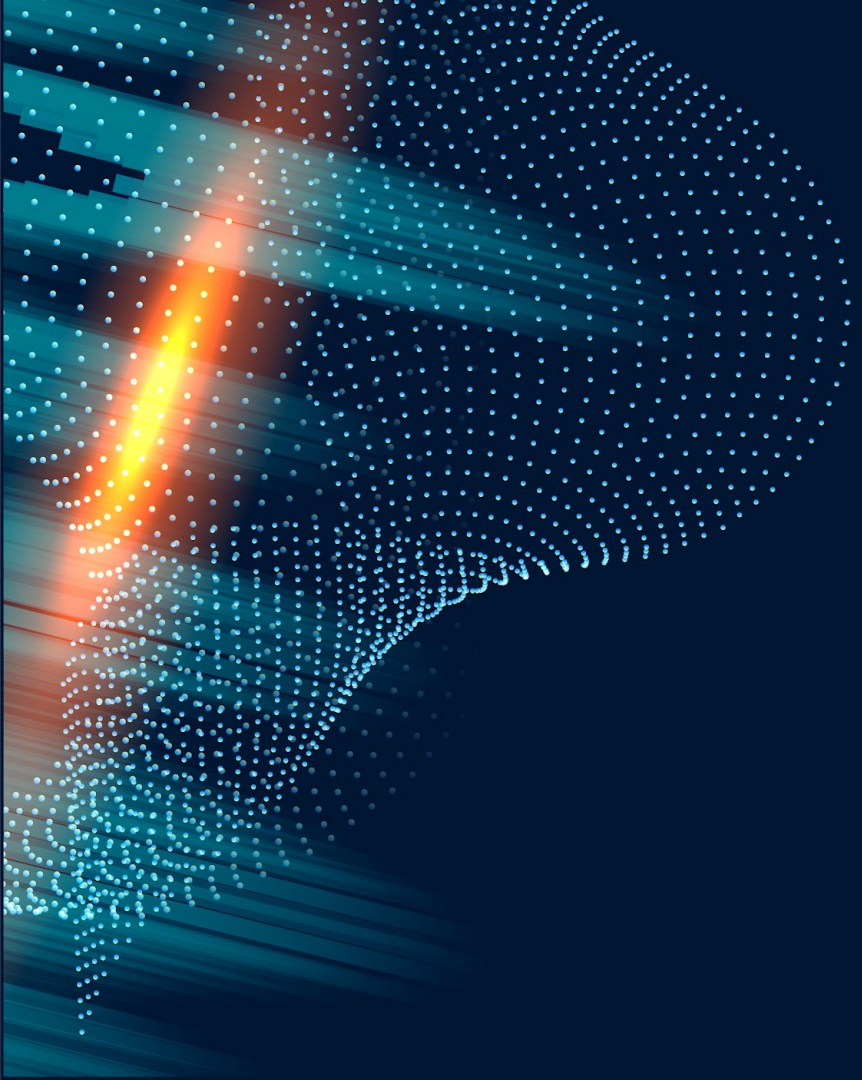
Artificial Intelligence project

Cinvestav Guadalajara

Carlos Cardenas
Emilio Tonix
Julia Abud



Part 5

Reinforcement learning
+ Neural networks (pytorch)

TASKS

16 March 2021 - 12 Abril 2021

- Implement
  - Reinforcement Learning
  - Neural Network using pytorch

Map: DefeatZerglingsAndBanelings

Description

A map with 9 Marines on the opposite side from a group of 6 Zerglings and 4 Banelings. Rewards are earned by using the Marines to defeat Zerglings and Banelings. Whenever all Zerglings and Banelings have been defeated, a new group of 6 Zerglings and 4 Banelings is spawned and the player is awarded 4 additional Marines at full health, with all other surviving Marines retaining their existing health (no restore). Whenever new units are spawned, all unit positions are reset to opposite sides of the map.

