10.014 1D Project, Fall 2023

Write a software/game to solve an interesting problem

Team F06K - Mini Arcade Games

Members:
Jatlyson Ang
Brandon Kim
Zachary Lee
Tok Jing Huan
Alson Tay
Brian Lim

Documentation

Air Hockey
It is required for this game to use the time library and the turtle library.

Turtle Library:
The turtle library was used to create the objects in the game such as the paddle, ball and displaying the score.

Time Library:
The time library ensures that the game does not start immediately upon running the code and after each player scores. It is also used to ensure that the terminal does not close immediately once either player scores == 3.

Set up screen

```
    screen = turtle.Screen()
    screen.title("Air Hockey")
    screen.bgcolor("black")
    screen.setup(width=700, height=500)
```

Creating Paddles

```
    paddle_a = turtle.Turtle()
    paddle_a.speed(0)
    paddle_a.shape("square")
    paddle_a.color("white")
    paddle_a.shapesize(stretch_wid=5,  stretch_len=1)   #  Make  the  paddle
vertical
    paddle_a.penup()
    paddle_a.goto(-340, 0)
    paddle_a_speed = 30  # Adjust the speed of paddle A
```

The code above is duplicated to create another paddle for player B.

Creating the Ball

```
    ball = turtle.Turtle()
    ball.speed(200)  # Adjust the speed of the ball
```

```
    ball.shape("square")
    ball.color("white")
    ball.penup()
    ball.goto(0, 0)
    ball.dx = 5  # Increase the speed of the ball in the x-direction
    ball.dy = 5  # Increase the speed of the ball in the y-direction
```

Scoring

A point is added to the player's score when the ball goes through the other player's paddle. Thereafter, the score is shown and the game resumes after 1 second. When a player scores 3 points, the game ends.

Main Game Loop:

This is the loop that runs the game. The pseudocode is as follows.

1. Initialize scores for both as zero.
2. For each time the ball goes past either paddle
    a. Score += 1 for the respective player
    b. When the score == 3 for either player, the game ends.
3. Display the final score for 3 seconds before closing the terminal

```
# Scoring
if ball.xcor() > 300:
    score_a += 1
    if score_a == 3:
        score_display.clear()
        score_display.write(f"Player  A:  {score_a}   Player  B:  {score_b}",
align="center", font=("Courier", 24, "normal"))
        time.sleep(2)
        Break
    else:
        ball.goto(0, 0)
        ball.dx *= -1
        score_display.clear()
        score_display.write(f"Player  A:  {score_a}   Player  B:  {score_b}",
align="center", font=("Courier", 24, "normal"))
        time.sleep(1)
```

Paddle Functions:

These functions are used to move the paddles in the game. An if statement was used to ensure the paddle does not go out of the screen.

```
def paddle_a_up():
    y = paddle_a.ycor()
    y += paddle_a_speed
    if y + 50 < 265:  # Check if the y-coordinate is within the screen limits
        paddle_a.sety(y)

def paddle_a_down():
    y = paddle_a.ycor()
    y -= paddle_a_speed
    if y - 50 > -265:  # Check if the y-coordinate is within the screen limits
        paddle_a.sety(y)
```

LumberJack

1. *Libraries used:*

```
import turtle
import time
import random
```

Turtle Library:
The turtle library was used to create the objects in the game such as the logs and to display the score.

Time Library:
The time library ensures that the game difficulty increases as time passes (logs fall faster)

Random Library:
The random library generates random numbers

2. Constants used in the game:

```
global log_speed, score
delay = 0.1
log_speed = 5
```

These constants define the game's parameters. delay controls the speed of the game loop, and log_speed sets the initial speed of falling logs. Global log_speed, score is used to reset the game after it is over. (Reset the log speed and score to the original)

3. Setting up the screen

```
win = turtle.Screen()
win.title("lumberjack")
win.bgcolor("yellow")
win.setup(width=600, height=600)
win.tracer(0)
```

This block initializes the game window using the turtle module. It sets the title, background color, and dimensions of the window. Here, a turtle screen is created with a yellow background. The window is set to a size of 600x600 pixels tracer(0) turns off automatic screen updates, which can improve performance.

