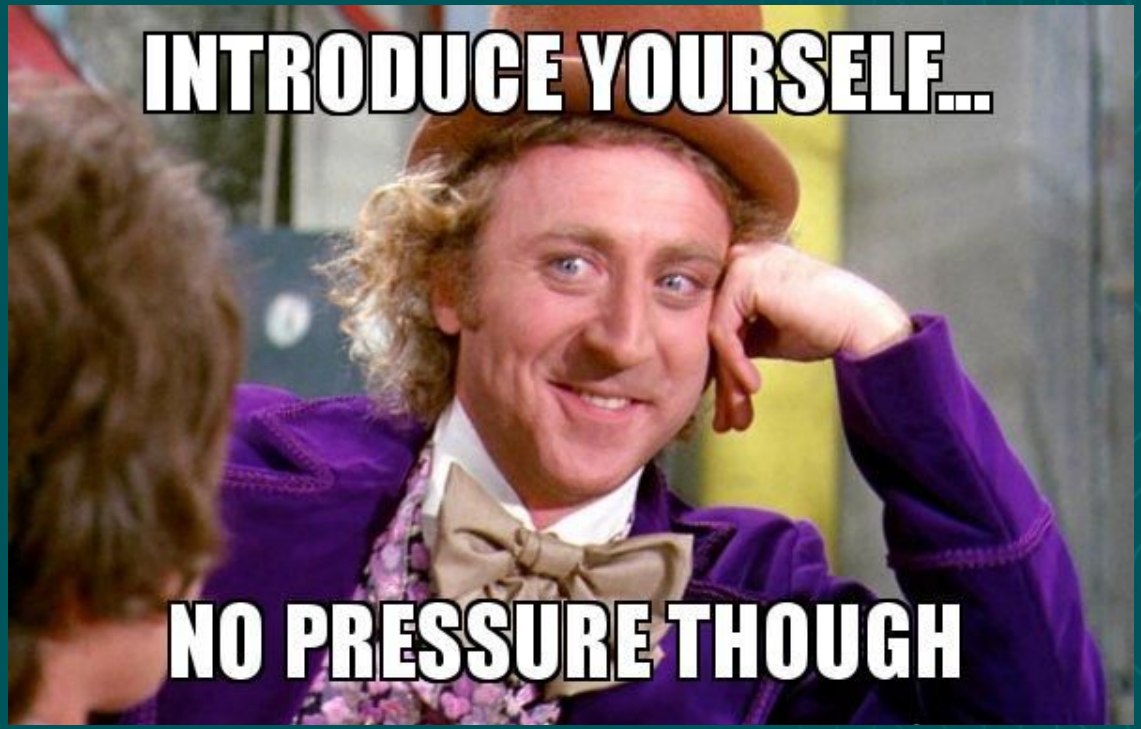
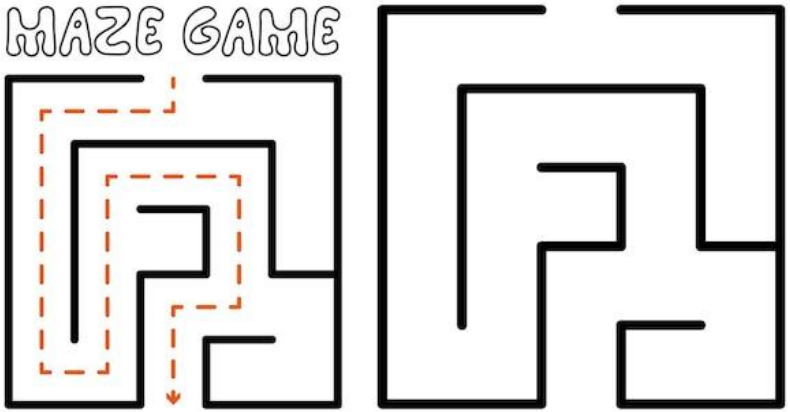
Game Design
(Session 1)

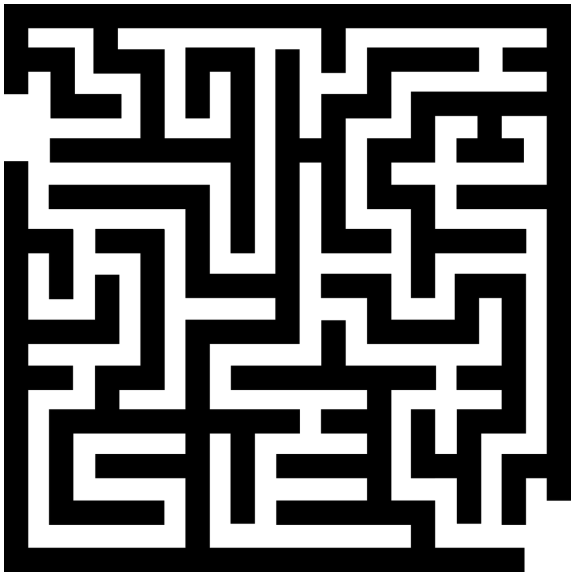
Introductions

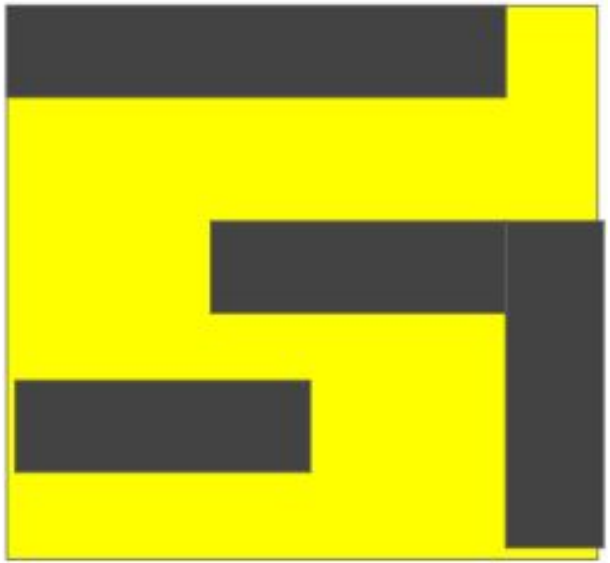


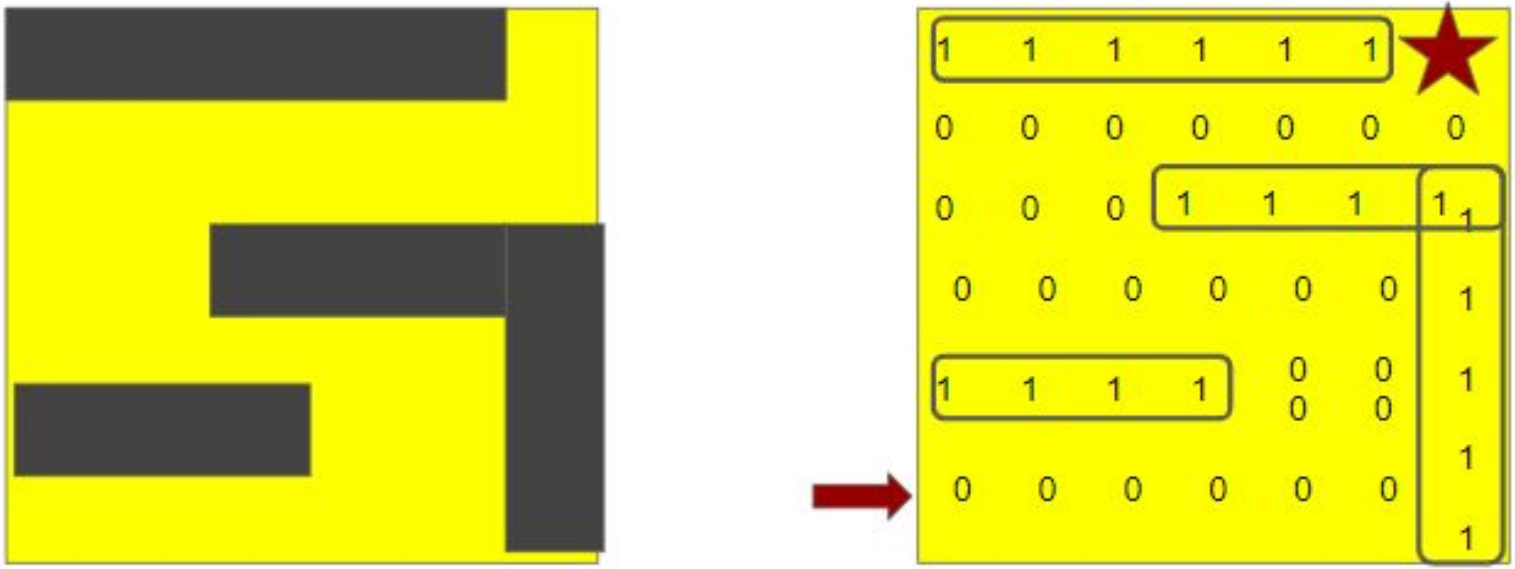
Maze Game


MAZE GAME

How to Represent Black and White Path






1 1 1 1 1 1
0 0 0 0 0 0 0
0 0 0 1 1 1 1
0 0 0 0 0 0
1 1 1 1
0 0 0 0 0 0

Random Maze Result

```
[[0. 1. 1. 0. 1. 1. 0. 1. 0. 1.]
 [0. 1. 1. 0. 1. 1. 0. 1. 0. 0.]
 [0. 1. 1. 0. 0. 1. 1. 1. 1. 1.]
 [1. 1. 0. 1. 1. 0. 1. 1. 0. 1.]
 [1. 1. 0. 0. 1. 1. 1. 1. 1. 1.]
 [0. 1. 1. 1. 0. 0. 1. 1. 0. 0.]
 [0. 1. 1. 0. 1. 0. 0. 1. 1. 1.]
 [0. 1. 1. 1. 0. 0. 1. 1. 1. 1.]
 [1. 1. 1. 0. 1. 0. 1. 0. 1. 0.]
 [1. 1. 0. 1. 1. 1. 0. 0. 1. 1.]]
```

Level 1:

Create maze board with 0 and 1s, of size 10x10 - randomly distributed

```
# first initialize maze with all 0 entries. we will modify later.
maze = np.zeros(shape=(10, 10))
print_maze(maze)
# how to access each element of matrix.
```

Level 1:

Create maze board with 0 and 1s, of size 10x10 - randomly distributed

```
# first initialize maze with all 0 entries. we will modify later.
maze = np.zeros(shape=(10, 10))
print_maze(maze)
# how to access each element of matrix.
```

```
for i in range(0, 10):
    for j in range(0, 10):
# since we will choose the walls to be random. how to write logic ?
```

Level 1:

Create maze board with 0 and 1s, of size 10x10 - randomly distributed

```
# first initialize maze with all 0 entries. we will modify later.
maze = np.zeros(shape=(10, 10))
print_maze(maze)
# how to access each element of matrix.
```

```
for i in range(0, 10):
    for j in range(0, 10):
# since we will choose the walls to be random. how to write logic ?
```

```
"some logic"
```

Get your Start and End point

```
# get start point
# we have 10X10 Maze. So for start we need to choose any 8 out of 10 columns.
# similarly for end points.
```

Get your Start and End point

```
# get start point
# we have 10X10 Maze. So for start we need to choose any 8 out of 10 columns.
# similarly for end points.
```

```
randomstart = random.randint(1, 8)
randomend = random.randint(1, 8)
# after getting the random column we need to get actual position of start and end.
```

