UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Faculty of Engineering, Built Environment and
Information Technology

EAI 320

Intelligent Systems

Practical 1 Guide

7 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.
- Do your 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.
- Clearly identify and reference all information from other sources.
- Do not include any computer code in appendices. The code provided to students should not be included.
- Text and code may not be included as images, and reports that do so will be considered not to have been submitted. Axis labels and information in figures may form part of a figure image.

A. Submission

Reports should only be submitted via the

- Practicals → Practical 1 → Practical 1 Report
- Practicals → Practical 1 → Practical 1 Code

links on the EAI 320 ClickUP page.

ClickUP will not accept code with the file extension .py, so the file extension should be changed .py to .txt before submission.

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

Rock-paper-scissors (RPS) is a popular game played between two adversaries, where each adversary simultaneously forms one of three shapes with an outstretched hand [3], or using some other representation. Throughout this course, the framework provided by [4] will be used to teach students how different artificial intelligence (AI) principles and algorithms can be applied to the game of RPS.

The RPS terminology that will be used in this course is shown below.

Object: One of three hand shapes (*viz.* rock (R), paper (P), or scissors (S)).

Move: One game where each opponent plays an object, resulting in a win, loss or draw for each player. The RPS framework used for this practical assignment [4] refers to a move as a round.

Match: A series of moves between two players to decide the overall winner of the game.

Sequence: A particular order of objects for one opponent (e.g. rock, rock, scissors).

This assignment will task students with using two common search strategies to find a hidden sequence that will allow it to exploit an opponent's weaknesses during a match of RPS.

III. Instructions

For this practical assignment, students will investigate the effectiveness of two uninformed search techniques, namely depth first search (DFS) and breadth first search (BFS). The algorithms will be used to explore different possible sequences in a game of RPS.

The four major goals for this practical are thus
1) implementing a search tree structure in Python,
2) implementing a DFS and BFS algorithm,
3) implementing a RPS agent that can beat `breakable.py`, which is provided in Listing 3 in Appendix B on page 6, and
4) writing a short technical report describing the findings.

---

[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: An example code segment showing the class for a tree.

```
1  class node(object):
2    def __init__(self,object,children = []):
3       self.object = object
4       self.children = children
```

Listing 2: Code showing how to create a tree using the class shown in Listing 1.

```
1  tree = node("Start", [
2   node("R",[
3       node("R"),
4       node("P"),
5       node("S")]),
6   node("P",[
7       node("R"),
8       node("P"),
9       node("S")]),
10  node("S",[
11      node("R"),
12      node("P"),
13      node("S")])
14                         ])
```
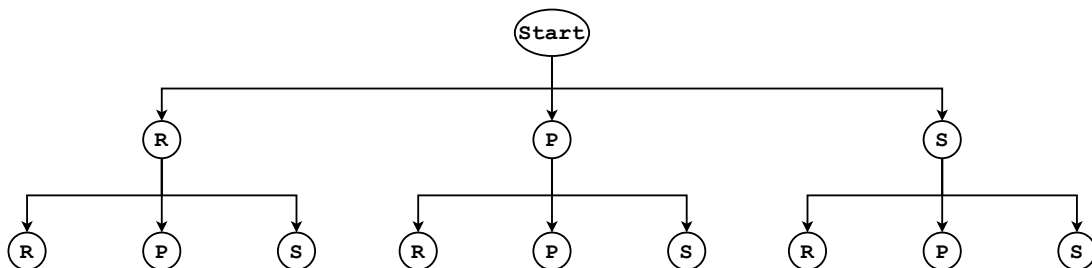

Fig. 1: The tree generated by the code in Listing 2.

A. Task 1

A search tree structure should be created in Python that can be used by any type of search algorithm. For this task the depth of the search tree must be dynamic and should only be limited by memory constraints.

Any process or method may be used to create the search tree. The recommended approach is to use a Python Class to represent nodes in a tree. Each of the nodes are linked and a recursive function can be used to build the tree.

An example code segment to create a tree class is provided in Listing 1, and a tree can be constructed using the code in Listing 2, resulting in the node tree shown in Figure 1.

1) Suggestion: A `depth` parameter may be passed to the constructor of the tree to allow the tree up to be constructed to the specified depth.

TABLE I: Sequences Generated by BFS Applied to the Tree in Figure 1.

Step	1	2	3	4	5	6	7	8	9	10	11	12
Sequence	R	P	S	RR	RP	RS	PR	PP	PS	SR	SP	SS

B. Task 2

The next task is to write BFS and DFS algorithms that can be used to incrementally search the tree built in Task 1. Each search algorithm must return a complete list of the sequences represented by each node, in the order in which they are visited by the algorithm. For example, applying the BFS algorithm to the tree in Figure 1 would give the result shown in Table I.

