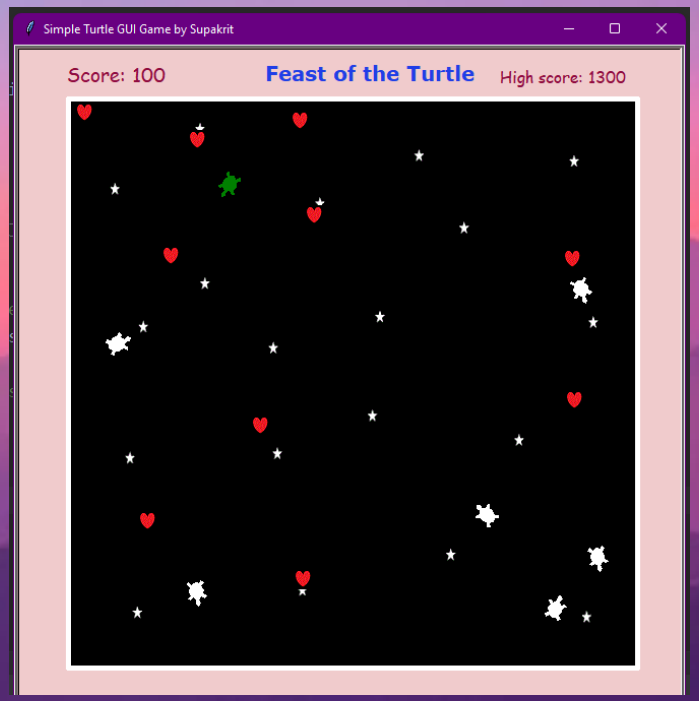
Feast of Turtle

Simple Turtle Game Project

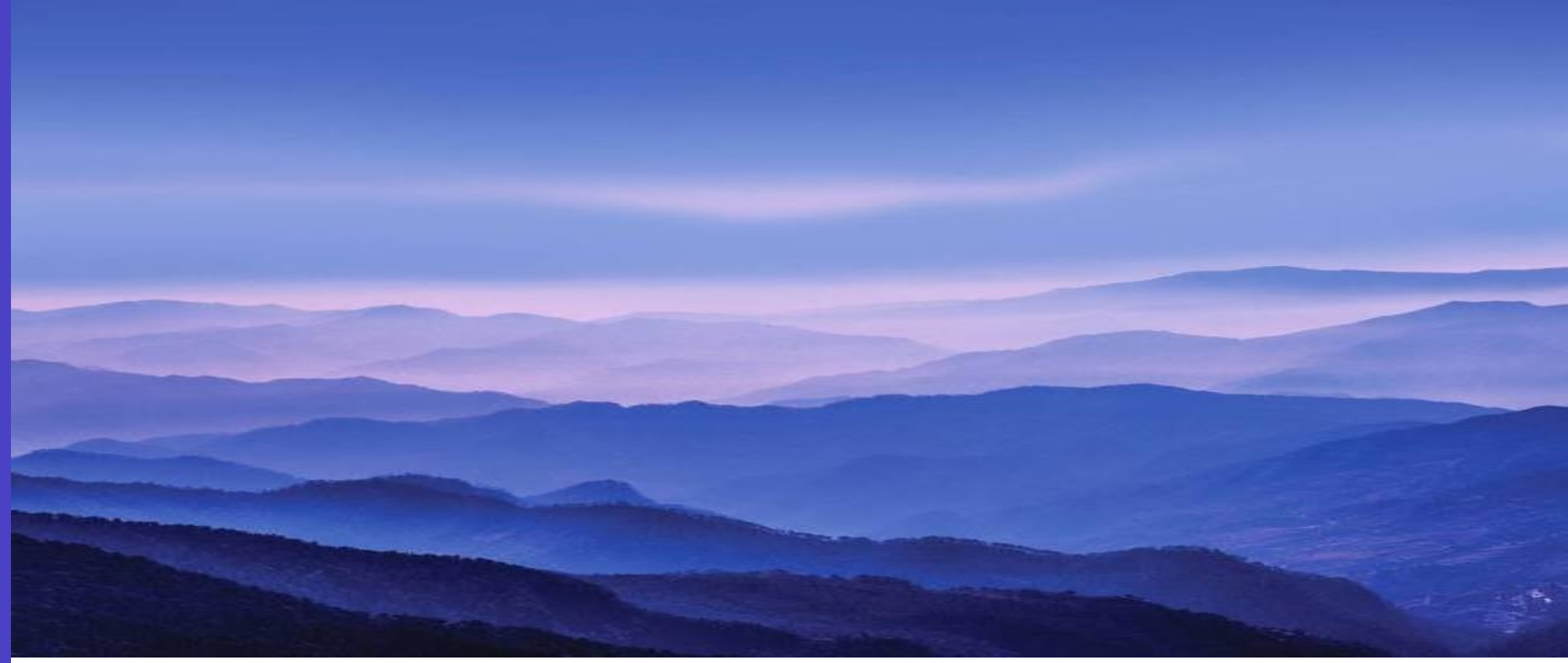


