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

EAI 320

Intelligent Systems

Practical 2 Guide V1.1

24 February 2020

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

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.
- 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 Sequence 1`
- `Practicals` → `Practical 2` → `Practical 2 Sequence 2`

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

Genetic algorithms (GAs) are numerical techniques that attempt to imitate evolution and natural selection. The motivation for this approach is the remarkable way in which natural systems adapt to their environments [3]. GAs have been applied to a very wide range of problems including optimisation [4], [5], computer programming [6], circuit design [6], the traveling salesman problem [4], and training of neural networks [7].

This assignment will require students to find a pseudo-optimal strategy for a rock-paper-scissors (RPS) agent by using a GA. The same RPS framework that was used for the previous assignment will be used again [8].

In accordance with the naming conventions used in the prescribed textbook, the string representing an individual will comprise 81 genes. Each gene will correspond to the object that an individual will play given a history of 2 moves (i.e. 4 objects). Fig. 1 provides a visual representation of an individual string. The histories, corresponding to the 81 different objects, should be in the same order as they would be visited by breadth first search (BFS) in a search tree of depth 4.

III. Instructions

A. Task 1

Each student has to implement a GA that can be used to find a pseudo-optimal strategy for a RPS agent, given some historical game data. More information can be found in Section 4.1, and more specifically Section 4.1.4, of the prescribed textbook [1].

[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.

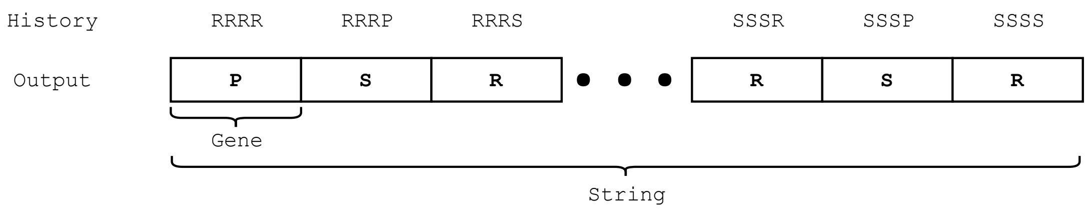

Fig. 1: A partial visual representation of an individual string. This represents a possible strategy for an RPS agent.

The student is expected to experiment with different population sizes and mutation rates, and all students should ensure that they answer the questions that are listed below as part of their discussion.

- How does the size of the population affect the performance of the GA?
- How is the selection step in the GA performed?
- How is the crossover step in the GA performed?
- How is the mutation step in the GA performed?
- How many iterations of the algorithm is required to find a good solution?
- What is the pseudo-optimal strategy that was found by the GA?

The technical report should also contain a graph where the fitness of the best individual from each generation is plotted against the number of generations. The average fitness of the population for each generation should also be overlayed onto this plot.

The historical game data is provided in comma-separated values (CSV) files, `data1.csv` and `data2.csv`. If we let agent 1 be the agent that the GA is trying to optimise and let agent 2 be the opposing agent, then we can represent the objects that agent 1 and agent 2 will play in the current round by x_t and y_t respectively. The history contained in the first column of `data1.csv` and `data2.csv` is then in the form $[x_{t-2}, y_{t-2}, x_{t-1}, y_{t-1}]$, and the second column represents the next object that the opponent played given the two game history. The optimal move for a given history is thus the move that will beat the opponent's move in the second column. The student may use this information in any way to design an evaluation function that can be used to identify strong and weak individuals during training.

Each student will have to submit three files for this assignment.

- A short, yet thorough, technical report with the Python code that was used to implement the GA added to the end of the report in the Appendix.
- A CSV file containing the optimal strategy found by the GA when using `data1.csv`. This file should contain 81 objects (R, P, or S), each separated only be a comma.
- A CSV file containing the optimal strategy found by the GA when using `data2.csv`. This file should contain 81 objects (R, P, or S), each separated only by a comma.

In summary, a program must be written to implement a GA that can be trained on a dataset (`data1.csv` or `data2.csv`) to generate a sequence of 81 moves. This sequence must then be written to a file using the same format as the file `rock_sequence.txt` shown in Listing 2 in section C. This file is read in by `agent.py` and used to determine the agent's next move against an opponent. The code for `agent.py` is shown in Listing 1 in section B.

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 deducted.

- Provide an introduction to the assignment.
- Explain how you have implemented a GA.
- Provide the results that were obtained.
- Provide a discussion of the results and a reflection on what you have learnt. Make sure to discuss your GA implementation and how different parameters of the algorithm effect its performance.
- Provide a conclusion for the assignment.
- Provide a list of references for the assignment.
- The Genetic Algorithm code must be included in the report appendix.
- Use the prescribed LaTeX format.

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 -r 1000 agent.py only_rock.py,only_paper.py,
            only_scissors.py,beat_previous.py,beat_common.py
  ```
  to test how successful the sequence generated by the implemented GA is.
- The histories given in the datasets were generated by playing against the bots listed in the command above.
- 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 as a result of the fact that `rpsrunner.py` is written in Python 3.
- The code submitted must be the final implementation of Task 1.
- TurnItIn will not accept code with the file extension `.py` or `.csv`, so the file extensions should be changed 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] W. P. du Plessis, "A genetic algorithm for impedance matching network design," Master's thesis, University of Pretoria, 2003.

[4] D. E. Goldberg, *Genetic algorithms in search, optimization, and machine learning*. New York, USA: Addison-Wesley, 1989.

[5] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs*. Berlin, Germany: Springer-Verlag, 1992.

[6] J. R. Koza, F. H. Bennet, D. Andre, and M. A. Keane, *Genetic programming III*. San Francisco, USA: Morgan Kaufmann, 1999.

[7] X. Yao, “A review of evolutionary artificial neural networks,” *Int. Journal Intelligent Systems*, vol. 8, no. 2, pp. 539–567, Jun. 1993.
[8] B. Knoll. (2011, 6 Feb.) Rock paper scissors programming competition. [Online]. Available: http://www.rpscontest.com/

Appendix A
Abbreviations

AL	Assistant Lecturer
BFS	breadth first search
CSV	comma-separated values
GA	genetic algorithm
PDF	portable document format
RPS	rock-paper-scissors

Appendix B
Source Code for the Agent agent.py

Listing 1: The source code for the agent `agent.py`.

