SAPIENZA
UNIVERSITÀ DI ROMA

Reasoning Agent Project

Students:
Nguyen Ngoc Dat
Govardhan Chitrada

Teacher:
Professor Giusepper De Giacomo

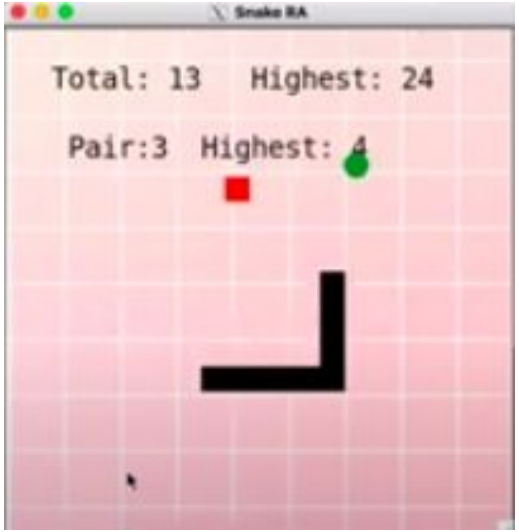
Contents

- Project container
- Learning Snake Agent
- Restraining bolts
- Reinforcement learning (DQN)
- Training and Results

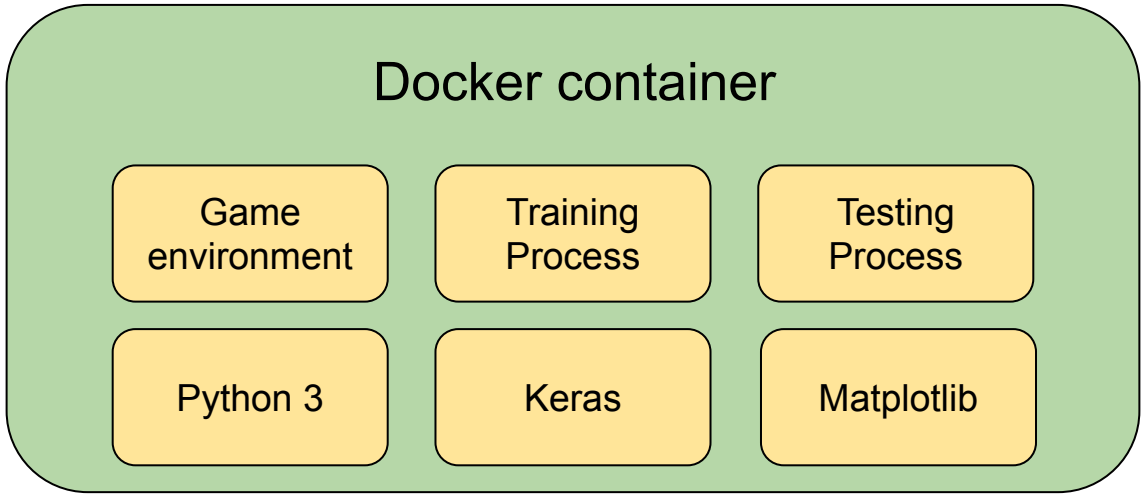
Introduction

- In the project, we using Restraining Bolts and DQN to play Snake game.

- Beside goal eating food of snake, we create new task for snake that is eating meat (red) interleave eating apple (green).