Get your Start and End point

```
# get start point
# we have 10X10 Maze. So for start we need to choose any 8 out of 10 columns.
# similarly for end points.
```

```
randomstart = random.randint(1, 8)
randomend = random.randint(1, 8)
# after getting the random column we need to get actual position of start and end.
```

```
start = np.array([0, randomstart])
end = np.array([9, randomend])
# check if start and end is correct.
```

Get your Start and End point

```
# get start point
# we have 10X10 Maze. So for start we need to choose any 8 out of 10 columns.
# similarly for end points.
```

```
randomstart = random.randint(1, 8)
randomend = random.randint(1, 8)
# after getting the random column we need to get actual position of start and end.
```

```
start = np.array([0, randomstart])
end = np.array([9, randomend])
# check if start and end is correct.
```

```
"print start point"
"print end point"
# start to be denoted as 8.
# end to be denoted as 5.
```

Get your Start and End point

```
# get start point
# we have 10X10 Maze. So for start we need to choose any 8 out of 10 columns.
# similarly for end points.
```

```
randomstart = random.randint(1, 8)
randomend = random.randint(1, 8)
# after getting the random column we need to get actual position of start and end.
```

```
start = np.array([0, randomstart])
end = np.array([9, randomend])
# check if start and end is correct.
```

```
"print start point"
"print end point"
# start to be denoted as 8.
# end to be denoted as 5.
```

```
print(maze)
```

Result

```
start point is : [0 7]
end point is :  [9 7]
```

Result

```
start point is : [0 7]
end point is :  [9 7]
```

Print the Maze with Start and End points

```
from colorama import Fore
```

Print the Maze with Start and End points

```
from colorama import Fore
```

```
35  def print_maze(maze):
36  # access each element of maze.
```

Print the Maze with Start and End points

```
from colorama import Fore
```

```
35  def print_maze(maze):
36  # access each element of maze.
```

```
37      for i in range(0, 10):
38          for j in range(0, 10):
39  # if its 0 means free space Replace it with __
```

Print the Maze with Start and End points

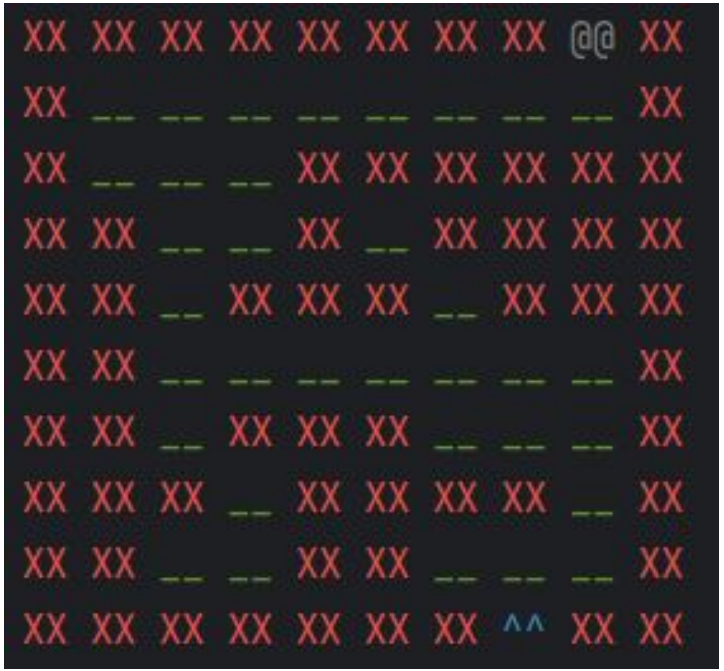
```
from colorama import Fore
```

```
35  def print_maze(maze):
36  # access each element of maze.
```

```
37      for i in range(0, 10):
38          for j in range(0, 10):
39  # if its 0 means free space Replace it with __
```

```
39  # if its 0 means free space Replace it with __
40   "somecode"
41  # 1 means walls. Replace it with XX.
42  "somecode"
43  # 8 means start point. Replace it with @@.
44  "somecode"
45  # 5 means end point. Replace it with ^^
46  "somecode"
47          print()
```

Result will look like...



How to reach the end ?

1. Capture the keys.

- A for moving left
- D for moving right
- W for moving up
- S for moving down

How to reach the end ?

```
# ask user for a s d w keys ?
```

How to reach the end ?

```
# ask user for a s d w keys ?
```

```
# if pressed key = some value, you need to update the maze.
    if (chr == "a") or (chr == "A"):
        print("move left - decrement col ")
        "do something"
    elif (chr == "d") or (chr == "D"):
        print("-------------")
        "do something"
    elif (chr == "s") or (chr == "S"):
        print("-------------")
        "do something"
    elif (chr == "w") or (chr == "W"):
        print("-------------")
        "do something"
```

How to reach the end ?

```
# ask user for a s d w keys ?
```

```
# if pressed key = some value, you need to update the maze.
    if (chr == "a") or (chr == "A"):
        print("move left - decrement col ")
        "do something"
    elif (chr == "d") or (chr == "D"):
        print("-------------")
        "do something"
    elif (chr == "s") or (chr == "S"):
        print("-------------")
        "do something"
    elif (chr == "w") or (chr == "W"):
        print("-------------")
        "do something"
```

```
# let's have a cancel button to exit the maze in between.
    elif (chr == "x") or (chr == "X"):
```

LOOPS

- Since you need to take input for every step till you reach at the end.

- Give it a try ?

Update the maze according to key Pressed?

```
def updatemaze (move):
    if move == "left":
        if start[1] != 0:
            if maze[start[0], start[1] - 1] != 1:
                start[1] = start[1] - 1
                maze[start[0], start[1] + 1] = 0
                maze[start[0], start[1]] = 8
```

Write code for Right?

Update the maze according to key Pressed?

```
if move == "up":
    if start[0] != 0:
        if maze[start[0] - 1, start[1]] != 1:
            start[0] = start[0] - 1
            maze[start[0] + 1, start[1]] = 0
            maze[start[0], start[1]] = 8
```

- **Write code for Down ?**

Congrats…
You have completed Random Maze Game

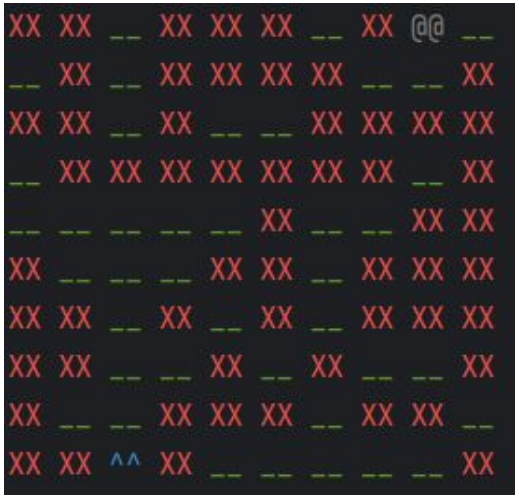
- Combine all the codes to get final output

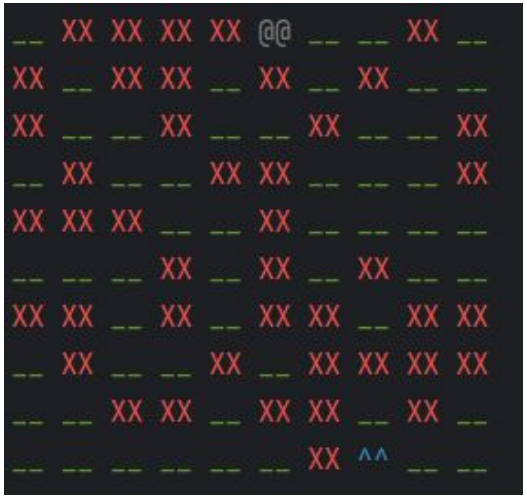
1. Create a random 10x10 Maze Board
2. Get your Start and End points
3. Print the Board in terms of XX and __
4. Capture the keypress
5. Update the Maze on keypress

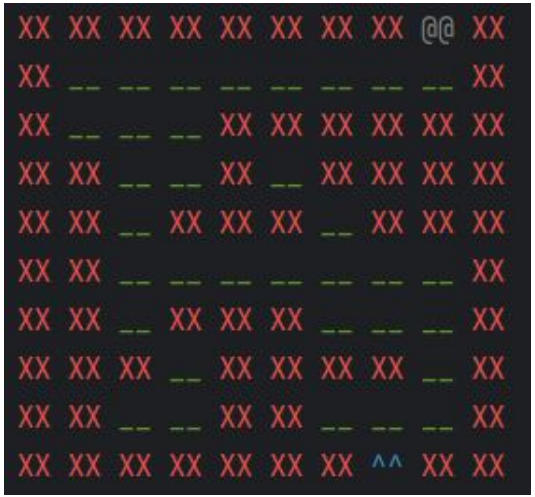
OUTPUT??


XX XX XX XX XX XX XX __ XX XX
XX XX XX XX __ __ __ __ XX XX
XX __ __ __ __ XX XX XX XX XX
XX __ __ XX XX XX __ XX XX XX
XX __ __ __ XX __ XX XX XX XX
XX __ __ __ __ __ XX XX XX XX
XX __ __ __ __ __ XX __ XX XX
XX XX __ XX __ XX XX __ __ XX
XX XX __ __ __ __ XX __ XX XX
XX XX @@ XX XX XX XX XX XX XX
yayyyy!! you won The Game

Did you find any problem??







Our Maze game design is A random.

- It means Guaranteed EXIT is not confirmed.
- How to Create Guaranteed exit ??
- TRY?

Let’s think while moving…

```
def adjacentcells(current):
#let have an empty list.. where u will push the adjacent of current cell.
    walls=[]
 "some logic"
            walls.append([current[0] - 1, current[1]])
```

Let’s think while moving...

```
def adjacentcells(current):
#let have an empty list.. where u will push the adjacent of current cell.
    walls=[]
 "some logic"
           walls.append([current[0] - 1, current[1]])
```

```
"some logic"
        walls.append([current[0], current[1]-1])
"some logic"
        walls.append([current[0]+1, current[1]])
"some logic"
        walls.append([current[0], current[1]+1])

    return walls
```

Loop 2:

Till u reach at the end, at every step you need to choose.

```
# before starting the loop let's make a current variable to keep track of position we are currently in.
current = start
# also tracking the position for previous cell.
previous = current
```

Loop 2:

Till u reach at the end, at every step you need to choose.

```
# before starting the loop let's make a current variable to keep track of position we are currently in.
current = start
# also tracking the position for previous cell.
previous = current
```

```
"initialize the loop till u reach to end"

    walls = adjacentcells(current)
    randomcell = previous
"choose a random wall from the set of adjacent walls. But remember do not choose the previous path."
```

Loop 2:

Till u reach at the end, at every step you need to choose.

```
# before starting the loop let's make a current variable to keep track of position we are currently in.
current = start
# also tracking the position for previous cell.
previous = current
```

```
"initialize the loop till u reach to end"

    walls = adjacentcells(current)
    randomcell = previous
"choose a random wall from the set of adjacent walls. But remember do not choose the previous path."
```

```
        "make the path if its not there."
        "update the rule"
```

Boundary for Maze

```
# at the outer edges there should be boundary like a square.
for i in range(0,10):
```

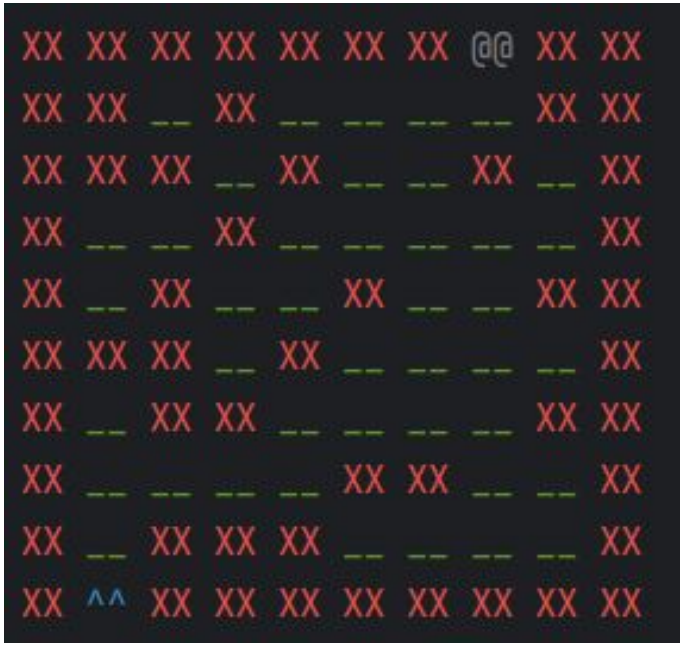
```
# remember there is a random start and end point at the periphery. Don't make them walls.
```

Final Maze??

