AI - Deep Queue Learning

Bootstrap Snake

Introduction to ai in game

Game

The Snake Game is a classic arcade game where the player controls a snake that moves around the screen, consuming food to grow longer. The game is implemented in Python using the Pygame library, providing a graphical interface for an interactive gaming experience.

Prerequisites

Ensure you have the necessary tools installed to run the Snake Game:

- **Python (version x.x.x)**
- **Pygame library**

Game Components

Welcome to the Snake Game Bootstrap! In this guide, we'll structure the essential class and functions for building a Snake game using Python and the Pygame library. The Snake game is a classic that involves controlling a snake to eat food and grow longer while avoiding collisions with the game boundaries and the snake's own body.

Enumerations and Constants

Define the necessary enumerations and constants used in the game.

- Direction Enum:
  - RIGHT, LEFT, UP, DOWN
- Colors:
  - WHITE, RED, BLUE1, BLUE2, BLACK
- Game Settings:
  - BLOCK_SIZE, SPEED

Point Class

Create “Point” using namedtuple to represent a point with x and y coordinates.

Main Class

To manage the game state and display, we'll create a Python class named `SnakeGame`. This class will handle the initialization of the game window, the clock, and other essential components.

```
class SnakeGame:
    def __init__(self, w=640, h=480):
        """
        Parameters:
        - w (int): Width of the game window.
        - h (int): Height of the game window.
        - display: pygame.display.set_mode((self.w, self.h))
        - clock: pygame.time.Clock()
        - reset: reset()
        """
```

In this initialization, we set up the game window dimensions, display, window caption, and clock. The reset function is called to initialize the game state.

Game Components

Game State Reset

The reset function resets the game state to its initial values. It is called during class initialization and can be used to restart the game.

```
def reset(self):
    """
    Resets the game state to its initial values.
    """
    # Initialize game state variables
    self.direction = Direction.RIGHT
    self.head = Point(self.w / 2, self.h / 2)
    self.snake = [self.head, Point(self.head.x - BLOCK_SIZE, self.head.y),
                  Point(self.head.x - (2 * BLOCK_SIZE), self.head.y)]
    self.score = 0
    self.food = None
    self._place_food()
    self.frame_iteration = 0
    self.reward = 0
```

Placing Food

The _place_food function is responsible for randomly placing food on the game grid. It ensures that the food is not positioned on the snake.

```
def _place_food(self):
    """
    Places food at a random location on the game grid.
    """
```

Game Components

Game Loop Step

The play_step function executes a single step in the game loop. It collects user input, moves the snake, checks for collisions, updates the UI, and returns relevant information.

```
def play_step(self, action):
 """
 Executes a single step in the game loop.

 Parameters:
 - action: The action taken by the player.

 Returns:
 - game_over (bool): True if the game is over, False otherwise.
 - score (int): The current score in the game.
 - reward (float): The reward obtained in this step.
 - total_reward (float): The total reward accumulated so far.
 """
```

Collision Detection

The is_collision function checks for collisions with walls or the snake's own body.

```
def is_collision(self, pt=None):
 """
 Checks for collisions with walls or the snake's own body.

 Parameters:
 - pt (Point): The point to check for collision (default is None, which uses the head).

 Returns:
 - collision (bool): True if there is a collision, False otherwise.
 """
```

Game Components

UI Update

The _update_ui function is responsible for updating the game display. It draws the snake, food, score, and reward on the screen.

```
def _update_ui(self):
 """
 Updates the game display.
 """
```

Snake Movement

The _move function handles the movement of the snake based on the provided action.

```
def _move(self, action):
 """
 Moves the snake based on the provided action.

 Parameters:
 - action: The action taken by the player.
 """
```

Conclusion

This Snake Game Bootstrap provides a foundation for developing a Snake game using Pygame. Use this structure as a starting point, and fill in the code for each function to implement the game logic. Enjoy building your Snake game!"

AI

Prerequisites

Ensure you have the necessary tools installed to run the Snake Game:

- **Numpy**
- **MatplotLib**
- **Torch**
- **IPython**

AI MODEL

1. Linear_QNet class

Purpose: To recreate a class that represents a simple neural network with a hidden layer.
What it does: This class creates a neural network to approximate a Q function in the context of reinforcement learning.

Hints: Look up the concepts of neural network, linear layers, activation functions (ReLU here), and how networks work in general in the context of reinforcement learning.

Pseudo-code :

```
class Linear_QNet:
    def __init__(self, input_size, hidden_size, output_size):
        # Initialise the weights and biases of the linear layers
    def forward(self, x):
        # Implement the forward pass of the network
        # Use ReLU as the activation function for the first layer
    def save(self, file_name='model.pth'):
        # Save the model weights in a file
```

2. QTrainer class

Purpose: To recreate a class which manages the training of the Q model.
What it does: This class uses the Q-learning algorithm to train the neural network to approximate the Q function.