C. Task 3

For this task, the code from Tasks 1 and 2 will be used to determine the objects that will be played against an agent called `breakable.py`. This agent is programmed to play randomly without ever repeating an object, until a certain sequence of objects, called the break sequence, is played. The break sequence will be unknown and will have a length between 2 and 5, inclusive. Once the break sequence is played, the agent `breakable.py` will repeat its last move an unknown number of times, before it starts playing randomly again. While the agent is repeating itself, it is possible to exploit it and win every move until the agent starts playing randomly again. At this stage, the known break sequence is simply played again to cause the agent to repeat its objects again.

In summary, the tree should be traversed using BFS and DFS respectively, until the break sequence is found. Once the break sequence is known, it can be used to defeat the agent. The script `rpsrunner.py` [4] will be used to evaluate submitted code and ensure that the agent `breakable.py` is successfully beaten. As a result of the fact that `rpsrunner.py` is written in Python 2, Python 2 should be used for all submissions.

The code submitted to `Practical 1 Code` should be your final implementation of Task 3 and should be a file that can be used by `rpsrunner.py` to play against `breakable.py`. Please remove all print statements and any other functions that were used for unit testing etc. before submitting your code.

1) Suggestions: It is possible to accidentally stumble upon the break sequence before reaching the correct node in the tree. For example, steps 3 and 4 in Table I could trigger the break sequence SR, even though the current sequence being tested is RR. If this occurs, the assumed break sequence will not work consistently, and searching of the tree should be resumed.

The variable bfs_dfs should be included at the top of your code to select between the BFS (bfs_dfs = 0) and DFS algorithms (bfs_dfs = 1).

The BFS and DFS can be implemented recursively or iteratively, but it is strongly recommended that your implementation is iterative.

If one implements this task successfully, one should expect to win 100% of the games against the agent `breakable.py`, but it is possible that this may not occur. Can you explain why?

IV. Report Content

Each student is required to submit a report providing the information listed below.

- Provide the results obtained.
- Provide a discussion of the results. Make sure to compare the BFS and DFS algorithms against one another and if the one performed better than the other on this problem, discuss why.
- Use the prescribed LaTeX format.
- **Do not** include the submitted computer code in the report.
- 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.

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.

References

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

[2] L. Strydom, *EAI 320 – Practical 1 Guide*, University of Pretoria, 7 Feb. 2019.

[3] (2019, 6 Feb.) Rock-paper-scissors – wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Rock-paper-scissors

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

Appendix A
Abbreviations

AI   artificial intelligence
AL   Assistant Lecturer
BFS   breadth first search
DFS   depth first search
P   paper
PDF   portable document format
R   rock
RPS   rock-paper-scissors
S   scissors

Listing 3: The source code for the agent breakable.py.

```python
# An agent which plays randomly without repeated the
    objects it plays until a specific sequence is played by
    its opponent. The agent then repeats the last object it
    played a random number of times before resuming its
    normal sequence.
#
# Uses ideas from information in agents submitted to http
    ://www.rpscontest.com
# Written by: W. P. du Plessis
# Last update: 2019-02-07


import random

if input == "":

    # The range of possible lengths of the break sequence.
    break_min = 2
    break_max = 5
    # The maximum number of repeats when repeating.
    repeat_max = 10

    # This line allows repeatable results.
    #random.seed(0)

    # The length of the break sequence is break_min to
        break_max hands.
    length = random.randint(break_min, break_max)

    # Generate the random break sequence.
    sequence = [ "X" ]*length
    for counter in range(length):
        sequence[counter] = random.choice(["R", "P", "S"])

    # Initialise the history.
    # Use "X" as the initial value because it does not
        match any of the hands.
    history = [ "X" ]*length

    # Initialise the variable for the number of repeats.
    repeat = 0

    # Play randomly for the first move.
    previous = random.choice(["R", "P", "S"])
```

```python
39  else:
40
41      # Update the history.
42      history.pop(0)
43      history.append(input)
44
45      # Initialise the repeat counter if the opponent's last
              hands match the break sequence.
46      if history == sequence:
47          repeat = random.randint(length, repeat_max)
48
49
50  # If the bot has entered a repeat cycle.
51  if repeat > 0:
52
53      # Reduce the repeat counter because a repeat will now
              take place.
54      repeat -= 1
55
56      # The history needs to be reset to ensure that repeats
              cannot be continuously triggered.
57      history = ["X"]*length
58
59  else:
60
61      # Play randomly while storing the value played and
              without repeating a move.
62      if previous == "R":
63          previous = random.choice(["P", "S"])
64      elif previous == "P":
65          previous = random.choice(["R", "S"])
66      else:
67          previous = random.choice(["R", "P"])
68
69
70  # Play the selected hand.
71  output = previous
```