player being to guide a virtual snake towards a target, while avoiding obstacles and other hazards.

One of the key features of Pico Snake is its use of IoT technology, which allows for real-time communication between the game board and a player's device. This means that players can interact with the game in real-time, and see the effects of their actions on the game board immediately.

Pico Snake also incorporates various sensors and actuators, which allow the game to respond to changes in the physical environment. For example, if the game board is tilted, the snake may move in a different direction, adding an additional level of complexity to the gameplay.

Overall, Pico Snake is a fun and interactive way to learn about IoT technology and how it can be used to create engaging and dynamic experiences. Whether you are a beginner or an experienced player, Pico Snake provides a challenging and rewarding gaming experience that is sure to keep you coming back for more. The whole thing is written using Python3 language.

The main objectives of the project are:

- To explore and demonstrate the capabilities of IoT technology
- To create a fun and engaging gaming experience
- To educate users about IoT technology
- To encourage creativity and problem-solving

- To promote teamwork and collaboration
- To provide a challenging and rewarding gaming experience for players of all skill levels
- To showcase how IoT devices can communicate and interact with each other in real-time

1.2 Project Overview:

The project explains the implementation of " **PICO SNAKE-A FUN GAME"**. The thesis is explained here with:

Chapter 1 Presents introduction to the overall thesis and the overview of the project and its applications are discussed.

Chapter 2 Presents the hardware description. It deals with the block diagram of the project and explains the purpose of each block.

Chapter 3 Presents the software description. It explains the implementation of the project using Thonny IDE.

Chapter 4 Presents the advantages, disadvantages and applications of the project.

Chapter 5 Presents the results, conclusion and future scope of the project.

CHAPTER 2: HARDWARE DESCRIPTION

2.1 Introduction:

In this chapter the block diagram of the project and design aspect of independent modules are considered. Block diagram is shown in fig: 2.1

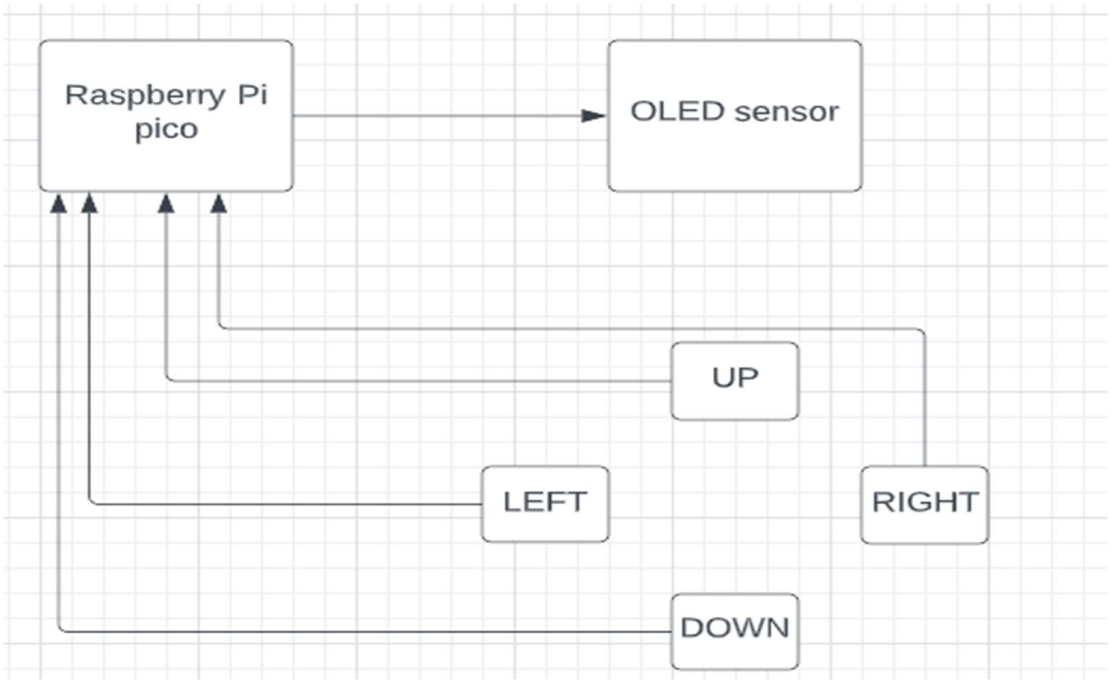


Fig. 2.1 Block diagram of Pico Snake Game

The main blocks of this project are:

- OLED sensor Module
- Raspberry Pi Pico Microcontroller
- Push buttons
- Jumper cable

OLED SENSOR MODULE:

Transparent OLED is a breakthrough transparent display technology that displays dynamic or interactive information on a transparent surface glass. This revolutionary display allows users to view what is shown on a glass video screen while still being able to see through it. Designers can overlay text, digital images, and video content onto physical objects or scenes that sit behind the glass.

Transparent OLED displays are self-emitting and utilize cutting-edge Organic Light Emitting Diode (OLED) technology to eliminate the need for a backlight or enclosure, making it possible to create truly see-through installations in a virtually frameless glass design.

Fig. 2.2 OLED Sensor Module

Features:

- The High accuracy: OLED sensors are known for their high accuracy and sensitivity, making them well-suited for a wide range of applications.
- Low power consumption: OLED sensors use less power than other types of sensors, making them ideal for use in portable and battery-powered devices.

- Fast response time: OLED sensors are able to detect and measure changes in the environment quickly, making them suitable for applications that require fast response times.
- Compact size: OLED sensors are typically small and lightweight, making them easy to integrate into a variety of devices and systems.
- Wide temperature range: OLED sensors are able to function over a wide temperature range, making them suitable for use in a variety of environments.
- High reliability: OLED sensors are known for their high reliability and long lifespan, making them ideal for use in mission-critical applications.

