Project Part B

Playing the Game

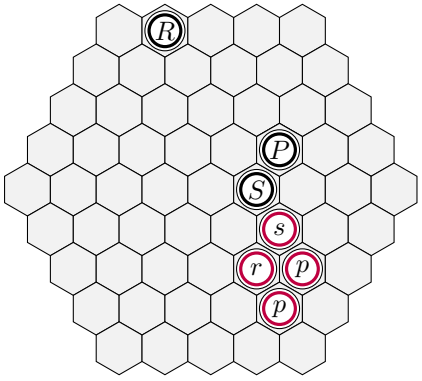
COMP30024 Artificial Intelligence

2021

1 Overview

In this second part of the project, we will play the full two-player version of *RoPaSci 360*. Before you read this specification you may wish to re-read the 'Rules for the Game of *RoPaSci 360*' document.

The aims for Project Part B are for you and your project partner to (1) practice applying the game-playing techniques discussed in lectures and tutorials, (2) develop your own strategies for playing *RoPaSci 360*, and (3) conduct your own research into more advanced algorithmic game-playing techniques; all for the purpose of creating the best *RoPaSci 360*–playing program the world has ever seen.



1.1 The task

Your task is twofold. Firstly, you will **design and implement a program** to play the game of *RoPaSci 360*. That is, given information about the evolving state of the game, your program will decide on an action to take on each of its turns (we provide a driver program to coordinate a game of *RoPaSci 360* between two such programs so that you can focus on implementing the game-playing strategy). Section 2 describes this programming task in detail, including information about how our driver program will communicate with your player program and how you can run the driver program.

Secondly, you will **write a report** discussing the strategies your program uses to play the game, the algorithms you have implemented, and other techniques you have used in your work, highlighting the most impressive aspects. Section 3 describes the intended structure of this document.

The rest of this specification covers administrative information about the project. For assessment criteria, see Section 4. For submission and deadline information see Section 5. Please seek our help if you have any questions about this project.

2 The program

You must create a program in the form of a Python 3.6 **module** named with **your team name**.

2.1 The `Player` class

When imported, your module must define a class named `Player` with at least the following three methods:

1. `def __init__(self, player):` Called once at the beginning of a game to initialise your player. Use this opportunity to set up an internal representation of the game state.

   The parameter `player` will be the string `"upper"` (if your program will play as Upper), or the string `"lower"` (if your program will play as Lower).

2. `def action(self):` Called at the beginning of each turn. Based on the current state of the game, your program should select and return an action to play this turn.

   The action must be represented based on the instructions for representing actions in the next section.

3. `def update(self, opponent_action, player_action):` Called at the end of each turn to inform your player of both players' chosen actions. Use this opportunity to update your internal representation of the game state.

   The parameter `opponent_action` will be your opponent's chosen action, and `player_action` will be the same action your player just returned through the `action` method. Both actions will be represented following the instructions for representing actions in the next section. The actions will always be allowed for the player (your method *does not* need to validate the actions against the game rules).

2.2 Representing actions

Our programs will need a consistent representation for actions. We will represent all actions as *three-tuples* containing first a string **action type** and then two **action arguments**, indexing hexes with the *axial coordinate system* from Part A (see Figure 1).

- To represent a **throw action**, use a tuple:

  `("THROW", `s`, (`r`, `q`))`

  where s is a string (`"r"` for rock, `"p"` for paper, or `"s"` for scissors, always in lower case) representing the symbol of token to throw, and (r, q) are the coordinates of the token.

- To represent a **slide action** or a **swing action**, use a tuple:

  `(`*atype*`, (`r_a`, `q_a`), (`r_b`, `q_b`))`

  where *atype* is the action type (`"SLIDE"` or `"SWING"`), (r_a, q_a) are the coordinates of the moving token before the action, and (r_b, q_b) are the new coordinates.