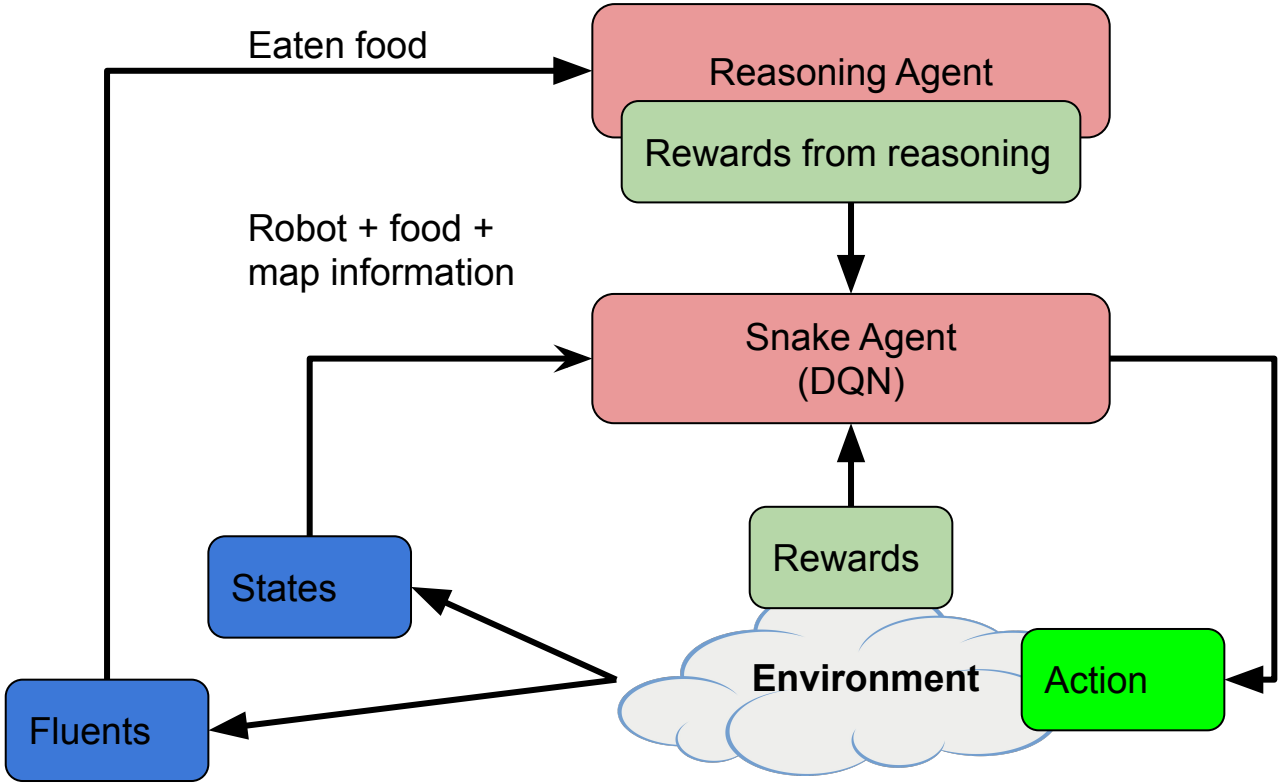


Project management



All code for training and testing including our models is available on:
https://github.com/DavidNguyen95/RA_project.git

Learning snake agent



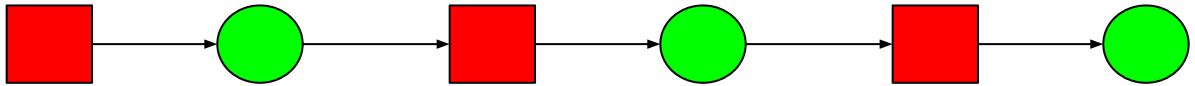
Learning snake agent

Learning Agent

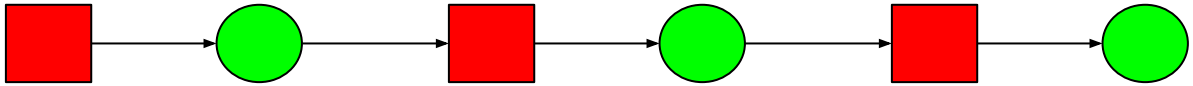
Features: Location of foods (meat and apple), and robot position information.
Actions: Move up,move down,move right, move left
Reward: Negative reward if robot die (hit wall or hit tail) and moving cost per step.
Positive reward when snake eat food

Restraining Bolts

Fluents: Type of food snake eat : "*no food" ; "apple" ; "meat"*
Reward: Eating meat interleave with eating apple