4. Creating the Lumberjack

```
lumberjack = turtle.Turtle()
lumberjack.speed(0)
lumberjack.shape("square")
lumberjack.color("darkblue")
```

```
lumberjack.shapesize(stretch_wid=1, stretch_len=2)
lumberjack.penup()
lumberjack.goto(0, -250)
```

This section creates the lumberjack turtle. It defines its appearance, color, size, and initial position at the bottom center of the screen. The lumberjack is a dark blue square shape (representing the lumberjack) located initially at the bottom center of the screen.

5. An empty list called logs is created to store the log turtles.

```
logs = [ ]
```

6. The variable score is initialized to zero to keep track of the player's score.

```
# Initialize the score
score = 0
```

7. This section sets up a turtle named score_display to show the player's score at the top of the screen.

```
score_display = turtle.Turtle()
score_display.speed(0)
score_display.color("black")
score_display.penup()
score_display.hideturtle()
score_display.goto(0, 260)
score_display.write("Score: {}".format(score), align="center", font=("Courier", 24, "normal"))
```

8. A turtle named game_over_display is created to display the "Game Over" message when needed.

```
game_over_display = turtle.Turtle()
game_over_display.speed(0)
game_over_display.color("red")
game_over_display.penup()
game_over_display.hideturtle()
game_over_display.goto(0, 0)
```

9. Function to create a log turtle

```
def create_log():
    log = turtle.Turtle()
    log.speed(0)
    log.shape("square")
    log.color("brown")
    log.shapesize(stretch_wid=1, stretch_len=random.uniform(1, 2))
    log.penup()
    x = random.randint(-290, 290)
    y = random.randint(100, 250)
```

```
    log.goto(x, y)
    logs.append(log)

# Create 6 initial log turtles
for _ in range(6):
    create_log()
```

This section defines a function create_log to create a log turtle with random size and position. The function is then called 6 times to initialize the logs list with 6 logs.

10. Functions for lumberjack movement

```
def move_left():
    x = lumberjack.xcor()
    if x > -280:
        lumberjack.setx(x - 20)

def move_right():
    x = lumberjack.xcor()
    if x < 280:
        lumberjack.setx(x + 20)
```

These functions, move_left and move_right, handle the left and right movement of the lumberjack when the corresponding arrow keys are pressed.

11. Set up keyboard bindings

```
win.listen()
win.onkey(move_left, "Left")
win.onkey(move_right, "Right")
```

These lines make the program listen for the left and right arrow key presses to invoke the corresponding movement functions.

12. Function for handling game over

```
def game_over():
    global score
    game_over_display.clear()
    game_over_display.write("Game Over", align="center", font=("Courier", 36,
"normal"))
    time.sleep(2)
    game_over_display.clear()
    score = 0
    reset_game()
```

13. Function to reset the game settings

```
def reset_game():
```

```
    global log_speed
    lumberjack.goto(0, -250)
    for log in logs:
        x = random.randint(-290, 290)
        y = random.randint(100, 250)
        log.goto(x, y)
        log.shapesize(stretch_wid=1, stretch_len=random.uniform(1, 2))
    log_speed = 5  # Reset log speed
    score_display.clear()
    score_display.write("Score: {}".format(score), align="center", font=("Courier", 24, "normal"))
```

For 12 and 13, these functions, game_over and reset_game, handle the game over scenario. The game_over function displays the "Game Over" message, waits for 2 seconds, clears the screen, resets the score to 0, and calls the reset_game function.

