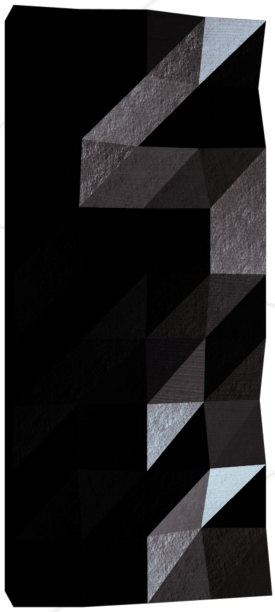


Learn2Slither

Reinforcement Learning

Summary: A snake that learns how to behave in an environment through trial and error.

Version: 1.00

Contents

Chapter I

Foreword

In the mysterious folds of space-time, where stars hum absurd melodies and planets organize unexpected meetings, there exists a peculiar corner of the universe known as "The Snaky Galaxy." This interstellar sector is renowned for its extraterrestrial serpents, debating with profound philosophical nuances while gracefully coiling around drifting asteroids in the void.

The serpents of the Snaky Galaxy, whose iridescent scales are imbued with existential questions, gather in cosmic spirals to discuss the grand mysteries of life, the universe, and everything in between two serpentine loops. Their hissing language is a strange symphony of serpentine thoughts, evoking twisted concepts and ideas that seem as elusive as an asteroid slipping through one's fingers.

Some claim that these space serpents are the guardians of cosmic secrets, while others whisper that they are the wise scholars of a parallel serpentine dimension. Regardless, the Snaky Galaxy remains a mysterious place where serpents, with their sinuous wisdom, continue to debate the great questions of life, winding their thoughts around each cosmic enigma with an elegance that could defy the very logic of the universe.

Douglas Adams, The Restaurant At The End Of The Universe

Chapter II

Introduction

This project is an artificial intelligence project about reinforcement learning.

Reinforcement Learning (RL) stands at the forefront of artificial intelligence, representing a paradigm where intelligent agents learn optimal decision-making strategies through interaction with their environment. Unlike traditional programming approaches, RL enables machines to learn by trial and error, continuously refining their behavior based on feedback in the form of rewards or punishments.

In this project, you will create a snake, moving on a board, and controlled by an intelligent agent. The board and the agent follow specific rules and constraints that are detailed below. We encourage you to carefully read the entire subject before starting any development: it would be unfortunate to create the board part without considering the rules that apply to the agent.

Obviously, the agent controlling the snake will implement reinforcement learning, to make choices. Each choice made by the agent will provide a positive or negative feedback from the board. The agent will then adapt to maximize positive feedback (or rewards) over time. This learning process is akin to how humans and animals learn from experience, making RL a powerful framework for addressing complex problems in various domains.

Chapter III

General rules

You can either use directly a computer in your favorite cluster, or a virtual machine.

- Regarding the virtual machine:
  - You are free to choose the operating system you want.
  - You will install on the virtual machine all the necessary software needed to realize your project.
- Regarding a computer on campus:
  - Make sure that all the needed tools are installed on the computer, or that you can install them locally on your account.
  - Make sure you have the space on your local session to install what you need for all the modules (use the goinfre if your campus has it).
  - You must have everything installed before the evaluations.
- As usual, your program should not quit unexpectedly (depending on the coding language) apart from undefined behaviors. If this happens, your project will be considered as non functional and will receive a `0` during the evaluation.
- Submit your core work (the board, the agent, and your trained models) to your assigned git repository. Only the work in the git repository will be graded: the "git clone" of the repository will be done in a new empty directory, where the appropriate runtime environment is available (in the virtual machine, in a venv for python, etc...).
- You are free to use any language of your choice for this project but we advise you to do it in **python** because there are many libraries that can help you, if you do this project in **python**, it must respect the norm.
  - pip install flake8
  - alias norminette_python=flake8
- Enjoy your snake training!

Chapter IV

Mandatory part

IV.1 Part 1: Environment/Board

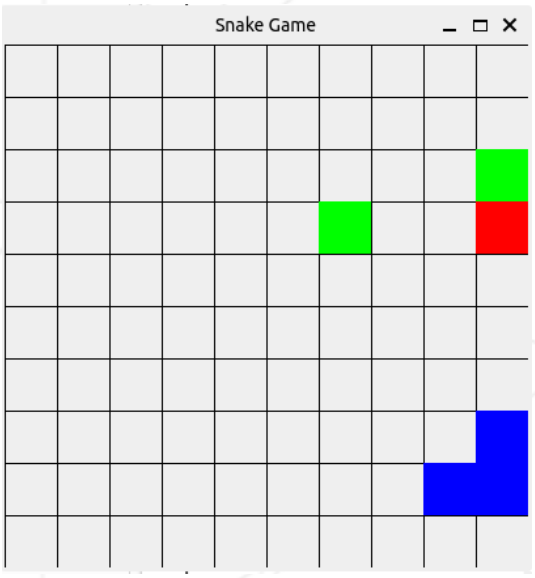
The environment of the snake is represented by the board. The following rules apply to the board:

- Board size: 10 cells by 10 cells.
- Two green apples, in a random cell of the board.
- One red apple, in a random cell of the board.
- The snake starts with a length of 3 cells, also placed randomly and contiguously on the board.
- If the snake hits a wall: Game over, this training session ends.
- If the snake collides with its own tail: Game over, this training session ends.
- The snake eats a green apple: snake's length increase by 1. A new green apple appears on the board.
- The snake eats a red apple: snake's length decrease by 1. A new red apple appears on the board.
- If the snake's length drops to 0: Game over, this training session ends.

