CAPSTONE PROJECT

SELF REINFORCEING AI-SNAKE GAME

PRESENTED BY
NAME : ATISH KUMAR MONDAL
COLLEGE : UNIVERSITY OF ENGINEERING
& & MANAGEMENT
DEPARTMENT : COMPUTER SCIENCE & TECHNOLOGY
EMAIL ID : atishkrmondal5@gmail.com
AICTE STUDENT ID:
STU680e215a9b1b51745756506

OUTLINE

- Problem Statement
- Proposed System/Solution
- System Development Approach
- Algorithm & Deployment
- Result
- Conclusion
- Future Scope
- References

PROBLEM STATEMENT

Traditional Snake games rely on hardcoded rules or human input for movement. These versions lack adaptability and cannot learn from experience. The challenge is to create a snake agent that:

- Learns from the environment.
- Makes decisions dynamically.
- Improves performance through trial and error.

Goal: Develop a self-learning Snake agent using Reinforcement Learning to play autonomously and adapt over time.

PROPOSED SOLUTION

To train an AI agent to play Snake, we use Deep Q-Learning, allowing it to learn strategies through rewards and trial-and-error within the game environment.

Solution Highlights:

- **Neural Network (DNQ_model.py):** Predicts Q-values (expected rewards) for three possible actions: straight, left, or right.
- **Agent Logic (A.I_player_model.py):** Manages training, decision-making, reward evaluation, experience storage, and replay training.
- **Reward System:**
  - **>** +1 for eating food
  - **>** game over for dying (collision)
  - **>** 0 for regular moves
- **Game Interface (snake_game_engine.py):** Runs the game, handles snake movement, food placement, and collision detection.
- **Exploration Strategy:** Uses epsilon-greedy approach to balance between trying new actions and sticking to learned strategies.
- **Training Visualization (game_assist.py)**: Plots scores and performance during training to monitor progress.

SYSTEM APPROACH

This section outlines the core components and tools used to implement the AI Snake Game using reinforcement learning:

System Requirements:

Python 3.8+
64-bit OS (Windows/Linux/macOS)
Minimum 4GB RAM

Libraries Used:

- **Pygame:** For developing the Snake game environment and rendering visuals.
- **PyTorch:** To build and train the deep Q-learning neural network.
- **Matplotlib:** For visualizing score trends and model performance.
- **Numpy:** For efficient numerical computations.
- **Collections (deque):** For storing gameplay memory efficiently.

SYSTEM APPROACH

Code Structure:

- **Snake_game_engine.py:** Handles game logic and environment.
- **DNQ_model.py:** Builds the neural network used for action prediction.
- **A.I_player_model.py:** Manages Q-learning, memory, and action decisions.
- **Game_assist.py:** Visualizes results such as scores and averages during training.

This structured modular approach ensures that each component has a dedicated responsibility, making the system easier to manage and extend.

ALGORITHM & DEPLOYMENT

Algorithm:

- Used Deep Q-Learning—a reinforcement learning method with a neural network to estimate Q-values.
- Ideal for dynamic, sequential environments like Snake.

Inputs:

- State vector includes:
- Snake direction
- Food location
- Immediate danger (straight, right, left)

ALGORITHM & DEPLOYMENT

Training:

- Epsilon-greedy policy
- Short-term training per move
- Experience replay for long-term learning
- Optimized via stochastic gradient descent