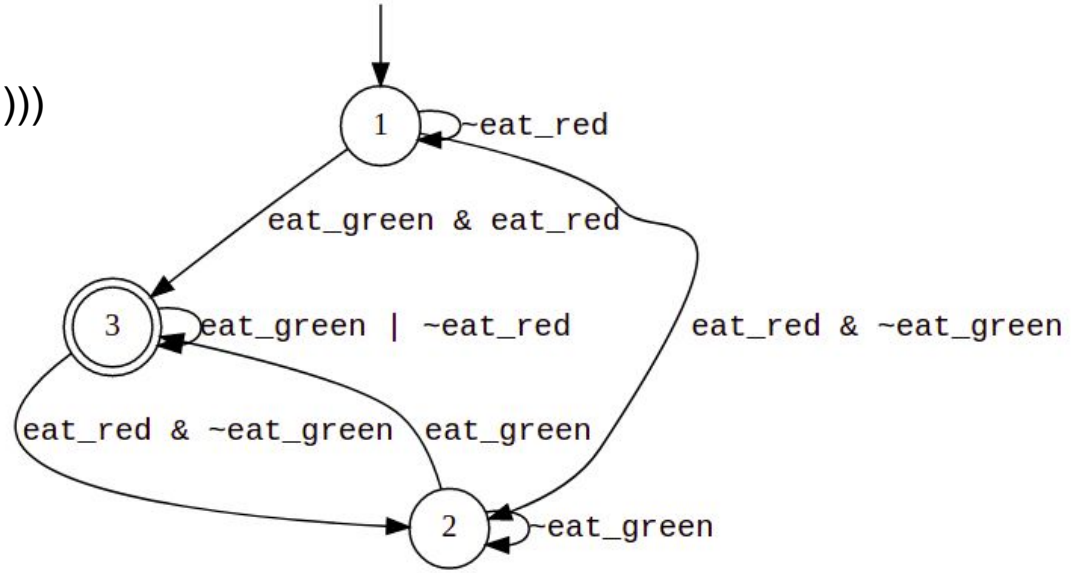


LTLf to DFA



F(eat_red) & G((eat_red -> F(eat_green)))

Using tools: MONA that integrate in whitemech/LTLf2DFA



Restraining Bolts

3 states: q1,q2,q3

q1: (False, False)

q2: (True, False)

q3: (True, True)

Where qi: (eat red,eat green)
For example: if snake eat red the first value is True

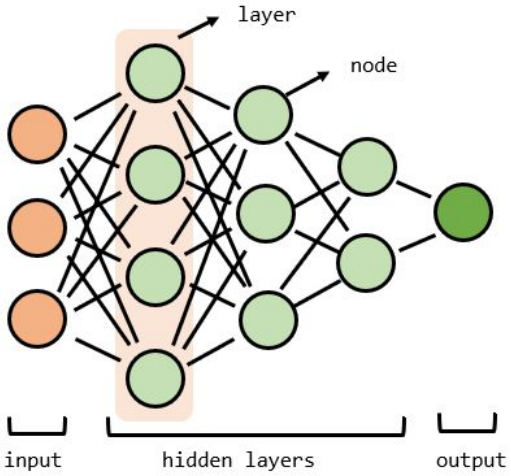
Rewards

q1 -> q2 : +10

q2 -> q3 : +60

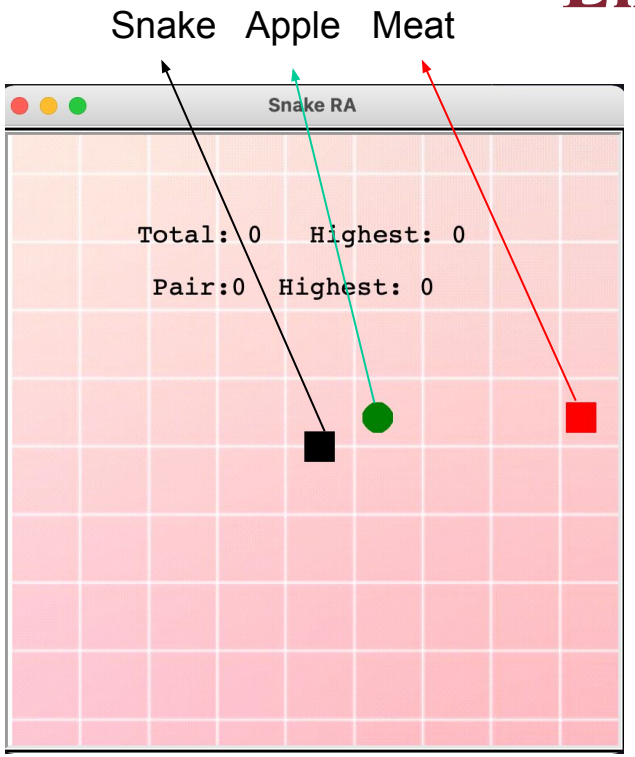
q3 -> q2 : +60

Deep Reinforcement Learning- DQN



Reinforcement learning can be sufficiently applicable to the environment where the all achievable states can be managed (iterated) and stored in standard computer RAM memory. However, the environment where the number of states overwhelms the capacity of contemporary computers (for Atari games there are 12833600 states) the standard Reinforcement Learning approach is not very applicable. Furthermore, in real environment, the Agent has to face with continuous states (not discrete), continuous variables and continuous control (action) problems.