Training sessions (or games, or rounds): An untrained agent will quickly fail. A unique training session can't be enough to learn: the agent will face hundreds or thousands of training sessions to increase its skills. The main program will offer a command line parameter to define how many training sessions should be executed.

To ensure the proper functioning of the project, you must provide a graphical interface displaying the board and its items through time (each choice of the agent updates the board). The display speed can be configured and at least one human-readable speed exists. Also, a step-by-step mode must be available.

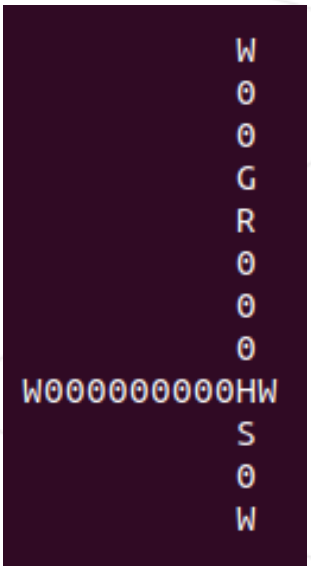
The display of the board could look like this:



- *Green: green apple*
- *Red: red apple*
- *Blue: snake*

IV.2 Part 2: State

Snake vision:
Your snake can only see in the 4 directions from its head. The following figure represents the terminal output that your program will compute, before asking the agent where the snake should move to. This output is matching the board represention from the previous figure.



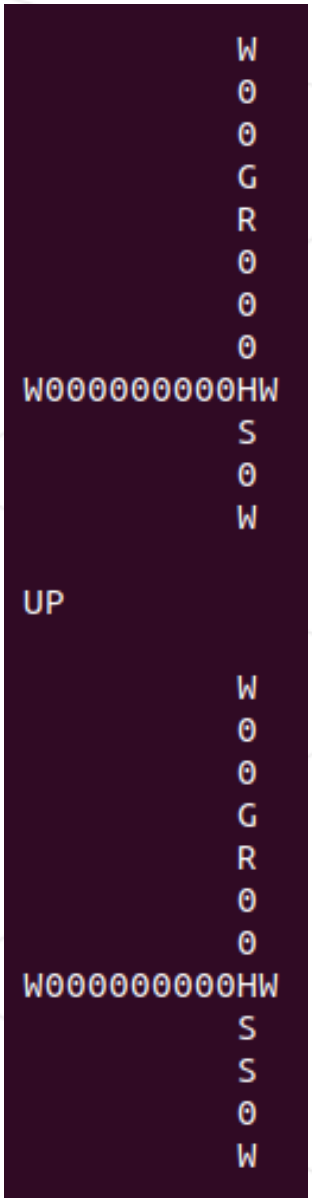
In this figure, each letter has of course a specific meaning:

- W = Wall
- H = Snake Head
- S = Snake body segment
- G = Green apple
- R = Red apple
- 0 = Empty space

You can only provide to the agent the information visible to your snake. Providing more information will result in a penalty of -42.

IV.3 Part 3: Action

Your agent can only perform 4 actions **(UP, LEFT, DOWN, RIGHT)**. It must take decisions solely based on the snake vision (State), from the head and to the 4 directions. No other information from the board should be used by the agent.



The board or environment is displayed graphically in a dedicated window. The state or vision and the action taken by the agent are displayed in your terminal, according to the previous figure.

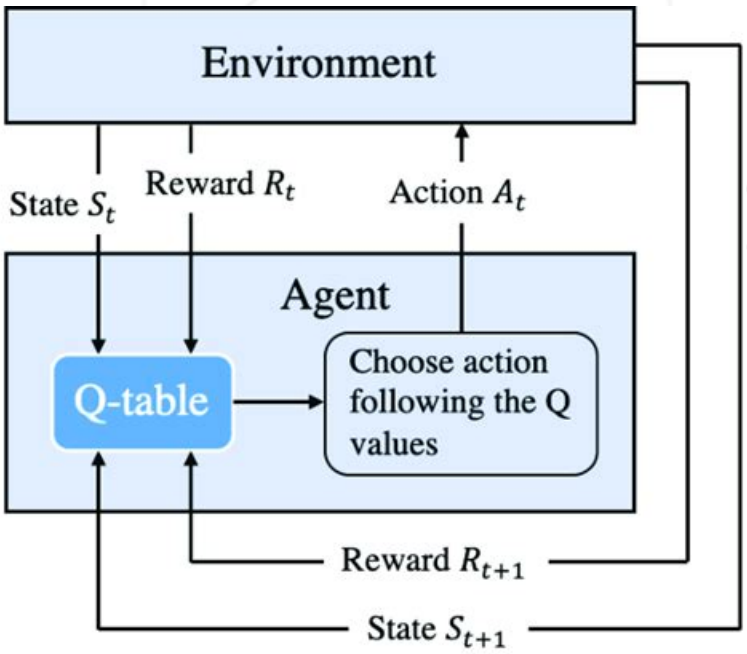
IV.4 Part 4: Rewards

The goal of your snake is to reach at least a length of 10 cells, and stay alive as long as possible. The action chosen by the agent will evolve through time in order to reach this goal, based on rewards granted by the environment (or board) after each action.
It's up to you to define the positives and negative rewards. An obvious approach could be:

- If the snake eats a red apple: a negative reward.
- If the snake eats a green apple: a positive reward.
- If the snake eats nothing: a smaller negative reward.
- If the snake is Game over (hit a wall, hit itself, null length): a bigger negative reward.

The reward for each action will increase or decrease the chance for the agent to make the same choice in the future when facing an identical situation.

IV.5 Part 5: Q-learning



You must implement a model that uses a Q function to evaluate the quality of an action in a specific state. This Q function can be implemented using Q-values in a Q-table or a Neural Network.

`No other model is allowed; otherwise, a score of 0 will be given.`

Updating the Q function: The Q-learning algorithm is designed to adjust the Q function (and the associated Q-values) based on the reward received after each action taken. You can train mutiple models using different update approaches for the Q function.

Exploration vs Exploitation: To discover new actions that might be beneficial, the agent sometimes needs to take random actions instead of always choosing the ones that seem to be the best.

Iterative learning: The agent repeats this process of interacting with the environment, taking actions, receiving rewards, and updating the Q function iteratively. When the training session is over, a new training session is started to continue the reinforcement learning.

Export and import models: At any time, it is possible to export a unique file that contains everything describing the current learning state of the agent. This file will mostly contains the Q-values. The same file can be imported by the agent to restore its learning

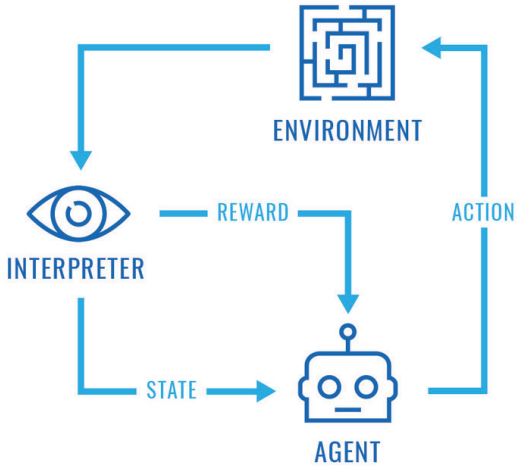
state to a specific level. Each file is a “model” of your AI. You will have to turn in several models in your git repository (see chapter V).

Exploitation without learning: A configuration switch of your agent will prevent the learning process from happening. The agent will just ignore reward from the board and the Q function will not be updated. This option is usefull to evaluate the success rate of a specific model without altering it.

During the training part of your agent, it is possible to remove the graphical display and the terminal outputs in order to speed up the process.

IV.6 Part 6: Technical structure

Please ensure that your program is modular as shown in the diagram below to easily evaluate each part of your project. It is up to you to find the most effective way for each module to communicate with each other.



Chapter V

Turn-in and Peer-Evaluation

Turn in your work (board, agent and models) using your `Git` repository, as usual. Only the work inside your repository will be evaluated during the defense.

You must have a "models" folder with your trained models inside, saved in a file format of your choice (e.g., .txt). Each model can be more or less trained, over few or many training sessions, and with alternate update strategies for the Q function.

You must have performed your training in advance and generated mutiple models; be cautious as it may take a lot of time.

Possible command line and display:

```
$> ./snake -sessions 10 -save models/10sess.txt -visual off
...
Game over, max length = 4, max duration = 17
Save learning state in models/10sess.txt
```

You must have at least 3 saved model files, trained respectively with 1, 10, and 100 training sessions. This should show how much your snake "learns" over multiple training sessions. You must be able to start a new session using one of your saved models, in conjuction with the “non-learning” feature, to verify the performance of each model. This will be tested during the defence.

Possible command line and display:

```
$> ./snake -visual on -load models/100sess.txt -sessions 10 -dontlearn -step-by-step
Load trained model from models/100sess.txt
...
Game over, max length = 7, max duration = 32
```

Your objective is to have a snake with a length of 10 or more by the end of the session, and an important lifetime. Therefore, you will likely need a lot of training sessions.

Possible command line and display:

```
$> ./snake -visual on -load models/1000sess.txt
...
Gave over, max length = 12, max duration = 55
```

Chapter VI

Bonus Part

Bonuses will only be considered if all mandatory parts are correct.

List of bonuses that may be accepted:

- Achieving a higher length at the end of a session (15, 20, 25, 30, 35).
- Creating a visually stunning display, with a lobby, a configuration panel, results and statistics, ...
- Allowing the board size to be changed with arguments (your snake should be able to 'play' with the same trained models regardless of the board size, in order to validate this bonus).