

Python snake game by RL

Code 1

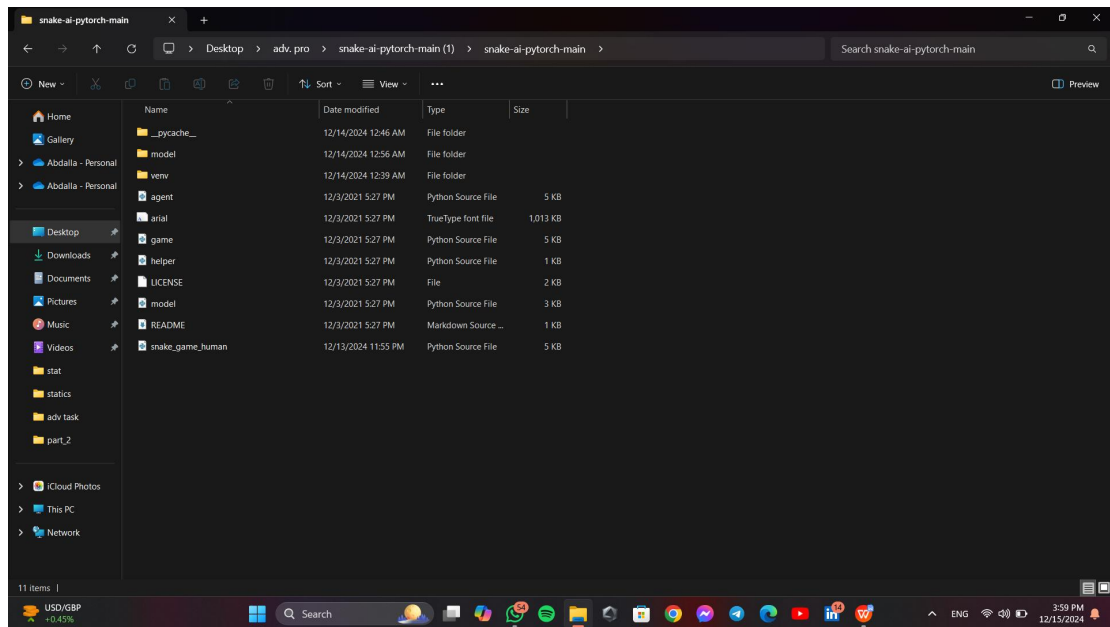
Video:

[Python + PyTorch + Pygame Reinforcement Learning - Train an AI to Play Snake](#)

“gethub link of the code description of video”

How to run code :

- (1) Unzip code folder
- (2) Open folder
- (3) Right click and choose “open terminal”