14. Main game loop

```
while True:
    win.update()

    # Move the logs
    for log in logs:
        y = log.ycor()
        x = log.xcor()
        log.sety(y - log_speed)
        log.setx(x + random.uniform(-20, 20))  # Random horizontal movement

        # Check for collision with the lumberjack
        if lumberjack.distance(log) < 20:
            game_over()

        # Check for logs reaching the bottom
        if log.ycor() < -290:
            y = random.randint(100, 250)
            x = random.randint(-290, 290)
            log.goto(x, y)
            log.shapesize(stretch_wid=1, stretch_len=random.uniform(1, 2))
            score += 1
            score_display.clear()
            score_display.write("Score: {}".format(score), align="center", font=("Courier", 24, "normal"))

        # Check for scoring
        if log.ycor() < lumberjack.ycor() < log.ycor() + 20 and log.xcor() - 20 < lumberjack.xcor() < log.xcor() + 20:
            score += 1
            score_display.clear()
            score_display.write("Score: {}".format(score), align="center", font=("Courier", 24, "normal"))

    time.sleep(delay)
```

The main game loop continuously updates the screen, moves the logs, checks for collisions and scoring, increases the log speed over time, and introduces a delay to control the game speed. The loop runs indefinitely, ensuring the game keeps running until manually terminated.

Space Shooter

It is required for this game to use the turtle, random and math library.

Determine directions:

get_direction(s1, s2)

This function takes in 2 sprites, s1 and s2. It then uses the .xcor() and y.cor() to return the x-coordinate and y-coordinate of s1 and s2 which is x1,x2 and y1,y2 respectively. It then uses the mathematical formula:
direction = arc tangent((y1-y2)/(x1-x2))
to determine the direction of s2 from s1.

Creating and setting sprite:

These functions are used to create different shapes for the various sprites as well as setting their movements and positions.

shooter_vertices
asteroid_vertices

These 2 variables were created, each consisting of a tuple of nested tuples, where each nested tuple was the coordinate of the vertices of the shapes of the player and the asteroids.
The method **.registershape("*name_of_shape*",*shape_vertices*)** was used to register both variables as shapes "player" and "asteroid" respectively.

```
class Sprite(turtle.Turtle):
    def __init__(self):
        turtle.Turtle.__init__(self)
```

A new class called sprite was created to apply the same attributes to all objects or sprites in the game which are:
self.speed(0) - Sprites do not move when created
self.penup() - Sprites do not draw a line behind them when moving

Setting Player:

player = Sprite() - Assigning attributes of new class to variable "player"
player.color("lightslateblue") - Changing color of the player sprite
player.shape("player") - Changing shape of player sprite to new shape created earlier

Setting Scoreboard:

pen = Sprite() - Assigning attributes of new class to variable "pen" which is the scoreboard
pen.color("white") - Changing color of the scoreboard
To set the scoreboard, a function **set_scoreboard(pen)** was created to position the scoreboard at the top of the screen with coordinates (0,250). It takes in a sprite "pen" and uses the **.write()** method to set what is written, the alignment of the text, the font of the scoreboard and the size of the font.

Setting Asteroids:

To set the asteroids, a function **set_asteroids(asteroids)** was created to create the asteroid sprite and set both its starting position and movement. It takes in a list of asteroids which will later be set to empty at first, and for each asteroid created, the **.append(asteroid)** method was used to add it to the list of asteroids.

It then uses a for loop **for i in range(7)** to create 7 asteroids and for each value of i, a sprite called asteroid will be created, assigned to the Sprite() attributes, have its color set using **asteroid.color("DarkKhaki")**, have its shape set to the previously registered shape using **asteroid.shape("asteroid")**, and have its speed set to either 0.03,0.04,0.05 or 0.06 using the *random library,* **asteroid.speed = random.randint(3,6)/100.**

To make the asteroid start at random positions across the edge of the screen, the following functions were used:
asteroid.goto(0, 0)
heading = random.randint(0, 360)
distance = random.randint(300, 400)
asteroid.setheading(heading)
asteroid.forward(distance)
Taking the center to be (0,0), this was to ensure that the asteroids started at random proximities from the center (which is where the player would be) and also be spread out randomly.

To set the direction of the asteroid's movement, the function created earlier was used:
get_direction(player,asteroid)
This was to get the direction of the player from the asteroid so the asteroid would always move towards the player.

