Fire Rescue Simulation

By

KARIM TAREK 195322

MOHAMED FARID 194308

THOMAS EHAB 181954

MOHAMED MADKOUR

Module: Artificial Neural Networks

Project Documentation

Module Leader: Dr. Sally Saad

Teaching Assistant: Eng. Ameera Amer

Data Set Used

In this project no dataset was used, we implemented the Reinforcement learning model where an agent learns to make decisions by interacting with an environment. The goal of the agent is to maximise a cumulative reward signal over time. Unlike supervised learning where a dataset with labelled examples is used, RL operates based on trial and error, learning from feedback received through interactions with the environment.

The agent observes the current state of the environment, takes an action based on its policy (a strategy for selecting actions), receives a reward from the environment based on the action taken, and transitions to a new state. The agent's objective is to learn a policy that maximises the expected cumulative reward over time.

The game UI features:

- Display: Rendered area with a fixed size of 640x480 pixels.
- Fire Man: Moves around as a series of blocks.
- Victim: Appears as a red block.
- Walls: Grey blocks outlining the game area, preventing the hero from moving out.
- Fires: Orange blocks randomly appearing, endangering the hero; collision ends the game.
- Score: Displayed at the top left, tracking player/agent points.
- Controls: Direction of the hero controlled by player or AI agent via reinforcement learning.

Design Details

1. Artificial Neural Network (ANN) Architecture:

- The neural network architecture used for Q-learning is a simple feedforward neural network.
- The network, named Linear_QNet, consists of two linear layers.
- The input size of the network depends on the state representation chosen for the game. This input size is determined by the number of features used to represent the game state.
- The hidden layer size is a parameter specified during initialization.
- The output size of the network corresponds to the number of possible actions the agent can take in the game (in this case, there are three actions: move left, move right, or continue straight).
- ReLU activation function is applied after the first linear layer to introduce non-linearity.

2. **Training**:

The QTrainer class is responsible for training the Q-network.
It uses the Mean Squared Error (MSE) loss function to measure the difference between predicted Q-values and target Q-values.
The optimizer used for training is Adam optimizer with a specified learning rate (lr).
The training step involves computing the Q-value targets for each state-action pair based on the Bellman equation, and then updating the weights of the neural network to minimise the loss between predicted and target Q-values.

3. Training Loop:

During each training iteration, the agent interacts with the environment, observes the current state, selects an action based on the current policy (epsilon-greedy), receives a reward, and transitions to the next state.
The train_step method in the QTrainer class is called to update the Q-network based on the observed transition.
Training continues for multiple episodes until convergence.

4. Model Saving:

The trained model is saved periodically using the save method of the Linear_QNet class. The saved model parameters are stored in a specified directory.

Implementation Details (Libraries Used)

1. Pygame:

Pygame is a set of Python modules designed for writing video games. It includes computer graphics and sound libraries. In this project, Pygame is used for:

- Handling the game display
- Drawing shapes and images on the screen
- Managing user input (keyboard events)
- Implementing game logic such as collision detection and updating game state

2. PyTorch:

PyTorch is an open-source machine learning library used for tasks such as deep learning and neural networks. In this project, PyTorch is used for implementing the AI agent. Specifically, it's used for:

- Defining neural network architectures (Linear_QNet class)
- Training the Q-learning model (QTrainer class)
- Performing forward passes through the neural network to get Q-values for state-action pairs

3. Other Libraries:

- random: Used for generating random numbers, particularly for placing food and fires randomly on the game board.
- enum: Used for defining enumerations for directions.
- collections.namedtuple: Used for defining named tuples to represent points in the game.
- numpy: Used for numerical computations, particularly for handling arrays and matrix operations.
- os: Used for file path manipulation and handling directories, specifically for saving and loading the trained model.

Implementation Details:

1. FireMan Game Class (FireManGameAI):

- Manages the game environment, including the Fire Man, food, fires, walls, and game state.
- Handles game logic such as moving the Fire Man, checking for collisions, updating the UI, and spreading fires.
- Provides a method (play_step) for the AI agent to interact with the game environment.