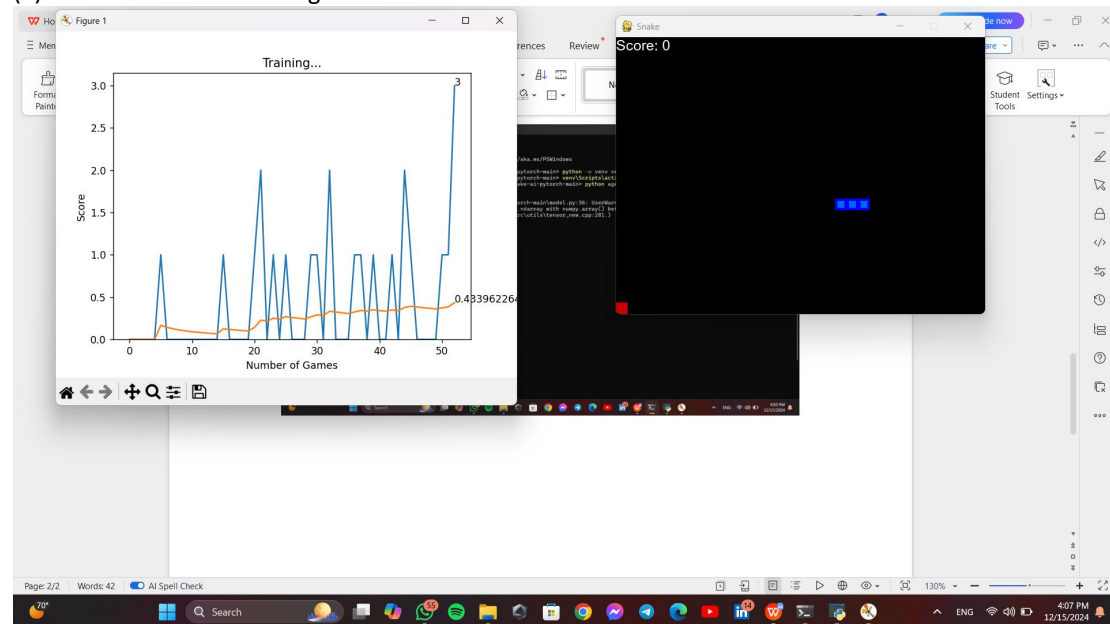
(4) Write the following command

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\abdalla\Desktop\adv. pro\sna...> python -m venv venv
PS C:\Users\abdalla\Desktop\adv. pro\sna...> venv\Scripts\activate
PS C:\Users\abdalla\Desktop\adv. pro\sna...> python agent.py
pygame 2.6.1 (SDL 2.28.4, Python 3.11.3)
Hello from the pygame community. https://www.pygame.org/contribute.html
C:\Users\abdalla\Desktop\adv. pro\sna...model.py:36: UserWarning: Creating a tensor from a list of numpy.ndarrays
is extremely slow. Please consider converting the list to a single numpy.ndarray with numpy.array() before converting to a tensor. (Triggered internally at
C:\actions-runner\posh\pytorch\pytorch\builder\windows\pytorch\torch\csrc\utils\tensor_new.cpp:281.)
  state = torch.tensor(state, dtype=torch.FloatTensor)
Game 1 Score 0 Record: 0
Figure(640x480)
Game 2 Score 0 Record: 0
Figure(800x600)
Game 3 Score 0 Record: 0
Figure(800x600)
Game 4 Score 0 Record: 0
Figure(800x600)
Game 5 Score 0 Record: 0
Figure(800x600)
Game 6 Score 1 Record: 1
Figure(800x600)
Game 7 Score 0 Record: 1
Figure(800x600)
Game 8 Score 0 Record: 1
Figure(800x600)
Game 9 Score 0 Record: 1
Figure(800x600)
Game 10 Score 0 Record: 1
Figure(800x600)
Game 11 Score 0 Record: 1
Figure(800x600)
Game 12 Score 0 Record: 1
Figure(800x600)
Game 13 Score 0 Record: 1
Figure(800x600)
Game 14 Score 0 Record: 1
```

(5) Code run and start training model



Code workflow and structure

- **agent.py**: The file where the reinforcement learning agent is implemented.
- **arial.ttf**: A font file, possibly used for displaying text in the game.
- **game.py**: Contains the logic for the Snake game environment.
- **helper.py**: Likely includes utility functions to assist in various tasks.
- **model.py**: Defines the neural network architecture used by the agent.
- **snake_game_human.py**: A script for playing the Snake game manually, without AI

1. agent.py

This file implements the reinforcement learning agent using a Q-learning approach:

Agent class manages the training and decision-making of the agent.

Attributes:

n_games: Tracks the number of games played.

epsilon: Controls exploration (randomness).

gamma: Discount factor for future rewards.

memory: A deque to store experiences for replay.

model: An instance of Linear_QNet for decision-making.

trainer: An instance of QTrainer for optimizing the model.

Methods:

get_state: Extracts the current game state as input for the model.

remember: Stores experiences for replay.

train_long_memory: Uses a batch of experiences for training.

train_short_memory: Trains on the most recent experience.

get_action: Chooses an action based on exploration-exploitation trade-off.

2. game.py

Defines the Snake game environment, compatible with both AI and human control:

SnakeGameAI class:

Attributes:

Screen size (w, h), block size, and speed settings.

Snake and food positions.

Methods:

play_step: Progresses the game by one step based on the given action.

reset: Resets the game for a new episode.

_place_food: Randomly positions the food.

_move: Updates the snake's position based on the action.

_is_collision: Checks if the snake collides with walls or itself.

_update_ui: Handles rendering the game on the screen.

3. model.py

Defines the neural network used for the Q-learning agent:

Linear_QNet class:

A simple feedforward neural network with:

Input layer size: Represents the game state

Hidden layer size: Processes the input with non-linear transformations.

Output layer size: Corresponds to possible actions (up, down, left, right).

Methods:

forward: Performs the forward pass to predict Q-values.

save: Saves the trained model to a file.

load: Loads a previously saved model.

Workflow

First step

the project structure is organized:

- agent.py - Reinforcement Learning (RL) agent.
- game.py - Game environment.
- model.py - Neural network for Q-learning.
- helper.py - Utility functions for data visualization and analysis.

Second step

Game Environment Design

Create the game environment using [Pygame](#) in [game.py](#):

- Define the snake's movement, food placement, and boundary conditions.
- Implement the logic for collision detection (with itself or walls).

Create a `play_step()` method to:

- Take an action (left, right, straight).
- Update the snake's position.
- Check for rewards (eating food).
- Determine if the game is over.

Third step

3. Agent Development

Design the RL agent in `agent.py`:

State Representation:

Define the game's state using features like the snake's position, food position, and direction.

Action Space:

Map actions (e.g., turn left, turn right, go straight) to integers.

Memory:

Use a deque to store past experiences (state, action, reward, next state).

Policy:

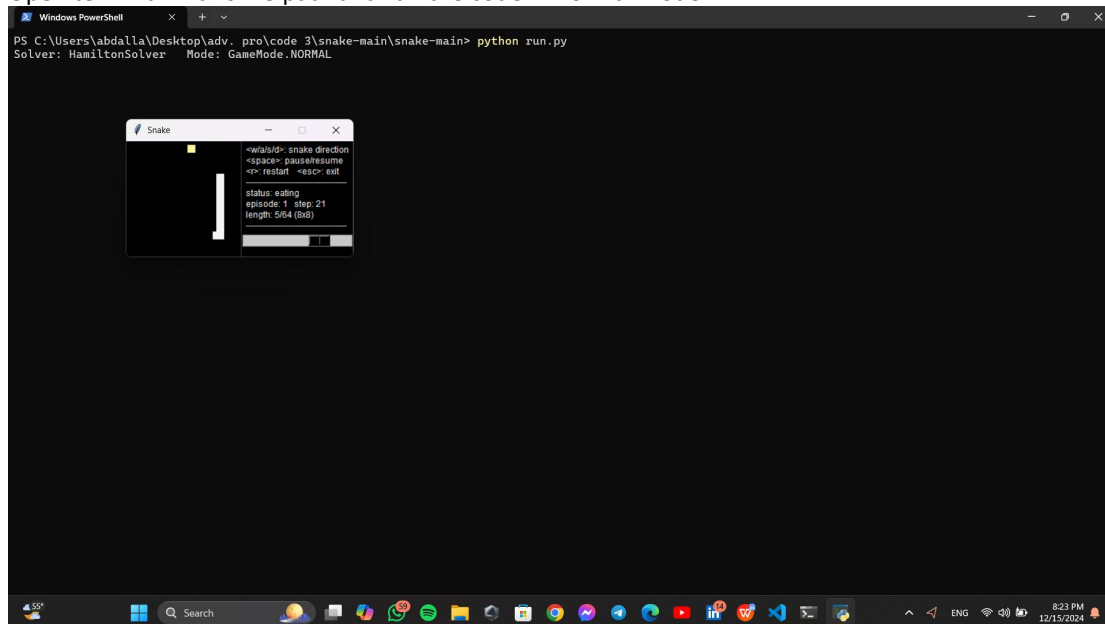
Implement an epsilon-greedy strategy to balance exploration (random actions) and exploitation (predicted actions).

Code 2

Video : null

How to run code

Open terminal in this file path and run the code in normal mode



Code workflow and structure

Code Structure

Root Level

run.py: Likely the main entry point to execute the project.

requirements.txt: Lists Python dependencies for the project.

README.md: Provides project documentation and usage instructions.

LICENSE: Specifies licensing terms.

docs/: Contains documentation, presentation files, and images to explain algorithms, solvers, and results.

tests/: Houses unit tests for various modules.

Core Modules

snake/: Main package for game logic and utilities.