Environment - Snake



This specific project the game ”Snake” is modified to fit certain requirements. The game is played on a grid of size 20x20. The snake is controlled by the agent. The agent is rewarded for eating food and for avoiding the walls and itself. The agent is penalized for running into itself. To this we also added two different food types such as meat and apple. In general, to stay healthy in real life people should balance their diet with fruits and meat. In the same way here the snake has to eat maximum amount of food and in the same way it has to alternate the food (We call it as pairs)

Actions, rewards and states

Food above the snake	0/1
Food below the snake	0/1
Food on right side of the snake	0/1
Food on left side of the snake	0/1
Wall above the snake	0/1
Wall on the right	0/1
Wall below the snake	0/1
Wall on the left	0/1
Snake direction is up	0/1
Snake direction is down	0/1
Snake direction is left	0/1
Snake direction is right	0/1

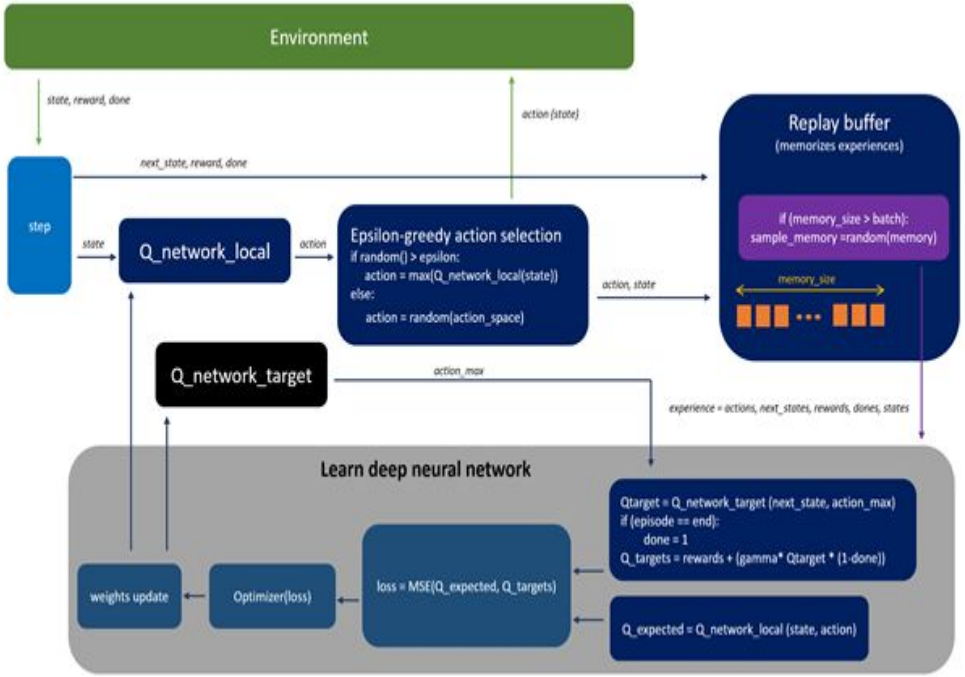
States of the Environment

Actions of the snake

For eating an apple	10
For coming closer to apple	1
For going away from apple	-1
For hitting the wall or itself	-100

Rewards for the agent

DQN Algorithm



1. *Initialize the Q* — network with random weights.
2. Initialize *replay buffer*
3. *Pre-process* the environment and store the initial state in the replay buffer.
4. Initialize the target Q — network with *random* weights.
5. For each episode:

(a) Choose an *action* at *random*.
(b) Take action and observe the *reward* and the next state.
(c) Store the *transition* in the replay buffer.
(d) *Sample* a random minibatch from the replay buffer.
(e) Update the *weights*.