Setting Missiles:
Similar to asteroids, an empty list was first assigned to a variable "missiles" and uses a for loop to create 3 missiles like what was done for asteroids. To create the shape of the missile, the "square" shape in turtle and the **.shapesize(stretch_wid = 0.3,stretch_len = 0.7)** method was used to create the missile.

missile.speed = 1 - Set missile speed to 1
missile.state = "ready" - Set the state of each missile created in the list

Setting up starting screen:
These were used to create the initial start screen that is shown before the game starts.

wn = turtle.Screen()
wn.setup(width=600, height=600) - Set size of screen
wn.title("Space Shooter Game") - Set title to be displayed on terminal
wn.bgcolor('black') - Set color of background
turtle.color('white') - Set color of instructios
turtle.write("< SPACE SHOOTER GAME > \nPress R to start \nLeft Key - Turn Left \nRight Key - Turn Right \nSpace Bar - Shoot",False,align = "center", font = ("Courier", 24, "normal")) - Display instructions aligned to the center and in what font and font size.
turtle.hideturtle() - Hide the turtle arrow
wn.tracer(0) - Ensure that all other sprites created do not appear before the game starts

Commands:
These functions are used to set keys for the various commands in the game.
rotate_left(): Player rotates 15 degrees anti clockwise

rotate_right(): Player rotates 15 degrees clockwise

fire_missile():

This function checks the state of the missile sprite, if it is in “ready” state, the missile goes to the center of screen where the player sprite is.
- Set direction that missile is fired to be same direction that the player is facing
- Changes state of missile

Main Game Function:

This is the function that runs the game.

start_game(). The pseudocode is as follows:

1. Set up game

a. Set ‘asteroids’ as an empty list
b. Set background to black
c. Set player score to 0
d. Set player’s position at (0,0)
e. Use **set_scoreboard(pen)** to reset scoreboard
f. Use **set_scoreboard(steroids)** to reset asteroids
g. Assign False to a variable Game_Over

2. While Game_Over is False

a. Set left arrow key to **rotate_left()**
b. Set right arrow key to **rotate_right()**
c. Set spacebar to **fire_missile()**
d. To move the missiles, for every missile in the list “missiles”

i. Check if the missile state is “fire”
ii. Missile moves forward at the set speed
iii. If the x-coordinate or y-coordinate of missile exceeds the screen, hide the missile and return its state to “ready”

e. To move the asteroids, for every asteroid in the list “asteroids”

i. Asteroids move forward at set speed
ii. To check for collision between asteroid and player, if the distance between center of player and asteroid is less than 20 pixels,
    1. Reset the Asteroids
    2. Reassign True to Game_Over
iii. To check for collision between missile and asteroid, if distance between center of asteroid and missile is less than 20,
    1. Reset Asteroids
    2. Reset missile to “ready” state
    3. Increase score by 10
    4. Change scoreboard

f. If Game_Over is True,

i. Hide all turtles
ii. Use **pen.write()** to show instructions to restart the game
iii. Use **pen.write()** to show game over message

wn.onkey(start_game,’r’) - Set ‘R’ key to restart game

Snake

This game requires the use of the random library for generating numbers and the tkinter library, which is utilized to create the UI.

Variables:

```
width, height = 600, 600
speed = 70
size = 30
snake_body = 3
snake_colour = "#50C878"
food_colour = "#D22B2B"
```

```
background_colour = "#000000"
```

width and **height** describe the size of the game window.
speed is how fast the snake moves in the game and size is the size of the snake and apple object in the game.
snake_body is the starting length of the snake.
snake_colour, food_colour and **background_colour** refers to the colour of the snake and apple object and the game background respectively.

Classes:

class Snake(): This class is used to create the snake object in the game. It uses the integer from the variable snake_body to generate the coordinates and graphics for the snake.

class Apple(): This class is used to create the apple object in the game. It uses the random library to randomly generate the coordinates of the apple object.

Functions:

def game_over(): This function stops the game by clearing the canvas and displays the game over text as well as the choice for the user to replay the game. The pseudocode is as follows.