Working principle of OLED:

OLEDs work in a similar way to conventional diodes and LEDs, but instead of using layers of n-type and p-type semiconductors, they use organic molecules to produce their electrons and holes. A simple OLED is made up of six different layers. On the top and bottom there are layers of protective glass or plastic. The top layer is called the seal and the bottom layer the substrate. In between those layers, there's a negative terminal (sometimes called the cathode) and a positive terminal (called the anode). Finally, in between the anode and cathode are two layers made from organic molecules called the emissive layer (where the light is produced, which is next to the cathode) and the conductive layer (next to the anode)

2.2 Raspberry Pi Pico:

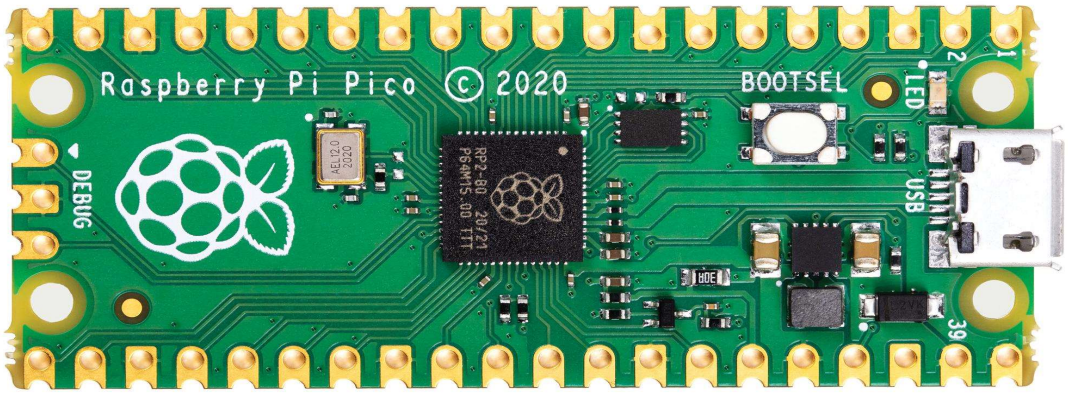


Fig. 2.3 RP2040

The Raspberry Pi Pico is a small, powerful microcontroller that is designed for use in a variety of applications. It is based on the Raspberry Pi platform, but is significantly smaller and more affordable than other Raspberry Pi boards. The Pico is equipped with a powerful Cortex-M0+ processor and a range of peripherals, including GPIO pins, PWM outputs, and an I2C interface. It can be programmed using a variety of languages, including C, Python, and MicroPython, making it versatile and easy to use. The Pico is an excellent choice for anyone looking to add powerful, low-cost microcontroller capabilities to their projects.

Summary

CPU type	Cortex M0+
Core count	2
Max speed	133 MHz
SRAM	264 kB in 6 banks
Flash internal	0 kB
Flash external	2 MB QSPI flash (up to 16 MB supported)
GPIOs	26 (incl. 4 ADC)
USB	1.1 Host / Slave
ADC	12 Bit @ 500 kSps
ADC Channels	5 (incl. temperature sensor)
SPI	2
UART	2
I^2C	2
PWM	16 channels
Timer	1x 64-bit
RTC	yes
Unique features	Programmable IO state machine, Boot ROM with USB mass storage bootloader

Fig 2.4 Specifications of Raspberry Pi Pico

Board:

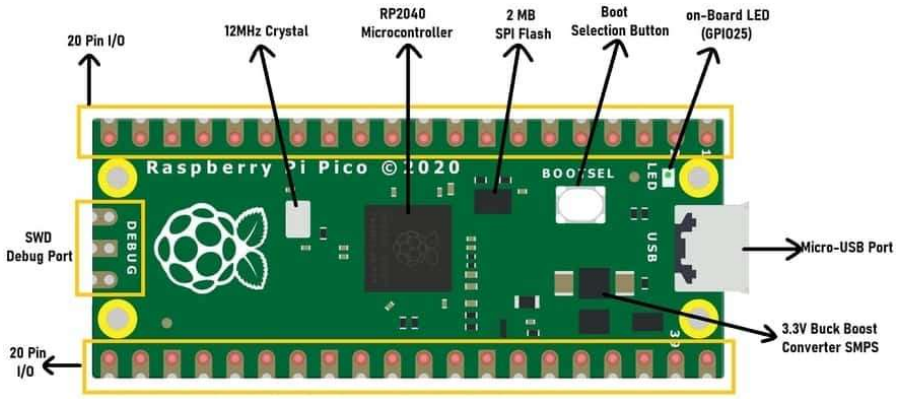


Fig 2.5 Board Description of Raspberry Pi Pico

The Raspberry Pi Pico is a small, powerful microcontroller that is based on the Raspberry Pi platform. It measures only 20mm x 51mm, making it significantly smaller and more compact than other Raspberry Pi boards. The Pico is equipped with a powerful Cortex-M0+ processor, which provides fast and efficient performance.

The Pico also includes a range of peripherals, including GPIO pins, PWM outputs, and an I2C interface. These peripherals allow the Pico to interact with a variety of sensors, actuators, and other devices, making it suitable for a wide range of applications. The Pico is also equipped with 2MB of onboard flash memory, which can be used to store program code and data.

