Project Part A

Single Player Freckers

COMP30024 Artificial Intelligence

March 2025

1 Overview

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

Both you and your partner are expected to read this specification *in full* before commencing the project, then at your earliest convenience, you should both meet up and come up with an action plan for tackling it together (see Section 4.2 for our expectations regarding teamwork).

1.1 Single player Freckers

Single player Freckers is a simplified version of the two-player game whereby you play as Red. The game starts with an arbitrary configuration of lily pads and Blue frogs (Blue does not move). The goal for Red is to get a **single** frog to the final row of the board ($r = 7$) in *as few actions as possible*.

There are a number of assumptions you should make when solving this problem:

1. In the given initial board state, there are six Blue frogs arbitrarily placed on the board, as well as zero or more lily pads. A single Red frog is placed in the first row ($r = 0$).
2. The *cost* of a "solution" to this problem is defined as the number of actions that must be played by Red to get their frog to the other side of the board.
3. If there is a *tie*, that is, there are multiple sequences of actions that attain a win for the **same** minimum *cost*, then any such sequence is considered an optimal solution.
4. There is no need to worry about turn limits or draws. Optimal solutions for the single player variant of the game should not come anywhere near 150 actions.
5. GROW actions are prohibited – that is, all actions that comprise a minimal cost solution must be MOVE actions. This means it is possible for there to be no solution to certain instances of the problem (i.e., if there are no lily pads that facilitate a path to the other side).

1.2 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.

2 The program

You have been given a template **Python 3.12** 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 MOVE 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. More information regarding the test cases is provided in Section 2.5.

Before continuing, download the template and follow the "Running the template code" guide on the assignment LMS page. Once your local development environment is set up, try running the command `python -m search < test-vis1.csv`

2.1 Program inputs

The `search(...)` function is given a Python dictionary as input, denoting the initial state of the board. Dictionary entries have the form `Coord(r, c): CellState`, where:

- `r` and `c` denote the coordinate on the board for a cell, (r, c)
- `CellState` is an enum which can be either:
  - `CellState.RED` (cell contains a Red frog), OR;
  - `CellState.BLUE` (cell contains a Blue frog), OR;
  - `CellState.LILY_PAD` (cell contains a lily pad)

This allows you to retrieve the state of a cell using a coordinate structure instance `Coord(r, c)` 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.

The `Coord` and `CellState` types are defined in the provided `core.py` file. You are welcome to reuse types defined in this file anywhere in your solution, but please don't modify them.

For the purposes of assessment, you may assume that any given input coordinates and cell states will be valid (i.e., within valid bounds and taking one of the states listed above).

2.2 Program outputs

The `search(...)` function must **return** a sequence of MOVE actions that form an optimal (minimal cost) solution to the given problem. This should be represented by a list of structures, each item having the form `MoveAction(Coord(r1, c1), [Direction.d1, ...])`.

Notice that a `MoveAction` accepts an arbitrary number of `Direction` objects via its constructor. This is used to describe a series of jumps, if needed. A move to a neighbouring cell (or a single jump) can be represented by a single `Direction`. The skeleton code comes with a hardcoded example illustrating these scenarios (it is also a solution to the first provided test case).

If there is no way for Red to win for a given input scenario, you should instead return `None` from the function, **not** an empty list.

Please take care when printing anything to "standard output" as this is used for printing the actual result. All lines beginning with `$SOLUTION` will be taken to be part of the final action sequence (see `__main__.py` in the template).

2.3 Efficiency

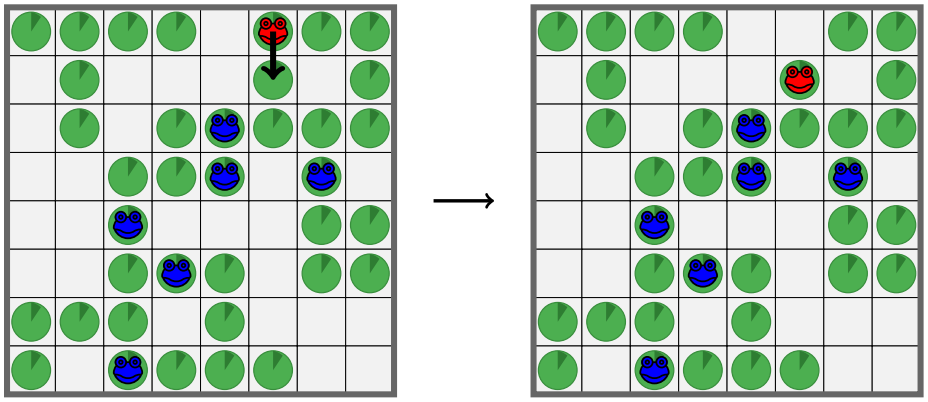
While correctness and optimality matters first and foremost, you should also consider *efficiency* when designing your solution (think about different input scenarios as discussed in Section 2.5). Consider profiling your solution and compare approaches *in practice* before assuming a theoretical optimisation actually provides a noteworthy efficiency improvement. Similarly, be sure to consider the performance and memory usage of data structures you utilise. Constant factors can and do matter in some instances.

