Project Part A

Single Player Infexion

COMP30024 Artificial Intelligence

March 2023

Overview

In this first part of the project, you will write a program to play a simplified "single player" version of Infexion. Before you read this specification, please make sure you have carefully read the entire 'Rules for the Game of Infexion' document on the LMS. Although you won't be writing an agent to play against an opponent just yet, you should be aiming to get very familiar with the game rules, board layout and corresponding hex coordinate system.

The aims for Project Part A are for you and your project partner to (1) refresh and extend your Python programming skills, (2) explore some of the algorithms you have encountered in lectures, and (3) become familiar with the Infexion game environment and coordinate system. This is also a chance for you to develop fundamental Python tools for working with the game: Some of the functions and classes you create now may be helpful later (when you are building your full game-playing program for Part B of the project).

We will be using **Gradescope** as the testing environment when assessing your code. You can (and should) regularly submit to Gradescope as a means to get immediate feedback on how you are progressing. The autograder is already equipped with a couple of "visible" test cases for this purpose. See the *Submission* section at the end of this document for details.

Single player Infexion

Imagine an Infexion game board which is already partially filled by Red and/or Blue tokens. Given the board state at this point, "single player" Infexion proceeds by having **only** Red play SPREAD actions thereon. Provided Red controls at least one token on the board, they should always be able

to find a finite sequence of SPREAD actions that results in a "win", as Blue cannot do anything to defend its position!

Importantly, in single player Infexion, we aren't just interested in securing a win as this is quite easy – rather, we want to win *in as few moves as possible*. In other words, an agent playing as Red should choose the *shortest* sequence of actions required to win. There are a number of assumptions you should make when solving this problem:

1. The coordinate system of the game board is the same as what is described in the Infexion game rules document.
2. In the given initial board state, there will be at least one Red token and at least one Blue token on the board. The total POWER will also be less than or equal to 49 as per the game rules.
3. The *cost* of a solution is defined as the number of SPREAD actions that must be played by Red to eliminate all Blue-controlled tokens and therefore win the game.
4. If there is a *tie*, that is, there are multiple sequences of SPREAD actions that attain a win for the **same** minimum *cost*, then any such sequence is considered an optimal solution to the problem.
5. The SPAWN action is prohibited in single player Infexion, so your solution should only comprise of SPREAD actions.
6. There is no need to worry about turn limits or draws – you will find that optimal solutions for the single player version of the game never utilise anything close to 343 moves.

Your tasks

Using a search strategy discussed in lectures, your tasks are to:

1. **Develop and implement a program** that *consistently* and *efficiently* identifies the shortest sequence of actions required to win, given an arbitrary (valid) board state as its "input".
2. **Write a brief report** discussing and analysing the strategy your program uses to solve this search problem.

These tasks are described in detail in the following sections.

The program

You have been given a template **Python 3.10** program in the form of a **module** called `search`. Your task is to complete the `search()` function in `program.py`. You may write any additional functions/classes as needed, just ensure any additional `.py` files are kept within the same module.

When completed, `search()` should return a list of SPREAD actions denoting the lowest cost win sequence, given an initial board state as input. Note that we have already supplied the necessary input/output code (in `__main__.py`), so you don't need to worry about this. To streamline the running of test cases, you can use `<` to redirect a `.csv` file to the program via the standard input stream.

Before continuing, download[1] and extract the template and run it from the **root** directory via the command `python -m search < test.csv`

Input format

The `search(...)` function is given a Python dictionary as input, denoting the initial board state. Dictionary entries have the form `(`r`, `q`):  (player, `k`)`, where:

- `player` is either `"r"` (cell controlled by Red) or `"b"` (cell controlled by Blue)
- r and q denote the coordinate on the board for the cell, (r, q)
- k is the POWER of the cell (how many tokens are stacked, $1 \leq k \leq 6$ as per the game description)

This means you can look up the current occupancy/power of a cell using a coordinate tuple (r, q) as a key. Just keep in mind that not all cells are necessarily occupied (the dictionary is a sparse representation), so check that the key exists before using it.