DQN AGENT

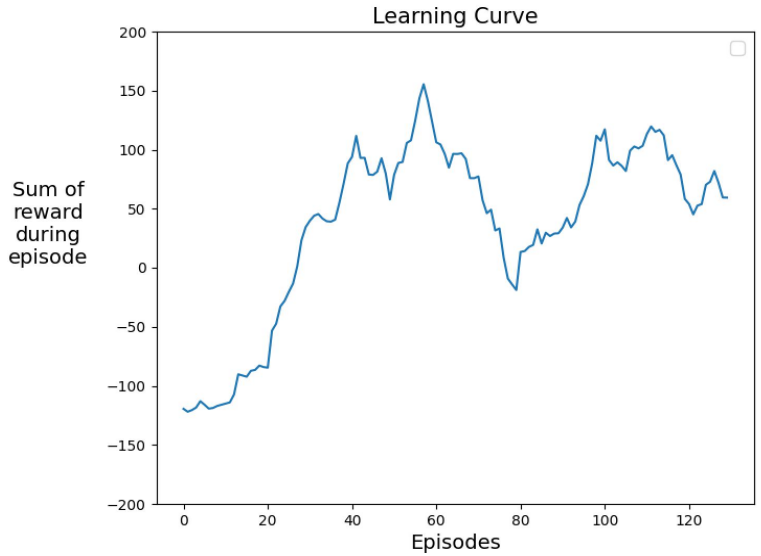
The methods reset(self), step(self,action) and getstate(self).It is also necessary to calculate the reward every time the agent takes a step.The agent learns to play snake (with experience replay) and to avoid the obstacles. The following four state spaces are used:

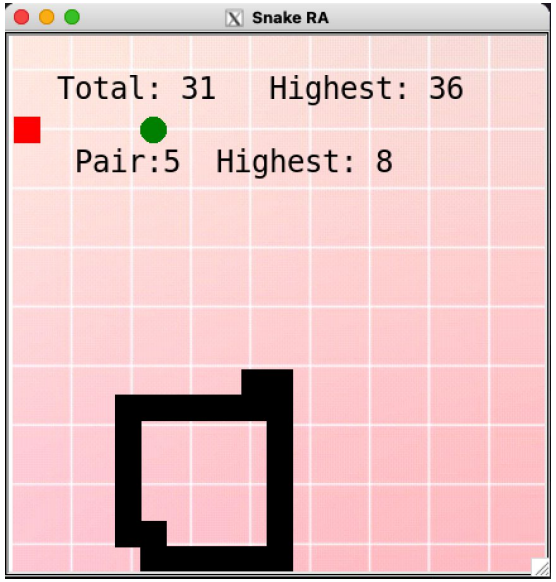
- *Observation* - the state of the environment.
- *Action* - the action taken by the agent.
- *Reward* - the reward received by the agent.
- *Next observation* - the next state of the environment

Param name	optimized values
epsilon	1
epsilon_min	0.01
gamma	0.95
batch_size	500
learning_rate	0.00025
layer_sizes	[128,128,128]

Results

Training the agent with the reasoning agent for 130 episodes with 10000 steps per episode. Approximately at 60 episodes the agent achieved a maximum reward and the learning curve shows that the agent is learning rapidly in first fifty episodes





Conclusions and Future works

The main motive of this project is to prove that complex conditions as restraining bolts can be as used as effective module for fast implementation into the already present environment without redoing the whole network.

- Using Restraining Bolts can help snake agent eating meat interleave eating apple
- The convergence of DQN need more episodes when using reasoning agent.
- The simulation environment is light computation so author does not need GPU
- In future work, we can put some obstacle in environment. Or extend more complex task for snake.

Reference

Ronen I. Brafman, Giuseppe De Giacomo, and Fabio Patrizi. ”LTLf/LDLf Non-Markovian Rewards“. In: AAAI. 2018.

Alberto Camacho et al. ”LTL and Beyond: Formal Languages for Reward Function Specification in Reinforcement Learning“. In: Aug. 2019, pp. 6065–6073. DOI: 10.24963/ijcai.2019/840.

Giuseppe De Giacomo et al. Reinforcement Learning for LTLf/LDLf Goals. July 2018.

Giuseppe De Giacomo et al. ”Foundations for Restraining Bolts: Reinforcement Learning with LTLf/LDLf Restraining Specifications“. In: ICAPS. 2019

THANK YOU