This project focuses on writing an AI which is capable of learning how to play the game "Snake". This game involves controlling an agent composed out of several blocks to eat food within a closed boundary, with the aim being to eat as many blocks as possible without dying. The following rules apply within the game:

* If the snake hits the edge of the boundary the game is over
* If the snake hits itself the game is over
* Each time the snake eats a piece of food a block is added to its length

To complete this task reinforcement learning will be employed and, in particular, Q-learning is used.

The general workflow of this task is now detailed. There are two main parts to this:

* Game, implemented using Pygame. Takes in an action from the game and returns the reward, whether the game is over and the current score
* Model, implemented using PyTorch. A deep neural net which takes in the current state and returns an action

The training has the following form:

1. State = get\_state(Game)
2. Action = get\_action(state)

🡪 Model.predict()

1. Reward, game\_over, score = Game.play\_step(action)
2. New\_state = get\_state(Game)
3. Remember previous states and new states
4. Model.train()

Some other parameters in the game must also be defined before implementing this game.

The rewards given for different outcomes at each time step:

* +10 if the snake eats a piece of food
* -10 if the snake bumps into a boundary or itself
* 0 otherwise

How we define an action in the game. Actions will be defined as three dimensional vectors. It is expected that there would be four actions for four directions. However, if the snake takes the action of the opposite direction, it will immediately bump into itself and die so this action is not considered. Therefore, actions are:

* [1,0,0] = straight (maintain current direction)
* [0,1,0] = right
* [0,0,1] = left (both left and right are in relation to current direction)

The “state” is the current information about different aspects of the environment and will be composed of eleven values, all of which are boolean:

* Danger straight
* Danger right
* Danger left
* Direction left
* Direction right
* Direction up
* Direction down
* Food left
* Food right
* Food up
* Food down

As for the neural network, a relatively simple architecture will be used to make decisions. This has eleven input values for the states, one hidden layer with four nodes and an output layer with three nodes. A graph of this is given below.

Diagram

Description automatically generated

The max value of the three outputs is then chosen as the action, i.e., .

Q-learning is then implemented using the Bellman equation which is detailed in the equation below.

Graphical user interface, text, application

Description automatically generated