In addition to its small size and powerful processor, the Pico is also easy to use. It can be

programmed using a variety of languages, including C, Python, and MicroPython. This allows users to quickly and easily develop software for the Pico, without the need for specialized knowledge or skills. The Pico also includes a built-in debugger and a range of tools and libraries, which can be used to develop, test, and debug software for the Pico.

Overall, the Raspberry Pi Pico is a versatile and powerful microcontroller that is well-suited to a wide range of applications. Its small size, powerful processor, and easy-to-use programming capabilities make it an excellent choice for anyone looking to add microcontroller capabilities to their projects.

Pin Definition:

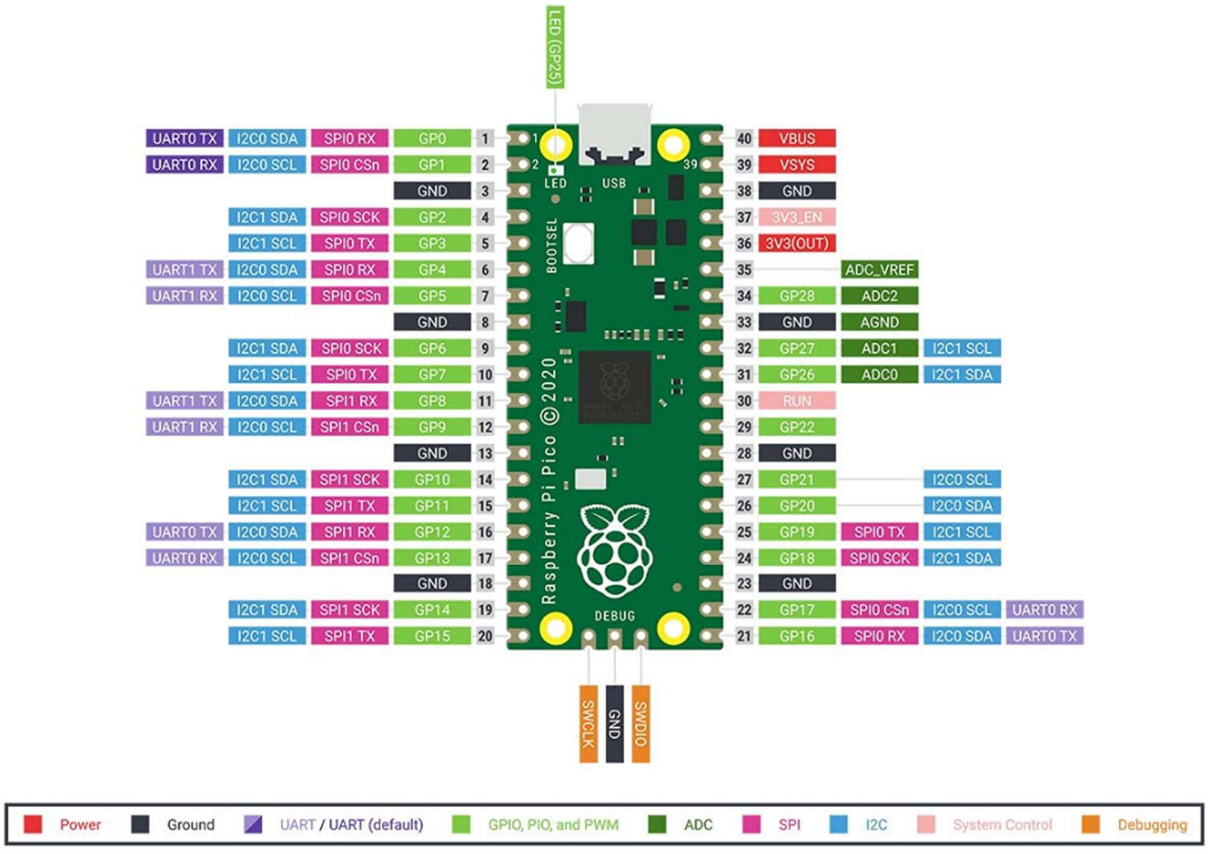


Fig 2.6 Pinout Description of Raspberry Pi PICO

CHAPTER 3: SOFTWARE DESCRIPTION

Thonny IDE:

Thonny is a Python integrated development environment (IDE) that is designed for beginners. It is an easy-to-use, cross-platform IDE that is specifically designed to help users learn Python and develop Python programs. Thonny includes a range of features that make it ideal for learning, including an intuitive user interface, built-in debugging tools, and a simple code editor. It also includes a range of educational resources, such as tutorials, examples, and documentation, which can help users to quickly get started with Python and develop their skills. Overall, Thonny is an excellent choice for anyone looking to learn Python and develop Python programs..

The User Interface:

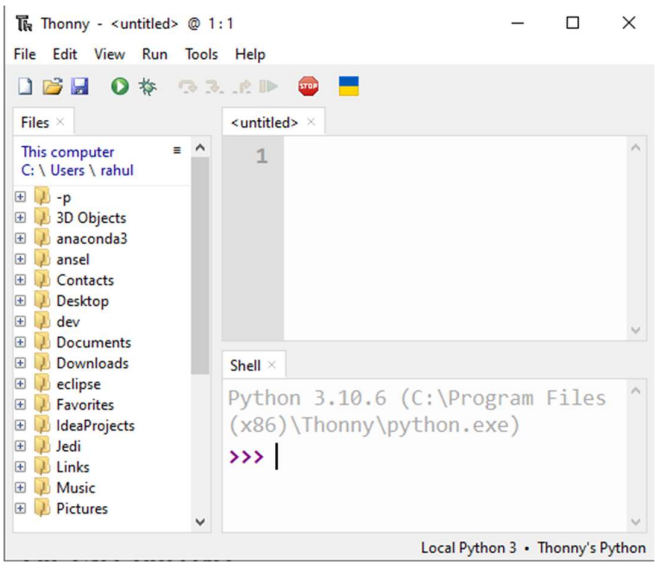


Fig 3.1 User Interface of Thonny

The UI of Thonny is designed to be intuitive and easy-to-use, even for users who are new to Python or programming in general. The interface includes a simple code editor, which allows users to write and edit their Python code. It also includes a range of built-in debugging tools, such as a variable explorer and a debugger, which can be used to help users understand and troubleshoot their code.