game.py: Contains the main game mechanics.
gui.py: Implements the graphical interface of the game.

snake/base/: Provides foundational components.

direc.py: Handles directions for the snake.
map.py: Manages the game map.
point.py: Represents points (coordinates) in the grid.
pos.py: Position-related utilities.
snake.py: Represents the snake's data and behavior.

snake/solver/: Implements AI solvers for the game.

base.py: Base class for solvers.
greedy.py: Greedy algorithm-based solver.
hamilton.py: Hamiltonian cycle solver.
path.py: Pathfinding algorithms.
dqn/: Deep Q-Network (DQN)-based solver.
history.py: Tracks DQN training history.
logger.py: Handles DQN logging.
memory.py: Implements replay memory for reinforcement learning.
snakeaction.py: Encapsulates snake actions for DQN.

snake/util/: Provides utility modules.

sumtree.py: Implements a sum tree for efficient prioritization.

Tools

plot_dqn_compare.py: Compares performance of different DQN configurations.
plot_dqn_history.py: Visualizes DQN training history.
print_ckpt.py: Processes and prints checkpoints, likely related to DQN models.

Workflow

Game Initialization:

The game logic is initialized using game.py.
The GUI for the game is set up with gui.py.

Gameplay:

Base modules in snake/base/ handle map creation, snake movement, and collision detection.

Player or AI controls the snake to navigate the grid.

AI Solvers:

The solver package provides different strategies for automated gameplay.

Greedy: Aims for immediate goals (e.g., shortest path to food).

Hamiltonian: Ensures the snake follows a safe cyclic path.

DQN: Uses reinforcement learning to optimize gameplay.

Reinforcement Learning:

dqn/ modules manage training, memory replay, and action decisions.

Training performance can be analyzed using the tools in the tools/ directory.

Testing and Validation:

Automated tests for each module ensure reliability (found in tests/).

Visualization:

Results and comparisons of algorithms or training runs can be visualized using scripts in tools/ and images in docs/images/.

Features

Multiple AI Solvers:

Greedy, Hamiltonian, and DQN solvers demonstrate different approaches to automating the Snake game.

Reinforcement Learning:

Includes a DQN-based solver to learn optimal strategies through experience.

GUI Integration:

A user-friendly graphical interface for playing or observing AI strategies.

Customizability:

Modular structure allows easy modification or addition of new solvers and features.

Data Visualization:

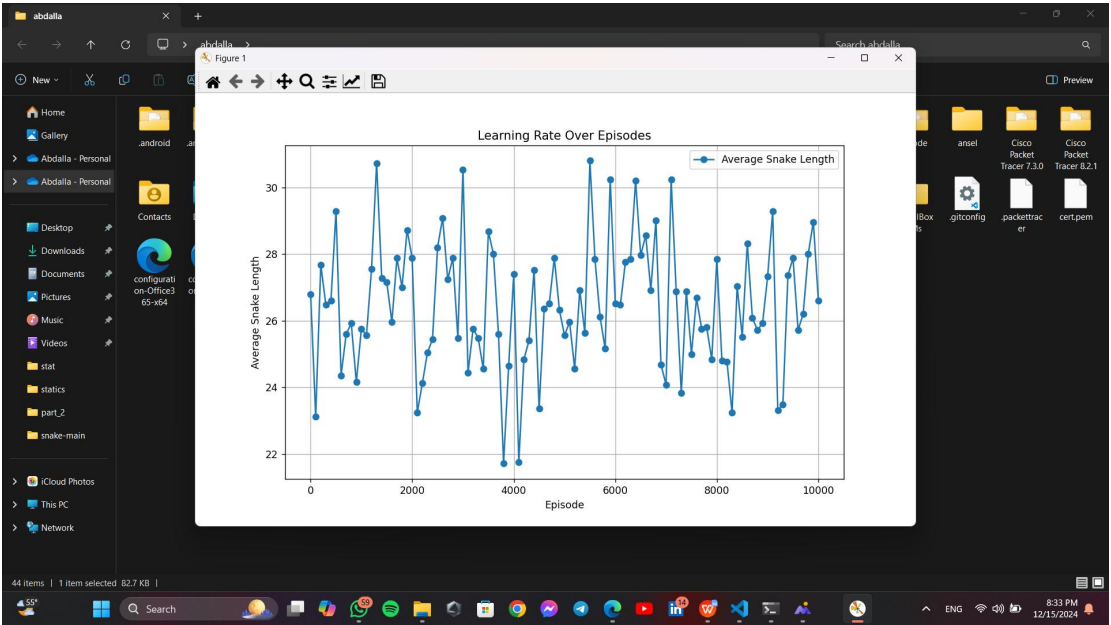
Provides scripts to analyze and visualize training performance and solver comparisons.

Unit Testing:

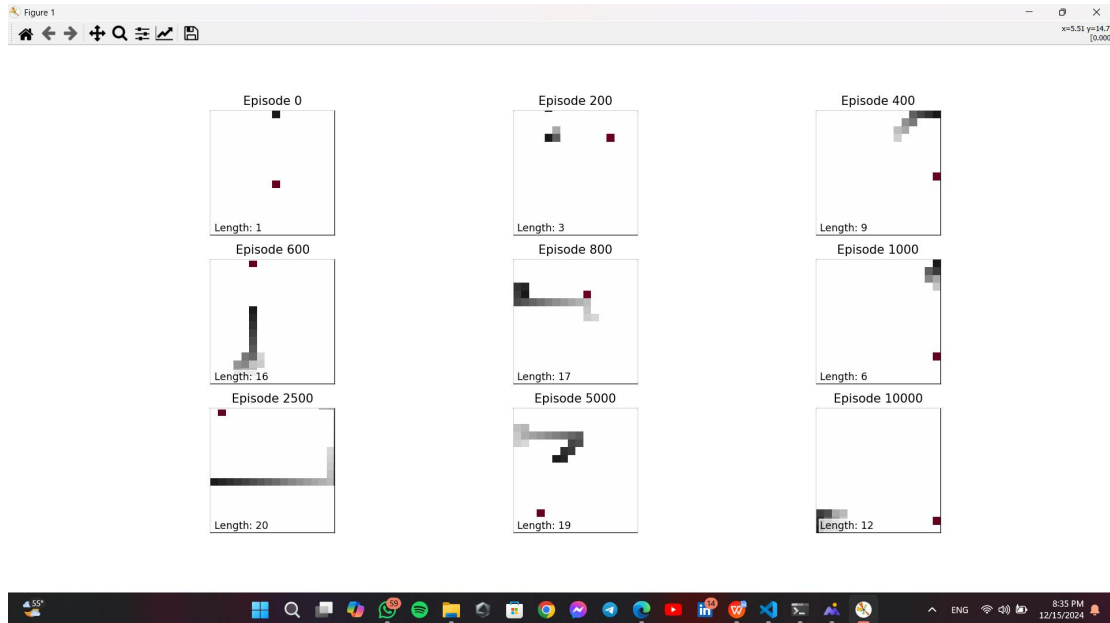
Comprehensive tests ensure the accuracy and stability of core components.

Code3

Video: null