- Combine the code for Adjacent walls, loops and boundary.

- Let’s Run the code.

Final output:



Announcements

Umm, Thank You, I Guess?

See you in the next session!

Hope you had fun :)

Game Design
(Session 2)

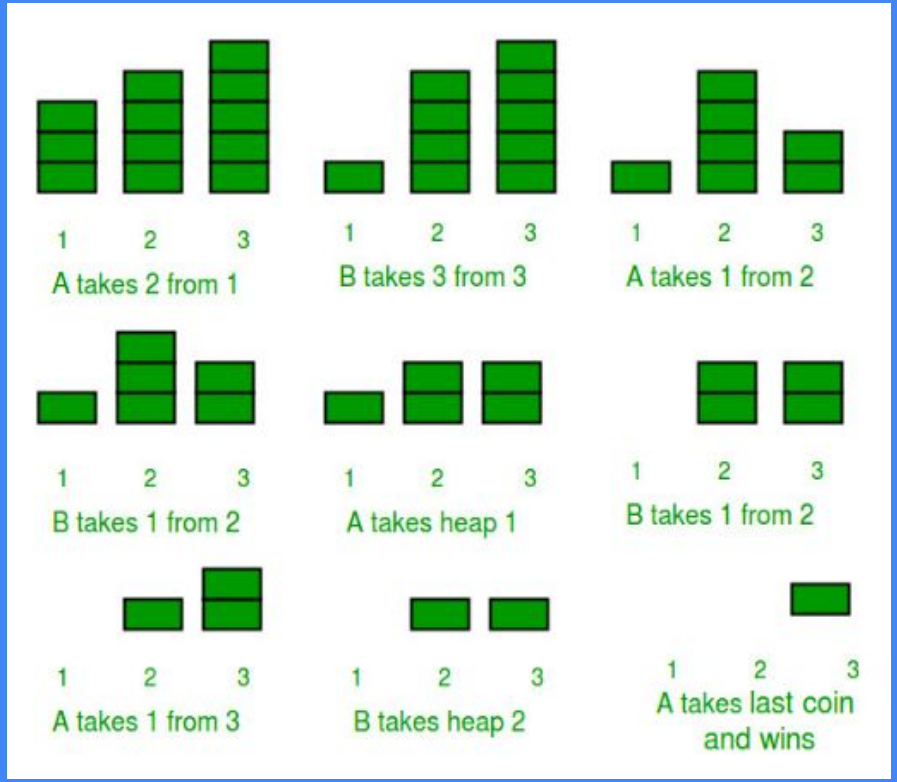
Dragons

Introduction

Given a number of piles in which each pile contains some numbers of stones/coins. In each turn, a player can choose only one pile and remove any number of stones (at least one) from that pile.

WHO WINS?

The player who takes away the last stone is the winner



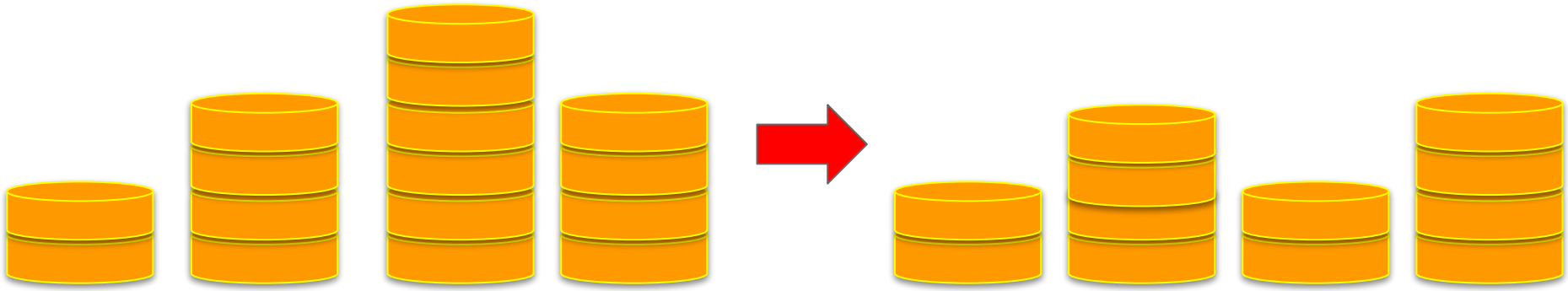
PHASE - 1

BUILDING A TWO-PLAYER VERSION

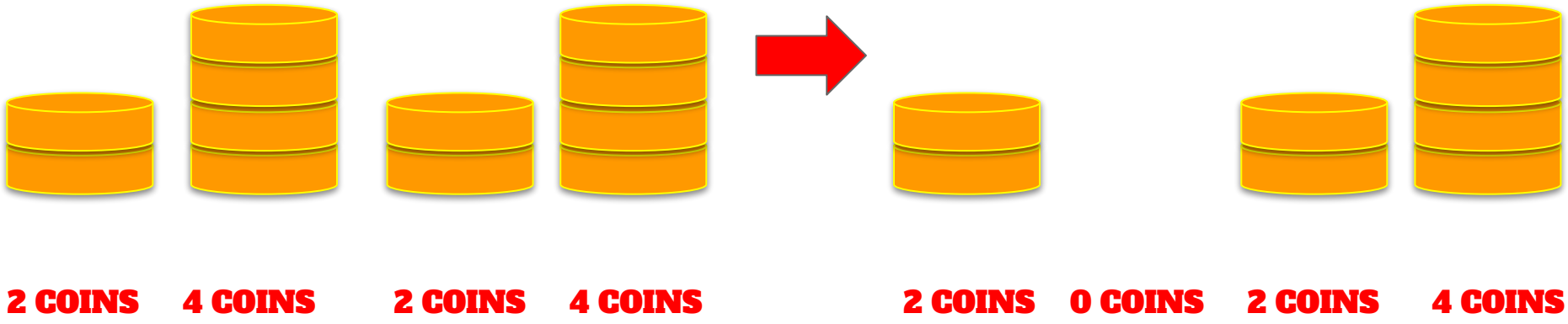
SAMPLE RUN - 1

2 COINS 4 COINS 6 COINS 4 COINS

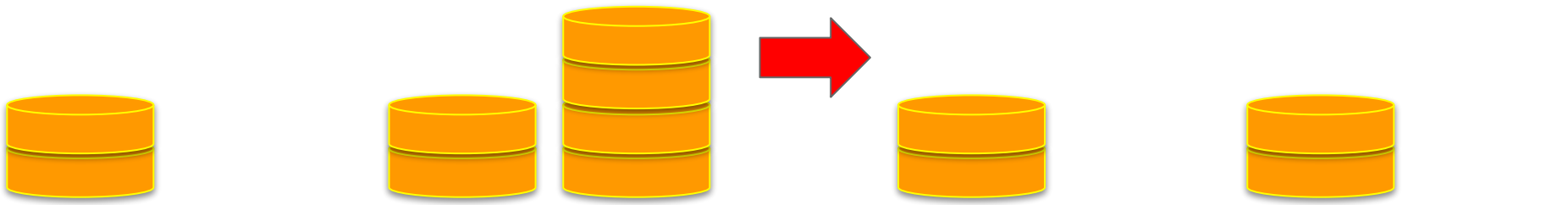
PLAYER 1 : MOVE: (3, 4) = FROM HEAP 3 , REMOVE 4 COINS



PLAYER 2 : MOVE: (2, 4) = FROM HEAP 2 , REMOVE 4 COINS

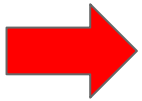


PLAYER 1 : MOVE: (4, 4) = FROM HEAP 4 , REMOVE 4 COINS



2 COINS 0 COINS 2 COINS 4 COINS 2 COINS 0 COINS 2 COINS 0 COINS

PLAYER 2 : MOVE: (1, 2) = FROM HEAP 1 , REMOVE 2 COINS



2 COINS 0 COINS 2 COINS 0 COINS 0 COINS 0 COINS 2 COINS 0 COINS

PLAYER 1 : MOVE: (3, 2) = FROM HEAP 3 , REMOVE 2 COINS

0 COINS 0 COINS 2 COINS 0 COINS

PLAYER 1 TAKES AWAY THE LAST COIN AND WINS THE GAME

0 COINS 0 COINS 0 COINS 0 COINS

PHASE - 2

PLAYING AGAINST THE COMPUTER

Simple Computer Opponent

How should the computer choose how many coins to remove from which heap ?

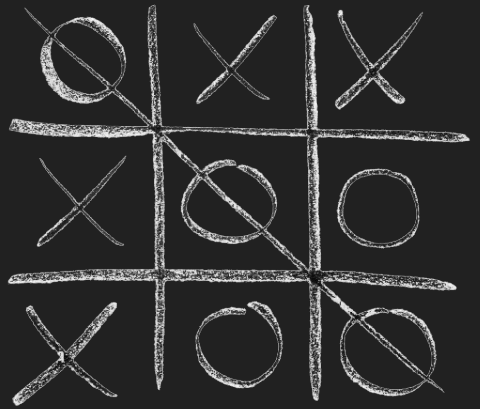
What's the least complicated strategy ?

FURTHER DISCUSSION

HOW TO MAKE BETTER/STRONGER COMPUTER OPPONENTS?

Game Design Session

Tic Tac Toe



1. Getting started with tkinter

What is GUI and tkinter?

- GUI stands for Graphical user interface. Unlike text based interfaces, GUI uses graphical and interactive components for the user to interact with.

- tkinter stands for Tk interface. It is an easy to use and standard GUI for Python.