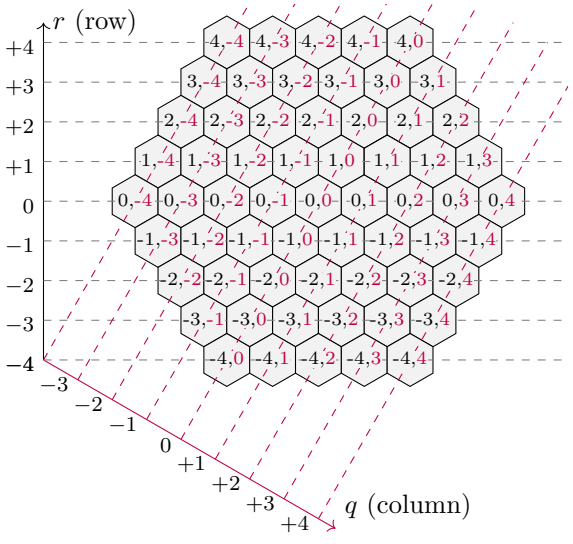


Figure 1: Each hex is addressed by a **row** (r) and **column** (q) pair, as shown. For notes on a similar axial coordinate system, see redblobgames.com/grids/hexagons/ (and **don't forget to acknowledge** any algorithms or code you use in your program).

2.3 Running your program

To play a game of *RoPaSci 360* with your program, we provide a driver program—a Python module called `referee`. For your information, the referee program has the following essential structure:

1. Set up a *RoPaSci 360* game. Initialise one `Player` class, as directed by the command line arguments, as each of Upper and Lower players (calling their `.__init__()` methods).
2. Repeat the following until the game ends:
   (a) Ask each player for their next action (calling their `.action()` methods).
   (b) Validate the actions and apply them to the game if they are allowed (otherwise, end the game with an error message). Display the resulting game state to the user.
   (c) Notify both players of the actions (calling their `.update()` methods).
3. After detecting one of the ending conditions, display the final result of the game to the user.

To play a game using `referee`, invoke it as follows. The referee module (the directory `referee/`) and the modules with your `Player` class(es) should be within your current directory:

```
python -m referee <upper module> <lower module>
```

where `python` is the name of a Python 3.6 interpreter[1] and `<upper module>` and `<lower module>` are the names of modules containing the classes `Player` to be used for Upper and Lower, respectively. The referee offers many additional options. To read about them, run '`python -m referee --help`'.

2.4 Program constraints

The following **resource limits** will be strictly enforced on your program during testing. This is to prevent your programs from gaining an unfair advantage just by using more memory and/or computation time. These limits apply to each player for an entire game. In particular, they *do not* apply to each turn separately. For help measuring or limiting your program's resource usage, see the referee's additional options (`--help`).

- A maximum computation time limit of **60 seconds per player, per game**.
- A maximum memory usage of **100MB per player, per game** (not including imported libraries).

You must not attempt to circumvent these constraints. For example, do not use multiple threads or attempt to communicate with other programs or the internet to access additional resources.

2.5 Allowed libraries

Your program should use **only standard Python libraries**, plus the optional third-party libraries **NumPy and SciPy** (these are the only libraries installed on dimefox). With acknowledgement, you may also include code from the AIMA textbook's Python library, where it is compatible with Python 3.6 and the above limited dependencies. Beyond these, **your program should not require any other libraries in order to play a game**.

However, to develop your program, you may use *any tools*, including tools using other programming languages. This is all allowed *as long as your `Player` class does not require these tools to be available when it plays a game* (because they are not available on `dimefox`).

For example, let's say you want to use machine learning techniques to improve your program. You could use third-party Python libraries such as scikit-learn/TensorFlow/PyTorch to build and train a model. You could then export the learned parameters of your model. Finally, you would have to (re)implement the prediction component of the model yourself, using only Python/NumPy/SciPy. Note that this final step is typically simpler than implementing the training algorithm, but may still be a significant task.

[1] Note that Python 3.6 is not available on `dimefox` by default. However, it can be used after running the command '`enable-python3`' (once per login).

