SNAKE GAME PROJECT

MUHAMMED SAVAŞ NO:210408003

Libraries:

pygame: This library is a library that makes game development easier with Python. It provides various functions such as creating game windows, drawing graphical objects, processing user input, and managing the game loop.

random: This library is used to generate random numbers and make random choices. It is frequently used in games, such as placing baits in random positions.

enum: This library allows creating enumerated constants. That is, it is used to group a group of literals under a certain type. For example, in this game example it is used to represent directions.

collections.deque: This library provides a bidirectional deque data structure. This data structure supports quickly adding or removing elements from the beginning or end of the list. In this gaming example it is often used for the BFS algorithm.

Basic Settings and Constants of the Game:

Screen Dimensions: SCREEN_WIDTH and SCREEN_HEIGHT constants determine the game's window size. These constants usually represent width and height values in pixels. In this game example, the game screen is 800 pixels wide and 600 pixels high.

Snake Dimensions: The SNAKE_SIZE constant determines the size of each segment of the snake. The snake usually consists of a series of rectangular segments. This constant determines the side length of these rectangles. In this game example, the size of the snake segments is 20 pixels.

Feed Dimensions: The FOOD_SIZE constant determines the size of the feed object. Lure is one of the objects that the snake will eat and can often be the same or different from the snake's segment sizes. In this game example, the size of the bait object is also 20 pixels.

Snake Speed: The SNAKE_SPEED constant determines the snake's movement speed. Each time the snake moves, it determines how far it will travel with each step. The higher this value, the faster the snake moves. In this game example, the snake moves with a distance of 20 pixels with each step.

Colours: The constants ORANGE, RED and GREEN define the game's color palette. In this game example, it is used to determine the colors of the snake, bait, and other objects. Each color is specified as a triple RGB (Red, Green, Blue) value.

Directions: An enum (enumerated constants) called Direction is defined. This enum represents the four basic directions (up, down,

left, right) in which the snake can move. Each direction is assigned a numerical value. This makes directions more readable and less likely to make mistakes. This is used in-game to determine which direction the snake will move.

Behavior and Characteristics of the Snake

__init__: This special method is called when a Snake object is created. This method sets the initial position (x and y coordinates), color, direction and growth of the snake. The snake's body starts out with just one segment.

move: This method moves the snake in the specified direction. The snake's body is updated with each step depending on the position of the leading segment. If the snake crosses the boundaries, it continues from the opposite side when the snake leaves the screen. This method also checks the growth status of the snake. If the snake's growth status is True, the snake's body will grow, otherwise the last segment of the snake's tail will be removed.

grow_snake: This method sets the snake's growth state to True. This method is called when the snake eats a bait and in the next step the snake's body lengthens.

draw: This method draws the snake on the screen. Each segment of the snake is drawn as a rectangle on a given surface. The color is determined by the location and size of the segment.

check_eat_food: This method checks whether the snake has eaten a food item. Using a given bait position (food_position) and a tolerance value, it checks whether the snake's head segment collides with the bait. Returns True if the snake has eaten the bait, otherwise False.

check_collision_with_other: This method checks whether the snake's head segment collided with the body of another snake. This is done by comparing the head segments of two snakes. If the head segments collide with each other, the snakes have collided with each other and True is returned. Otherwise, False is returned.

Feeds

Feed Class (Food):

__init__ Method: This method is called when a feed object is created. The bait is created in a random location. The x and y coordinates are randomly selected within a certain range of steps to be within the game screen and in a position where the snake can eat. This pitch range is determined using FOOD_SIZE, which is the size of the snake segments.

draw Method: This method draws the feed object on the screen. The bait is drawn as a red rectangle. The position and size of the rectangle are based on the x and y coordinates of the feed object and the constant FOOD_SIZE.

