

COMP3702 Artificial Intelligence (Semester 2, 2023)

Assignment 2: DRAGONGAME MDP

Key information:

- **Due: 3pm, Wednesday 20 September**
- This assignment assesses your skills in developing algorithms for solving MDPs.
- Assignment 2 contributes 20% to your final grade.
- This assignment consists of two parts: (1) programming and (2) a report.
- This is an individual assignment.
- Both the code and report are to be submitted via Gradescope (<https://www.gradescope.com/>). You can find a link to the COMP3702 Gradescope site on Blackboard.
- Your program (Part 1, 50/100) will be graded using the Gradescope code autograder, using testcases similar to those in the support code provided at <https://gitlab.com/3702-2023/a2-support>.
- Your report (Part 2, 50/100) should fit the template provided, be in .pdf format and named according to the format a2-COMP3702-[SID].pdf, where SID is your student ID. Reports will be graded by the teaching team.

The DRAGONGAME AI Environment

“Untitled Dragon Game”¹ or simply DRAGONGAME, is a 2.5D Platformer game in which the player must collect all of the gems in each level and reach the exit portal, making use of a jump-and-glide movement mechanic, and avoiding landing on lava tiles. DRAGONGAME is inspired by the “Spyro the Dragon” game series from the original PlayStation. **For this assignment, there are some changes to the game environment indicated in pink font. In Assignment 2, actions may have non-deterministic outcomes!**

To optimally solve a level, your AI agent must find a **policy (mapping from states to actions)** which collects all gems and reaches the exit while incurring the minimum possible **expected** action cost.

Levels in DRAGONGAME are composed of a 2D grid of tiles, where each tile contains a character representing the tile type. An example game level is shown in Figure 1.



Figure 1: Example game level of DRAGONGAME, showing character-based and GUI visualiser representations










¹The full game title was inspired by Untitled Goose Game, an Indie game developed by Australian developers in 2019

Game state representation

Each game state is represented as a character array, representing the tiles and their position on the board. In the visualizer and interactive sessions, the tile descriptions are graphical assets, whereas in the input file these are single characters.












Levels can contain the tile types described in Table 1.

Table 1: Table of tiles in DRAGONGAME, their corresponding symbol and effect

Tile	Symbol in Input File	Symbol in Visualiser	Effect
Solid	'X'		The player cannot move into a Solid tile. Walk and jump actions are valid when the player is directly above a Solid tile.
Ladder	'='		The player can move through Ladder tiles. Walk, jump, glide and drop actions are all valid when the player is directly above a Ladder tile. <i>When performing a walk action on top of a ladder tile, there is a small chance to fall downwards by 2 tiles (when the position 2 tiles below is non-solid).</i>
Air	' '		The player can move through Air tiles. Glide and drop actions are all valid when the player is directly above an Air tile.
Lava	'*'		<i>Moving into a Lava tile or a tile directly above a Lava tile results in Game Over.</i>
Super Jump	'J'		<i>When the player performs a 'Jump' action while on top of a Super Jump tile, the player will move upwards by between 2 and 5 tiles (random outcome - probabilities given in input file). If blocked by a solid tile, probabilities for moving to each of the blocked tiles roll over to the furthest non-blocked tile. For all other actions, Super Jump tiles behave as 'Solid' tiles.</i>
Super Charge	'C'		<i>When the player performs a 'Walk Left' or 'Walk Right' action while on top of a Super Charge tile, the player will move (left or right) until on top of the last adjoining Super Charge tile (see example in Table 2), then move in the same direction by between 2 and 5 tiles (random outcome - probabilities given in input file). If blocked by a solid tile, probabilities for moving to each of the blocked tiles roll over to the furthest non-blocked tile. For all other actions, Super Charge tiles behave as 'Solid' tiles.</i>
Gem	'G'		Gems are collected (disappearing from view) when the player moves onto the tile containing the gem. The player must collect all gems in order to complete the level. Gem tiles behave as 'Air' tiles, and become 'Air' tiles after the gem is collected.
Exit	'E'		Moving to the Exit tile after collecting all gems completes the level. Exit tiles behave as 'Air' tiles.
Player	'P'		The player starts at the position in the input file where this tile occurs. The player always starts on an 'Air' tile. In the visualiser, the type of tile behind the player is shown in place of the spaces.

An example of performing the Walk Right action on a Super Charge tile is shown in Table 2:

Table 2: Example Walk Right action on a Super Charge tile

			>>>	>>>	>>>	(P)	(P)	(P)	(P)		
											

The  on the green tile represents the current player position, '>>>' represents tiles which are skipped over, and each (P) represents a possible new position of the player.