Mr. Supakrit Aphonmaeklong

6410545789 sec 450

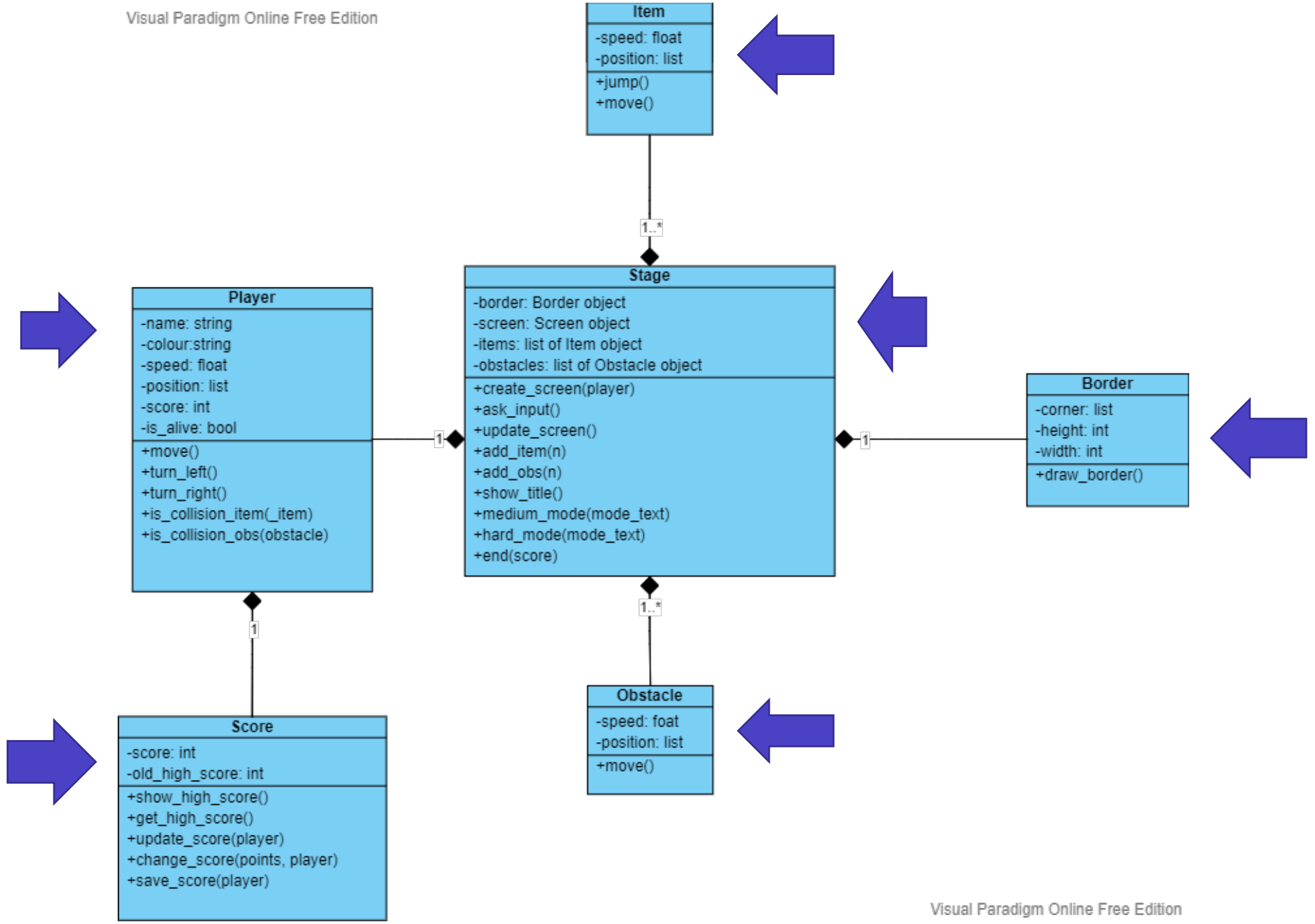
Why I choose this project?

