UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA
Faculty of Engineering, Built Environment and Information Technology

EAI 320

Intelligent Systems

Practical 2 Guide

14 February 2019

Concept: Prof. Warren P. du Plessis
Guide written by: Llewellyn Strydom

I. General Instructions

This section contains general instructions which are applicable to all reports.

- The prescribed LaTeX format should be used, and the generated portable document format (PDF) file should be submitted.
- The commented Python source code should also be submitted.
- Reports must be submitted via the EAI 320 ClickUP page. Do not email reports to the lecturer or Assistant Lecturer (AL) as emailed reports will be considered not to have been submitted.
- The names of submitted files must have the format provided below.

  `eai320_prac_{practical number}_{student number}.{extension}`
- No late assignments will be accepted. No excuses for late submission will be accepted.
- Each student must do their own work. Academic dishonesty is unacceptable and cases will be reported to the university Legal Office for suspension.
- The report must include the standard cover-page declaration for individual assignments provided in the General Study Guide of the Department of Electrical, Electronic and Computer Engineering. The declaration is not included in the LaTeX template automatically and needs to be explicitly uncommented to indicate agreement.
- All information from other sources must be clearly identified and referenced.
- Text and code may not be included as images, and reports that do so will be considered not to have been submitted. Axis labels, legends, and other information normally included in figures may form part of a figure image.
- Computer code should not be included in the report as it is submitted separately.
- The code provided to students should also not be included in submissions.
- Any attempt to interfere with the operation of the framework used (e.g. to modify the scores of agents) will be regarded as academic dishonestly.

A. *Submission*

Reports should only be submitted via the

- `Practicals` → `Practical 2` → `Practical 2 Report`
- `Practicals` → `Practical 2` → `Practical 2 Code`

links on the EAI 320 ClickUP page.

Only assignments submitted in via ClickUP will be accepted. Do not email reports to the lecturer or AL as emailed reports will be ignored.

Late reports will not be accepted! Accepting late reports is extremely unfair on those students who submit their work timeously because their tardy colleagues are effectively given additional time to complete the same work. Students are advised to submit the day before the deadline to avoid inevitable problems with ClickUP, internet connections, unsynchronised clocks, load shedding, hard-drive failure, computer theft, etc.. Students who choose to submit close to the deadline accept the risk associated with their actions, and no excuses for late submissions will be accepted.

Students will be allowed to submit updated copies of their reports until the deadline, so there will be no excuse for submitting late. Rather be marked on an incomplete early version of your report than fail to submit anything.

B. Academic Dishonesty

Academic dishonesty is completely unacceptable. Students should thus familiarise themselves with the University of Pretoria's rules on academic dishonesty summarised in the study guide and the university's rules. Students found guilty of academic dishonesty will be reported to the Legal Office of the University of Pretoria for suspension.

Students are required to include the standard cover page for individual assignments provided in the General Study Guide of the Department of Electrical, Electronic and Computer Engineering as part of their reports. This standard cover page includes a statement that the student submitting the report is aware of the fact that academic dishonesty is unacceptable and a statement that the submitted work is the work of that student. Failure to include this cover page will mean that the report will be considered not to have been submitted.

While students are encouraged to work together to better understand the work, each student is required to independently write their own code and report. No part of any student's work may be the same as any part of another student's work.

Students should clearly indicate material from other sources and provide complete references to those sources. Examples of commonly-used sources include the textbook [1] and this document [2]. Note that this does not mean that students may reuse code and/or information found in books, on the internet, or in other sources as students are required to complete the tasks themselves.[1]

II. Scenario

This assignment will task students with solving a problem in a partially observable environment in order to beat an opponent in a game of rock-paper-scissors (RPS). The same framework that was used for the previous assignment will be used again [3].

III. Instructions

Students have to write an RPS client that is able to exploit an adversary that always plays from a known sequence, though the agent does not know where in the sequence the adversary will start playing. This problem can be solved in the same way that other search problems with observations are solved using belief states. More information can be found in Section 4.4, and more specifically Section 4.4.2, of the prescribed textbook [1].

The two major goals for this practical are thus

1) implementing a search algorithm that works in a partially observable environment, in order to beat `sequence_random.py`, which is provided in Listing 2 in Appendix B on page 5, and
2) writing a short technical report describing the findings.

A. Task 1

An agent has to be implemented that can beat `sequence_random.py`. This should be achieved by updating the agent's belief states as more information about the adversary becomes available. The agent will be able to determine the sequence that `sequence_random.py` will be playing from by reading the sequence from a file at the start of every match. Once the position in the sequence that the adversary is playing from is known, the agent should be able to win all ensuing rounds.