Starting with tkinter

- Run the following command to ensure it is installed correctly.

python -m tkinter

- Create a blank tkinter window:

```
import tkinter as tk

window = tk.Tk()
window.mainloop()
```

Basics of tkinter

- There are 3 basic widgets in tkinter. These are Label, Button, Entry.
- **Labels -** It can be used to display text of various sizes and styles.

```python
import tkinter as tk

my_window = tk.Tk()
my_label = tk.Label(
    text='I am a label widget with custom properties',
    background='black',
    foreground='white',
    font=('Times New Roman', 20)
)
my_label.pack()
my_window.mainloop()
```

Basics of tkinter

- **Button:** It can be used to call a command when clicked.

```
1   import tkinter as tk
2
3   def press_button():
4       print('The button is pressed.')
5
6   window = tk.Tk()
7   button = tk.Button(text='Click me', command=press_button)
8   button.pack()
9   window.mainloop()
```

Basics of tkinter

- **Entry:** Interactive widget to get user input.

```python
import tkinter as tk

window = tk.Tk()
entry = tk.Entry()

def submit():
    print(entry.get())

button = tk.Button(text='Submit', command=submit)
entry.pack()
button.pack()
window.mainloop()
```

2. Let’s start designing the game

Design a two player game in terminal

Design a two player game using tkinter

Designing the game

Step 1: Create the grid.

Designing the game

Step 1: Create the grid.

```
1  import tkinter as tk
2
3  window = tk.Tk()
4  window.resizable(False, False)
5  window.title("Tic Tac Toe")
6
7  tk.Label(window, text="Tic Tac Toe", font=('Ariel', 25)).pack()
8
```

Designing the game

Step 2: Create functions for reset button and play area

Designing the game

Step 2: Create functions for reset button and play area

```
4   def reset_button(button):
5       button.configure(text="", bg='white')
6
7   def create_XO_point(x, y):
8       button = tk.Button(play_area, text="", width=10, height=5)
9       button.grid(row=x, column=y)
10      return button
```

Designing the game

Step 3: Add a Play Area

Designing the game

Step 3: Add a Play Area

```
14  play_area = tk.Frame(window, width=300, height=300, bg='white')
15  play_area.pack(pady=10, padx=10)
16
17  XO_buttons = []
18  for x in range(1, 4):
19      for y in range(1, 4):
20          button = create_XO_point(x, y)
21          XO_buttons.append((button))
22
23  window.mainloop()
```

Designing the game

Step 4: Make it dynamic

Designing the game

Step 4: Make it dynamic

```
19  current_chr = "X"
20  X_points = []
21  O_points = []
22
23  def set_point(x, y, button, value):
24      global current_chr
25      if not value:
26          button.configure(text=current_chr, bg='snow', fg='black')
27          value = current_chr
28          if current_chr == "X":
29              X_points.append((x, y))
30              current_chr = "O"
31          else:
32              O_points.append((x, y))
33              current_chr = "X"
34
```

Designing the game

Step 4: Make it dynamic - where else are we suppose to add values variable?

```
19  current_chr = "X"
20  X_points = []
21  O_points = []
22
23  def set_point(x, y, button, value):
24      global current_chr
25      if not value:
26          button.configure(text=current_chr, bg='snow', fg='black')
27          value = current_chr
28          if current_chr == "X":
29              X_points.append((x, y))
30              current_chr = "O"
31          else:
32              O_points.append((x, y))
33              current_chr = "X"
34
```

Designing the game

Step 5: Check win function

Designing the game

Step 5: Check win function

```
for possibility in winning_possibilities:
    if all(point in X_points for point in possibility):
        print("X won!")
        reset_points()
        return
    elif all(point in O_points for point in possibility):
        print("O won!")
        reset_points()
        return

if len(X_points) + len(O_points) == 9:
    print("Draw!")
    reset_points()
```

Designing the game

Step 6: Reset points

Designing the game

Step 6: Reset points

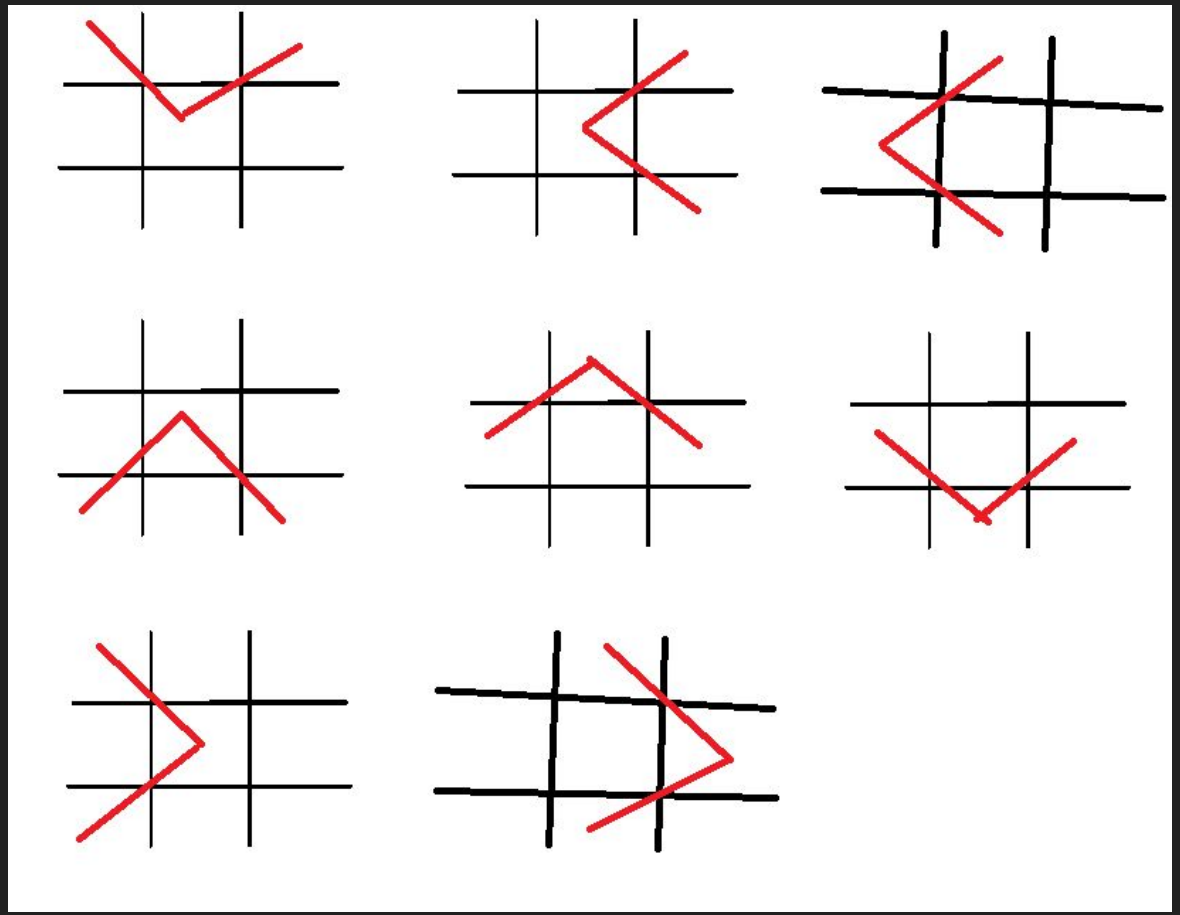
```
3 usages
def reset_points():
    for button, value in XO_buttons:
        reset_button(button, value)
    X_points.clear()
    O_points.clear()
```

Step 7: Integrate

DONE?

3. Variations of Tic Tac Toe

TWIST IN THE GAME……..



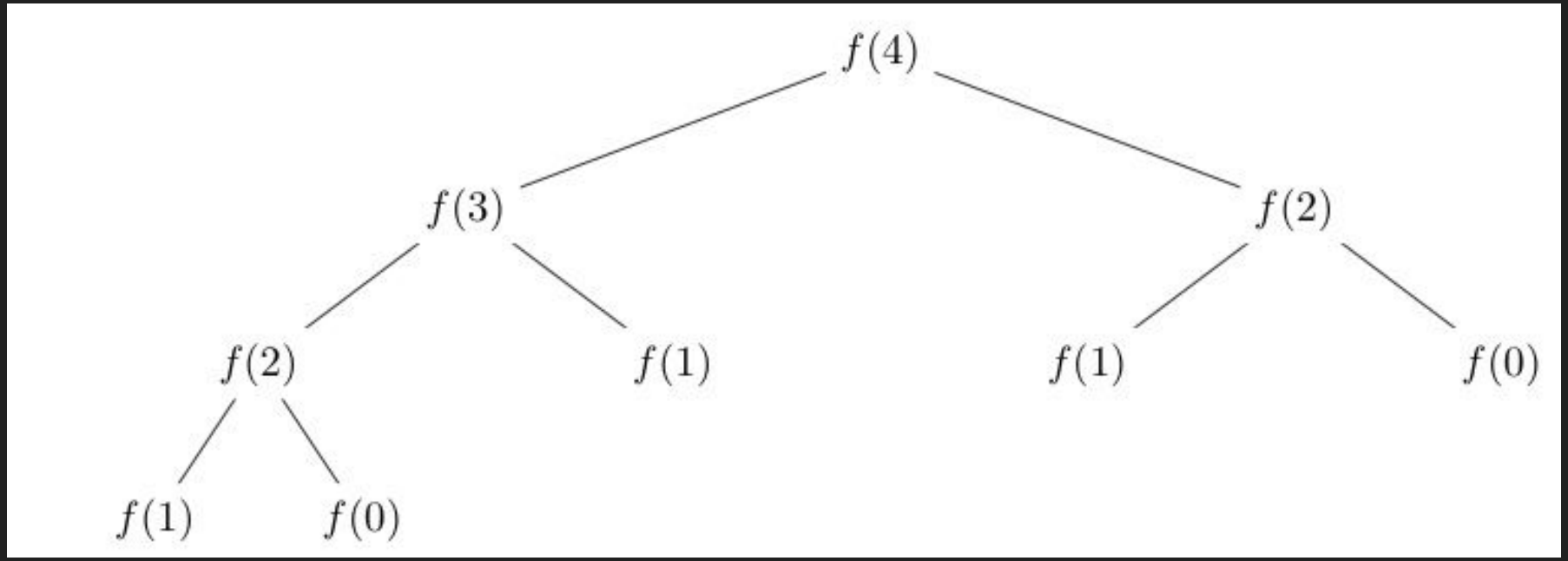
5 x 5 tic tac toe

4. Winning Strategy

But imp concepts before that


Me
Is this a good meme?
Me
Is this a good meme?
imgflip.com

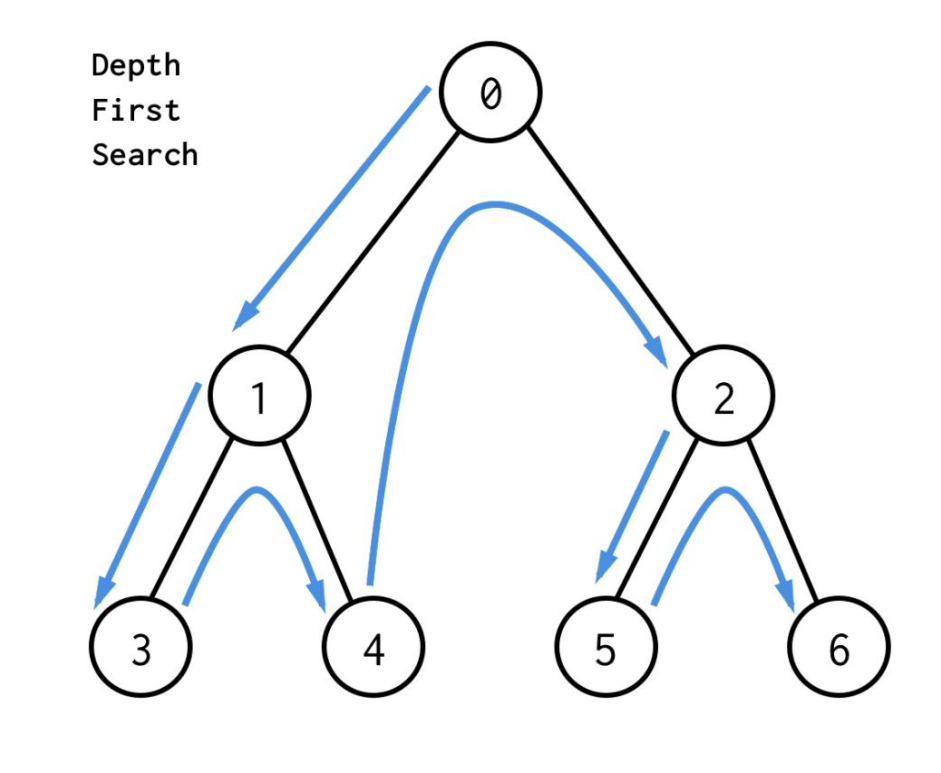
```python
def fibonacci(n):
    if n <= 1:
        return n
    else:
        return(fibonacci(n-1) + fibonacci(n-2))
```

