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

EAI 320

Intelligent Systems

Practical 1 Guide V1.1

Updated on 18 February 2024

Concept: Prof. Warren P. du Plessis
Guide written by: Llewellyn Strydom
Guide revised by: Michael Teles, Steven Pinto and Heinrich Keyser
Due date: TBC

I. 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 [1], or using some other representation. Throughout this course, the framework provided by [2] 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 handshapes (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 [2] 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.

II. 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 an RPS agent that can beat `breakable.py`, which is provided in Listing 1 in Appendix C on page 5, and
4) appending the results at the end of your code files as comments.

III. Guide

Firstly, It is advised that a dynamic search tree structure is created that can be used by any type of search algorithm. 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 is linked and a recursive function can be used to build the tree.

An example of a tree with a depth of two is shown in Fig. 1.

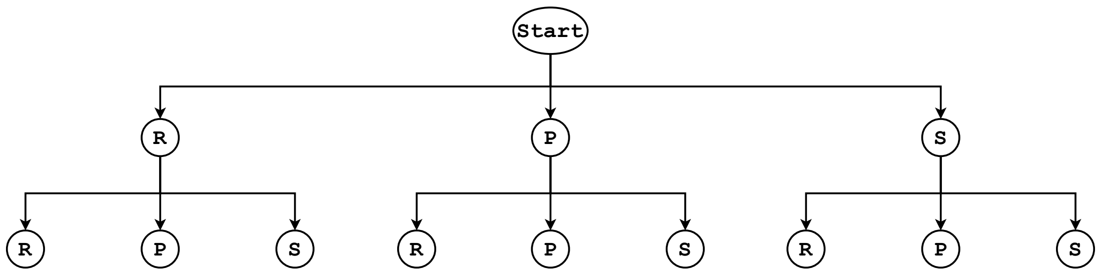

Fig. 1: Search tree structure

Suggestion: A `depth` parameter may be passed to the constructor of the tree to allow the tree to be constructed up to the specified depth, where the depth represents the number of moves.

Next, write BFS and DFS algorithms that can be used to search the tree incrementally. 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 Fig. 1 would give the result shown in TABLE I.

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

Your search algorithm can then be tested against the `breakable.py` agent. 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 can be 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.

It is recommended that the tree should be created and searched at the same time, dynamically, instead of building a tree at the start of a round and searching it as the round continues.

Suggestions: The primary goal of this assignment is that the specified search algorithms should be implemented. As a result, other approaches to solving the problem are not acceptable (e.g. playing randomly until a repeat is detected). Furthermore, the properties of the various search algorithms should be compared, so modifications to the search algorithms should not be implemented as such modifications may obscure the properties of the search algorithms (e.g. storing the previous five objects played to avoid the need for further search after the accidental breaks described in the next paragraph).

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

Implementing the entire search process before the first match takes place is not recommended as some of the aspects of the various search algorithms will not be clearly seen by this approach.

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? Add the explanation as a comment at the end of your code.Be sure to briefly compare all aspects of BFS and DFS as a comment at the end of your code.

IV. Submission Requirements

A. Code Instructions

Each group is required to submit code based on the requirements listed below. A mark of zero will be awarded if the submitted code 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.
- The group number, as well as the names and student numbers of the group members, must be included as comments at the top of the submitted file.
- Submitted code will be evaluated using the command

  ```
  rpsrunner.py <your_filename>.py breakable.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 3.
- The code submitted must be the final implementation of **Task 3**.
- Please add the final output (the rpsrunner output) as a comment in your code script.
- The first line of the code must declare a variable bfs_dfs to select between the BFS (bfs_dfs = 0) and DFS algorithms (bfs_dfs = 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] (2019, 6 Feb.) Rock-paper-scissors – wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Rock-paper-scissors

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

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

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

Appendix A
General Instructions

This section contains general instructions which apply to all submissions.

- The commented Python source code should be submitted.
- Submissions must be submitted via the EAI 320 ClickUp page. Do not email submission to the lecturer or assistant lecturer (AL) as emailed submissions will be considered not to have been submitted.
- The names of submitted files must have the following format:
  `eai320_prac_{practical number}_{group number}.{extension}`
- No late assignments will be accepted. No excuses for late submission will be accepted.
- Each group must do their own work. Academic dishonesty is unacceptable and cases will be reported to the University Legal Office for suspension.
- All information from other sources must be clearly identified and referenced.
- Any attempt to interfere with the operation of the framework used (e.g. to modify the scores of agents) will be regarded as academic dishonesty.
- **Only one student should submit the code and report to the ClickUp link.** Please follow the instructions meticulously to avoid the possibility of receiving a high plagiarism score for your group.
- Multiple submissions from all group members are allowed on the AMS.
- It is recommended that you upload the required code and documents to both systems as you work to avoid last-minute submission problems.

A. Submission

Code should be submitted both to the **AMS** and via the link on the EAI 320 **ClickUp** page: `Practicals` → `Practical 1` → `Practical 1 Code`.

Late submissions will not be accepted. Accepting late submissions is extremely unfair to those groups who submit their work on time 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, 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.

Groups will be allowed to submit updated copies of their reports and code 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.

Groups are required to include the standard cover page for group 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 group submitting the report is aware of the fact that academic dishonesty is unacceptable and a statement that the submitted work is the work of those students. 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 group is required to independently write their own code and report. No part of any group's work may be the same as any part of another group's work.

Groups should clearly indicate material from other sources and provide complete references to those sources. Examples of commonly used sources include the textbook [3] and this document [4]. Note that this does not mean that groups 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]

Appendix B
Abbreviations

AI	artificial intelligence
AL	assistant lecturer
BFS	breadth first search
DFS	depth first search
P	paper
R	rock
RPS	rock-paper-scissors
S	scissors

Appendix C
Source Code for the Agent breakable.py

Listing 1: 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.
```

[1]The objective of all academic assignments is that students learn fundamental aspects 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.

```
    repeat_max = 10

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

    # The length of the break sequence is break_min to break_max objects.
    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 objects.
    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”])

else:

    # Update the history.
    history.pop(0)
    history.append(input)

    # Initialise the repeat counter if the opponent's last objects match the break sequence.
    if history == sequence:
        repeat = random.randint(length, repeat_max)


# If the bot has entered a repeat cycle.
if repeat > 0:

    # Reduce the repeat counter because a repeat will now take place.
    repeat -= 1

    # The history needs to be reset to ensure that repeats cannot be continuously triggered.
    history = [ ”X” ]*length

else:
```

```python

    # Play randomly while storing the value played and
        without repeating a move.
    if previous == "R":
        previous = random.choice(["P", "S"])
    elif previous == "P":
        previous = random.choice(["R", "S"])
    else:
        previous = random.choice(["R", "P"])

# Play the selected object.
output = previous
```