You are welcome to use any data structures that come with the template code as part of your solution, but do so critically and ensure they are appropriate for what you are trying to achieve.

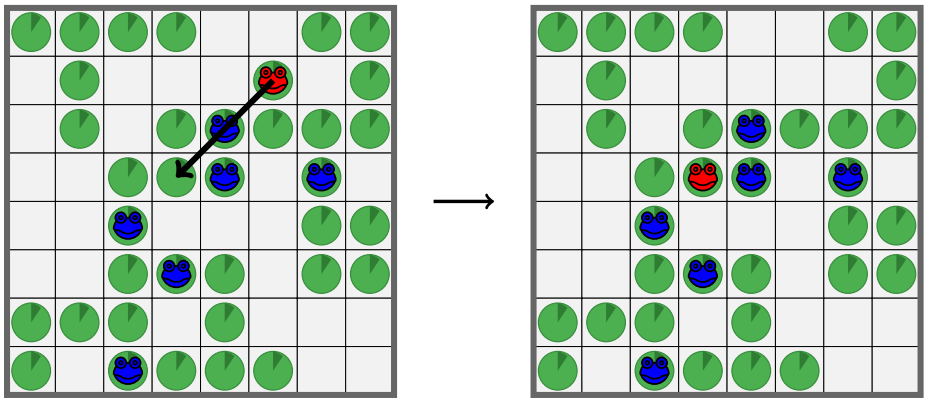
2.4 Example

See Figure 1 for an example solution to the `test-vis1.csv` test case which comes with the template code. You might notice there are alternative action sequences Red could have taken to win in the same number of moves. If your solution happens to compute a different sequence of the same (minimal) cost, that's perfectly fine.

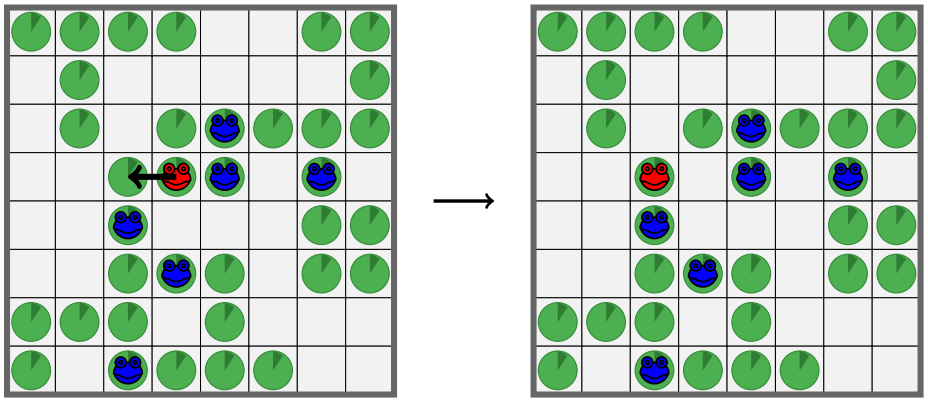
Take a look at the given template code and you'll see this same solution currently "hardcoded" into the `search()` function. Obviously you should write code to solve the *actual* problem yourself but the given example should provide clarity around the structure of the output.