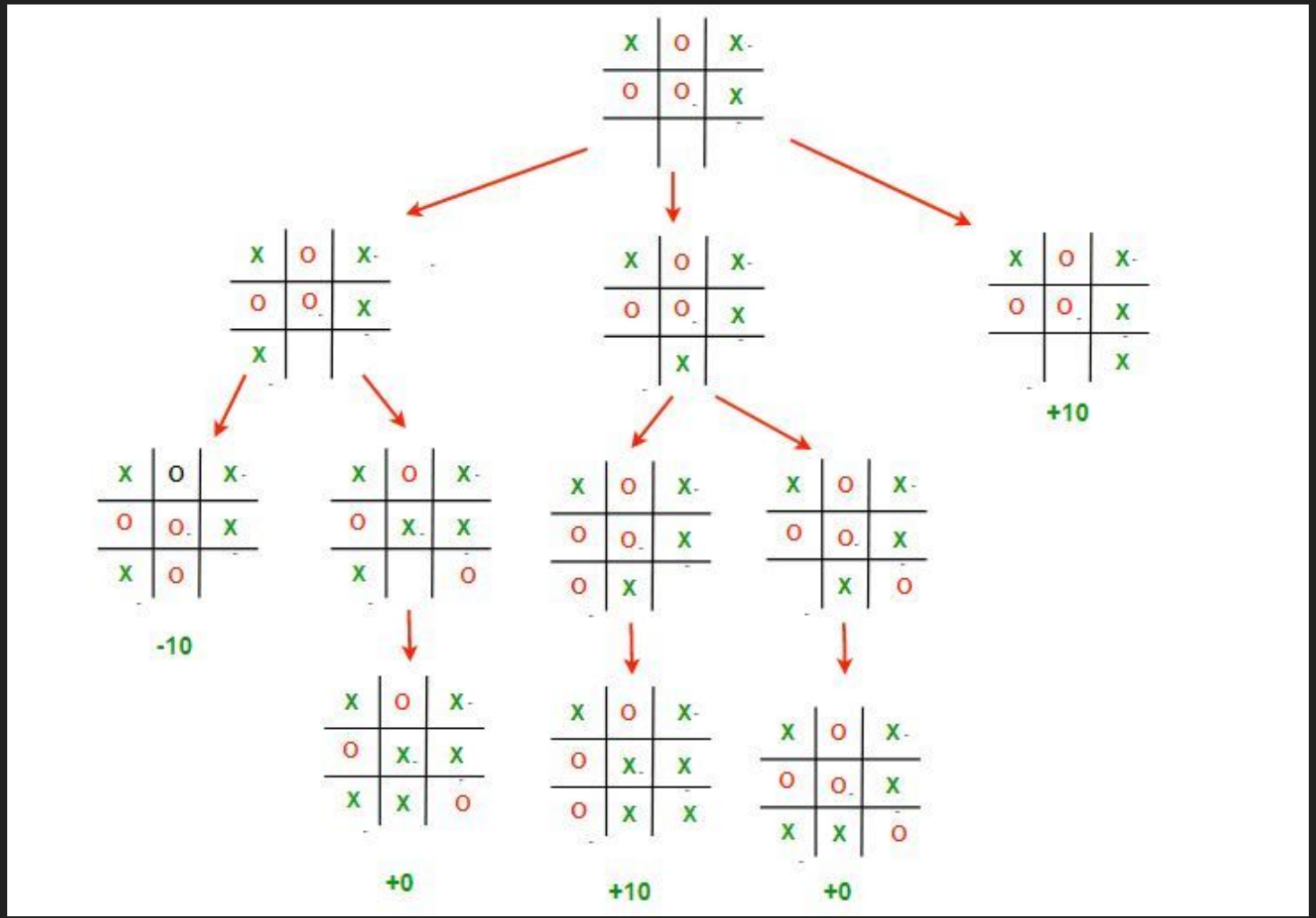

f(4)
f(3)
f(2)
f(2)
f(1)
f(1)
f(0)
f(1)
f(0)


When you finally
understand recursion:

Depth First Search

Depth First Search



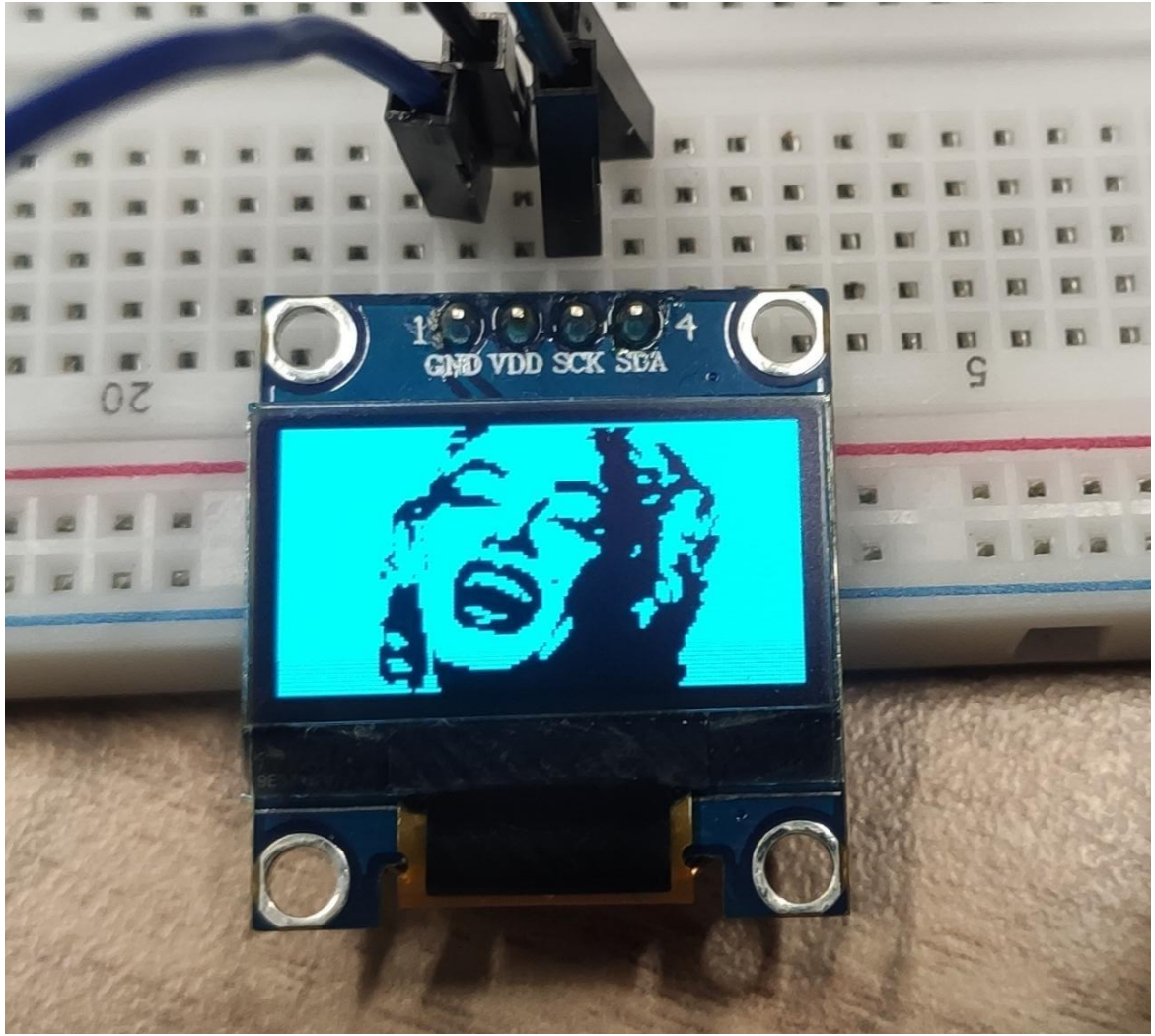

+10
-10
+0
+10
+0

Umm, Thank You, I Guess?

See you in the next session!

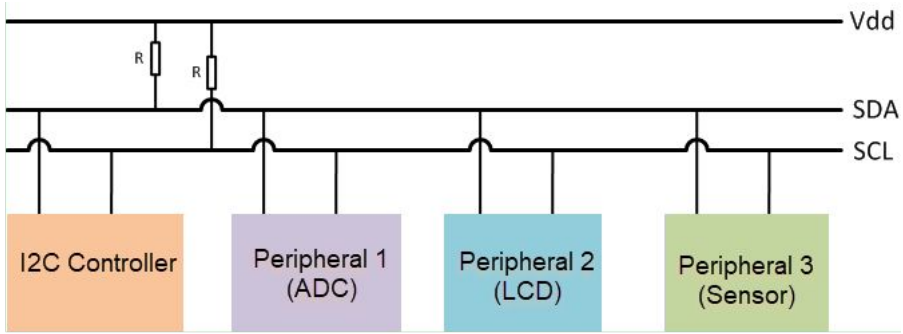
Hope you had fun :)

Game Design
(Session 4)


1
4
GND VDD SCK SDA

About I2C (Inter-Integrated Circuit)

A communication protocol used to communicate with each other using just two wires: a serial data line (**SDA**) and a serial clock line (**SCL**), connected in a bus configuration. Design involves a master device, which initiates and controls the communication, and slave devices, which respond to commands and provide data.