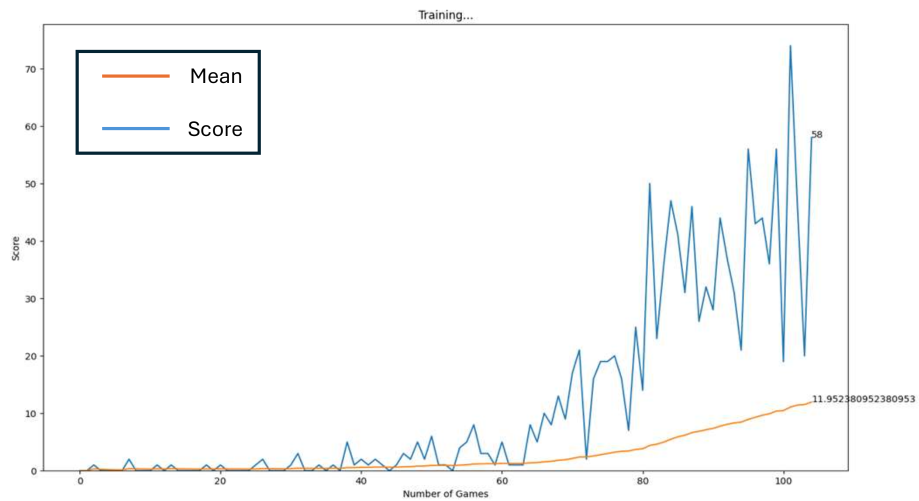
Prediction:

Model outputs Q-values; highest is chosen as the next move.

Deployment:

- Runs in Pygame.
- Trained model loaded into agent.py for autonomous play.
- Visualization via matplotlib.

RESULT



Scores vs Number of Games

- Results based on accounted observation of the A.I agent performing actions in 105 games.
- The graph is plotted taking in account the scores (y axis) against number of games (y axis) played by the A.I agent.
- The orange line shows the mean score value of the total score against the total games played .
- Further insights are stated in the next slide:

RESULT

Performance Insights:

- Scores increased steadily over 100 games i.e. :
  >Initially low.
  >Improves score after ~50 games.
  >Better score is been obtained at ~100 game mark.
- Thus in the graph the leaning curve consistently increases.
- Mean score improved from 0 - ~ 12.
- Maximum score obtained out of 105 games are 74 showing agent’s capacity to survive, explore new possibilities and collect food effectively.
- Stability of the Reinforcement learning model is medium.
- Over all agent growth over time is good as per the scores obtained.

CONCLUSION

- Successfully developed a self-improving AI Snake Game using Reinforcement Learning (DQN).
- The AI agent learns through trial and error, improving gameplay with experience.
- Incorporated key techniques: state representation, epsilon-greedy action selection, and experience replay.
- Integrated performance visualizations for tracking training effectiveness.
- System enhancements like game-over caps, score logging, and performance plots provided better control and insights.

Key Outcomes :

- Agent performance improved significantly over training cycles.
- Achieved stable and increasing average scores across training iterations.
- Performance metrics (score distribution) gave insight into learning behavior.

FUTURE SCOPE

- **Cross-Game Adaptability :**

  The reinforcement learning model can be extended to other 2D/3D games like Pac-Man, Flappy Bird, or simple maze solvers.

- **Real-Time Game Optimization:**

  AI can adjust difficulty dynamically based on player performance, enhancing user engagement.

- **Learning from Player Experience :**

  NPCs and game objects can analyze player behavior, adapting their strategies and responses over time.

- **Personalized Virtual Worlds :**

  Environments and challenges evolve with user interaction, enabling intelligent and customized gameplay.

- **Research & Education :**

  Useful for AI education, game theory research, and autonomous agent simulations.

REFERENCES

- **Sutton, R. S., & Barto, A. G. (2018)** – Reinforcement Learning: An Introduction (2nd Ed.) http://incompleteideas.net/book/the-book-2nd.html
- **Mnih, V. et al. (2015)** – Human-level control through deep reinforcement learning, Nature https://www.nature.com/articles/nature14236
- **PyTorch Documentation** – Deep Learning Framework https://pytorch.org/docs/stable/index.html
- **OpenAI Gym** – Toolkit for developing and comparing RL algorithms https://www.gymlibrary.dev/
- **Matplotlib Documentation** – Data visualization in Python https://matplotlib.org/stable/contents.html
- **GitHub Repositories & Tutorials** – Practical RL implementations in games (e.g., Snake RL AI) github.com/python-engineer/snake-ai-pytorch

Further more the project is available in my Github repository (Self_Reiforceing_A.i_Snake_Game) :

Git hub repository link :

https://github.com/Atish004/Self_Reiforceing_A.i_Snake_Game.git

Thank you