Hints: Research the concepts of Q-learning, loss function (MSELoss here), optimisation (Adam here), and how to update the weights of the network.

AI MODEL

```
class QTrainer:

    def __init__(self, model, lr, gamma):
        # Initialise optimiser, loss function, and other parameters

    def train_step(self, state, action, reward, next_state, done):
        # Pre-process input data
        # Calculate predicted Q-values using current state
        # Calculate target values using Q-learning algorithm
        # Calculate loss and perform backpropagation
```

General notes:

Feel free to explore the PyTorch documentation for a better understanding of each function used.

AI HELPER

In AI, data visualisation is essential for understanding the performance of your model and observing how it learns over time. In the context of the snake game with Q-Learning, the visualisation function provided in the code is crucial.

Here is a detailed explanation of this function:
Visualisation function: plot
Purpose: This function is used to graphically display the scores achieved by the snake over the course of learning, as well as the average score.

Why is it important? Visualisation allows you to visually track the model's progress through the games, identify trends, and spot any problems.
Tips:

- **Use plt.ion() to enable Matplotlib's interactive mode, allowing real-time updates to the graph.**
- **display.clear_output() and display.display(plt.gcf()) refresh the display each time the graph is updated.**
- **Individual scores and the average score are plotted on the same graph for easy comparison.**
- **plt.pause(.1) adds a short pause to allow real-time display.**

AI HELPER

```
import matplotlib.pyplot as plt
from IPython import display

# Activate Matplotlib's interactive mode
plt.ion()

def plot(scores, mean_scores):
 # Clear the previous output and display the current figure

 # Clear the previous contents of the graph

 # Axis titles and labels

 # Plot individual scores and mean scores

 # Adjust y-axis to start at 0

 # Display the last scores at the end of the corresponding lines

 # Display the legend

 # Display the graph in real time

 # Pause briefly to allow real-time display
```

Use this plot function to visually monitor your model's performance during training. It will give you an immediate overview of the evolution of scores, which can be crucial for adjusting parameters and improving your AI.

AI AGENT

To recreate the snake game with Q-Learning, follow these step-by-step instructions.

1. Importing libraries

Objective: Import the necessary libraries.
Tip: Make sure you have installed the required libraries using pip install pygame torch.

```
import torch
import random
import numpy as np
from collections import deque
from game import SnakeGameAI, Direction, Point
from model import Linear_QNet, QTrainer
from helper import plot
```

2. Define the parameters
Objective: To define the game and training parameters.
Tips: Parameters such as MAX_MEMORY, BATCH_SIZE, and LR affect the learning of the model. Experiment with different values for best results.

```
MAX_MEMORY = 100_000
BATCH_SIZE = 1000
LR = 0.001
```

3. Define the Agent Class
Objective: Create the Agent class that represents the snake AI.
Tips: This class encapsulates the AI logic, the agent memory, the Q model, and the training functions.

AI AGENT

```
class Agent:
 # ... (see code provided)
```

4. Get_state function

Purpose: To define the function to get the current state of the game.

Tips: This function captures various information about the snake's environment, such as dangers and the position of food.

```
 def get_state(self, game):
 # ... (see code provided)
```

5. remember and train_long_memory functions

Aim: To use the functions to store experiences and train the model from memory.

Tips: The memory is a circular queue that stores past experiences. The train_long_memory function uses a random sample of memory for training.

```
 def remember(self, state, action, reward, next_state, done):
 # ... (see code provided)

 def train_long_memory(self):
 # ... (see code provided)
```

6. train_short_memory function

Purpose: To implement the function for short-term training with a single experiment.

Tips: This function is used for training after each movement of the snake.

```
 def train_short_memory(self, state, action, reward, next_state, done):
 # ... (see code provided)
```

AI AGENT

7. Get_action function

Purpose: To define the function for obtaining the action to be performed.

Tips: This function uses exploration/exploitation by randomly choosing certain actions at the start, then gradually exploiting the model's predictions.

```
def get_action(self, state):
# ... (see code provided)
```

8. Train function

Purpose: To create the main function for training the model and displaying the results.

Tips: The main loop of the game, where the snake performs movements, is here. The scores are recorded and the model is trained at the end of each game.

```
def train():
# ... (see code provided)
```

9. Running the programme

Objective: To start learning the snake.

Advice: Make sure that the game.py file containing the SnakeGameAI class, the model.py file with the Linear_QNet and QTrainer classes, and the helper.py file for the plot function are present.

```
if __name__ == '__main__':
 train()
```

Final notes :

Ensure that the game.py, model.py, and helper.py files are implemented correctly to avoid errors at runtime.

Explore the details of the functions provided in the source files to better understand how the game and the AI work.

Don't hesitate to adjust parameters and experiment to improve the model's performance.