The Icons:

Across the top you'll see several icons. Let's explore what each of them does. You'll see an image of the icons below, with a letter above each one. We will use these letters to talk about each of the icons:

Fig 3.2 Icon pack of Thonny

Working our way from left to right, below is a description of each of the icons in the image.

A: The paper icon allows you to create a new file. Typically in Python you want to separate your programs into separate files. You'll use this button later in the tutorial to create your first program in Thonny!

B: The open folder icon allows you to open a file that already exists on your computer. This might be useful if you come back to a program that you worked on previously.

C: The floppy disk icon allows you to save your code. Press this early and often. You'll use this later to save your first Thonny Python program.

D: The play icon allows you to run your code. Remember that the code you write is meant to be

executed. Running your code means you're telling Python, "Do what I told you to do!" (In other words, "Read through my code and execute what I wrote.")

E: The bug icon allows you to debug your code. It's inevitable that you will encounter bugs when you're writing code. A bug is another word for a problem. Bugs can come in many forms, sometimes appearing when you use inappropriate syntax and sometimes when your logic is incorrect.

Thonny's bug button is typically used to spot and investigate bugs. You'll work with this later in the tutorial. By the way, if you're wondering why they're called bugs, there's also a fun story of how it came about!

F-H: The arrow icons allow you to run your programs step by step. This can be very useful when you're debugging or, in other words, trying to find those nasty bugs in your code. These icons are used after you press the bug icon. You'll notice as you hit each arrow, a yellow highlighted bar will indicate which line or section Python is currently evaluating:

- The **F** arrow tells Python to take a big step, meaning jumping to the next line or block of code.
- The **G** arrow tells Python to take a small step, meaning diving deep into each component of an expression.
- The **H** arrow tells Python to exit out of the debugger.

I: The resume icon allows you to return to play mode from debug mode. This is useful in the instance when you no longer want to go step by step through the code, and instead want your program to finish running.

J: The stop icon allows you to stop running your code. This can be particularly useful if, let's say,

your code runs a program that opens a new window, and you want to stop that program

Other UI Features:

To see more of the other features that Thonny has to offer, navigate to the menu bar and select the View dropdown. You should see that Shell has a check mark next to it, which is why you see the Shell section in Thonny's application window:

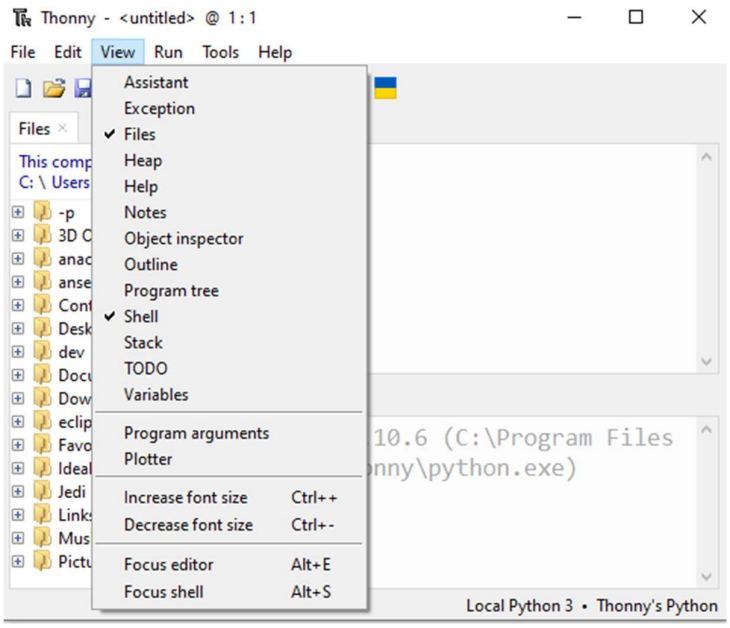


Fig 3.3 More UI of Thonny

Let's explore some of the other offerings, specifically those that will be useful to a beginning:

1. **Help:** You'll select the *Help* view if you want more information about working with Thonny. Currently this section offers more reading on the following topics: *Running Programs Step-wise,* how to install *3rd Party Packages*, or *using Scientific Python Packages*.
2. **Variables:** This feature can be very valuable. A variable in Python is a value that you define in code. Variables can be numbers, strings, or other complex data structures. This section allows you to see the values assigned to all of the variables in your program.
3. **Assistant:** The Assistant is there to give you helpful hints when you hit Exceptions or other types of errors.

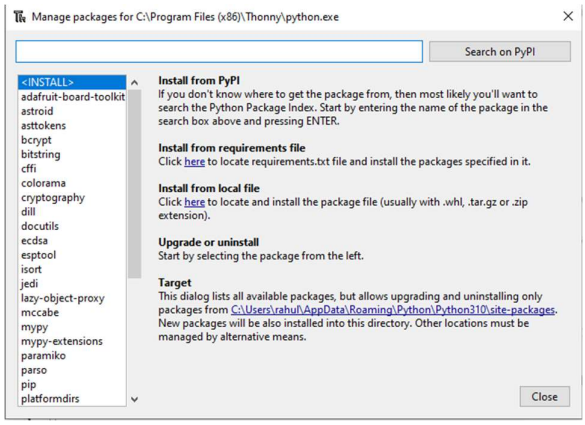
The other features will become useful as you advance your skills. Check them out once you get more comfortable with Thonny!