About I2C Adapter

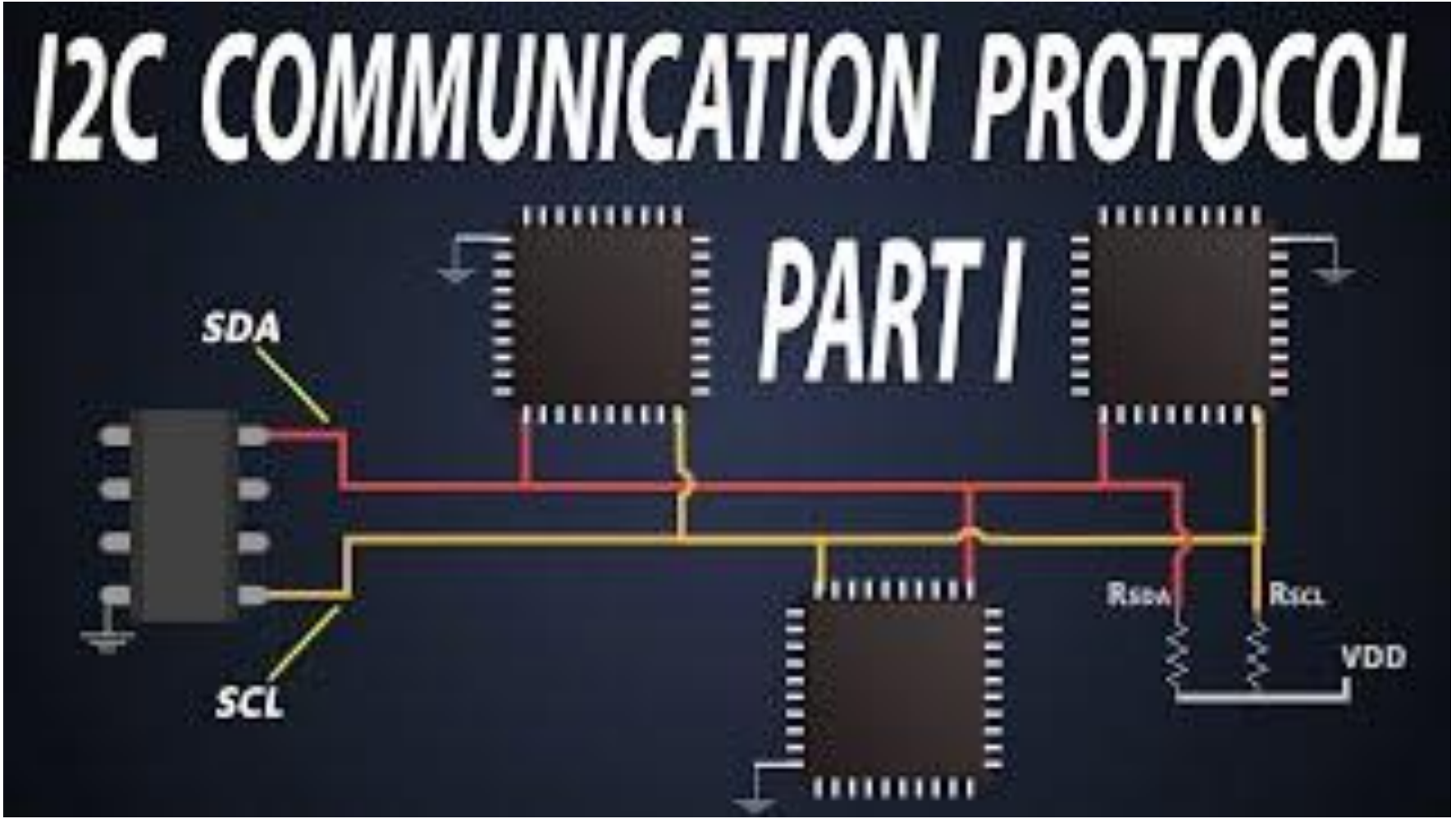
Using I2C Adapter, we can use this protocol in our circuit.

Due to the Master-Slave relationship and the bus configuration, the process of connecting and communicating with multiple devices is simplified by using a minimal number of wires.

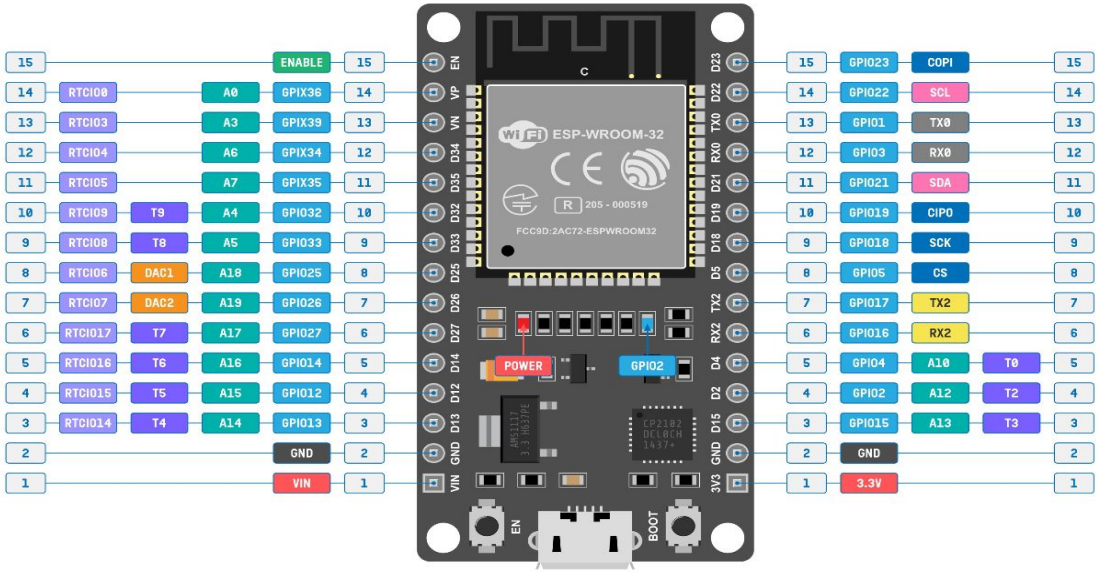
It also makes it easier if in future we want to add more devices to the circuit.

Hence, in the case of the LCD, we will use this more often.

I2C ADAPTER


I2C COMMUNICATION PROTOCOL
PART I
SDA
SCL
RSDA
RSCL
VDD

DOIT ESP32 DEVKIT V1 PINOUT



PHYSICAL PIN	POSITIVE SUPPLY	DAC OUTPUTS	SPI PINS
CONTROL PINS	GROUND SUPPLY	TOUCH INPUTS	UART PINS
GPIO PINS	ADC INPUTS	I2C PINS	EXCLUDED PINS

- GPIO pins **34**, **35**, **36** and **39** are input only.
- **TX0** and **RX0** (**Serial0**) are used for serial programming.
- **TX2** and **RX2** can be accessed as **Serial2**.
- Default SPI is VSPI. Both VSPI and HSPI pins can be set to any GPIO pins.
- All GPIO pins support PWM and interrupts.
- Builtin LED is connected to **GPIO2**.
- Some GPIO pins are used for interfacing flash memory and thus are not shown.

Rev. 0.1, 17-12-2022
Design: Vishnu Mohanan

https://rb.gy/vkd21

```ino
#include <Wire.h>
#include <U8g2lib.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

U8G2_SH1106_128X64_NONAME_F_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE);

void setup(void) {
  u8g2.begin();
}

void loop(void) {
  u8g2.firstPage();
  do {
    u8g2.setFont(u8g2_font_ncenB14_tr);
    u8g2.drawStr(3, 35, "Hello World!");
  } while (u8g2.nextPage());
}
```

What are potentiometers?

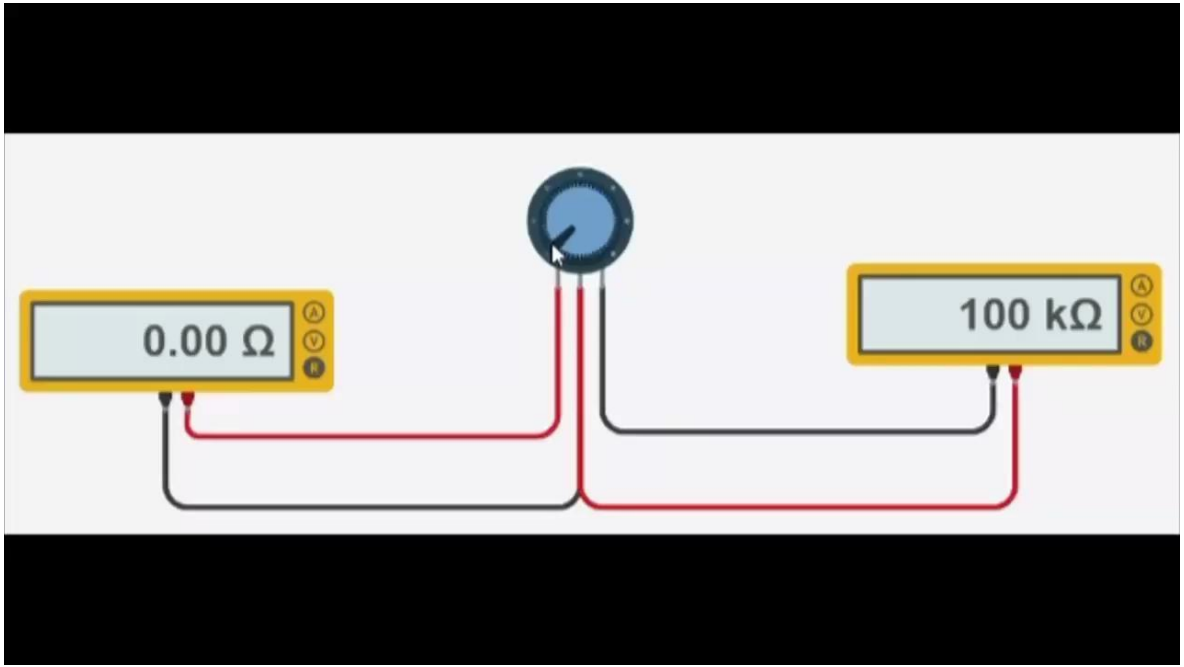
- *"A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider."* - Wikipedia

The Virgin Standard Pot

The Chad Fender Pure Vintage 250K Split Shaft Potentiometer

But what are potentiometers actually !?
(I am bored of this theory)

- In short: Most Common Variable Resistor (more in electronics class)
- The three terminals: The end two terminals have fixed resistance
    - The terminal in between (in relation to other) provide variable resistance



Make circuit using OLED, Potentiometer and ESP32

Necessary Libraries and Constants

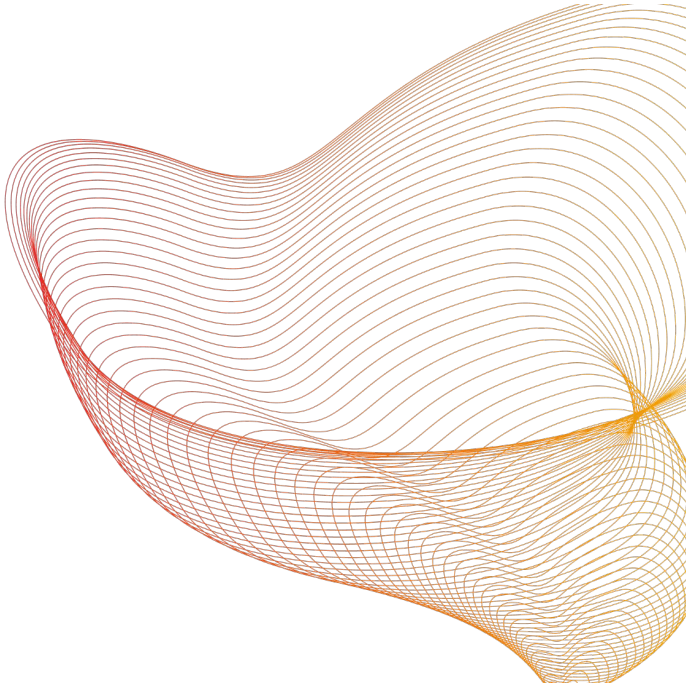
01 Defines constants for screen dimensions and pin assignments