You may assume that all coordinates are valid as per the game description, and there are no overlapping cell coordinates in the cell state definitions (for each tuple, (r, q) will be unique to that tuple only). In other words, there is no need for "input validation" on your part – when we test your work, the `search()` function is always given a valid board configuration as per the Infexion game specification.

[1] The skeleton code is provided on the Canvas assignment page where you found this document.

Output format

The `search(...)` function must return the sequence of SPREAD actions forming an optimal (minimal cost) solution to the given problem. This should be represented by a list of tuples, where each tuple is of the form `(r, q, dr, dq)`. The first two values r and q represent the hex cell **position** where the SPREAD action originates, (r, q), whereas dr and dq represent the hex **direction** of the action. Simply put, (dr, dq) must be one of: $(0, 1)$, $(-1, 1)$, $(-1, 0)$, $(0, -1)$, $(1, -1)$, *or*, $(1, 0)$.

Note that you should **not** print anything to standard output in your final submission, as this is reserved for printing the actual result. All lines beginning with `SPREAD` will be taken to be part of the final action sequence (see `__main__.py` in the template).

Efficiency

While correctness and optimality matters first and foremost, you should also consider *efficiency* when designing your solution (remember to carefully consider different input scenarios). For example, you may find that a heuristic helps to improve the performance of your search. On the other hand, there could be a time or space trade-off in using a heuristic which results in it being very costly to compute in practice – it might be even slower than a simpler brute-force approach! Therefore, consider profiling your solution and compare approaches before assuming an optimisation actually provides a notable efficiency improvement. Similarly, be sure to consider the performance and memory usage of data structures you utilise.

Example

See Figure 1 for an example solution for the `test.csv` test case (this comes with the template code). Despite Red initially only controlling a single POWER 2 cell, they are able to take control of all Blue tokens in only five SPREAD actions.

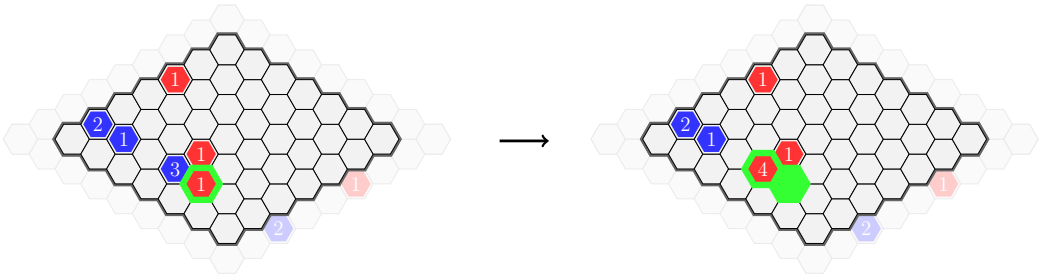
You might notice there are a few alternative action sequences Red could have taken to win in the same number of moves. If your solution happens to compute a different one of the same (minimal) cost, that's perfectly fine.

Take a look at the given template code and you'll see this same solution currently "hardcoded" into the `search()` function. Obviously you should write code to solve the *actual* problem yourself but the given example should provide clarity around the structure of the output.

(a) SPREAD `(5, 6, -1, 1)`

(b) SPREAD `(3, 1, 0, 1)`

(c) SPREAD `(3, 2, -1, 1)`



(d) SPREAD `(1, 4, 0, -1)`

(e) SPREAD `(1, 3, 0, -1)`

Figure 1: Example solution visualisation.

The report

Finally, you must briefly discuss your approach to solving this problem in a separate file called `report.pdf` (to be submitted alongside your program). Your discussion should address the following:

1. With reference to the lectures, which search strategy did you use? Discuss implementation details, including choice of relevant data structures, and analyse the time/space complexity of your solution.
2. If you used a heuristic as part of your approach, clearly state how it is computed and show that it speeds up the search (in general) without compromising optimality. If you did not use a heuristic, justify this choice.
3. Imagine that SPAWN actions were also allowed in single player Infexion (not *just* SPREAD actions). Discuss how this impacts the complexity of the search problem in general, and also how your team's solution would need to be modified in order to accommodate it.