3 The report

Finally, you must discuss the strategic and algorithmic aspects of your game-playing program and the techniques you have applied in a separate file called `report.pdf`.

This report is your opportunity to highlight your application of techniques discussed in class and beyond, and to demonstrate the most impressive aspects of your project work.

3.1 Report structure

You may choose any high-level structure of your report. Aim to present your work in a logical way, using sections with clear titles separating different topics of discussion.

Here are some suggestions for topics you might like to include in your report. Note: Not all of these topics or questions will be applicable to your project, depending on your approach. That's completely normal. You should focus on the topics which make sense for you and your work. Also, if you have other topics to discuss beyond those listed here, feel free to include them.

- **Describe your approach:** How does your game-playing program select actions throughout the game?

  Example questions: What search algorithm have you chosen, and why? Have you made any modifications to an existing algorithm? How have you handled the simultaneous-play nature of this game? What are the features of your evaluation function, and what are their strategic motivations? If you have applied machine learning, how does this fit into your overall approach? What learning methodology have you followed, and why?

- **Performance evaluation:** How effective is your game-playing program?

  Example questions: How have you judged your program's performance? Have you compared multiple programs based on different approaches, and, if so, how have you selected which is the most effective?

- **Other aspects:** Are there any other important creative or technical aspects of your work?

  Examples: algorithmic optimisations, specialised data structures, any other significant efficiency optimisations, alternative or enhanced algorithms beyond those discussed in class, or any other significant ideas you have incorporated from your independent research.

- **Supporting work**: Have you completed any other work to assist you in the process of developing your game-playing program?

  Examples: developing additional programs or tools to help you understand the game or your program's behaviour, or scripts or modifications to the provided driver program to help you more thoroughly compare different versions of your program or strategy.

You should focus on making your writing succinct and getting the level of detail right. The appropriate length for your report will depend on the extent of your work and so aiming for succinct writing will be more appropriate than aiming for a specific word or page count.

For example, there's probably no need to present detailed code inside your report. Moreover, there's no need to re-explain ideas we have discussed in class (and if you have applied a technique or idea that you think we may not be familiar with, then it would be appropriate to write a brief summary of the idea and provide a reference through which we can obtain more information).

3.2 Report constraints

While the structure and contents of your report are flexible, your report must satisfy the following constraints:

- Your report **must not be longer than 6 pages** (excluding references, if any).
- Your report can be written using any means but **must be submitted as a PDF document**.

4 Assessment

Your team's Project Part B submission will be assessed out of 22 marks, and contribute 22% to your final score for the subject. Of these 22 marks:

- **11 marks** will be allocated to the level of performance of your final player.

  Of these, **4 marks** are available for a player capable of consistently completing games without syntax/import/runtime errors, without invalid actions, and without violating the time or space constraints.

  The remaining marks are awarded based on the results of testing your player against a suite of hidden 'benchmark' opponents of increasing difficulty, as described below. In each case, the mark will be based on the number of games won by your player (multiple test games will be played against each opponent with your player playing as Upper and Lower in equal proportion, and draws counting for half as much as wins). All tests will use **Python 3.6** on the **student Unix machines** (for example, `dimefox`[2]).

  3 marks available: Opponents who choose randomly from their set of allowed actions each turn.

  2 marks available: 'Greedy' opponents who select the most immediately promising action available each turn, without considering your player's responses (for various definitions of 'most promising').

  2 marks available: Opponents using the adversarial search techniques discussed in class and a simple evaluation function to look an increasing number of turns ahead.

- **11 marks** will be allocated to the successful application of techniques demonstrated in your work.

  We will review your report (and, on occasion, your code[3]) to assess your application of adversarial game-playing techniques, including your game-playing strategy, your choice of adversarial search algorithm, and your evaluation function. For top marks, we will also assess your level of exploration beyond techniques discussed in class for enhancing the effectiveness of your player. **Note that your report will be the primary means for us to assess this component of the project,** so please use it as an opportunity to highlight your successful application of techniques. For more detail, see the following criteria:

  0–5 marks: Work that does not demonstrate a successful application of important techniques discussed in class for playing adversarial games.