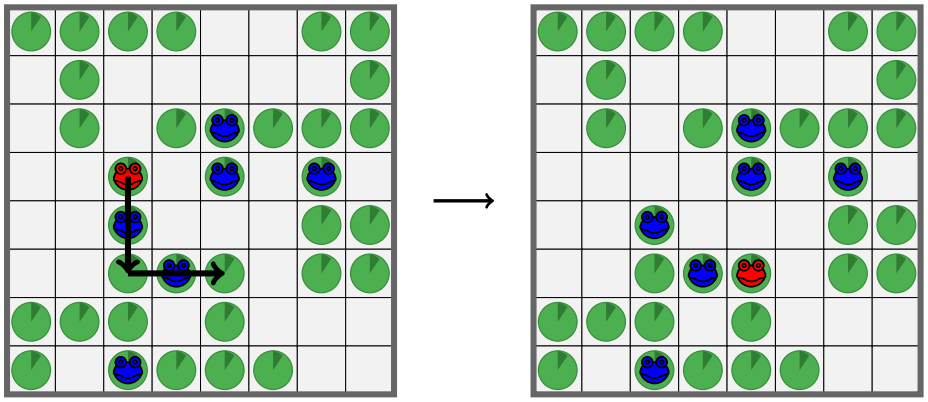
(a) MOVE$(0, 5)$ [DOWN]



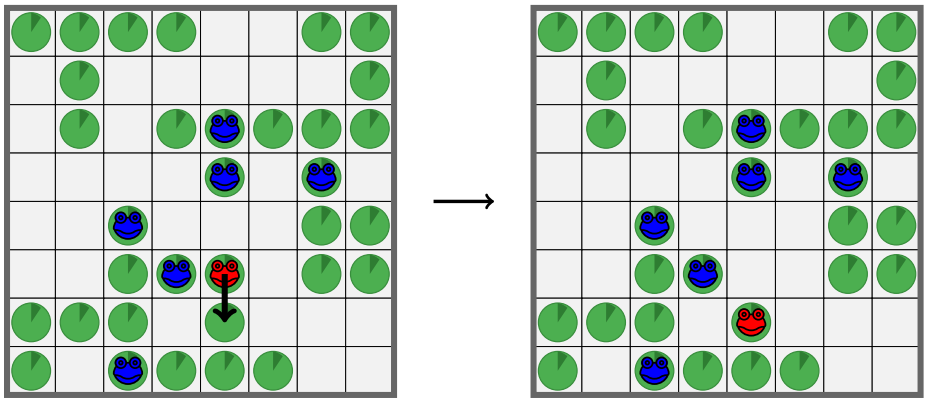
(b) MOVE$(1, 5)$ [DOWN-LEFT]



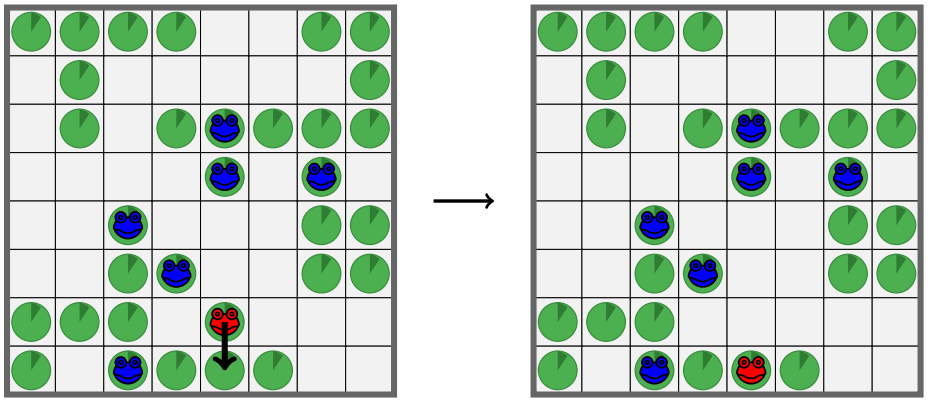
(c) MOVE$(3, 3)$ [LEFT]



(d) MOVE$(3, 2)$ [DOWN, RIGHT]



(e) MOVE$(5, 4)$ [DOWN]



(f) MOVE$(6, 4)$ [DOWN]

Figure 1: Example solution visualisation for `test-vis1.csv`. In this example, it is possible for Red to win in six moves.

2.5 Testing

Thorough testing of your work is critical in order to ensure that your program functions as expected across a wide array of possible inputs. Testing also helps identify bugs early on during development.

To help get you started, we have provided two sample test cases in the template, which are also live on Gradescope:

- `test-vis1.csv` – a test case with an optimal solution comprising three actions. A solution to this test has already been illustrated in Figure 1.
- `test-vis2.csv` – a test case where there is no possible solution.

