University of Cape Town
Department of Computer Science

CSC3022F
ML Assignment 2 - Reinforcement Learning

May, 2023

In this Assignment, you will teach RL Agents to pickup packages on a grid-world. The environment you will be using is called the Four-Rooms domain (See Figure 1), an environment commonly used to test the efficacy of RL Algorithms. You must implement Q-learning as your RL aglorithm with the goal of teaching your agent to pickup the package(s) in three scenarios of increasing complexity. General information about this Assignment and the Scenarios can be found below.

When implementing your solutions, take note of the following:

1. The environment is a 2D grid-world of size *13 x 13* but rows 0 and 12 and columns 0 and 12 are boundaries (not traversable) so it is technically a grid-world of size *11 x 11*. The environment has one start state that is randomly assigned when the environment is created and a terminal state that is determined when the number of packages left to collect reaches 0.

2. You have been given two scripts to start with: *FourRooms.py* and *ExecutionSkeleton.py*. **You are not allowed to modify *FourRooms.py*** but, you can, and will, copy and modify *ExecutionSkeleton.py* for each of the scenarios you tackle. Documentation for *FourRooms.py* can be found in Appendix A.

3. Your agents will only have a partial representation of the environment which consists of their location (x, y) and the number of packages left k. They are not allowed to know the location of the package(s), hallways (cells that connect two rooms together) or the boundaries of the environment. They must figure that out themselves.

4. Your agent has four actions: {*UP, DOWN, LEFT, RIGHT*}. Depending on the scenario flags used (read Section 4), these actions may be deterministic or stochastic. If the actions are stochastic, every action the agent takes may result in a unexpected state to state transition occurring (this is handled by *FourRooms.py*).

5. You will need to design a suitable reward function for each scenario such that your agents produce the expected behaviour for each scenario. You will also need to experiment with different learning rates and discount factors and see which values work best for your Agents.

6. The goal of your Agent is to derive an optimal policy such that it collects all of the packages in the environment in accordance with each scenario description.

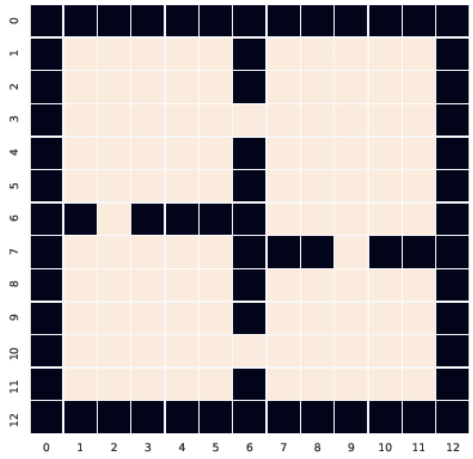

Figure 1: A visual representation of the Four-Rooms domain courtesy of Goyal et al. (2019).

1 Scenario 1: Simple Package Collection

In this first scenario, your Agent must collect 1 package located somewhere in the environment. Your implementation should be done in a file called *Scenario1.py* and should be invoked as follows:

```
python Scenario1.py
```

For this scenario, you should specify *scenario = 'simple'* when creating your *FourRooms* object and call *fourRoomsObj.newEpoch()* whenever your agent starts a new training epoch. When learning is complete, you should call *fourRoomsObj.showPath(-1)* to show the final path your agent took to get from its start state to the terminal state. *showPath* will display an image on screen so you will need to set the *savefig* optional parameter to save the image to disk if you are on nightmare.

Completing this part of the assigned work will earn you up to 30% for the Assignment.

2 Scenario 2: Multiple Package Collection

In this second scenario, your Agent must collect 3 packages located somewhere in the environment. Your implementation should be done in a file called *Scenario2.py* and should be invoked as follows:

```
python Scenario2.py
```

For this scenario, you should specify *scenario = 'multi'* when creating your *FourRooms* object and call *fourRoomsObj.newEpoch()* whenever your agent starts a new training epoch.

When learning is complete, you should call *fourRoomsObj.showPath(-1)* to show the final path your agent took to get from its start state to the terminal state. *showPath* will display an image on screen so you will need to set the *savefig* optional parameter to save the image to disk if you are on nightmare.

Completing this part of the assigned work will earn you up to 30% for the Assignment.