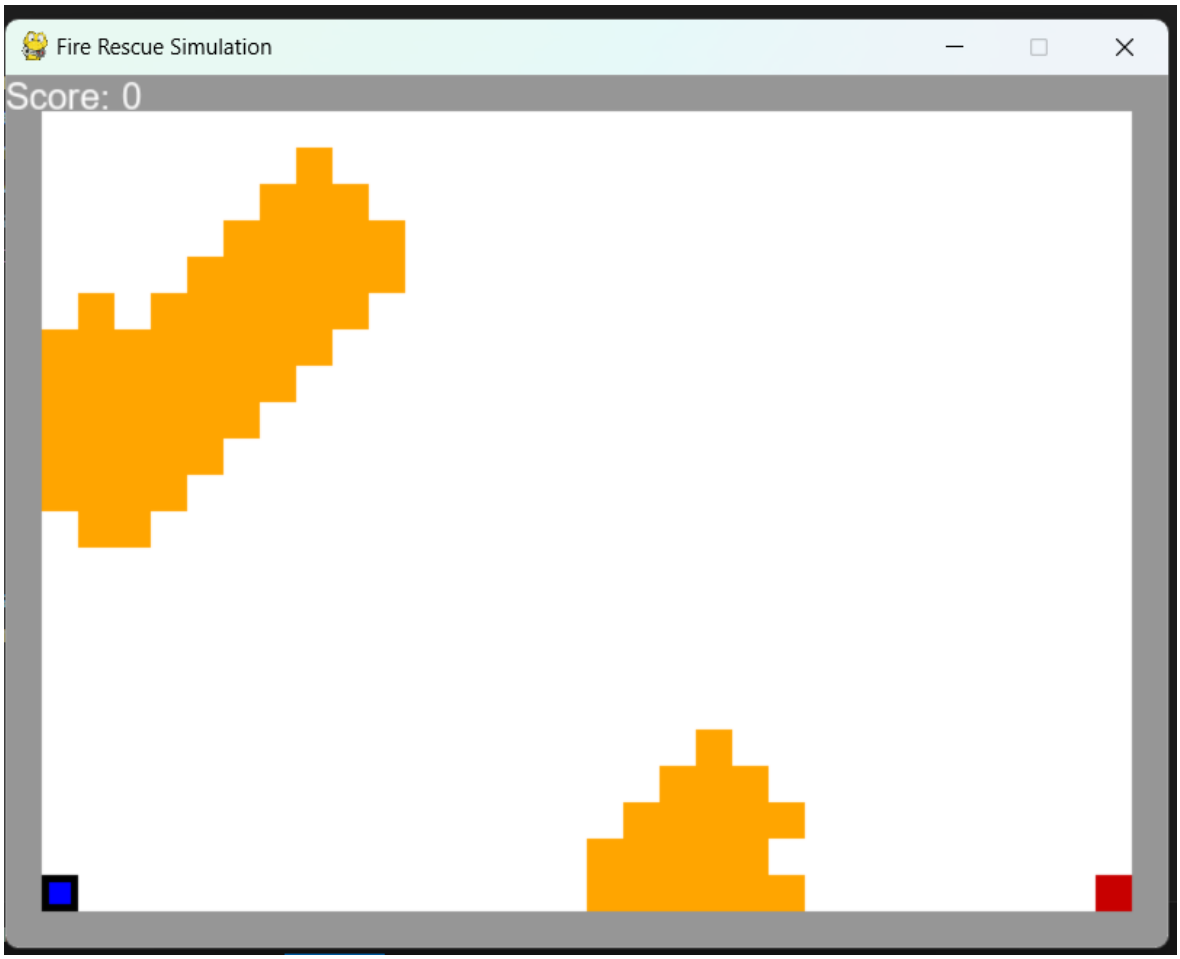
2. AI Model and Trainer Classes:

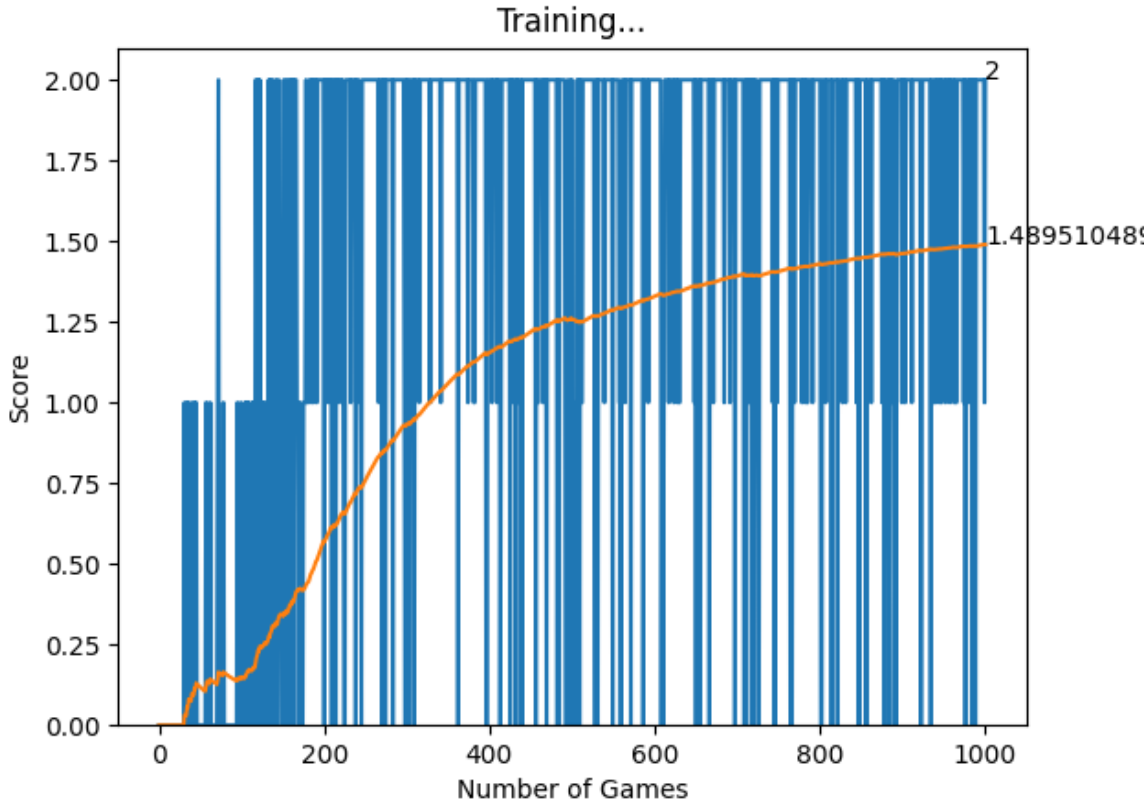
- Linear_QNet: Defines a simple feedforward neural network with one hidden layer for Q-learning.
- QTrainer: Manages the training process of the Q-learning model using the provided Q-values and rewards.

3. Integration:

- The Pygame-based game environment interacts with the PyTorch-based AI agent. The AI agent makes decisions based on the current state of the game and receives rewards accordingly.
- The Q-learning model is trained iteratively as the game progresses, updating the Q-values based on the observed rewards and transitions.

Screenshots of the output





This is the plot of the training of the agent on the fire rescue simulation environment. The agent was trained for 1000 games on a speed of 400. The average score and the mean score are plotted. The score of 2 means that the hero rescued the victim successfully the score of 1 means that the agent reached the victim but failed to rescue them or the victim was not reachable due to colliding with fire therefor the hero returned to the starting point. Score of 0 means that the agent did collide with a wall or fire before reaching the victim or reached the allowed number of steps.

Accuracy Achieved

To test for accuracy, we considered a positive reward to be a true positive and a negative reward to be a false positive as a true reward indicates desirable behaviour of the model and a negative reward indicates undesirable behaviour. 270 games were simulated using the pretrained model saved after being trained on 100 games. 247 games yielded positive rewards and 23 games yielded negative rewards. These are the evaluation metrics of the

pretrained model.

```
Number of elements in the array: 270
Number of positive rewards: 245
Number of negative rewards: 23
Accuracy: 0.9148148148148149
Precision: 0.9148148148148149
Recall: 1.0
F1 Score: 0.9555125725338491
```

Reward function Explained

The reward function here was obtained through trial and error over a few weeks. Some earlier iterations of the reward function yielded undesirable behaviour such as the hero colliding with the fire instead of saving the victim as the numbers of steps had a large penalty so the hero preferred to have fewer steps and not save the victim to have a better reward. However after a few iterations of the reward function a successful implementation of the rewards was reached and it is as follows.

- If the hero collides with the fires or has over two hundred steps or four hundred of the victim is being rescued then the reward is -10 with an additional penalty of number of steps over 50

```
109             if self.is_collision() or self.frame_iteration > 200*len(self.rescuers):
110                 game_over = True
111                 if self.victim_count == 2:
112                     reward = 10-self.frame_iteration/50
113                 reward = -10-self.frame_iteration/50
114                 return reward, game_over, self.score,self.frame_iteration
```

- If the hero reaches the victim a reward of 10 was awarded with an additional penalty of number of steps over 50

```
16              if self.hero == self.victim:
17                  self.score += 1
18                  reward = 10-self.frame_iteration/50
```

- If the hero rescues the victim (takes him back to the starting point) a reward of 20 is given with an additional penalty of number of steps over 50

```
121                 else:
122                     game_over = True
123                     reward=20-self.frame_iteration/50
```