Initial State

- 9 Marines in a vertical line at a random side of the map (preselected)
- 6 Zerglings and 4 Banelings in a group at the opposite side of the map from the Marines

Rewards

- Zergling defeated: +5
- Baneling defeated: +5
- Marine defeated: -1

End Conditions

- Time elapsed
- All Marines defeated

Time Limit

- 120 seconds

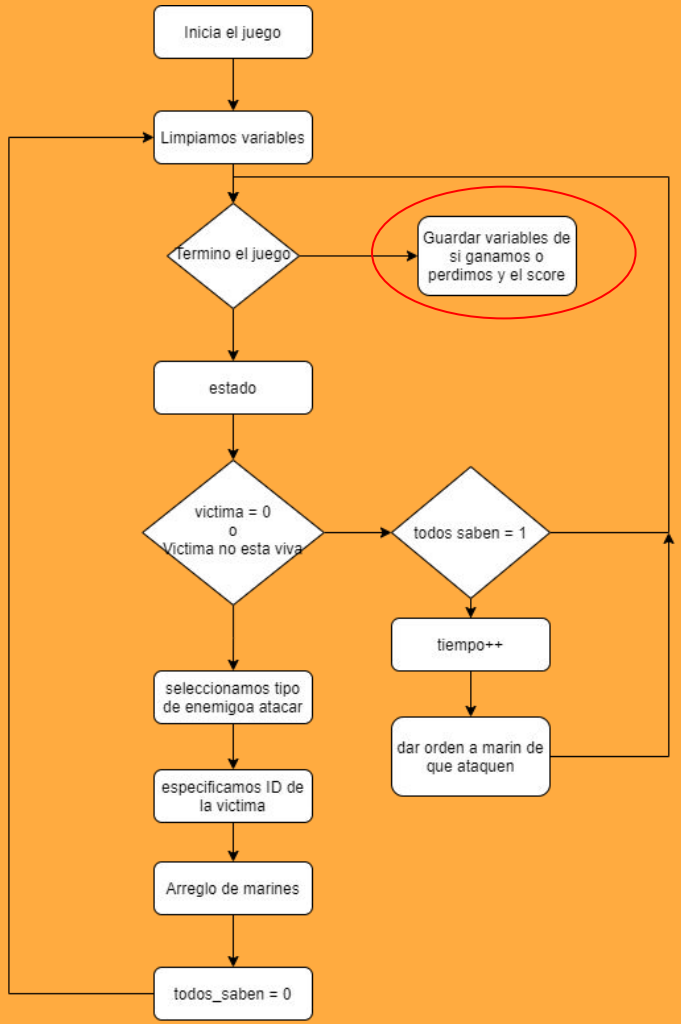
Additional Notes

- Fog of War disabled
- No camera movement required (single-screen)
- This map and DefeatRoaches are currently the only maps in the set that can include an automatic, mid-episode state change for player-controlled units. The Marine units are automatically moved back to a neutral position (at a random side of the map opposite the Roaches) when new units are spawned, which occurs whenever the current set of Zerglings and Banelings is defeated. This is done in order to guarantee that new units do not spawn within combat range of one another.

Enemies

- **6 Zerglings and 4 Banelings**

Reinforcement learning



```
Carlos, a day ago | 1 author (Carlos)
class LinearDeepQNetwork(nn.Module):
    def __init__(self, lr, n_actions, input_dims):
        super(LinearDeepQNetwork,self).__init__()

        self.fc1 = nn.Linear(input_dims, 64)
        self.fc2 = nn.Linear(64, n_actions)

        self.optimizer = optim.Adam(self.parameters(), lr=lr)
        self.loss = nn.MSELoss()
        self.device = 'cpu'

        self.to(self.device)
    def forward(self,state):
        layer1 = F.relu(self.fc1(state))
        actions = self.fc2(layer1)

        return actions
```