```python
# An agent which plays a list of specified responses to a list of two-round histories.
#
# Uses ideas from information in agents submitted to http://www.rpscontest.com
# Written by: L. Strydom
# Updated by: M. Teles
# Last update: 2020-02-17

import numpy as np
import csv

# All the possible histories.
dictionary = [['R', 'R', 'R', 'R'], ['R', 'R', 'R', 'P'],
              ['R', 'R', 'R', 'S'], ['R', 'R', 'P', 'R'],
              ['R', 'R', 'P', 'P'], ['R', 'R', 'P', 'S'],
              ['R', 'R', 'S', 'R'], ['R', 'R', 'S', 'P'],
              ['R', 'R', 'S', 'S'], ['R', 'P', 'R', 'R'],
              ['R', 'P', 'R', 'P'], ['R', 'P', 'R', 'S'],
              ['R', 'P', 'P', 'R'], ['R', 'P', 'P', 'P'],
              ['R', 'P', 'P', 'S'], ['R', 'P', 'S', 'R'],
              ['R', 'P', 'S', 'P'], ['R', 'P', 'S', 'S'],
              ['R', 'S', 'R', 'R'], ['R', 'S', 'R', 'P'],
              ['R', 'S', 'R', 'S'], ['R', 'S', 'P', 'R'],
              ['R', 'S', 'P', 'P'], ['R', 'S', 'P', 'S'],
              ['R', 'S', 'S', 'R'], ['R', 'S', 'S', 'P'],
              ['R', 'S', 'S', 'S'], ['P', 'R', 'R', 'R'],
              ['P', 'R', 'R', 'P'], ['P', 'R', 'R', 'S'],
              ['P', 'R', 'P', 'R'], ['P', 'R', 'P', 'P'],
              ['P', 'R', 'P', 'S'], ['P', 'R', 'S', 'R'],
              ['P', 'R', 'S', 'P'], ['P', 'R', 'S', 'S'],
              ['P', 'P', 'R', 'R'], ['P', 'P', 'R', 'P'],
              ['P', 'P', 'R', 'S'], ['P', 'P', 'P', 'R'],
              ['P', 'P', 'P', 'P'], ['P', 'P', 'P', 'S'],
              ['P', 'P', 'S', 'R'], ['P', 'P', 'S', 'P'],
              ['P', 'P', 'S', 'S'], ['P', 'S', 'R', 'R'],
              ['P', 'S', 'R', 'P'], ['P', 'S', 'R', 'S'],
              ['P', 'S', 'P', 'R'], ['P', 'S', 'P', 'P'],
              ['P', 'S', 'P', 'S'], ['P', 'S', 'S', 'R'],
              ['P', 'S', 'S', 'P'], ['P', 'S', 'S', 'S'],
              ['S', 'R', 'R', 'R'], ['S', 'R', 'R', 'P'],
              ['S', 'R', 'R', 'S'], ['S', 'R', 'P', 'R'],
              ['S', 'R', 'P', 'P'], ['S', 'R', 'P', 'S'],
              ['S', 'R', 'S', 'R'], ['S', 'R', 'S', 'P'],
              ['S', 'R', 'S', 'S'], ['S', 'P', 'R', 'R'],
              ['S', 'P', 'R', 'P'], ['S', 'P', 'R', 'S'],
              ['S', 'P', 'P', 'R'], ['S', 'P', 'P', 'P'],
              ['S', 'P', 'P', 'S'], ['S', 'P', 'S', 'R'],
              ['S', 'P', 'S', 'P'], ['S', 'P', 'S', 'S'],
              ['S', 'S', 'R', 'R'], ['S', 'S', 'R', 'P'],
              ['S', 'S', 'R', 'S'], ['S', 'S', 'P', 'R'],
              ['S', 'S', 'P', 'P'], ['S', 'S', 'P', 'S'],
              ['S', 'S', 'S', 'R'], ['S', 'S', 'S', 'P'],
              ['S', 'S', 'S', 'S']]


# The values in the csv file should be the sequence generated using the output of the GA.
sequence_filename = "rock_sequence.txt" # example CSV file containing a sequence of 81 objects
with open(sequence_filename, newline='') as f:
    reader = csv.reader(f)
    response = list(reader)[0] # The responses to each history, loaded from the generated CSV file.



if input == '':
    history = ['X']*4
    output = np.random.choice(['R', 'P', 'S'])
else:
    history.pop(0)
    history.append(input)
    try:
        index = dictionary.index(history)
        output = response[index]
    except:
        output = np.random.choice(['R', 'P', 'S'])
    history.pop(0)
    history.append(output)
```

Appendix C
Example Sequence ROCK_SEQUENCE.TXT

Listing 2: Example CSV file sequence `rock_sequence.txt`.

```
R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R,R
```