Game Class (Game):

__init__ Method: This method is called when a game object is created. The game's window is created and its title is set. It also starts the game clock. The game's player and AI snakes are created, a Food

object is created, and starting scores (score_player and score_ai) are set for the player and AI.

handle_events Method: This method handles player inputs (such as pressing keyboard keys). If the game window is closed, the game is terminated (QUIT event). It also allows the player to direct their snake by pressing keyboard keys (KEYDOWN events).

BFS Method:

A method called bfs is defined.

Given a starting point (start), a target point (target) and the snake's body (snake_body), the method tries to find the shortest path to reach the target point from the starting point.

A queue and a set are created to hold the visited points.

It is added to the queue with the starting point and an empty path list.

The following steps are performed until the queue is empty (i.e., the goal is reached):

An element is removed from the queue (using popleft()). This element is a copy of the current node and the path taken to reach this node.

If the current node is equal to the destination point, the tracked path is returned.

If the current node has been visited before, the transaction is invalidated and the loop continues.

If the current node has not been visited, it is added to the set of visited nodes.

The coordinates of the current node are taken (x, y).

Moves up, down, left and right of the current node. However, these movements are made on the condition that they do not hit the segments of the snake and do not leave the playing field.

The current location and path are added to the queue.

If the destination cannot be reached, an empty path is returned.

Collision Situations:

move_ai Method:

This method controls the movement of the AI snake.

First, the position (x, y) of the head segment of the artificial intelligence snake is taken and the target point is determined. The target point is the location of the bait object.

Then, an error probability (error_rate) is checked. If a random number is less than the probability of error, the movement of the AI snake is determined randomly. Otherwise, a path to the destination is found using the BFS algorithm.

If a path is found, the direction of movement of the artificial intelligence snake is determined as the first step of the found path.

Finally, the AI snake (self.ai) is moved (self.ai.move()).

check_collisions Method:

This method checks for collision cases.

First, the collision of the player snake (self.player) and the artificial intelligence snake (self.ai) is checked. If a collision occurs, "Player

loses!" will appear on the screen. is printed and the game is terminated.

Then, the AI snake is checked to collide with the player snake. If a collision occurs, "AI loses!" will appear on the screen. is printed and the game is terminated.

check_food_collision Method:

This method checks whether the snakes have eaten the bait.

First, it is checked whether the player snake can eat the bait. If the player snake has eaten a bait, the player score is increased, the player snake is made to grow and a new bait is created.

Then, it is checked whether the artificial intelligence snake can eat the food. If the artificial intelligence snake has eaten a bait, the artificial intelligence score is increased, the artificial intelligence snake is made to grow and a new bait is created.

Control the Operation of the Game

draw_score Method:

This method draws the player and AI player's scores on the screen.

First, a font is determined (pygame.font.SysFont(None, 30)).

Then, the player and AI player's scores are created with the appropriate colors (GREEN and ORANGE) to be plotted on the screen.

Player score and AI player score are printed on specific coordinates of the screen ((10, 10) and (10, 40)).

run Method:

This method runs the main loop of the game.

The game runs continuously in an endless loop.

At each step, the screen is filled with a black color (self.surface.fill((0, 0, 0))).

User inputs (handle_events method) are processed.

The player and the AI player are moved (self.player.move() and self.move_ai()).

Collisions are checked (self.check_collisions() and self.check_food_collision()).

The player snake, artificial intelligence snake and baits (self.player.draw(), self.ai.draw(), self.food.draw()) are drawn on the screen.

Player scores (draw_score method) are drawn on the screen.

The screen is updated (pygame.display.update()).

The speed of the game is controlled by setting the game clock (self.clock) with a certain FPS (frames per second) (self.clock.tick(10)).

Starting the Game (if name == "main"):

This section allows the game to be launched when the Python script is run directly.

The pygame library is initialized (pygame.init()).

A Game object is created (game = Game()).

The game is started by calling the run method (game.run()).

MUHAMMED SAVAŞ NO:210408003