Actions

At each time step, the player is prompted to select an action. Each action has an associated cost, representing the amount of energy used by performing that action. Each action also has requirements which must be satisfied by the current state in order for the action to be valid. The set of available actions, costs and requirements for each action are shown in Table 3.

Table 3: Table of available actions, costs and requirements

Action	Symbol	Cost	Description	Validity Requirements
Walk Left	wl	1.0	Move left by 1 position; if above a Ladder tile, random chance to move down by 2; if above a Super Charge tile, move a variable amount (see example in Table 2)	Current player must be above a Solid or Ladder tile, and new player position must not be a Solid tile.
Walk Right	wr	1.0	Move right by 1 position; if above a Ladder tile, random chance to move down by 2; if above a Super Charge tile, move a variable amount (see example in Table 2)	
Jump	j	2.0	Move up by 1 position; if above a Super Jump tile, move up by between 2 and 5 (random outcome)	
Glide Left 1	gl1	0.7	Move left by between 0 and 2 (random outcome) and down by 1.	Current player must be above a Ladder or Air tile, and all tiles in the axis aligned rectangle enclosing both the current position and new position must be non-solid (i.e. Air or Ladder tile). See example below.
Glide Left 2	gl2	1.0	Move left by between 1 and 3 (random outcome) and down by 1	
Glide Left 3	gl3	1.2	Move left by between 2 and 4 (random outcome) and down by 1	
Glide Right 1	gr1	0.7	Move right by between 0 and 2 (random outcome) and down by 1	
Glide Right 2	gr2	1.0	Move right by between 1 and 3 (random outcome) and down by 1	
Glide Right 3	gr3	1.2	Move right by between 2 and 4 (random outcome) and down by 1	
Drop 1	d1	0.3	Move down by 1	Current player must be above a Ladder or Air tile, and all cells in the line between the current position and new position must be non-solid (i.e. Air or Ladder tile).
Drop 2	d2	0.4	Move down by 2	
Drop 3	d3	0.5	Move down by 3	

Example of glide action validity requirements for GLIDE_RIGHT_2 ('gr2'):

Current Position	Must be Non-Solid	Must be Non-Solid
Must be Non-Solid	Must be Non-Solid	New Position

Interactive mode

A good way to gain an understanding of the game is to play it. You can play the game to get a feel for how it works by launching an interactive game session from the terminal with the following command:

```
$ python play_game.py <input_file>.txt
```

where <input_file>.txt is a valid testcase file from the support code with path relative to the current directory, e.g. `testcases/L1.txt`

In interactive mode, type the symbol for your chosen action and press enter to perform the action. Type 'q' and press enter to quit the game.

DRAGONGAME as an MDP

In this assignment, you will write the components of a program to play DRAGONGAME, with the objective of finding a high-quality solution to the problem using various sequential decision-making algorithms based on the Markov decision process (MDP) framework. This assignment will test your skills in defining a MDP for a practical problem and developing effective algorithms for large MDPs.

What is provided to you

We provide supporting code in Python, in the form of:

1. A class representing the DRAGONGAME environment and a number of helper functions (in `game_env.py`)
 - The constructor takes an input file (testcase) and converts it into a DRAGONGAME map
2. A class representing the DRAGONGAME game state (in `game_state.py`)
3. A graphical user interface for visualising the game state (in `gui.py`)
4. A solution file template (`solution.py`)
5. A tester (in `tester.py`)
6. Testcases to test and evaluate your solution (in `./testcases`)

The support code can be found at: <https://gitlab.com/3702-2023/a2-support>. Please read the README.md file which provides a description of the provided files. Autograding of code will be done through Gradescope, so that you can test your submission and continue to improve it based on this feedback — you are strongly encouraged to make use of this feedback.

Your assignment task

Your task is to develop planning algorithms for determining the optimal policy (mapping from states to actions) for the agent (i.e. the Dragon), and to write a report on your algorithms' performance. You will be graded on both your submitted **program (Part 1, 50%)** and the **report (Part 2, 50%)**. These percentages will be scaled to the 20% course weighting for this assessment item.

The provided support code formulates DRAGONGAME as an MDP, and your task is to submit code implementing the following MDP algorithms:

1. Value Iteration (VI)
2. Policy Iteration (PI)

Once you have implemented and tested the algorithms above, you are to complete the questions listed in the section "Part 2 - The Report" and submit the report to Gradescope.

More detail of what is required for the programming and report parts are given below.

Part 1 — The programming task

Your program will be graded using the Gradescope autograder, using testcases similar to those in the support code provided at <https://gitlab.com/3702-2023/a2-support>.