The Package Manager:

The package manager will allow you to install packages that you will need to use with your program. Specifically, it allows you to add more tools to your toolbox. Thonny has the built-in benefit of handling any conflicts with other Python interpreters.

To access the package manager, go to the menu bar and select Tools > Manage Packages… This should pop open a new window with a search field. Type simplecalculator into that field and click the Search button.

The output should look similar to this:



CHAPTER 4: ADVANTAGES AND DISADVANTAGES

Some of the advantages of a Pico Snake Game are:

- Provides an interactive and immersive gaming experience
- Teaches users about IoT technology
- Encourages creativity and problem-solving
- Promotes teamwork and collaboration
- Offers a wide range of difficulty levels
- Provides a portable gaming experience
- Can be played in a variety of environments
- Utilizes IoT technology to allow for real-time communication and interaction with the game
- Incorporates sensors and actuators that enable the game to respond to changes in the physical environment
- Provides a challenging and rewarding gaming experience for players of all skill levels

Some of the disadvantages of a Pico Snake Game are:

- Dependency on technology: As an IoT-based game, Pico Snake relies on technology to function, which means it may not be available if there are issues with connectivity or the device used to play the game.
- Limited replay value: Depending on the design of the game, Pico Snake may have limited replay value, as players may eventually exhaust all of the available challenges and obstacles.
- Complexity: Pico Snake may be too complex or challenging for some players, which could make it less enjoyable or less accessible to certain users.
- Cost: Depending on the design and components used, Pico Snake may be more expensive to produce and maintain than other types of games.
- Limited physical interaction: As a virtual game, Pico Snake may not provide the same level of physical interaction as other types of games, which could be a disadvantage for some players.
- Sensitivity to environmental changes: The use of sensors and actuators in Pico Snake means that the game may be more sensitive to changes in the physical environment, which could impact the gameplay in unexpected ways.

CHAPTER 5: RESULTS

5.1 Results:

The project "**PICO SNAKE-A FUN GAME**" was designed a system which uses a combination of OLED Sensor module and a Raspberry Pi Pico so that the snake can move and can be controlled with the push buttons provided.

All figures in sequence step by step

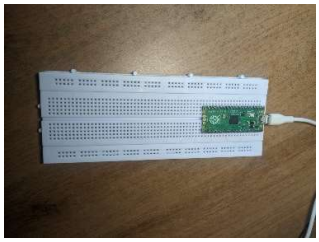


Fig 5.1

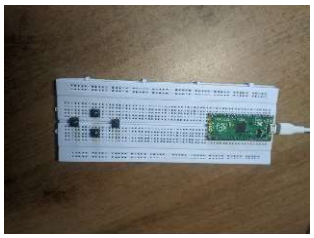


Fig 5.2

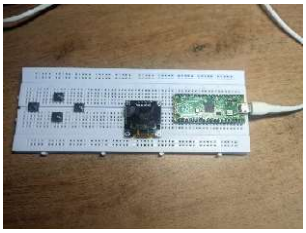


Fig 5.3

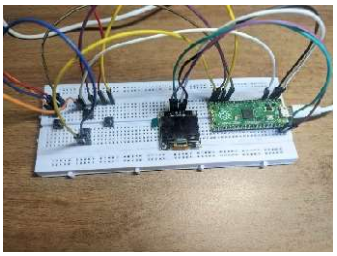


Fig 5.4

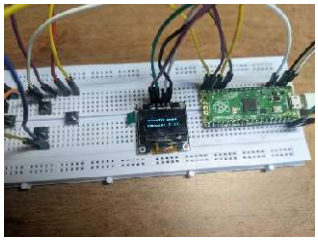


Fig 5.5

Fig 5.6

Fig 5.1 : Pico is connected to Breadboard

Fig 5.2 : Push buttons are connected with Pico

Fig 5.3 : OLED Sensor Module is connected with the Pico

Fig 5.4 : All Jumper wires are connected

Fig 5.5 : Final project After it’s powered On

Fig 5.6 : Final Packaged Project

5.2 Conclusion:

In conclusion, Pico Snake is a fun and engaging game that utilizes IoT technology to provide a unique and interactive gaming experience. While it has a number of advantages, such as encouraging creativity and problem-solving, promoting teamwork and collaboration, and offering a wide range of difficulty levels, it also has some potential disadvantages, such as being dependent on technology, having limited replay value, and potentially being too complex or challenging for some players. Overall, Pico Snake is a well-designed game that offers a challenging and rewarding gaming experience for players of all skill levels.

5.3 Future Scope:

The future scope of Pico Snake, as a fun and interactive IoT-based game, is vast. Some possible areas for future development include:

- Expanding the range of challenges and obstacles: To keep players coming back for more, Pico Snake could incorporate new and varied challenges and obstacles to keep gameplay fresh and exciting.
- Adding multiplayer capabilities: Pico Snake could be designed to allow for multiplayer gameplay, which could encourage teamwork and collaboration among players.
- Enhancing the use of sensors and actuators: As technology continues to advance, Pico Snake could incorporate more advanced sensors and actuators to enable the game to respond to a wider range of environmental changes.
- Incorporating augmented reality (AR) or virtual reality (VR) elements: Pico Snake could be enhanced with AR or VR elements to provide an even more immersive and interactive gaming experience.
- Developing educational modules: Pico Snake could be used as a tool to teach users about IoT technology and its potential applications, as well as other related topics such as programming and electronics

Overall, the future scope of Pico Snake is wide open, and there is potential for continued growth and development as technology continues to evolve.