What is it about?



- Memories of game I played in kindergarten.
- The concept of this game is to eat moving items while avoiding moving obstacles to gain the highest score.

Class Diagram



My code

Feast of Turtle

player module

```python
class Player(Turtle):
    """
    Maintain Player object which movement can be controlled.
    PLayer class is a subclass of Turtle class.
    Initialize with name, colour, speed, is_alive, and score properties.
    """
    def __init__(self, name, color):
        super().__init__()
        self.penup()
        self.shape('turtle')
        self.color(color)
        self.colour = color
        self.speed = 2.0
        self.name = name
        self.position = [0, 0]
        self.score = 0
        self.is_alive = True
```

player module

```python
def move(self):
    """move the Player instance with its speed attribute"""
    self.forward(self.speed)
    # turn left when player hit the border
    if self.xcor() > 270 or self.xcor() < -270:
        self.left(60)
    if self.ycor() > 270 or self.ycor() < -270:
        self.left(60)
    # update position attribute
    self.position = [self.xcor(), self.ycor()]

def is_collision_item(self, _item):
    """check if player hit the item or not"""
    a = self.xcor() - _item.xcor()
    b = self.ycor() - _item.ycor()
    distance = math.sqrt((a ** 2) + (b ** 2))
    if distance < 20:
        return True
    else:
        return False

def is_collision_obs(self, obstacle):
    """check if player hit the obstacle or not"""
    a = self.xcor() - obstacle.xcor()
    b = self.ycor() - obstacle.ycor()
    distance = math.sqrt((a ** 2) + (b ** 2))
    if distance < 20:
        return True
    else:
        return False
```

item module

```
class Item(Turtle):
    """
    Maintain Item object which move randomly.
    Item class is a subclass of Turtle class.
    Initialize with speed and position properties.
    """
    def __init__(self):
        super().__init__()
        self.penup()
        self.shape('item.gif')
        self.speed = 0.5
        # random the spawn location of item across the screen
        self.goto(x=random.randint(-270, 270), y=random.randint(-270, 270))
        self.setheading(random.randint(0, 360))
        self.position = [self.xcor(), self.ycor()]
```

item module

```
def jump(self):
    """teleport randomly on the screen and update a position property"""
    self.goto(x=random.randint(-270, 270), y=random.randint(-270, 270))
    self.setheading(random.randint(0, 360))
    self.position = [self.xcor(), self.ycor()]

def move(self):
    """move the Item instance with its speed attribute"""
    self.forward(self.speed)
    # turn left when item hit the border
    if self.xcor() > 270 or self.xcor() < -270:
        self.left(60)
    if self.ycor() > 270 or self.ycor() < -270:
        self.left(60)
    # update position property
    self.position = [self.xcor(), self.ycor()]
```