Interaction with the testcases and autograder

We now provide you with some details explaining how your code will interact with the testcases and the autograder (with special thanks to Nick Collins for his efforts making this work seamlessly).

Implement your solution using the supplied `solution.py` template file. You are required to fill in the constructor and the following method stubs of the Solver class:

- | | |
|-------------------------------------|-----------------------------------|
| • <code>vi_initialise()</code> | • <code>pi_initialise()</code> |
| • <code>vi_is_converged()</code> | • <code>pi_is_converged()</code> |
| • <code>vi_iteration()</code> | • <code>pi_iteration()</code> |
| • <code>vi_get_state_value()</code> | • <code>pi_select_action()</code> |
| • <code>vi_select_action()</code> | |

You can add additional **helper methods and classes** (either in `solution.py` or in files you create) if you wish. To ensure your code is handled correctly by the autograder, you should avoid using any try-except blocks in your implementation of the above methods (as this can interfere with our time-out handling). Refer to the documentation in `solution.py` for more details.

If you are unable to solve certain testcases, you may specify which testcases to attempt in `testcases_to_attempt`.

Grading rubric for the programming component (total marks: 50/100)

For marking, we will use 5 different testcases to evaluate your solution. Each test case is scored out of 10.0 marks in the following categories:

- a) Completion/reaching goal (0.5 pts pass/fail)
- b) Number of Iterations vs target (up to 1 pt with partial marks)
- c) Average time per iteration vs target (up to 1 pt with partial marks) [disqualified if max time per iteration is much greater than the mean]
- d) Convergence of values/policy (1pt pass/fail)
- e) Values - distance from reference solution values (up to 1 pt with partial marks) [assessed for VI only]
- f) Reward vs target (up to 1 pt with partial marks)

This totals to 5.5 marks per testcase for VI, 4.5 marks per testcase for PI, resulting in 10 marks per testcase, and 50 marks total over 5 testcases.

Part 2 — The report

The report tests your understanding of MDP algorithms and the methods you have used in your code, and contributes 50/100 of your assignment mark.

Question 1. MDP problem definition (18 marks)

- a) Define the State space, Action space, Transition function, and Reward function components of the DRAGONGAME MDP planning agent as well as where these are represented in your code. (10 marks)
- b) Describe the purpose of a discount factor in MDPs. (3 marks)
- c) State and briefly justify what the following dimensions of complexity are of your agent in the DRAGONGAME MDP. (See <https://artint.info/3e/html/ArtInt3e.Ch1.S5.html> for definitions) (5 marks)
 - Planning Horizon
 - Sensing Uncertainty
 - Effect Uncertainty
 - Computational Limits
 - Learning

Question 2. Comparison of algorithms (17 marks)

This question requires comparison of your implementation of value iteration (VI) and policy iteration (PI). If you did not implement PI, you may receive partial marks for this question by providing insightful relevant comments on your implementation of VI. For example, if you tried standard VI and asynchronous VI, you may compare these two approaches for partial marks.

- a) Describe your implementations of Value Iteration and Policy Iteration in one sentence each. Include details such as whether you used asynchronous updates, and how you handled policy evaluation in PI. (2 marks)
- b) Pick three representative testcases to compare the performance of VI and PI, reporting the numerical values for the following performance measures: (6 marks)

- Time to converge to the solution.
 - Number of iterations to converge to the solution.
- c) Discuss the difference between the numbers you found for VI and PI. Explain and provide reasons for why the differences either make sense, or do not make sense. (9 marks)

Question 3. Investigating optimal policy variation

(15 marks)

One consideration in the solution of a Markov Decision Process (i.e. the optimal policy) is the trade off between a risky higher reward vs a lower risk lower reward, which depends on the probabilities of non-deterministic dynamics of the environment and the rewards associated with certain states and actions.

Consider testcase L4.txt, and explore how the policy of the agent changes with *ladder_fall_prob* and *game_over_penalty*. The agent must cross from the bottom left (to collect a gem) to the bottom right (to reach the exit). There are 3 possible paths from left to right (bottom, middle and top). Because of the chance to fall when performing a walk action while on top of a ladder, there is a risk of falling into the lava in the centre.

