**Prompt**

Can you refactor the code you produced just before the RL questions so the AI logic will not use BFS but instead load the DQN model and incorporate that instead. Preserve all the other logic including the sounds and the scoring in the final code.

**Reasoned for 42 seconds**

**Refactoring the snake game**

I’m evaluating the integration of a DQN model to control the snake in place of BFS, while ensuring the preservation of sound and scoring functionalities.

**Implementing a DQN model**

I’m thinking about integrating a pre-trained DQN model with Python's stable\_baselines3 into the snake game, ensuring it replaces the original BFS logic and maintains the game’s sound and scoring features.

**Clarifying code specifics**

I’m deconstructing the task into segments: adding traps, coding the scoring system, and incorporating RL. The focus is on transitioning from BFS logic to a DQN model.

**Switching gears**

I’m pulling together the idea that the snake game code, originally using BFS and Pygame, needs to be updated to integrate a DQN model for AI decision-making, while preserving other game elements.

**Integrating DQN logic**

I’m pulling together the effort to merge a stable\_baselines3 DQN model into the snake game code, where AI logic will replace BFS and provide actions using the model’s predict function.

**Refining the game**

Integrating Pygame elements, AI logic with a DQN model, diverse sound effects, and a dynamic scoring system to create a rich, interactive self-playing snake game.

**Crafting observations**

I’m refining the game loop to use a flattened 2D grid (snake, apple, trap) as input for the DQN model, ensuring AI logic aligns with Pygame's format.

**Integrating observation**

I’m ensuring the DQN model, trained on a gym-like environment, uses a flattened 2D grid observation from the snake game. This setup aligns the model's expectations with the game environment.

**Refining the model input**

I’m considering the game loop to process model predictions. The observation will be a reshaped numpy array, ensuring compatibility with stable\_baselines3 DQN model expectations.

**Refining the game loop**

I’m pulling together constants, helper functions, and a main loop that manages AI decision-making, ensuring all elements are effectively integrated for the snake game redesign.

**Setting up**

I’m initializing Pygame, loading sounds, setting up the game window, and ensuring the DQN model is ready for predictions. Now, I’m preparing to define the observation function.

**Constructing the grid**

I’m outlining a function to craft a numpy array grid, coding snake segments, apple, and traps. This grid will be passed to the DQN model for prediction.

**Determining input shape**

I’m focusing on the DQN model's expected input, whether a flattened vector or 2D grid, to ensure the observation aligns with its training setup.

**Confirming shapes**

I’m confirming the observation shape as either a flattened vector or a 2D array based on the DQN model's training configuration and the gym environment's setup.

**Mapping actions**

I’m detailing how the action from the model predicts the snake's movement direction, ensuring the game loop integrates sound, scoring, and collision logic.

**Crafting the game mechanics**

I’m adding a function to fetch observations, ensuring it aligns with constants like cell size and grid dimensions, and integrating previous game mechanics and logic.

**Integrating constants**

I’m piecing together configuration constants like cell size and grid dimensions, along with color schemes and direction mappings, to shape the game mechanics and visual aesthetics.

**Crafting the game**

I’m constructing the snake game with Pygame, integrating AI decision-making, collision handling, and sound effects, ensuring the user experience is engaging and smooth.

**Enhancing gameplay**

I'm working through the snake game, adding collision with traps, visual updates for fruit, traps, and snake segments, and ensuring sound effects for enhanced user engagement.

elow is a complete refactored version of the Snake game that preserves the scoring, sounds, traps, and random-colored apples—but now the AI decision making uses a pre‐trained DQN model (loaded from disk) rather than BFS. (If the DQN model cannot be loaded, the game will fall back to random moves.) Make sure that you have the files chirp.wav, crash.wav, and your trained DQN model file (e.g. dqn\_snake\_model.zip) in the same folder as this script. You also need to have installed the required packages:

pip install pygame stable-baselines3 numpy

import pygame

import random

import sys

import numpy as np

from stable\_baselines3 import DQN # Used to load the trained model

# === Configuration Constants ===

CELL\_SIZE = 20

GRID\_WIDTH = 20

GRID\_HEIGHT = 20

WINDOW\_WIDTH = CELL\_SIZE \* GRID\_WIDTH

WINDOW\_HEIGHT = CELL\_SIZE \* GRID\_HEIGHT

FPS = 10 # Frames per second

# Colors (RGB)

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

GREEN = (0, 255, 0) # Snake color

PURPLE = (128, 0, 128) # Trap color

# Candidate apple colors (avoid conflicts with snake/traps/black/white)

APPLE\_COLORS = [

(255, 0, 0), # Red

(255, 165, 0), # Orange

(255, 255, 0), # Yellow

(0, 0, 255), # Blue

(0, 255, 255), # Cyan

(255, 0, 255) # Magenta

]

