Artificial Intelligence

Final Assignment: Reinforcement Learning
Assigned: 7 June
Due: Tuesday, 21 June, 19h30m

Prof. Felipe Meneguzzi
Leonardo Rosa Amado (assistant)

November 1, 2016

1 Probabilistic Hyrule's Maze

You must work on this project in groups of at most *two students*. You are free to discuss high-level design issues with the people in your class, but every aspect of your actual implementation must be entirely your own work. Furthermore, there can be no textual similarities in the reports generated by each student. Plagiarism, no matter the degree, will result in forfeiture of the entire grade of this assignment.

The Legend of Zelda is a series of games originally designed by Shigeru Miyamoto[1], Takashi Tezuka and Eiji Aonuma whose first game was released in 1986. Games in the series are played on a fantasy world called Hyrule, which (up until the Super Nintendo) is often represented as a 2D grid map in which the character has to navigate, avoiding obstacles and reaching a goal[2]. In this assignment, we greatly simplify Link's movement by allowing only orthogonal movements within the grassland environment. However, unlike in the original assignment, Link's movement is now stochastic in such a way that whenever Link chooses a direction, he has 0.7 probability of arriving at the square in desired direction and 0.15 probability of arriving on a square at either side of the current square An example of this dynamics for the action UP is shown below in Figure 1.

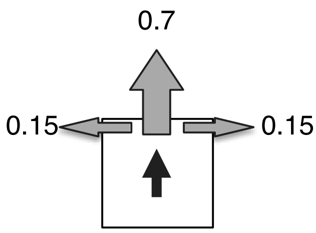

Figure 1: The possible result with probabilities for Link's UP movement

[1]Yes, the same guy from Mario.
[2]Following with a patriarchal view of society, our helpless princess Zelda, is usually captured by some villain, and our hero Link, must rescue her.

In this assignment, we help Link learn, via reinforcement learning, to navigate and maximise rewards within a map, aiming to reach the move between an initial state and the goal state, represented by a treasure chest. Thus, we may specify problems such as the one shown in Figure 2

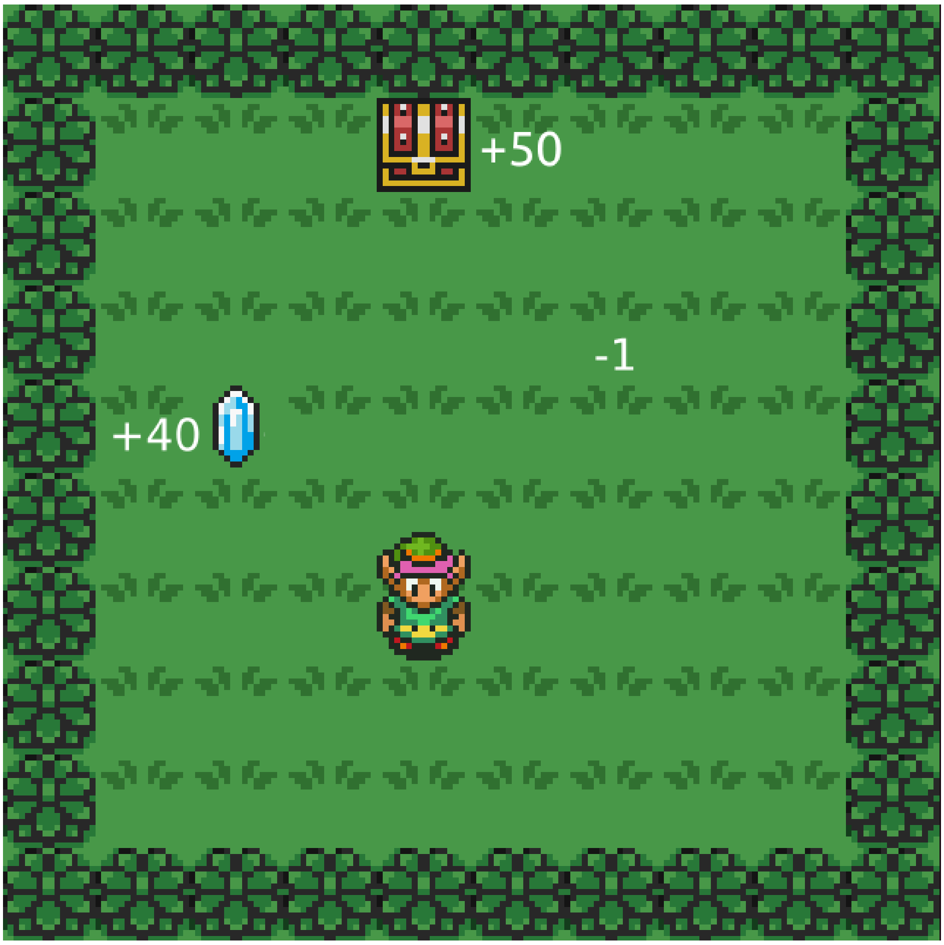

Figure 2: Example of an arbitrary map with rewards, +50 for the chest, +40 for the rupee and -1 for everything else.