Your report can be written using any means but must be submitted as **a PDF document**. Your report should be between 0.5 and 2 pages in length, and must not be longer than 2 pages (excluding references, if any). The quality and readability of your report matters, and marks won't be given where discussion is vague or irrelevant to topics discussed in the subject.

Assessment

Your team's Project Part A submission will be assessed out of 8 marks, and contribute 8% to your final score for the subject. Of these 8 marks:

- **5 marks** will be for the correctness of your program, based on running your program through a collection of automated test cases. The tests will run with **Python 3.10** on **Gradescope**. Programs that do not run in this environment will be considered incorrect. There will be a **30 second time limit** per test case, and credit will not be awarded for tests where a timeout occurs. All test cases will be solvable by our sample solution well within this time limit.

  You can minimise the risk of incompatibilities by submitting to Gradescope early and often – there are a couple of "visible" tests live already. You can re-submit as many times as you like, so make sure you take advantage of this. You should write your own tests as well (see `test.csv` for the format), as the visible tests don't cover all input scenarios.
- **3 marks** will be for the clarity and accuracy of the discussion in your `report.pdf` file, with 1 mark allocated to each of the three points listed above. A mark will be deducted if the report is longer than 2 pages or not a PDF document.

Your program should use **only standard Python libraries**, plus the optional third-party library **NumPy**[2] (this is just for extra flexibility – use of NumPy is not *required*). With acknowledgement, you may also include code from the AIMA textbook's Python library, where it is compatible with Python 3.10 and the above limited dependencies.

Code style/project organisation

While marks are not *dedicated* to code style and project organisation, you should write readable code in case the marker of your project needs to cross-check discussion in your report with your implementation. In particular, avoid including code that is **unused**. Report marks may be indirectly lost if it's difficult to ascertain what's going on in your implementation as a result of such issues.

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 [3] 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. When you submit your assignment, you are claiming that the work is your own, except where explicitly acknowledged.

4. If you use LLM tools such as ChatGPT, these **must be attributed** like any other external source – you should state exactly how you've used them in your report (under "References"). Technology to detect use of such tools is constantly evolving, and we will endeavour to use what is available come marking (or even retrospectively) to detect *dishonest* use of it. We do, however, believe such tools can be useful when used in the context of proper understanding of a subject area – in short, use them responsibly, ethically, and be aware of their limitations!

[2] Currently the latest version, NumPy v1.24, is available in the Gradescope environment
[3] Link: academicintegrity.unimelb.edu.au

Submission

The deadline is **11:00PM on Tuesday the 4th April, Melbourne time (AEST).** You may submit multiple times – only the latest submission will be marked.

The procedure for submission via Gradescope is almost identical to that of the "Project Team Member Nominations" submission, however in this case you are submitting multiple files. You must include the `team.py` file you originally submitted at the root of your submission (unmodified) so that your team can be properly identified (if you must modify it for some reason, please email us or create a private thread on Ed).

Here's how the directory structure for your submission should look:

```
/
├── team.py ..... The exact same file as the original you submitted
├── report.pdf ..... Your report for this assignment (must be a PDF)
└── search ..... Your Python module for this assignment
    ├── __main__.py ..... The original file from the template unmodified
    ├── program.py
    └ ..... You may have other .py source files (optional)
```

If you create a `.zip` archive with this structure internally, you should be able to just drag and drop it into the Gradescope "upload" box, and this will preserve the directory structure. Alternatively, if you have set up a GitHub repository to collaborate with your project partner, this can also be an efficient way to streamline submissions via Gradescope – again, ensure the same directory structure is used as shown above.

You may submit **multiple times** on Gradescope, and you are in fact **strongly encouraged** to do this early and often in order to test your work in the assessment environment. If you do make multiple submissions, we will mark the **latest** submission made.

Late submissions will incur a penalty of **one mark per working day** (out of the 8 total marks allocated for this part of the project).

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 will usually be declined.