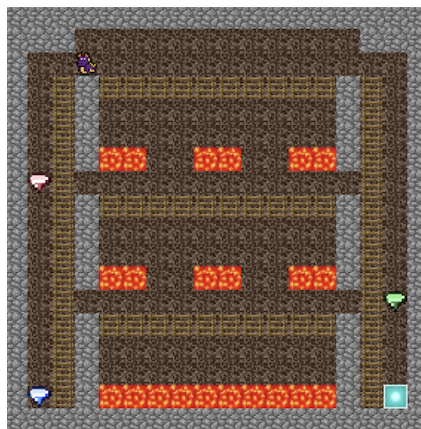


Figure 2: Testcase L4.txt

- The **bottom path** has the lowest cost (fewest jumps needed) and highest risk, as the lava tiles above prevent using jump+glide (which is more expensive than walking but has no fall risk).
- The **middle path** also has lava tiles above preventing jump+glide, but if the agent falls once it can glide and land in a safe part of the bottom path for a second chance at making it across.
- The **top path** has enough headroom to allow jump+glide (which eliminates the risk of falling), but requires a large number of jumps to reach, so is expensive.

The level of risk presented by the lower paths can be tuned by adjusting the *ladder_fall_prob* and the *game_over_penalty*.

If you did not implement PI, you may change the solver type to VI in order to answer this question.

- Describe how you expect the optimal path to change as the *ladder_fall_prob* and *game_over_penalty* values change. Use facts about the algorithms or Bellman optimality equation to justify why you expect these changes to have such effects. (7.5 marks)
- Picking three suitable values for *ladder_fall_prob*, and three suitable values for the *game_over_penalty*, explore how the optimal policy changes over the 9 combinations of these factors. You should present the results in a table, indicating whether the agent's optimal policy is to traverse the top, middle or bottom path, or something else, using colours to denote the optimal behaviour for each combination. Do the experimental results align with what you thought should happen? If not, why? (7.5 marks)

Academic Misconduct

The University defines Academic Misconduct as involving “a range of unethical behaviours that are designed to give a student an unfair and unearned advantage over their peers.” UQ takes Academic Misconduct very seriously and any suspected cases will be investigated through the University's standard policy (<https://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct>). If you are found guilty, you may be expelled from the University with no award.

It is the responsibility of the student to ensure that you understand what constitutes Academic Misconduct and to ensure that you do not break the rules. If you are unclear about what is required, please ask.

It is also the responsibility of the student to take reasonable precautions to guard against unauthorised access by others to his/her work, however stored in whatever format, both before and after assessment.

In the coding part of COMP3702 assignments, you are allowed to draw on publicly-accessible resources and provided tutorial solutions, but you must make reference or attribution to its source, by doing the following:

- All blocks of code that you take from public sources must be referenced in adjacent comments in your code or at the top of your code.
- Please also include a list of references indicating code you have drawn on in your `solution.py` docstring.

If you have utilised Generative AI tools such as ChatGPT, you must cite the tool and version in your code as well as describe in the report. A failure to reference generative AI use may constitute student misconduct under the Student Code of Conduct.

However, you must not show your code to, or share your code with, any other student under any circumstances. You must not post your code to public discussion forums (including Ed Discussion) or save your code in publicly accessible repositories (check your security settings). You must not look at or copy code from any other student.

All submitted files (code and report) will be subject to electronic plagiarism detection and misconduct proceedings will be instituted against students where plagiarism or collusion is suspected. The electronic plagiarism detection can detect similarities in code structure even if comments, variable names, formatting etc. are modified. If you collude to develop your code or answer your report questions, you will be caught.

For more information, please consult the following University web pages:

- Information regarding Academic Integrity and Misconduct:
 - <https://my.uq.edu.au/information-and-services/manage-my-program/student-integrity-and-conduct/academic-integrity-and-student-conduct>
 - <http://ppl.app.uq.edu.au/content/3.60.04-student-integrity-and-misconduct>
- Information on Student Services:
 - <https://www.uq.edu.au/student-services/>

Late submission

Students should not leave assignment preparation until the last minute and must plan their workloads to meet advertised or notified deadlines. It is your responsibility to manage your time effectively.

It may take the autograder up to an hour to grade your submission. It is your responsibility to ensure you are uploading your code early enough and often enough that you are able to resolve any issues that may be revealed by the autograder *before the deadline*. Submitting non-functional code just before the deadline, and not allowing enough time to update your code in response to autograder feedback is not considered a valid reason to submit late without penalty.

Assessment submissions received after the due time (or any approved extended deadline) will be subject to a 100% late penalty. A one-hour grace period will be applied to the due time after which time the 100% late

penalty will be imposed. This grace period is designed to deal with issues that might arise during submission (e.g. delays with Blackboard or Turnitin) and should not be considered a shift of the due time. Please keep a record of your submission time.

In the event of exceptional circumstances, you may submit a request for an extension. You can find guidelines on acceptable reasons for an extension here <https://my.uq.edu.au/information-and-services/manage-my-program/exams-and-assessment/applying-extension>. All requests for extension must be submitted on the UQ Application for Extension of Assessment form at least 48 hours prior to the submission deadline.