obstacle module

```python
class Obstacle(Turtle):
    """
    Maintain Obstacle object which move randomly.
    Obstacle class is a subclass of Turtle class.
    Initialize with speed and position properties.
    """
    def __init__(self, born_location):
        super().__init__()
        self.penup()
        self.color('white')
        self.shape('turtle')
        self.speed = 1.0
        # random the spawn location of obstacle across the screen
        self.goto(born_location)
        self.setheading(random.randint(0, 360))
        self.position = [self.xcor(), self.ycor()]
```

obstacle module

```
def move(self):
    """move the Obstacle instance with its speed attribute"""
    # Obstacle instance has a circular moving pattern
    self.forward(self.speed)
    self.left(0.5)
    # turn left when obstacle hit the border
    if self.xcor() > 270 or self.xcor() < -270:
        self.left(60)
    if self.ycor() > 270 or self.ycor() < -270:
        self.left(60)
    # update position property
    self.position = [self.xcor(), self.ycor()]
```

border module

```
class Border(Turtle):
    def __init__(self, corner, width, height):
        """
        Maintain Border object with a rectangular shape.
        Border class is a subclass of Turtle class.
        Initialize with corner, width,and height properties.
        """
        super().__init__()
        self.penup()
        self.hideturtle()
        self.speed(0)
        self.color('white')
        self.pensize(5)
        self.corner = corner
        self.height = height
        self.width = width
```

border module

```
def draw_border(self):
    """draw a border by using its corner, width, and height"""
    self.penup()
    self.hideturtle()
    self.speed(0)
    self.color('white')
    self.pensize(5)
    self.penup()
    self.goto(self.corner[0], self.corner[1])
    self.pendown()
    self.goto(self.corner[0], self.height)
    self.goto(self.width, self.height)
    self.goto(self.width, self.corner[1])
    self.goto(self.corner[0], self.corner[1])
```

score module

```
class Score(Turtle):
    """
    Maintain Score object which can track your score and save it.
    Score class is a subclass of Turtle class.
    Initialize with score property and old_high_score attribute.
    """
    def __init__(self):
        super().__init__()
        self.penup()
        self.hideturtle()
        self.speed(0)
        self.color('#900C3F')
        self.score = 0
        self.old_high_score = 0
```

score module

```
@staticmethod
def get_high_score():
    """extract data from json file and return a int of high score """
    try:
        with open('game_data.json', 'r') as data_file:
            data = json.load(data_file)
    except FileNotFoundError:
        return 0
    else:
        scores = []
        for each_dict in data.values():
            scores.append(each_dict['score'])
        high_score = max(scores)
        return int(high_score)
```

score module

```python
@staticmethod
def save_score(player):
    """at the end of the game save username, color, and score on json file"""
    # create new_data dict storing username as key and color and score as value
    new_data = {
        player.name: {
            'color': player.colour,
            'score': player.score
        }
    }
    try:
        with open('game_data.json', 'r') as data_file:
            data = json.load(data_file)
    except FileNotFoundError:
        with open('game_data.json', 'w') as data_file:
            json.dump(new_data, data_file, indent=4)
    else:
        data.update(new_data)
        with open('game_data.json', 'w') as data_file:
            json.dump(data, data_file, indent=4)
```

stage module

```
class Stage(Turtle):
    """
    Represent a stage for all objects such as border and a screen, as well
    as a Player object, Item object, Obstacle objects.
    Stage class is a subclass of Turtle class.
    Stage class provides many important methods for the game.
    """
    def __init__(self, border, screen):
        super().__init__()
        self.border = border
        self.screen = screen
        # contain list of items and obstacles
        self.items = []
        self.obstacles = []
```