3 Scenario 3: Ordered Multiple Package Collection

In this third scenario, your Agent must collect 3 packages located somewhere in the environment. These packages are marked as red (R), green (G) and blue (B) and your agent must collect them in that order. Collecting a package in the wrong order will terminate a simulation epoch early. Your implementation should be done in a file called *Scenario3.py* and should be invoked as follows:

```
python Scenario3.py
```

For this scenario, you should specify *scenario = 'rgb'* when creating your *FourRooms* object and call *fourRoomsObj.newEpoch()* whenever your agent starts a new training epoch. When learning is complete, you should call *fourRoomsObj.showPath(-1)* to show the final path your agent took to get from its start state to the terminal state. *showPath* will display an image on screen so you will need to set the *savefig* optional parameter to save the image to disk if you are on nightmare.

Completing this part of the assigned work will earn you up to 30% for the Assignment.

4 Scenario 4: Stochastic Actions

For each of your scenarios, add a *'–stochastic'* optional flag that a user may specify on program execution. This will make the actions space stochastic such that when an Agent takes an action, there is a 20% chance that the agent may instead move to a different grid cell than anticipated.

You can make each scenario use a stochastic action space by setting *stochastic = True* when creating your *FourRooms* object.

Completing this part of the assigned work will earn you up to 10% for the Assignment (weighted equally across all scenarios).

Note: You will need to ensure that both *numpy* and *matplotib* are included in your *requirements.txt* file. *FourRooms.py* will not work without it.

You can use the *sys* module the get arguments passed to your program and you can use a package such as *argparse* to parse the arguments passed to your program (although you may find this step unnecessary given the simplicity of the arguments themselves).

In a compressed archive named as your student number (e.g. GWRBRA001.zip/tar.gz/tar.xz), place your Python scripts, makefile, git repo, readme, and any other applicable files. Upload that archive to the assignments tab on **Vula before 10.00 AM, 16 May, 2022**

Please Note the following for ALL ML Assignments:

1. A working Makefile must be submitted. If the tutor cannot build a python virtual environment on nightmare.cs by typing make, you will only receive **50%** of your final mark.

2. You must use version control from the get-go. This means that there must be a .git folder alongside the code in your project folder. A **10%** penalty will apply should you fail to include a local repository in your submission.

   With regards to git usage, please note the following:

   -10% - usage of git is absent. This refers to both the absence of a git repo and undeniable evidence that the student used git as a last minute attempt to avoid being penalized.

   -5% - Commit messages are meaningless or lack descriptive clarity. eg: "First", "Second", "Histogram" and "fixed bug" are examples of bad commit messages. A student who is found to have violated this requirement for numerous commits will receive this penalty.

   -5% - frequency of commits. Git practices advocate for frequent commits that are small in scope. Students should ideally be committing their work after a single feature has been added, removed or modified. Tutors will look at the contents of each commit to determine whether this penalty is applicable. A student who commits seemingly unrelated work in large batches on two or more occasions will receive this penalty.

   Please note that all of the git related penalties are cumulative and are capped at -10% (ie: You may not receive more than -10% for git related penalties). The assignment brief has been updated to reflect this new information.

   We cannot provide a definitive number of commits that determine whether or not your git usage is appropriate. It is entirely solution dependent and needs to be accessed on an individual level. All we are looking for is that a student has actually taken the time to think about what actually constitutes a feature in the context of their solution and applied git best practices accordingly.

3. You must provide a README file explaining what each file submitted does and how it fits into the program as a whole. The README file should **not** explain any theory that you have used. The README is used by the tutors if they encounter any problems.

4. Do **not** hand in any binary files. Do **not** add binaries (.o files and your executable) to your local repository.

5. Please ensure that your tarball works and is not corrupt (you can check this by trying to downloading your submission and extracting the contents of your tarball - make this a habit!). Corrupt or non-working tarballs will not be marked - **no exceptions.**

6. A 10% penalty per day will be incurred for all late submissions. No hand-ins will be accepted if later than 3 days.

7. **DO NOT COPY. All code submitted must be your own. *Copying is punishable by 0 and can cause a blotch on your academic record.* Scripts will be used to check that code submitted is unique.**

8. You are **NOT** allowed to simply download python packages to solve your coding problems. Unless specifically stated in the Assignment brief, we expect all of your code to be your own.