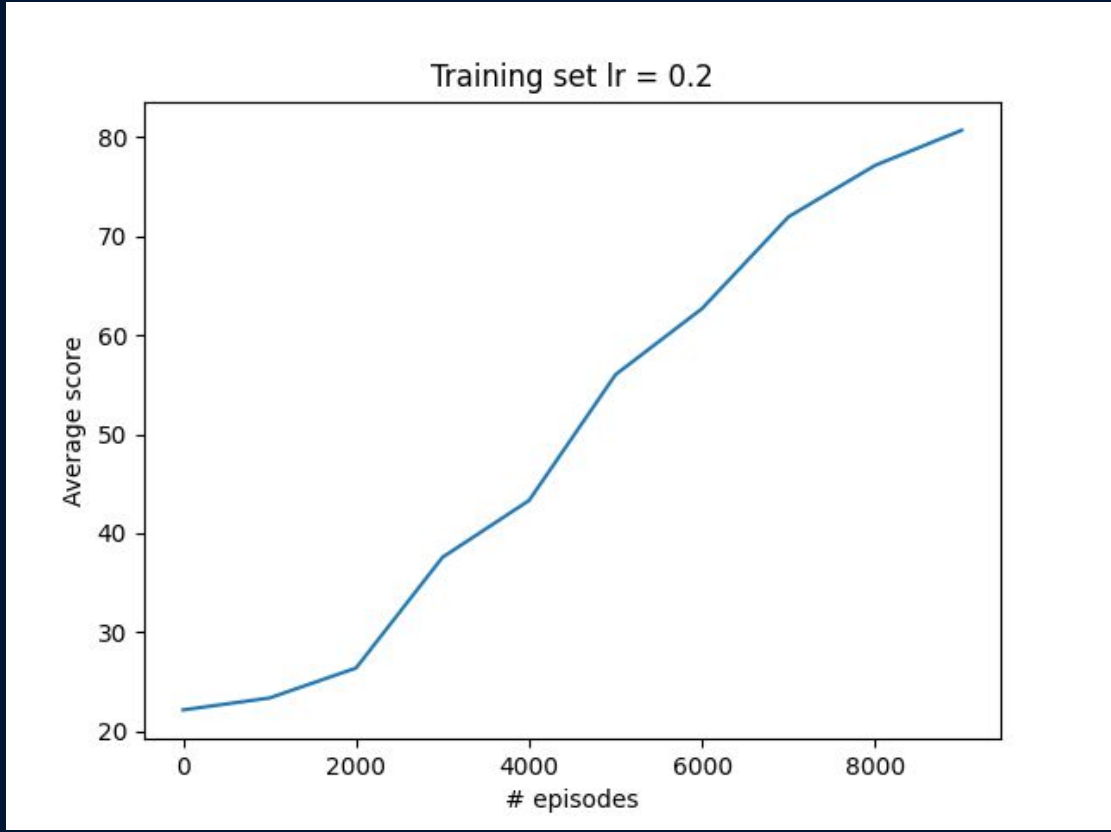
```python
class nnq():
    def __init__(self, input_dims, n_actions, lr, gamma=0.80, epsilon=1.0, eps_dec=1e-5, eps_min=0.01):
        self.lr=lr
        self.input_dims = input_dims
        self.n_actions = n_actions
        self.gamma = gamma
        self.epsilon = epsilon
        self.eps_dec = eps_dec
        self.eps_min = eps_min
        self.action_space = [i for i in range(self.n_actions)]
        self.Q = LinearDeepQNetwork(self.lr,self.n_actions,self.input_dims)

    def choose_action(self, observations):
        if np.random.rand() > self.epsilon:
            state = T.tensor(observations, dtype=T.float).to(self.Q.device)
            actions = self.Q.forward(state)
            action = T.argmax(actions).item()
        else:
            action = np.random.choice(self.action_space)
        return action

    def decrement_epsilon(self):
        self.epsilon = self. epsilon - self.eps_dec if self.epsilon > self.eps_min else self.eps_min

    def learn(self, state,action, reward,state_):
        self.Q.optimizer.zero_grad()
        states = T.tensor(state,dtype=T.float).to(self.Q.device)
        actions = T.tensor(action).to(self.Q.device)
        rewards = T.tensor(reward).to(self.Q.device)
        states_ = T.tensor(state_, dtype=T.float).to(self.Q.device)
        q_pred = self.Q.forward(states)[actions]
        q_next = self.Q.forward(states_).max()
        q_target = reward + self.gamma*q_next
        loss = self.Q.loss(q_target, q_pred).to(self.Q.device)
        loss.backward()
        self.Q.optimizer.step()
        self.decrement_epsilon()
```

