PySnake

Kaj Habegger & Fabio Lenherr

Miniproject

Table of Contents

1. Introduction

We decided to do a small reinforcement learning project. As we have never done anything in terms of reinforcement learning before, we used this video as a starting point. It is a tutorial of creating a Snake game and design a reinforcement learning AI model which is able to play the game after some iterations.

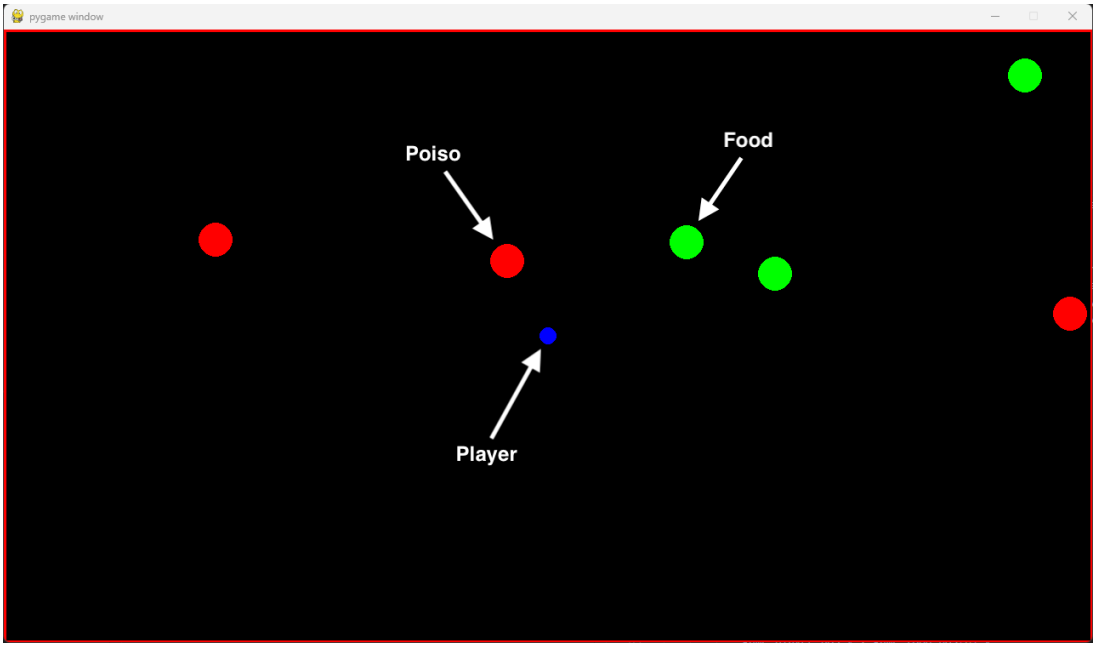
2. Tooling

Like the guy in the tutorial we used Pytorch to design our AI model and Pygame to create our game. It is worth pointing out that we planned to use Pytorch anyway because we wanted to try out another tool than Tensorflow. Furthermore, Pytorch also supports AMD graphic cards and not only Nvidia graphic cards like Tensorflow. Pygame is a composition of Python modules like computer graphic and sound libraries. Creating a game with Pygame is straightforward.

3. Making Of

3.1. The Game

We wanted to create our own little game and not just copy the game from the tutorial we watched. Therefore, we designed a two dimensional game where the player is a small ball and has to collect or eat other balls. There are two different types of balls to eat; green and red ones. By collecting a green ball the player grows in size and gets smaller when collecting red balls respectively. If the player has minimal size and collects a red ball the game is over. Touching the wall also ends the game. As the player grows each time eating a green ball it gets more difficult over time to avoid the wall as well as red balls.



3.2. The AI

Although we looked at reinforcement learning in a theoretical part at the lectures it was still an unknown field in a practical way. Therefore, we initially invested some time to understand how the basics work. We did this by reading through the code base of the already mentioned tutorial. Furthermore, some parts are copied from the tutorial code base. The AI of our project is divided into an agent- and a model part.

3.2.1. Model

The model consists of two classes. One of which is the Linear_QNet. The Linear_QNet holds the two methods for forward propagation as well as the save method to save the model to a file. It is basically the model network representing the input, hidden and output layer. The other class in the model is the QTrainer it holds only one function which is called train_step. This train_step method is used after each action the agent performs to optimize the model. Roughly described the train_step method updates the loss and performs backward propagation.

3.2.2. Agent

The Agent consists of a single class which is called agent aswell. Besides the methods it holds all important parameters for our AI. Those parameters are:

- Epsilon: Controls how much percent of the AI actions should initially be random rather than predicted by the model. This parameter is important as the AI has no idea what it should do at the beginning. As the AI learns over time this parameter value gets lowered after each game over.
- Gamma: The discount rate is a parameter that determines the relative importance of immediate rewards compared to future rewards. It represents the extent to which the agent values immediate rewards over delayed rewards. A discount rate less than 1 means the agent considers future rewards, while a discount rate closer to 0 places more emphasis on immediate rewards.
- Memory: The memory is represented by a deque datastructure. It is basically the memory of the AI where it can store current states, actions, rewards, next states and if it is currently in game over mode.
- Model: This field holds an instance of the previously described Linear_QNet class. Furthermore, here we define how many neurons the input-, hidden- and output-layer has.
- Trainer: This field holds an instance of the previously mentioned QTrainer.

CODING BUGS

TRYING OUT DIFFERENT VALUES (reward, reward by how close the player is to obstacles)

MULTI OBSTACLES

STATE INPUTS