# Mapping from discrete actions to movement directions.

# 0: UP, 1: DOWN, 2: LEFT, 3: RIGHT

ACTION\_TO\_DIRECTION = {

0: (0, -1),

1: (0, 1),

2: (-1, 0),

3: (1, 0)

}

# Dictionary to get the opposite direction for prevention of reversal.

OPPOSITE\_DIRECTION = {

(0, -1): (0, 1),

(0, 1): (0, -1),

(-1, 0): (1, 0),

(1, 0): (-1, 0)

}

# === Helper Functions ===

def get\_random\_free\_position(occupied):

"""Return a random (x, y) position on the grid that is not occupied."""

while True:

pos = (random.randint(0, GRID\_WIDTH - 1), random.randint(0, GRID\_HEIGHT - 1))

if pos not in occupied:

return pos

def get\_observation(snake, apple, traps):

"""

Create a grid-based observation as a 2D NumPy array.

0 = empty, 1 = snake, 2 = apple, 3 = trap.

"""

obs = np.zeros((GRID\_HEIGHT, GRID\_WIDTH), dtype=np.int8)

for (x, y) in snake:

obs[y, x] = 1

ax, ay = apple

obs[ay, ax] = 2

for (x, y) in traps:

obs[y, x] = 3

return obs

def draw\_grid(surface):

"""Draw grid lines on the provided surface."""

for x in range(0, WINDOW\_WIDTH, CELL\_SIZE):

pygame.draw.line(surface, WHITE, (x, 0), (x, WINDOW\_HEIGHT))

for y in range(0, WINDOW\_HEIGHT, CELL\_SIZE):

pygame.draw.line(surface, WHITE, (0, y), (WINDOW\_WIDTH, y))

def play\_crash\_sound(crash\_sound):

"""

Play the crash sound 2–3 times consecutively with a short delay between plays.

"""

if crash\_sound:

times = random.randint(2, 3)

for \_ in range(times):

crash\_sound.play()

pygame.time.wait(300)

# === Main Game Function ===

def main():

pygame.init()

pygame.mixer.init()

# Load sounds.

try:

chirp\_sound = pygame.mixer.Sound("chirp.wav")

except pygame.error:

print("Could not load chirp.wav. Please ensure it is in the same folder.")

chirp\_sound = None

try:

crash\_sound = pygame.mixer.Sound("crash.wav")

except pygame.error:

print("Could not load crash.wav. Please ensure it is in the same folder.")

crash\_sound = None

# Load the pre-trained DQN model.

try:

model = DQN.load("dqn\_snake\_model")

print("DQN model loaded successfully.")

except Exception as e:

print("Error loading DQN model:", e)

model = None

screen = pygame.display.set\_mode((WINDOW\_WIDTH, WINDOW\_HEIGHT))

pygame.display.set\_caption("AI Snake Game with DQN")

clock = pygame.time.Clock()

# Initialize snake: start with 3 segments.

snake = [

(GRID\_WIDTH // 2, GRID\_HEIGHT // 2),

(GRID\_WIDTH // 2 - 1, GRID\_HEIGHT // 2),

(GRID\_WIDTH // 2 - 2, GRID\_HEIGHT // 2)

]

current\_direction = (1, 0) # Initially moving right.