  6–7 marks: Work that demonstrates a successful application of the important techniques discussed in class for playing adversarial games, possibly with some theoretical, strategic, or algorithmic enhancements to these techniques.

  8–9 marks: Work that demonstrates a successful application of the important techniques discussed in class for playing adversarial games, along with *many* theoretical, strategic, or algorithmic enhancements to these techniques, possibly including some *significant* enhancements based on independent research into algorithmic game-playing or original strategic insights into the game.

  10–11 marks: Work that demonstrates a *highly* successful application of important techniques discussed in class for playing adversarial games, along with *many significant* theoretical, strategic, or algorithmic enhancements to those techniques, based on independent research into algorithmic game-playing or original strategic insights into the game, leading to excellent player performance.

As per this marking scheme, it is possible to secure a satisfactory mark by successfully applying the techniques discussed in class. Beyond this, the project is open-ended. Every year, we are impressed by what students come up with. However, a word of guidance: We recommend starting with a simple approach before attempting more ambitious techniques, in case these techniques don't work out in the end.

[2] We strongly recommended that you test your program on `dimefox` before submission. Note that Python 3.6 is not available on `dimefox` by default, but it can be used after running the command '`enable-python3`' (once per login).

[3] We will not assess the 'quality' of your submitted code. We may seek to clarify and verify claims in your report by referring to your implementation. For at least this reason, you should submit well-structured, readable, and well-documented code.

4.1 Academic integrity

Unfortunately, we regularly detect and investigate potential academic misconduct and sometimes this leads to formal disciplinary action from the university. Below are some guidelines on academic integrity for this project. Please refer to the university's academic integrity website (academicintegrity.unimelb.edu.au), or ask the teaching team, if you need further clarification.

1. You are encouraged to discuss ideas with your fellow students, but **it is not acceptable to share code between teams, nor to use code written by anyone else**. Do not show your code to another team or ask to see another team's code.

2. You are encouraged to use code-sharing/collaboration services, such as GitHub, *within* your team. However, **you must ensure that your code is never visible to students outside your team**. Set your online repository to 'private' mode, so that only your team members can access it.

3. You are encouraged to study additional resources to improve your Python skills. However, **any code adapted or included from an external source must be clearly acknowledged**. If you use code from a website, you should include a link to the source alongside the code.

4. If external or adapted code represents a significant component of your program, you should also acknowledge it in your report. Note that for the purposes of assessing your successful application of techniques, using substantial amounts of externally sourced code will count for less than an original implementation. However, it's still better to properly acknowledge all external code than to submit it as your own in breach of the university's policy.

5 Submission

One submission is required from each team. That is, one team member is responsible for submitting all of the necessary files that make up your team's solution.

You must submit a single compressed archive file (e.g. a `.zip` or `.tar.gz` file) containing all files making up your solution via the 'Project Part B Submission' item in the 'Assessments' section of the LMS. This compressed file should contain all Python files required to run your program (with the correct directory structure), along with your report. In addition, if you have created any extra files to assist you while working on this project,[4] then all of these files are worth including when you submit your solution.

The submission deadline is **11:00PM on Wednesday the 12$^{\text{th}}$ May, Melbourne time (AEST).**

You may submit multiple times. We will mark the latest submission made by either member of your team unless we are advised otherwise. You may submit late. Late submissions will incur a penalty of **two marks per working day** (or part thereof) late.

5.1 Extensions

If you require an extension, please email the lecturers using the subject 'COMP30024 Extension Request' at the earliest possible opportunity. If you have a medical reason for your request, you will be asked to provide a medical certificate. Requests for extensions received after the deadline may be declined.

[4] For example you may have created alternative player classes, a modified referee, additional programs to test your player or its strategy, programs to create training data for machine learning, or programs for any other purpose not directly related to implementing your player class. As long as these files are not too large, you are encouraged to include them with your submission (and mention them in your report).