The code snippet in Listing 1 will allow one to read the sequence at the start of each match.

[1]The objective of all academic assignments is fundamentally that students learn by completing the assignments. Merely reusing code and/or information found elsewhere defeats this objective because a key part of the learning process is performing the tasks oneself.

Listing 1: A code snippet for reading a sequence from `sequence.pkl`.

```
import cPickle

if input == "":
        sequence = cPickle.load(open('sequence.pkl'))
```

1) Suggestion: The first agent, from `rpsrunner.py`'s point of view, should always be `sequence_random.py` to ensure that the file is updated before it needs to be read. The command

`rpsrunner.py -t 1 -r 50 sequence_random.py <your_filename>.py`

will ensure the correct parameters for a new game. Only one thread should be used to ensure that the file containing the preset sequence is not overwritten as would occur when additional threads are instantiated. The nature of this assignment means that 50 rounds will be enough to ensure statistical significance.

B. Task 2

The task is to examine the amount of time the agent from Task 1 takes to determine the exact position in the sequence for different sequence lengths. The results should be visualised using a graph.

1) Suggestion: Once the agent has found the exact position in the sequence, write the number of rounds that have been played to a file, together with the length of the sequence the adversary is playing from. Another Python script can then be called at a later stage to read and plot this data.

IV. Submission Requirements

Marks will primarily be awarded on the basis of the motivations provided. Merely obtaining the correct results is not nearly as important as describing how the results were obtained.

A. Report Content

Each student is required to submit a report providing the information listed below. Failure to comply with these instructions will lead to marks being deduced.

- Provide the results obtained.
- Provide a discussion of the results. Make sure to discuss the relationship between the sequence length and the number of rounds required to find the exact position of the adversary.
- Use the prescribed LaTeX format.
- The submitted computer code must **not** be included in the report, otherwise TurnItIn will flag a 100% similarity.
- All the material necessary to produce a formal document (e.g. an abstract, introduction, and conclusion). These parts of a document are not included in practical guides to shorten them.

B. Code Instructions

Each student is required to submit code conforming to the requirements listed below. A mark of zero will be awarded if the submitted code file does not run, produces errors, or does not follow the instructions.

- All code must be commented to a point where the implementation of the underlying algorithm can be determined.
- The submission must be a single file.
- Submitted code will be evaluated using the command
  `rpsrunner.py -t 1 -r 50 sequence_random.py <your_filename>.py`
  to ensure that the agent `breakable.py` is successfully beaten.
- All print statements and any other functions that were used for unit testing etc. must be removed before submission.
- All submissions must be written using Python 2.7 as a result of the fact that `rpsrunner.py` is written in Python 2.7.
- The code submitted must be the final implementation of Task 1.
- TurnItIn will not accept code with the file extension `.py`, so the file extension should be changed from `.py` to `.txt` before submission.

References

[1] S. J. Russell and P. Norvig, *Artificial intelligence: A modern approach*, 3rd ed. Prentice Hall, 2010.

[2] L. Strydom, *EAI 320 – Practical 2 Guide*, University of Pretoria, 13 Feb. 2019.

[3] B. Knoll. (2011, 6 Feb.) Rock paper scissors programming competition. [Online]. Available: http://www.rpscontest.com/

Appendix A
Abbreviations

AL	Assistant Lecturer
PDF	portable document format
RPS	rock-paper-scissors

Appendix B
Source Code for the Agent `sequence_random.py`

Listing 2: The source code for the agent `sequence_random.py`.

```
# An agent which plays from a random sequence of objects. The agent starts from a
# random position in the sequence each time it is restarted.
#
# Uses ideas from information in agents submitted to http://www.rpscontest.com
# Written by: W. P. du Plessis
# Adapted by: L. Strydom
# Last update: 2019-02-13

import numpy as np
import cPickle



if input == "":

    # Generate a random length from a log uniform distribution.
    lengths = np.logspace(1., 10., num=10, base=3.)
    sequence_length = int(np.random.choice(lengths))

    # Generate a random sequence.
    sequence = np.random.choice(['R', 'P', 'S'], sequence_length, replace=True)
    # Write the random sequence to a file
    cPickle.dump(sequence, open('sequence.pkl', 'w'))

    # Generate a random position in the sequence to start.
    position = np.random.randint(0, sequence_length)

else:

    # Increment the current position in the sequence without overflowing.
    position += 1
    if position >= sequence_length:
        position = 0


# Play the selected object.
output = sequence[position]
```