2 Overview

For this assignment, you will be implementing **the Q-Learning algorithm** to compute policies for the navigation problem posed to Link. This implementation should be introduced in the code as:

1. The Q-Learning update rule ($Q[s, a] \leftarrow Q[s, a] + \alpha(r + \gamma \max_{a'} Q[s', a'] - Q[s, a])$).
2. Every method necessary for Q-Learning to function properly (highlighted with TODO in the given code).

3 Implementation and Deliverables

At the bare minimum, you are required to implement only the API in file **link.py**. The API for the game is almost exactly the same as our first assignment and we have omitted any specific API for your learning method to give you the maximum freedom to develop their own internal APIs (and minimise coincidental similarities in implementation), and that well developed code is a component of the final mark. Since this assignment will not be unit tested, you need not stick with any particular implementation structure for your update, but we recommend using only the Link class to include the learning method (or methods if you decide to implement more than one).

The key part of your implementation will be plugged into the Environment and Link classes, which are where the game trains the agent, generates new states and computes the immediate rewards.

At the end of this assignment, you will upload **one zip file** named `s<IDs>.zip`[3]. In this zip file, there must be one folder named `s<IDs>`(same as before). This folder must contain your implementation code (following Github's repository organization), where all Python files (.py) are in the folder's root.

- **Do not** modify the files for the game logic, your zip file should unzip into a directory exactly like the one you received.
- You are **not allowed** to use any external libraries besides the one supplied with the assignment package.
- one `report.pdf` file containing the report as instructed below;
- any unit tests you developed to test your helper classes.

Your deliverables must be handed in via Moodle using the appropriate upload room: Final Assignment Upload.

4 Grading

In order to properly evaluate your work and thought process, you will write a 2-page report in the AAAI conference format explaining your implementation and experiments. These guidelines are to be followed **exactly**. **Reports that are less than two pages of actual content, or not in format will receive 0 marks for the report criterion.** This report will be included in the deliverables of the assignment. The formatting instructions are available at the AAAI website[4]. The report must have the following sections:

- An introduction motivating your choice of reinforcement learning algorithm, outlining the remainder of the paper;
- One section describing in further detail the algorithm and how you implemented it;

[3] Where `IDs` are the registry IDs (without check digits) separated by a hyphen, e.g. if the assignment was made by 03453235-3 and 23454314-4, then you must create `s03453235-23454314.zip`

[4] `http://www.aaai.org/Publications/Templates/AuthorKit.zip`

- One experimentation section where the performance of your learning algorithm is measured using each of the maps included in the maps directory (and any other you decide to include).
- One conclusion section, where you will summarise your experience in programming the Q-Learning algorithm and discuss its performance, and any limitations encountered in learning the policy for the given maps.

Grading will take consider elements of your programming, experimentation and reporting of the work done. The criteria, as well as their weight in the final grade is as follows:

- Quality of the implementation (30%) — correctness of the algorithm implementation, coding style and organization;
- Efficiency of the learning algorithm (20%) — your implementation's ability to generate a policy that leads to the end state with maximum efficiency overall after a certain number of learning episodes;
- Overall report readability (20%) — how accessible and coherent your explanation of your algorithm is and how coherent your experiments are explained;
- Experiments (30%) — how coherent the proposed experiments are in measuring the performance of your learning algorithm;
- Feature learning Q-Learning (20% bonus) — if you have finished a working on the Q-Learning implementation, implement the Feature Learning Q-Learning to generalize the domain model.

5 Code-Specific Advice

- You can change Link's movement speed modifying the `MOVE_SPEED` constant in *common.py* file;
- All classes and methods which you must complete yourself are marked with the `TODO` keyword (search for that using your favorite editor);

6 Miscellaneous Advice

Here are some lessons we learned in creating our own solution and writing papers/reports:

- It took the instructor approximately 3 hours to code and debug the solution for this problem from scratch based on the pseudocode at AIMA, plan your time accordingly;
- In your report, key indicators of performance are: the overall reward after a number of runs, and number of learning episodes before convergence;
- Notice that a single run of any calculated policy is not guaranteed to have a specific reward, given the stochastic nature of the domain.