# Place the first apple.

occupied = set(snake)

apple = get\_random\_free\_position(occupied)

apple\_color = random.choice(APPLE\_COLORS)

# Initialize traps.

traps = []

trap\_interval = 1000 # Add one trap every 1000 ms (1 second).

last\_trap\_time = pygame.time.get\_ticks()

score = 0

running = True

while running:

clock.tick(FPS)

current\_time = pygame.time.get\_ticks()

# Process events.

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

# Add a new trap every second.

if current\_time - last\_trap\_time >= trap\_interval:

occupied\_for\_trap = set(snake) | set(traps) | {apple}

new\_trap = get\_random\_free\_position(occupied\_for\_trap)

traps.append(new\_trap)

last\_trap\_time = current\_time

# --- AI Decision Making using DQN ---

obs = get\_observation(snake, apple, traps)

# Expand dimensions to add batch dimension (model expects shape: (1, height, width)).

if model is not None:

action, \_ = model.predict(obs[None, ...], deterministic=True)

action = int(action)

else:

# Fallback: choose a random valid action.

action = random.choice([0, 1, 2, 3])

new\_direction = ACTION\_TO\_DIRECTION[action]

# Prevent immediate reversal if the snake has more than one segment.

if len(snake) > 1 and new\_direction == OPPOSITE\_DIRECTION[current\_direction]:

new\_direction = current\_direction

current\_direction = new\_direction

# --- Move the Snake ---

head\_x, head\_y = snake[0]

dx, dy = current\_direction

new\_head = (head\_x + dx, head\_y + dy)

# Check for collision with walls.

if (new\_head[0] < 0 or new\_head[0] >= GRID\_WIDTH or

new\_head[1] < 0 or new\_head[1] >= GRID\_HEIGHT):

play\_crash\_sound(crash\_sound)

print("Game over! Final score:", score)

running = False

continue

# Check for collision with itself.

if new\_head in snake:

play\_crash\_sound(crash\_sound)

print("Game over! Final score:", score)

running = False

continue

# If the snake eats the apple.

if new\_head == apple:

snake.insert(0, new\_head) # Grow the snake.

apple\_reward = 10 + len(snake) + len(traps)

score += apple\_reward

if chirp\_sound:

chirp\_sound.play()

# Place a new apple (avoid snake and traps) with a random color.

occupied = set(snake) | set(traps)

apple = get\_random\_free\_position(occupied)

apple\_color = random.choice(APPLE\_COLORS)

# If the snake hits a trap.

elif new\_head in traps:

snake.insert(0, new\_head)

new\_length = max(1, len(snake) // 2)

snake = snake[:new\_length]

score -= 10

play\_crash\_sound(crash\_sound)

else:

# Normal move: add new head and remove tail.

snake.insert(0, new\_head)

snake.pop()

# --- Rendering ---

screen.fill(BLACK)

draw\_grid(screen)

# Draw the apple.

ax, ay = apple

pygame.draw.rect(screen, apple\_color, (ax \* CELL\_SIZE, ay \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

# Draw traps.

for (tx, ty) in traps:

pygame.draw.rect(screen, PURPLE, (tx \* CELL\_SIZE, ty \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

# Draw the snake.

for (sx, sy) in snake:

pygame.draw.rect(screen, GREEN, (sx \* CELL\_SIZE, sy \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

# Draw the score.

font = pygame.font.SysFont("Arial", 24)

score\_text = font.render("Score: " + str(score), True, WHITE)

screen.blit(score\_text, (10, 10))

pygame.display.update()

pygame.quit()

sys.exit()

if \_\_name\_\_ == "\_\_main\_\_":

main()

**How This Works**

1. **Loading the Model:**  
   The code attempts to load a trained DQN model from dqn\_snake\_model.zip using Stable Baselines3. If the model cannot be loaded, the AI will choose random actions as a fallback.
2. **Observation Construction:**  
   The current game state (snake body, apple, and traps) is converted into a 2D NumPy array where each cell is encoded as follows:
   * 0: Empty
   * 1: Snake
   * 2: Apple
   * 3: Trap  
     This observation is then expanded (to add a batch dimension) before being passed to the model.
3. **Action Selection:**  
   The loaded DQN model’s predict method is used to choose an action (0–3), which is then mapped to a movement direction. The code also prevents the snake from immediately reversing direction.
4. **Game Logic:**  
   The rest of the game logic remains the same—handling collisions, eating apples (with scoring and chirp sound), hitting traps (with crash sound and length reduction), and drawing the grid, snake, traps, and score.
5. **Integration:**  
   The final script is self‑contained and can be used both as a demonstration of the trained DQN agent playing the game and as a “production” version of your AI-powered snake game.

Simply run the script once you have the required files in place, and you’ll see your AI-controlled snake game in action with all the original features preserved. Enjoy!

o3-mini-high

Top of Form

Search

Bottom of Form