REFERENCES

The sites which were used while doing this project:

1. how2electronics.com
2. www.raspberrypi.com
3. in.element14.com
4. www.electroduino.com
5. www.advantage.co.uk
6. www.elprocus.com
7. https://www.wikipedia.org
8. hackaday.com
9. https://youtu.be/5r_6mbYlLVo

CHAPTER 6: TEAMWORK

8.1. SUMMARY OF TEAM WORK

6.1.1 ATTRIBUTES

1	Attends group meetings regularly and arrives on time.
2	Contributes meaningfully to group discussions.
3	Completes group assignments on time.
4	Prepares work in a quality manner.
5	Demonstrates a cooperative and supportive attitude.
6	Contributes significantly to the success of the project.

6.1.2 SCORE

1=strongly disagree; **2**=disagree; **3**=agree; **4**=strongly agree

Student 1: Kirti Shree (1941012382)

Student 2: Pallav Raj (1941012877)

	Evaluated by	
Student 1	Attributes	Student 1
	1	4
	2	4
	3	4
	4	4
	5	4
	6	4
	Grand Total	24

	Evaluated by	
Student 2	Attributes	Student 2
	1	4
	2	4
	3	4
	4	4
	5	4
	6	4
	Grand Total	24

Pallav Raj

Signature of Student 1

Kirti Shree

Signature of Student 2

Appendix

```
from machine import Pin, I2C, ADC,PWM
from ssd1306 import SSD1306_I2C
from utime import sleep, ticks_ms
from random import randrange

# Terminologies:
# Everything is made up of four arc sprites: a,b,c,d.
# Sequence defines the order of arcs to create the slithering "S" curve.
# each arc is followed by two characters that indicate where origin where the arc should be drawn at
# relative to the snake segment that arc represents.
# These three characters are collectively called a "code"

# turntable is a very tedious lookup dictionary.
# key is the current arc and its current and desired turning direction
# value is what segment should be drawn to accomplish that turning.

# snake is an array of segments. Each segment contains x,y,Code.
# x,y is actual snake segment position.
# Code is the arc (sprite) and the two character code specifycing where the origin of the sprite should be drawn.

snake = []
scale = 1
spd = 0.2 #0.05

WIDTH,HEIGHT = 128, 64
border=1
gameWidth = int(WIDTH/3/scale)
gameHeight =int(HEIGHT/3/scale)
arenaWidth=gameWidth*3*scale
arenaHeight=gameHeight*3*scale
isDead = False
startWasPressed = False

#=== SPRITES ===
# ..XX  ...X  XX..  X...  .XX.
# .X..  ...X  ..X.  X...  X..X
# X...  ..X.  ...X  .X..  X..X
# X...  XX..  ...X  ..XX  .XX.
# a     b     c     d
a = [(0,0),(0,1),(1,2),(2,3),(3,3)]
b = [(0,0),(1,0),(2,1),(3,2),(3,3)]
c = [(3,0),(3,1),(2,2),(1,3),(0,3)]
d = [(3,0),(2,0),(1,1),(0,2),(0,3)]
#apple = [(),(),(),(),(),(),(),(),]

lrSeq = "c-=a-=b--d--c-=" #Left to Right (positive direction curve down first)
rlSeq = "d=-b=-a==c==d=-" #Left to Right (NEGATIVE direction curve down first)
duSeq = "c=-b=-a--d--c=-" #Bottom to top (positive direction curve left first)
udSeq = "d-=a-=b==c==d-=" #Bottom to top (NEGATIVE direction curve left first)

turnTable = {
# Left to right
"a+==+": "a", # a smooth, c backtracks
```

```
"a+==-": "d", # was d, b continues but with gap, d backtracks but no gap.
"b+==+": "d", # d tight, alternatively b if larger radius desired.
"b+==-": "c", # a loops, c is tight turn
"c+==+": "b", # b is tight turn
"c+==-": "a", # a is tight turn
"d+==+": "c", # c is continuing curve but with gap, a backtracks
"d+==-": "d", # d is larger arc, b backtracks

# Right to left
"a-==+": "d", # d tight, b loops
"a-==-": "c", # c tight, a larger arc
"b-==+": "a", # a continues but gap, c backtracks
"b-==-": "b", # d backtracks, b larger arc
"c-==+": "c", # a backtracks, c larger arc
"c-==-": "d", # b backtracks, d continues with gap
"d-==+": "a", # a tight, c larger arc
"d-==-": "a", # a tight turn, c bad coil

# Up
"a=++=": "d", # d tight, a big arc
"a=+-=": "c", # b loops, c tight
"b=++=": "b", # b big arc, c backtracks
"b=+-=": "a", # d backtracks, a continue with break
"c=++=": "a", # a tight, d loops
"c=+-=": "b", # c big arc, b tight
"d=++=": "c", # c continue with break, b backtracks
"d=+-=": "d", # d big arc, a backtracks

# Down
"a=-+=": "c", # b continue with gap, c backtracks
"a=--=": "d", # a big arc, d backtracks
"b=-+=": "d", # d tight, a loops
"b=--=": "c", # b big arc, c tight
"c=-+=": "b", # b backtracks, c big loop
"c=--=": "d", # d continue with gap, a backtracks
"d=-+=": "a", # d big arc, a tight
"d=--=": "b" # b tight, c loops
}


def plot(x,y,sprite,isDraw):
    for p in sprite:
        oled.rect(border+x*3*scale+p[0]*scale,border+63-y*3*scale-p[1]*scale, scale,scale, 1 if isDraw else 0)

def draw(c,r,a):
    plot(c,r,a,True)

def erase(c,r,a):
    plot(c,r,a,False)

def toSprite(spriteName):
    sprite = d