1. Clear canvas and display game over text
2. Display restart game option and bind the **start_game()** function to double click.
3. Clear the old scoreboard

def run_game(*arg): This function runs the main game by clearing the canvas and then creating the apple and snake object. The pseudocode is as follows.

1. Clear the canvas
2. Create a snake and an apple object
3. Run the **game_controls()** function on the snake and apple object created.

def game_controls(snake, apple): This function contains the controls for the game, such as making sure the snake moves properly, inserting more body parts into the snake, updating the scoreboard. It takes in a snake object and an apple object as input to get their coordinates. The pseudocode is as follows.

1. Get the coordinates of the head of snake and assign them to 2 variable x and y.
2. Check the initial direction of snake and adjust the coordinates accordingly. Etc. if the **direction == 'right'**, move the x-coordinate by one size.
3. Update the new coordinates of the head of snake
4. Update the graphics of the head of snake
5. Check if the snake head is touching the apple.

i. If True, update scoreboard, delete the old apple and create a new one. If the apple is inside the snake, delete the apple and create another one.
ii. Else, update the tail of the snake.

6. Check for collisions in the game using the **check_collisions()** function.

i. If True, run the **game_over()** function.
ii. Else, repeat the **game_controls()** function using the **window.after()** function.

def movement(new_direction): This function is responsible for the movement of the snake. It takes in **new_direction**, which is a string and checks to make sure the **new_direction** is different from the current direction before changing the direction of the snake.

def check_collisions(snake): This function takes in the snake object as input to get the coordinates of it. It then checks whether the snake is colliding with the border of the game or

with its own body and returns True if there are any collisions and False if not. The pseudocode is as follows.

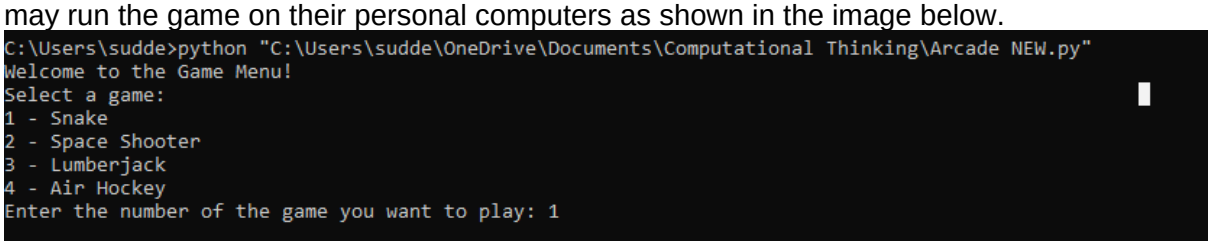
1. Obtain coordinates of snake head and store it in **x** and **y**.
2. Check if head of snake is beyond border. Return True if it is beyond the border
3. Check if head of snake is colliding with its own body. Return True if the snake touches itself
4. Return False if all checks are passed.

def main(): This function defines the global variables in the game such as direction and score, and creates the label for the scoreboard. It also binds the keyboard inputs for the movement in the game.

1. Initialize initial score and direction
2. Create the label for the scoreboard
3. Bind keyboard buttons with the tkinter window
4. Run the **run_game()** function.

User Interface

We have decided to let the code be run on the command line interface for ease of use. Users may run the game on their personal computers as shown in the image below.



def game_menu(): Displays the user interface once the Python file is run, allowing the user to choose which game he/she wants to play.

def parse_arguments(): Uses ‘argparse’ module to return the game based on the selected game number. For instance, if the user enters “1”, it will indicate that the user wants to play Snake as it corresponds to game number 1.

if __name__ == "__main__": Ensures that the code only runs if the script is executed directly. Runs the game based on the user's choice. Ensures that the user’s input is nothing other than “1”, “2”, “3”, “4”

References

1. Python Game Tutorial: Pong - YouTube
2. Turtle Colors
3. TKinter Documentation
4. TKinter Binding
5. TKinter GUI Tutorial