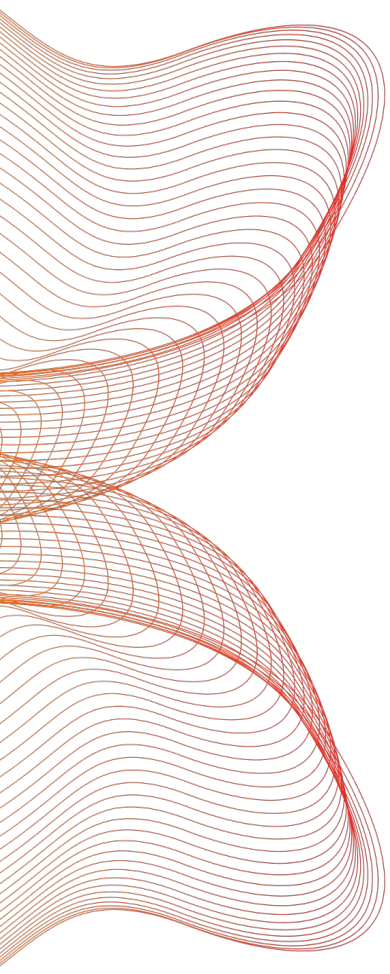
02 The code includes necessary libraries

Global Variables

01 Includes paddle positions, ball position and speed, game score, and game over flag

02 Declared to store various game parameters and states





Setup Function

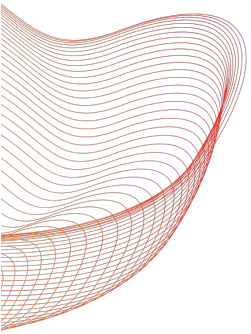
- Pin mode is set for the potentiometer input
- The display is initialized and configured
- Serial communication is initialized for printing the score
- The welcome message is displayed on the OLED screen
- A delay of 2 seconds is added before starting the game
- The random number generator is seeded with an analog reading

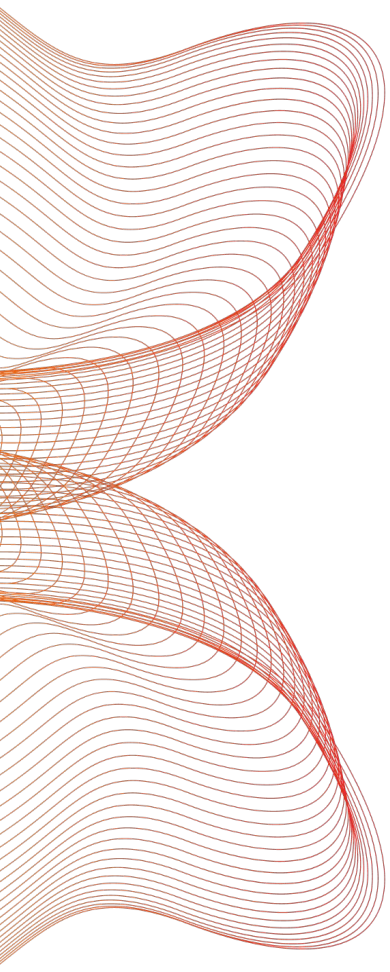
Loop Function

If the game is over, a game over message is displayed on the screen

If there is input available from the serial monitor and it is 'y', the game is reset

Otherwise, the function returns and stops updating the game





Potentiometer Input

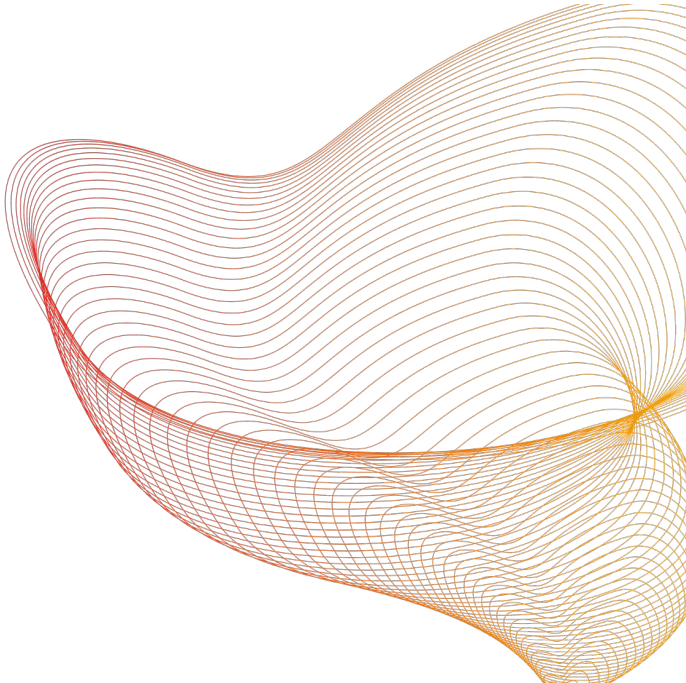
01 The value is mapped to the screen height and the resulting value is constrained to stay within the screen bounds

02 The potentiometer value is read to determine the position of the first paddle (`paddle1Y`)

Second Paddle Positioning

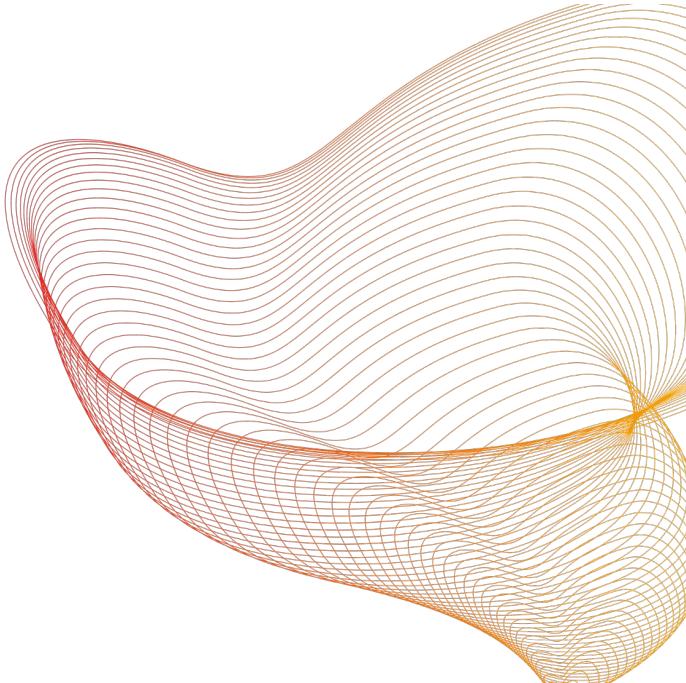
01 It adjusts its position to match the vertical movement of the ball

02 The second paddle (`paddle2Y`) is positioned relative to the ball's Y position



Ball Movement

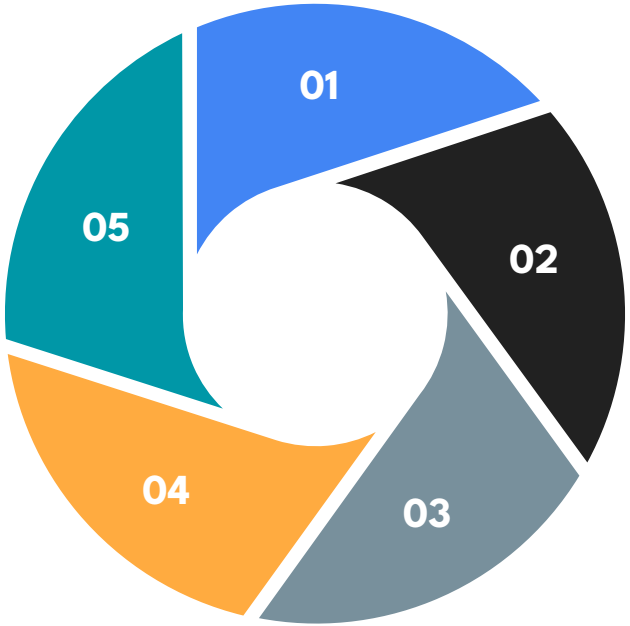
- The ball's position is updated based on its current position and speed



Collision Detection

If the ball hits the top or bottom edge of the screen, the ball's Y direction is reversed

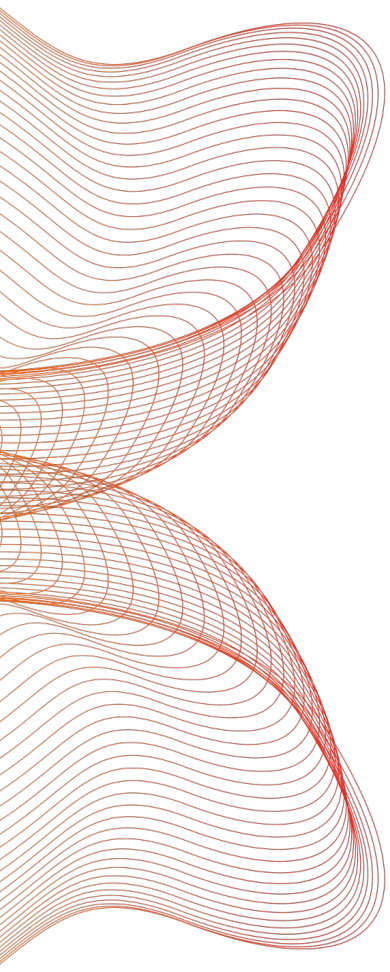
If the ball hits the first paddle (`paddle1Y`), the ball's X direction is reversed, the score is incremented, and the score is printed to the serial monitor



Various game scenarios are handled through collision detection

If the ball hits the left or right edge of the screen, the ball speed is reset and the ball is moved back to the left or right edge

If the ball hits the second paddle (`paddle2Y`), the ball's X direction is reversed



Additional Functions

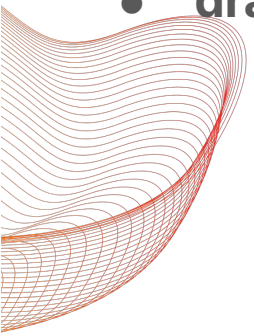
01 `resetBall()` - Resets the ball's position and speed to the center of the screen and assigns random initial speeds for X and Y directions

02 `displayWelcomeMessage()` - Displays the welcome message on the OLED screen

HINT - Functions to Google

- setPowerSave
- clearBuffer
- clearBuffer
- sendBuffer
- setCursor
- drawBox

- Serial.read()
- randomSeed
- analogRead
- constrain
- map
- abs



Umm, Thank You, I Guess?

See you in the next session!