    if spriteName=='a': sprite = a
    if spriteName=='b': sprite = b
    if spriteName=='c': sprite = c
    return sprite
```

```
def toOffset(offsetSymbol):
    result = 0
    if offsetSymbol=='-': result = -1
    if offsetSymbol=='+': result = +1
    return result

def fromOffset(dx,dy):
    sym = "-=+"
    return sym[dx+1] + sym[dy+1]

def deltaToSeq(dx,dy):
    if dx==1: seq = lrSeq
    if dx==-1: seq = rlSeq
    if dy==1: seq = duSeq
    if dy==-1: seq = udSeq
    return seq

def toCode(spriteName, dx,dy):
    seq = deltaToSeq(dx,dy)
    pos = seq.find(spriteName)
    return seq[pos:pos+3]

def initSnakeLR(x,y):
    global snake, dx, dy
    dx,dy=+1,0
    snake = []

    for i in range(4): # Go backward so the rightmost is head.
        j = i*3
        code = lrSeq[j:j+3]
        snake.append( (x, y, code) ) # x,y is virtual head. Sprites might be drawn at an offset
        x -= 1

def initSnakeRL(x,y):
    global snake, dx, dy
    dx,dy=-1,0
    snake = []
    for i in range(4): # Go backward so the leftmost is head.
        j = i*3
        code = rlSeq[j:j+3]
        snake.append( (x, y, code) ) # x,y is virtual head. Sprites might be drawn at an offset
        x += 1

def initSnakeUp(x,y):
    global snake, dx, dy
    dx,dy=0,+1
    snake = []
    for i in range(4): # Go Downward so the bottom most is head.
        j = i*3
        code = duSeq[j:j+3]
        snake.append( (x, y, code) ) # x,y is virtual head. Sprites might be drawn at an offset
        y -= 1

def initSnakeDown(x,y):
    global snake, dx, dy
    dx,dy=0,-1
    snake = []
```

```
    for i in range(4): # Go upward so the topmost is head.
        j = i*3
        code = udSeq[j:j+3]
        snake.append( (x, y, code) ) # x,y is virtual head. Sprites might be drawn at an offset
        y += 1

def plotSeg(x,y, segCode, isDraw): # segCode is like d--, x,y is virtual position before offsets
    print("plotting", segCode, "at", x,y, "Apple at", appleX, appleY)
    spriteName = segCode[0] # grab the sprite name (ie d)
    sprite = toSprite(spriteName)
    x += toOffset(segCode[1])
    y += toOffset(segCode[2])
    plot( x, y, sprite, isDraw )

def drawSeg(x,y, segCode):
    plotSeg(x,y, segCode, True)

def eraseSeg(x,y, segCode):
    plotSeg(x,y, segCode, False)

def drawSnake():
    for s in snake:
        # each s is virtual x,y position followed by the segCode
        drawSeg(s[0], s[1], s[2])
    oled.show()

def ChopTail():
    global snake
    #-- Remove tail --
    tailIndex = len(snake)-1
    tail = snake[tailIndex] # d--, x, y
    eraseSeg(tail[0],tail[1], tail[2])
    snake.pop(tailIndex)

def CheckWalls():
    global isDead
    head = snake[0]
    headX, headY = head[0],head[1]
    if headX<=0: isDead=True
    if headY<=0: isDead=True
    if headX>=gameWidth: isDead=True
    if headY>=gameHeight: isDead=True
    if isDead: print("IS DEAD!")

def DrawWalls():
    oled.rect(0,0,arenaWidth+2, arenaHeight+1, 1) #OLED height is 1 pixel too short :-(

def moveSnake(dx,dy):
    global snake, isDead

    # Each direction has its own sequence of sprites
    seq = deltaToSeq(dx,dy)
    print("MoveSnake. dx,dy", dx, dy)

    #-- New Head --
    curHead = snake[0]
    curHeadCode = curHead[2]
```

```
    curSeg = curHeadCode[0]
    nuX = curHead[0] + dx
    nuY = curHead[1] + dy
    pos = seq.rfind(curSeg) # seq is listed head to tail, so to find new head we need to go backward in the seq.
    nuHeadCode = seq[pos-3:pos]

    nuHead = (nuX, nuY, nuHeadCode)
    print("curHead", curHead, " --> nuHeadCode", nuHeadCode)

    if isInSnake(nuX,nuY): # Ran into self
        isDead=True

    #-- Append New Head --
    snake.insert(0, nuHead)

    #-- Head coincides with apple? --
    if abs(nuX-appleX)<=1 and abs(nuY-appleY)<=1:
    #if nuX==appleX and nuY==appleY:
        print("Ate apple at ",appleX, appleY)
        drawApple(0) # erase the apple we just ate
        # Don't erase tail so snake becomes one segment longer after eating apple
        # Also, create a new apple (outside the snake)
        randomApple()
        drawApple(1) # draw new apple
        SpeedUp()
    else:
        if not isDead:
            ChopTail() # Normal path is to erase tail as snake slithers around

    #draw new head after erasing apple
    drawSeg(nuX, nuY, nuHeadCode)

    oled.show()

def buttonPressedHandle(p):
    global startWasPressed
    startWasPressed = True

def setupUI():
    global buttonRight, buttonLeft, buttonUp, buttonDown, buttonStart

    buttonRight = Pin(13, Pin.IN, Pin.PULL_UP)
    buttonLeft = Pin(14, Pin.IN, Pin.PULL_UP)