stage module

```
def create_screen(self, player):
    """maintain a interactive screen with a title and a wallpaper"""
    # create GUI screen
    self.show_title()
    self.screen.bgcolor('#F0CBCC')
    self.screen.title('Simple Turtle GUI Game by Supakrit')
    self.screen.setup(width=660, height=660)
    self.screen.bgpic('sWallper.gif')
    self.screen.register_shape('item.gif')
    # update screen manually
    self.screen.tracer(0)
    # create border
    self.border.draw_border()
    # listens for screen input such as left and Right keys
    self.screen.listen()
    self.screen.onkey(fun=player.turn_left, key='Left')
    self.screen.onkey(fun=player.turn_right, key='Right')
```

stage module

```
    def update_screen(self):
        """update the screen which is a class property"""
        self.screen.update()

    def add_item(self, n):
        """add item to its items list for n times"""
        for i in range(n):
            self.items.append(Item())
```

```
def add_obs(self, n, player, late):
    """add obstacle to its obstacles list for n time"""
    born_location_list = [[-200, -200], [-100, 100], [220, 100], [-150, 250],
                          [100, -150], [150, 200], [0, -180]]
    for i in range(n):
        # obstacle will have a random born location on the screen
        born_location = [random.randint(-270, 270), random.randint(-270, 270)]
```

stage module

```
def medium_mode(self, mode_text):
    """increase the speed of obstacles and display 'medium mode' on the screen"""
    mode_text.hideturtle()
    for obs in self.obstacles:
        obs.speed = 2
    mode_text.penup()
    mode_text.goto(180, -311)
    mode_text.color('#700815')
    mode_text.write(f'Medium Mode', font=("Verdana", 12, 'bold'), align='center')

def hard_mode(self, mode_text):
    """increase the speed of obstacles and display 'hard mode' on the screen"""
    mode_text.hideturtle()
    for obs in self.obstacles:
        obs.speed = 3
    mode_text.penup()
    mode_text.goto(180, -311)
    mode_text.color('#700815')
    mode_text.write(f'Hard Mode', font=("Verdana", 12, 'bold'), align='center')
```

stage module

```
def medium_mode(self, mode_text):
    """increase the speed of obstacles and display 'medium mode' on the screen"""
    mode_text.hideturtle()
    for obs in self.obstacles:
        obs.speed = 2
    mode_text.penup()
    mode_text.goto(180, -311)
    mode_text.color('#700815')
    mode_text.write(f'Medium Mode', font=("Verdana", 12, 'bold'), align='center')

def hard_mode(self, mode_text):
    """increase the speed of obstacles and display 'hard mode' on the screen"""
    mode_text.hideturtle()
    for obs in self.obstacles:
        obs.speed = 3
    mode_text.penup()
    mode_text.goto(180, -311)
    mode_text.color('#700815')
    mode_text.write(f'Hard Mode', font=("Verdana", 12, 'bold'), align='center')
```

stage module

```
def end(self, score):
    """display your score on the screen"""
    text = Turtle()
    # clear the old screen
    self.screen.clear()
    Stage.show_title()
    self.screen.bgcolor('#F0CBCC')
    self.screen.title('Simple Turtle GUI Game by Supakrit')
    self.screen.setup(width=660, height=660)
    self.screen.bgpic('sWallper.gif')
    # show the high score on the top of the screen
    score.show_high_score()
    text.penup()
    text.hideturtle()
    text.color('#8CF310')
    text.goto(0, 0)
    style = ('Courier', 30, 'italic')
    # check if player break the previous high score
    if score.score > score.old_high_score:
        text.goto(0, -50)
        text.write(f'Game over:\nYou got a HIGH SCORE!\n\n'
                   f'Your score is {score.score}', font=style, align='center')
    else:
        text.write(f'Game over:\n\n'
                   f'Your score is {score.score}', font=style, align='center')
    text.goto(80, -270)
    end_style = ("Verdana",12, "normal")
    text.write('click the screen to exit', font=end_style)
    self.screen.exitonclick()
```

app module

```python
from turtle import Turtle, Screen
import json
from player import Player
from score import Score
from border import Border
from stage import Stage
import time


def app_mode():...


def game_mode(instruction):...


def view_mode(screen, user_data):...


def view_score(username):...


# Main
app_mode()
```