```
Episode 6600 Average snake length without exploration: 29.56
Episode 6700 Average snake length without exploration: 26.96
Episode 6800 Average snake length without exploration: 28.48
Episode 6900 Average snake length without exploration: 17.04
Episode 7000 Average snake length without exploration: 14.68
Episode 7100 Average snake length without exploration: 29.2
Episode 7200 Average snake length without exploration: 15.92
Episode 7300 Average snake length without exploration: 29.24
Episode 7400 Average snake length without exploration: 29.4
Episode 7500 Average snake length without exploration: 26.88
Episode 7600 Average snake length without exploration: 35.32
Episode 7700 Average snake length without exploration: 30.76
Episode 7800 Average snake length without exploration: 29.32
Episode 7900 Average snake length without exploration: 31.44
Episode 8000 Average snake length without exploration: 18.92
Episode 8100 Average snake length without exploration: 24.76
Episode 8200 Average snake length without exploration: 19.12
Episode 8300 Average snake length without exploration: 20.32
Episode 8400 Average snake length without exploration: 24.64
Episode 8500 Average snake length without exploration: 25.4
Episode 8600 Average snake length without exploration: 28.28
Episode 8700 Average snake length without exploration: 26.76
Episode 8800 Average snake length without exploration: 31.44
Episode 8900 Average snake length without exploration: 27.16
Episode 9000 Average snake length without exploration: 28.92
Episode 9100 Average snake length without exploration: 26.44
Episode 9200 Average snake length without exploration: 27.28
Episode 9300 Average snake length without exploration: 27.36
Episode 9400 Average snake length without exploration: 27.64
Episode 9500 Average snake length without exploration: 31.64
Episode 9600 Average snake length without exploration: 27.32
Episode 9700 Average snake length without exploration: 31.4
Episode 9800 Average snake length without exploration: 31.68
Episode 9900 Average snake length without exploration: 32.24
Episode 10000 Average snake length without exploration: 23.72
Warning: QT_DEVICE_PIXEL_RATIO is deprecated. Instead use:
QT_AUTO_SCREEN_SCALE_FACTOR to enable platform plugin controlled per-screen factors.
QT_SCREEN_SCALE_FACTORS to set per-screen DPI.
QT_SCALE_FACTOR to set the application global scale factor.
qt.qpa.fonts: Unable to open default EUDEC font: "EUDEC.TTE"
Generating data for animation...
```



Videos on snake game :

[How to train AI to play snake using deep reinforcement learning](#)

[Python Game Programming Tutorial: Snake Game Part 1](#)

Refances

<https://github.com/patrickloeber/snake-ai-pytorch>

Snake-Reinforcement-Learning

[eidenyoshida/Snake-Reinforcement-Learning: Applying basic reinforcement learning principles using the Snake game in Python](#)

[eidenyoshida/Snake-Reinforcement-Learning: Applying basic reinforcement learning principles using the Snake game in Python](#)

[Rishit-katiyar/snake-reinforcement-learning: This repository contains a Python implementation of the classic Snake game using reinforcement learning techniques. The game is built with Pygame and includes features such as obstacle generation, Q-learning, and visualization of training progress.](#)