    buttonDown = Pin(15, Pin.IN, Pin.PULL_UP)
    buttonUp = Pin(12, Pin.IN, Pin.PULL_UP)

    buttonStart = Pin(27, Pin.IN, Pin.PULL_UP)
    buttonStart.irq(trigger=Pin.IRQ_FALLING, handler=buttonPressedHandle)

def changeDir(nuDx, nuDy):
    global snake, dx, dy

    if dx==nuDx and dy==nuDy:
        return

    print("changeDir. BEGIN from", dx, dy, "to", nuDx, nuDy)
```

```
    headSeg = snake[0] #x,y,spriteCode
    headSpriteName = headSeg[2][0] # take first char of sprite code -> sprite name
    print("changeDir. headSeg", headSeg, "headSpriteName", headSpriteName)
    key = headSpriteName + fromOffset(dx,dy) + fromOffset(nuDx, nuDy)
    print("changeDir. key", key)

    if key in turnTable:
        nuSpriteName = turnTable[key]
        print("changeDir. Turntable value is nuSpriteName", nuSpriteName)
        nuCode = toCode(nuSpriteName, nuDx, nuDy)
        nuX, nuY = headSeg[0]+nuDx, headSeg[1]+nuDy
        print("changeDir. nuX,nuY,nuCode", nuX,nuY,nuCode)
        snake.insert(0, (nuX,nuY,nuCode))
        drawSeg(nuX, nuY, nuCode) # new head is created due to turning, so draw this new head!

        ChopTail()
        dx,dy=nuDx,nuDy

def _map(x, in_min, in_max, out_min, out_max):
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min

def dirToDeltas(dir):
    results = (0,0)
    if dir==0: results = (0,+1)
    if dir==1: results = (+1,0)
    if dir==2: results = (0,-1)
    if dir==3: results = (-1,0)
    return results


def CheckButtons():
    if buttonLeft.value()==False:  changeDir(-1,0)
    if buttonRight.value()==False:  changeDir(1,0)
    if buttonUp.value()==False: changeDir(0,1)
    if buttonDown.value()==False: changeDir(0,-1)

def isInSnake(x,y):
    for seg in snake:
        if x==seg[0] and y==seg[1]:
            return True
    return False

def randomApple():
    global appleX, appleY
    hasCollision = True
    while hasCollision:
        appleX, appleY = randrange(1,gameWidth), randrange(1,gameHeight)
        hasCollision = isInSnake(appleX, appleY)
    print("New apple is at", appleX, appleY )

def drawApple(color):
    x,y=appleX, appleY
    p = [-1,+1] # position offset
    appleSize = 3*scale
    oled.rect(border+x*3*scale+p[0]*scale,border+63-y*3*scale-p[1]*scale, appleSize,appleSize, color)
#   oled.rect(border+x*3*scale+p[0]*scale,border+63-y*3*scale-p[1]*scale, scale,scale, 1 if isDraw else 0)
```

```
def CenteredText(msg):
    hOffset = int((WIDTH - 7*len(msg))/2)
    oled.fill(0)
    oled.text(msg, hOffset, int(HEIGHT/2))
    oled.show()

def AreYouReady():
    global startWasPressed
    isBlank=False
    startWasPressed = False
    while not startWasPressed:
        oled.fill(0)
        if not isBlank: CenteredText("Are you ready?  ")
        sleep(1)
        CenteredText("PICO SNAKE")
        oled.show()
        sleep(.5)
        isBlank = not isBlank

def GameOver():
    global startWasPressed


    print("gameover")
    startWasPressed = False
    while not startWasPressed:
        buzzer = PWM(Pin(9))
        buzzer.freq(500)
        buzzer.duty_u16(1000)
        sleep(1)
        buzzer.duty_u16(0)
        led = Pin(8, Pin.OUT)
        led.value(1)
        CenteredText("GAME OVER")
        sleep(1)
        CenteredText("1941012877")
        sleep(0.5)
        CenteredText("1941012382")
        sleep(1)
        CenteredText("IOT PROJECT")
        sleep(2)

        led.value(0)

        #-- draw end game --
        oled.fill(0)
        DrawWalls()
        drawApple(1)
        drawSnake()
        sleep(.7)

    # wait for button release
    while buttonStart.value() == 0:
        sleep(0.5)

    global isDead
    isDead = False;
```

```
def SpeedUp():
    global spd
    spd -= 0.01
    if spd<0: spd=0

def main():
    sleep(1)
    i2c=I2C(0,sda=Pin(16), scl=Pin(17), freq=400000)
    global oled
    oled = SSD1306_I2C(128, 64, i2c)

    setupUI()

    oled.fill(0)
    x=int(WIDTH/3/scale/2)
    y=int(HEIGHT/3/scale/2)
    global snakeX, snakeY

    while True:
        AreYouReady()

        snakeX, snakeY = int(WIDTH/3/scale/2), int(HEIGHT/3/scale/2)

        #initSnakeRL(x,y)
        initSnakeLR(x,y)
        #initSnakeUp(x,y)
        #initSnakeDown(x,y)

        oled.fill(0)
        randomApple()
        drawApple(1)

        while not isDead:
            DrawWalls() # redraw walls because screen is too short. Snake actually overlaps with bottom wall :-(
            CheckWalls()
            if not isDead:
                moveSnake(dx,dy)
                sleep(spd)
                CheckButtons()

        GameOver()

main()
```

THE END .

IOT MID-TERM Project

Pallav Raj

Kirti Shree