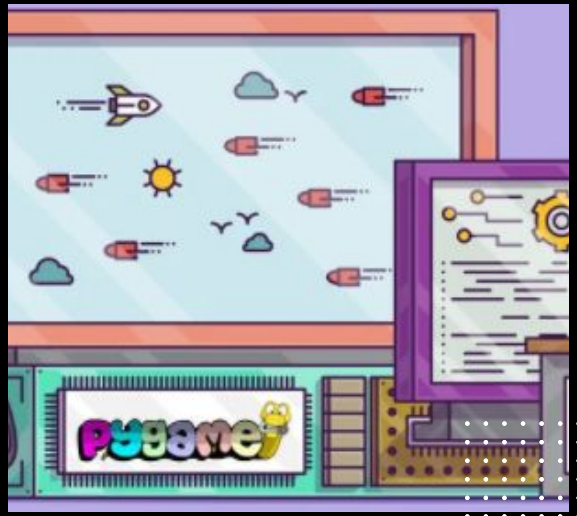
Hope you had fun :)

Game Design
(Session 5)

GUI in Python

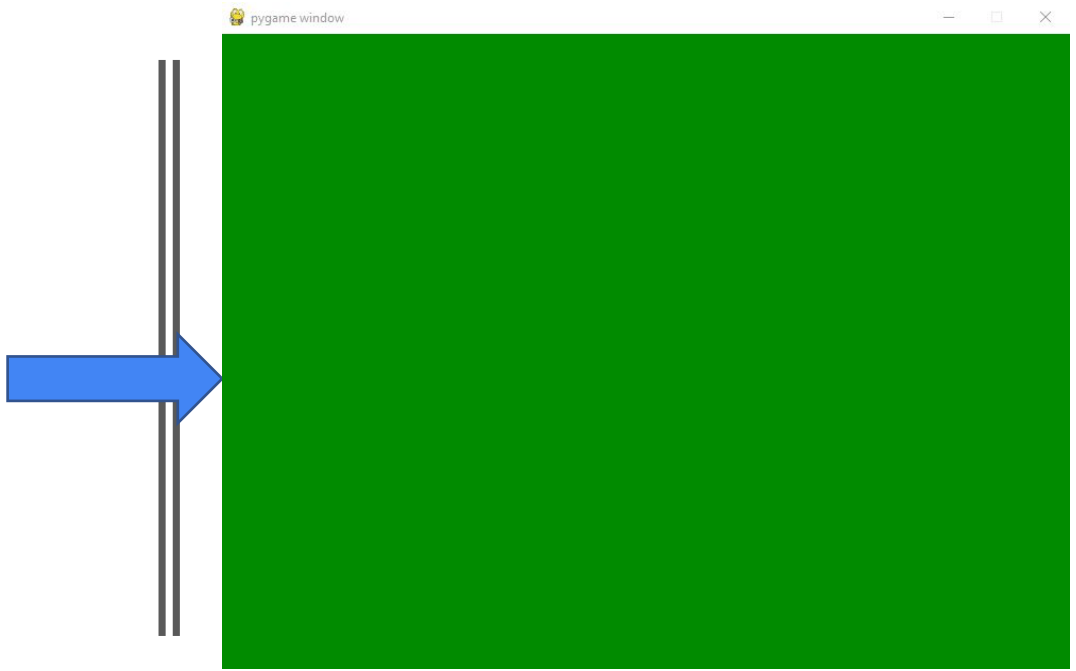
- Pygame is a cross-platform set of Python modules which is used to create video games.
- Pygame is suitable to create applications that can be wrapped in a standalone executable
- It consists of computer graphics and sound libraries designed to be used with the Python programming language.

- Installation through Pycharm: File > Settings > Project Interpreter > +
    - Search pygame and click on **install package** button



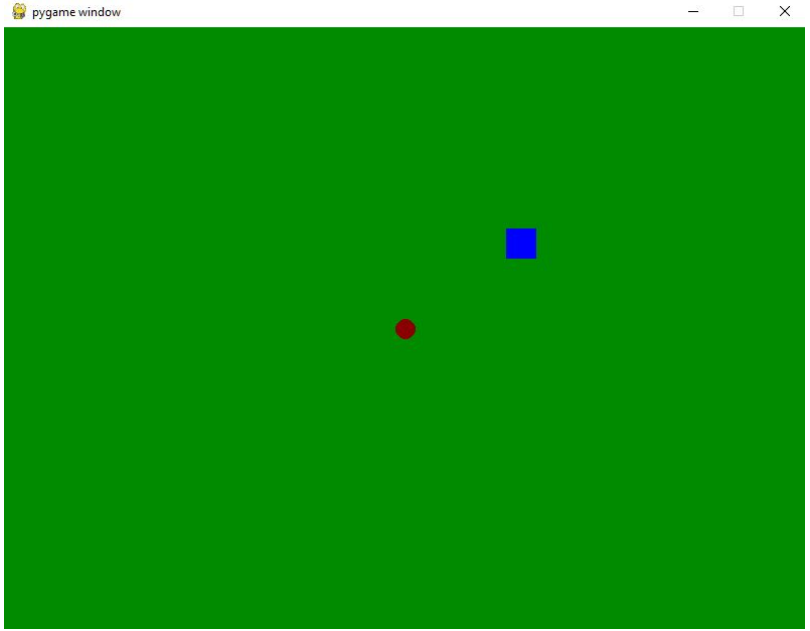
Starting with pygame: Game window

```python
# Import and initialize the pygame library
import pygame
pygame.init()

# Set up the drawing window
width = 800
height = 600
screen = pygame.display.set_mode((width, height))
background_color = (0,139,0)

# Run until the user asks to quit
game_over = False
while not game_over:

    # Did the user click the window close button?
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            game_over = True

    # Fill the background with white
    screen.fill(background_color)

    # Update the display
    pygame.display.update()

# Done! Time to quit.
pygame.quit()
```



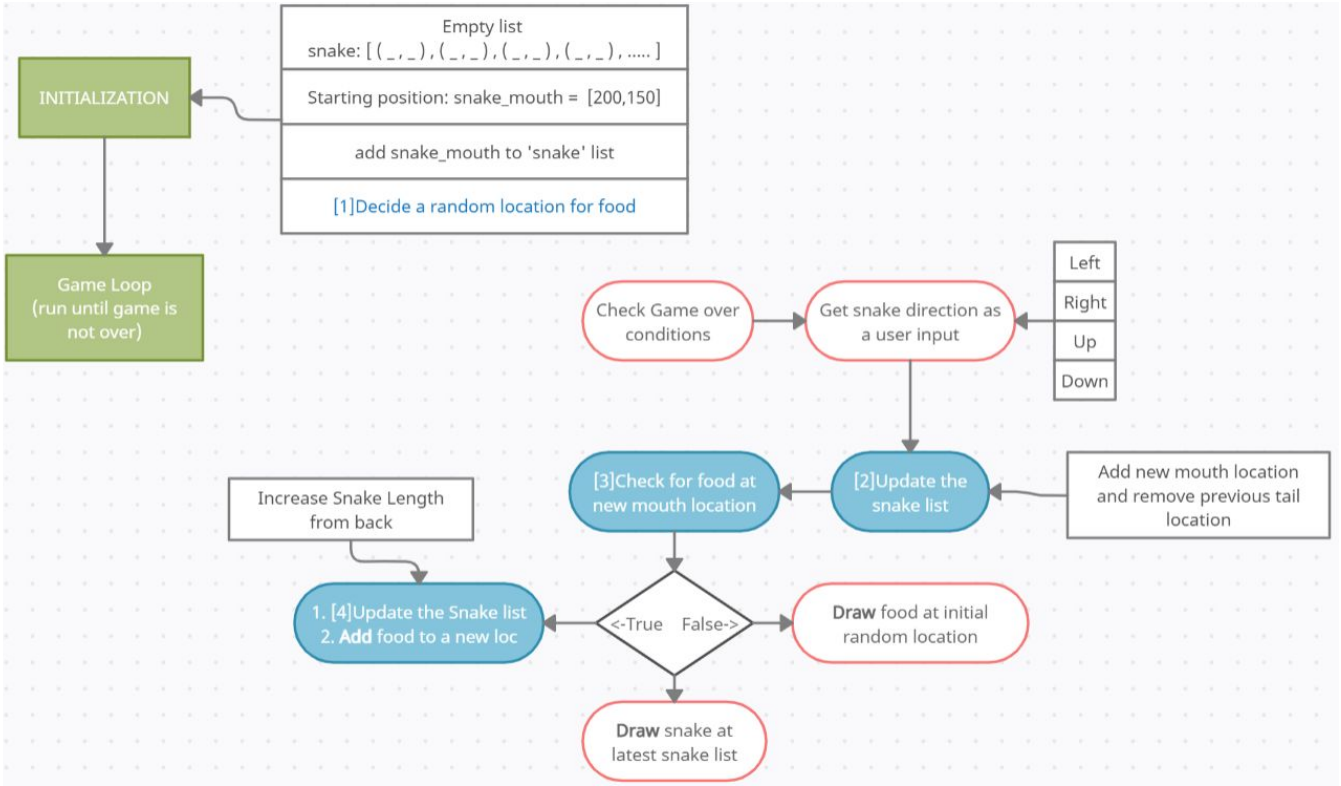
Starting with pygame: Adding objects

```
10  food_color = (139,0,0)
11  snake_color=(0,0,255)
12  snake_height = 30
13  snake_width = 30
14
15  # Run until the user asks to quit
16  game_over = False
17  while not game_over:
18
19      for event in pygame.event.get():
20          # Did the user click the window close button?
21          if event.type == pygame.QUIT:
22              game_over = True
23
24      # Fill the background with white
25      screen.fill(background_color)
26
27      # Draw objects on screen
28      pygame.draw.circle(screen, food_color, (400, 300), 10) # food
29      pygame.draw.rect(screen, snake_color, [500,200, snake_width, snake_height])
```



- We will first define a game flow with functions as black boxes
- Later these functions will be defined according to our use

Game Flow


INITIALIZATION
Empty list
snake: [(_ , _) , (_ , _) , (_ , _) , (_ , _) , ...]
Starting position: snake_mouth = [200,150]
add snake_mouth to 'snake' list
[1]Decide a random location for food
Game Loop
(run until game is not over)
Check Game over conditions
Get snake direction as a user input
Left
Right
Up
Down
[3]Check for food at new mouth location
[2]Update the snake list
Add new mouth location and remove previous tail location
Increase Snake Length from back
1. [4]Update the Snake list
2. Add food to a new loc
<-True
False->
Draw food at initial random location
Draw snake at latest snake list

Functions

- In the flow chart, we used four user defined functions
  1. Decide random location for food

```
def add_food():
    food_x =random.randint(0,width)
    food_y =random.randint(0,height)
    return [food_x,food_y]

food_posn = add_food()
```

Functions

- In the flow chart, we used four user defined functions
  2. Update snake list as a result of movement

```
def move(direction,snake):
    updated_mouth = move_cell(direction,snake[0])
    new_snake=[]
    new_snake.append(updated_mouth)
    for i in range(len(snake)-1):
        new_snake.append(snake[i])
    return new_snake
```

Functions

- In the flow chart, we used four user defined functions
  3. Check if food is present at mouth location

```
def is_food_present(snake,food_posn):
    snake_mouth = snake[0]
    snake_mouth_center = [snake_mouth[0],snake_mouth[1]]
    distance = ((food_posn[0]-snake_mouth_center[0])**2 + (food_posn[1]-snake_mouth_center[1])**2)**0.5
    print(snake_mouth)
    if distance<12:
        return True
    else:
        return False
```

Functions

- In the flow chart, we used four user defined functions
  4. Update snake list as a result of eating food

```
def update_food(snake,direction):
    last_block = snake[-1]
    if direction == 1:
        new_block = [last_block[0] + snake_width, last_block[1]]
    if direction == 2:
        new_block = [last_block[0] - snake_width, last_block[1]]
    if direction == 3:
        new_block = [last_block[0], last_block[1] + snake_height]
    if direction == 4:
        new_block = [last_block[0], last_block[1] - snake_height]
    snake.append(new_block)

    return snake
```

While loop

- Use the flowchart and the functions to write on your own!

While loop

```
while not game_over:
    display.fill(background_color)
    for event in pygame.event.get():
        if event.type==pygame.QUIT:
            game_over=True
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_LEFT:
                direction=1
            if event.key == pygame.K_RIGHT:
                direction=2
            if event.key == pygame.K_UP:
                direction=3
            if event.key == pygame.K_DOWN:
                direction=4
    if(is_food):
        print(snake)
    pygame.display.update()
    fpsClock.tick(FPS)
```