References

Anirudh Goyal, Shagun Sodhani, Jonathan Binas, Xue Bin Peng, Sergey Levine, and Yoshua Bengio. 2019. Reinforcement learning with competitive ensembles of information-constrained primitives. *arXiv preprint arXiv:1906.10667* (2019).

A *FourRooms.py* Documentation

A.1 Initialization:

```
FourRooms.__init__(scenario : str, stochastic : bool = False) -> FourRooms
```

Creates and initializes a FourRooms object. You just invoke it by creating the object *FourRooms()*. The *scenario* property must be one of the following values: 'simple', 'multi' or 'rgb'. You can set *stochastic=True* if you want your agent to have to traverse in a non-deterministic environment.

A.2 GetPosition:

```
FourRooms.getPosition() -> (int, int)
```

This function will return the location of the agent in the grid-world. The location is returned as a tuple (x, y).

A.3 GetPackagesRemaining:

```
FourRooms.getPackagesRemaining() -> int
```

This function will return the number of packages left to collect. The value returned is an int.

A.4 TakeAction:

```
FourRooms.takeAction(action : int) -> (int, (int, int), int, bool)
```

This function simulates an agent's action in the environment. It returns a 4-tuple consisting of the type of grid cell the agent occupies, the agents new location (which is a 2-tuple (x, y)), how many packages are left to collect and whether the agent has reached a terminal state (which is a boolean).

The functions takes in an int that corresponds to the action constants described in Section A.8. The grid cell type also returns an int that corresponds to the cell type constants described in Section A.8. That means the function can be invoked as follows:

```
cellType, newPos, packagesLeft, isTerminal = fourRoomsObj.takeAction(FourRooms.UP)
```

For scenarios 1 and 2, you don't care about the colour of the grid cell, so any value above zero represents a package and zero represents an empty grid cell.

A.5 IsTerminal:

```
FourRooms.isTerminal() -> bool
```

This function will return true if the simulation run is over. The value returned is a boolean. In scenarios 1 and 2, *isTerminal()* will only return *True* when all of the packages are collected. In scenario 3, *isTerminal()* will also return *True* if a package is picked up in the wrong order.

A.6 NewEpoch:

```
FourRooms.newEpoch() -> None
```

This function will reset the environment to its starting conditions. It should be called when a training epoch is finished.

A.7 ShowPath:

```
FourRooms.showPath(index : int, savefig : str = None) -> None
```

This function will display the path of the agent took for the *index'th* epoch. Given that python lists can be negative indexed, *showPath(-1)* will show the agent's path for the last recorded epoch.

By default, this function displays the agent path using the matplotlib GUI. If you want to save the image to disk, you can set *savefig* to the string path specifying the name and destination you want the png to be written to: *path/to/image/image.png*

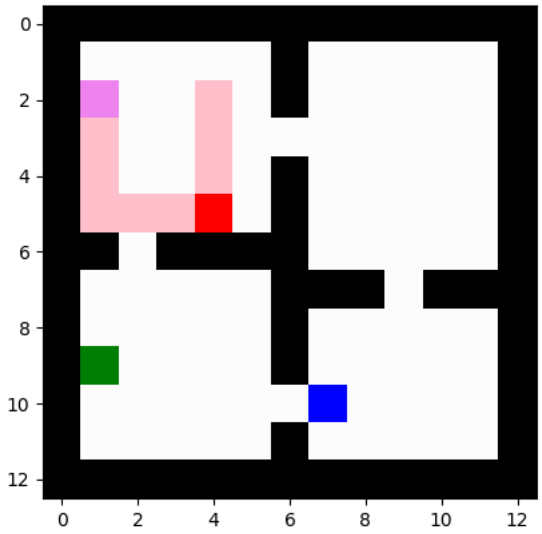

Figure 2: An example path generated using *FourRooms.py*. The Red, Blue and Green Cells are the Red, Blue and Green packages respectively. The Violet Cell is the start location of the agent and the Pink cells are the path the agent took through the environment.

A.8 Constants:

```
# Action Constants
FourRooms.UP = 0
FourRooms.DOWN = 1
FourRooms.LEFT = 2
FourRooms.RIGHT = 3

# Grid Cell Type Constants
FourRooms.EMPTY = 0
FourRooms.RED = 1
FourRooms.GREEN = 2
FourRooms.BLUE = 3
```