If you open these files, you will see that the initial 8×8 board state is represented as 8 lines of 8 comma-separated characters. Each character is either a "blank" (an empty cell), an `R` (Red frog), a `B` (Blue frog) or a `*` (a lily pad). This file format is already parsed for you in the `__main__.py` file that comes in the template. It is a good idea to study this file to see how this works, but it should *not* be modified.

You can and should write your own tests. The given two cases cover distinct input scenarios but are not "exhaustive" in their own right.

2.6 Gradescope

To help you check that your work is compatible with our marking environment, both of the supplied tests are also live on Gradescope (where you'll submit your project) and can be run at any point before the deadline. Whenever you make a submission, both tests are automatically run on the spot, and the resulting outputs are compared against our sample solution outputs.

You should periodically submit to Gradescope to confirm that your work is compatible with our marking environment. There is no penalty for multiple submissions prior to the deadline, nor is there any hard limit to how many you can make. By default, only the most recent submission will be marked.

When we mark your work, a number of "hidden" tests of varying difficulty will be run in the exact same environment on Gradescope. Three of these (in addition to the two provided/visible cases) will be semi-visible before the deadline, that is, you will be able to see an estimated mark for each but without specific feedback. Each test case (hidden or not) will be given a maximum execution time of **30 seconds** before being forcibly terminated.

3 The report

You must also briefly discuss your approach to solving this problem in a separate file called `report.pdf`, 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 based approach, justify this choice.
3. Imagine that all six Red frogs had to be moved to the other side of the board to secure a win (not just one). Discuss how this impacts the nature of the search problem, and 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.

4 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 3.12** 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. You may re-submit as many times as you like, so make sure you take advantage of this. As discussed previously, you should write your own tests in addition to the two given tests, as they 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**[1] (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 3.12 and the above limited dependencies.

4.1 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.

4.2 Teamwork

This project is to be completed in teams of two. Both you and your partner are expected to contribute an equal amount of work throughout the entire duration of the project. While each person may focus on different aspects of the project, both should understand each other's work *in full* before submission (including all code).

Both partners are *also* expected to be proactive in communicating with each other, including meeting up early in the process and planning ahead. There will inevitably be deadlines in other subjects for one or both of you, and you'll need to plan around this (extensions won't be granted on this basis). Ensure that you set up regular ongoing meetings so that you don't lose track of what each person is doing.

We recommend using a code repository (e.g., on GitHub) to collaborate on the coding portion of the project. For the report, you may wish to use cloud based document editing software such as Google docs. This not only assists with keeping your work in sync and backed up, but also makes "auditing" easier from our end if there ends up being a dispute over contributions.

Where there is clear evidence that one person hasn't contributed adequately, despite their partner acting in good faith to collaborate with them as equals, individual marks will be awarded to better reflect each person's work.

In the event that there are teamwork issues, please **first** discuss your concerns with your partner *in writing* comfortably before the deadline. If the situation does not improve promptly, please notify us as soon as possible so that we can attempt to mediate while there is still time remaining (an email to the lecturers mailbox will suffice).

[1] Currently the latest version, NumPy v1.26, is available in the Gradescope environment

4.3 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 [2] 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!

5 Submission

The deadline is **11:00PM on Monday 7 April, Melbourne time (AEST).** You may submit multiple times, but only the latest submission will be marked.

The procedure for submission via Gradescope is similar to that of the "Project Team Member Nominations" submission, however in this case you are submitting multiple files. You must include your (unmodified) `team.py` file in the top level directory of your submission so that your team can be correctly identified.

Note that only **one** team member needs to submit the final work. Once submitted, they must then add their partner to the submission within the Gradescope interface (top right corner). If both

[2] Link: academicintegrity.unimelb.edu.au

team members submit individually we will randomly pick one of the submissions to mark and link team members based on information in the `team.py` file. Projects won't be remarked if the wrong one was picked in such a scenario (i.e., if one was an old submission).

Here's how the file tree 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
│   ├── core.py ..... The original file from the template unmodified
│   ├── program.py
│   └ ..... You may have other .py source files (optional)
└ ..... You can include other files like test cases (optional)
```

If you create a `.zip` archive with this structure internally, you should be able to "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 be an efficient way to streamline submissions via Gradescope – again, ensure the same directory structure is used inside the code repository 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.