Results

[22.16, 23.37, 26.39, 37.58, 43.31, 56.02, 62.69, 71.96, 77.15, 80.69]

Total of 10,000 episodes

LR 0.2

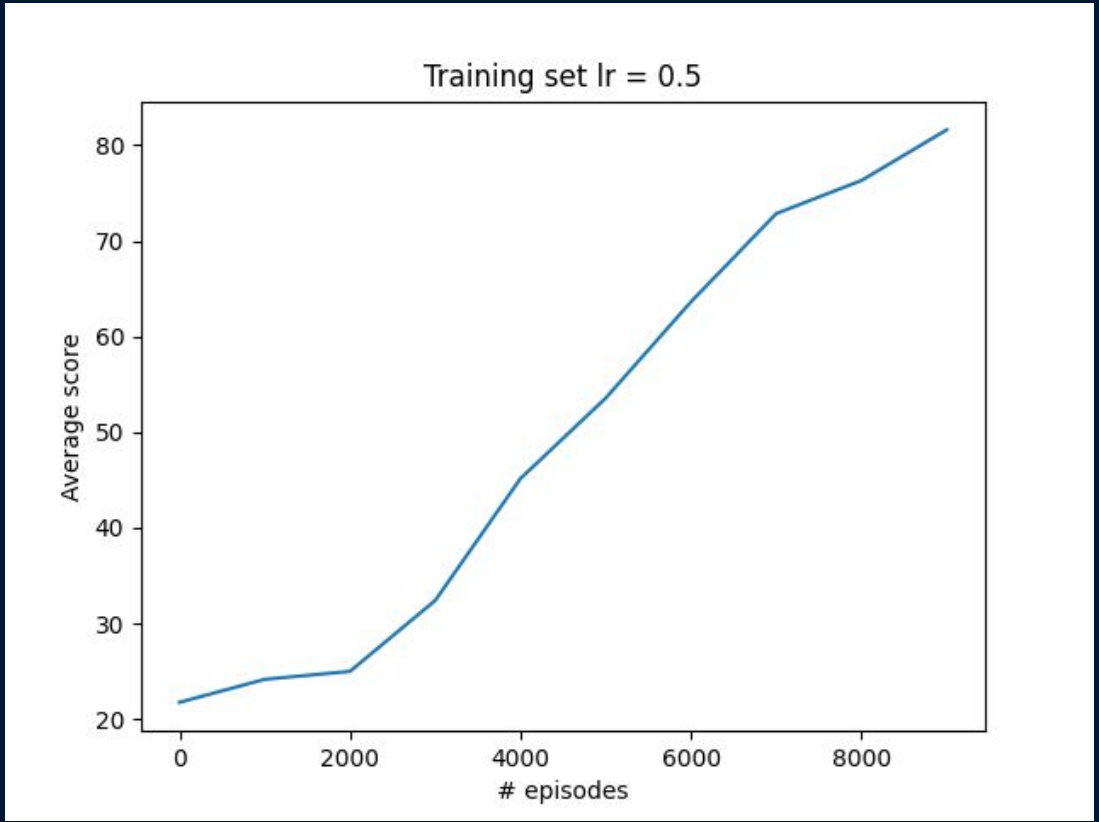


Results

[21.76, 24.17, 25.0, 32.41, 45.15, 53.56, 63.62, 72.84, 76.3, 81.6]

Total of 10,000 episodes

LR 0.5



Results

[20.99, 22.62, 25.57, 36.27, 43.22, 54.9, 65.37, 70.82, 75.32, 84.65]

Total of 10,000 episodes

LR 0.1