

# COMP3702 Artificial Intelligence (Semester 2, 2021)

## Assignment 1: Search in DRAGONGAME

### Key information:

- **Due: 4pm, Friday 27 August 2021**
- This assignment will assess your skills in developing discrete search techniques for challenging problems.
- Assignment 1 contributes 15% to your final grade.
- This assignment consists of two parts: (1) programming and (2) a report.
- This is an individual assignment.
- Both code and report are to be submitted via Gradescope (<https://www.gradescope.com/>). You can find instructions on how to register for the COMP3702 Gradescope site on Blackboard.
- Your program (Part 1) will be graded using the Gradescope code autograder, using the testcases in the support code provided at <https://gitlab.com/3702-2021/a1-support>.
- Your report (Part 2) should fit the template provided, be in .pdf format and named according to the format a1-[courseCode]-[SID].pdf. Reports will be graded by the teaching team.

### The DRAGONGAME AI Environment

“Untitled Dragon Game”<sup>1</sup> 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.

To optimally solve a level, your AI agent must find a sequence of actions which collects all gems and reaches the exit while incurring the minimum possible 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.

```
XXXXXXXXXXXXXXXXXXXXX
XX                      XXXXX
X G                      XXXX
XXX=X                    XX
X  =                    G    X
X  =                    XX=   X
X  = G                  =    X
X=XXXXXX                =    X
X=   X                  =   G X
X=   X                  =  XXXX
X=                    =X = XX X
X=XX G                =XX =   EX
X=PXXX XXXXXXXX      =X
XXXXXXXX**XXXXXXXX**XXX
```

Figure 1: Example game level of DRAGONGAME

---

<sup>1</sup>This full game title was inspired by Untitled Goose Game, an Indie game developed by some Australians 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 triples of characters, 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'	'XXX'	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.
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	'*'	'***'	The player cannot move into a Lava tile. Moving into a tile directly above a Lava tile results in Game Over.
Gem	'G'	' 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'	' E '	Moving to the Exit tile after collecting all gems completes the level. Exit tiles behave as 'Air' tiles.
Player	'P'	' 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.

## 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 2.

Table 2: Table of available actions, costs and requirements

Action	Symbol	Cost	Description	Validity Requirements
Walk Left	wl	1.0	Move left by 1 position	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	
Jump	j	2.0	Move up by 1 position.	
Glide Left 1	gl1	0.7	Move left by 1 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 2 and down by 1	
Glide Left 3	gl3	1.2	Move left by 3 and down by 1	
Glide Right 1	gr1	0.7	Move right by 1 and down by 1	
Glide Right 2	gr2	1.0	Move right by 2 and down by 1	
Glide Right 3	gr3	1.2	Move right by 3 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).

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 a search problem

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 search algorithms. This assignment will test your skills in defining a search space for a practical problem and developing good heuristics to make your program more efficient.

## What is provided to you

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

1. A class representing DRAGONGAME game map and a number of helper functions
2. A parser method to take an input file (testcase) and convert it into a DRAGONGAME map
3. A state visualiser
4. A tester
5. Testcases to test and evaluate your solution
6. A solution file template

The support code can be found at: <https://gitlab.com/3702-2021/a1-support>. 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 a program that outputs a path (series of actions) for the agent (i.e. the Dragon), and to provide a written report explaining your design decisions and analysing your algorithms' performance. You will be graded on both your submitted **program (Part 1, 60%)** and the **report (Part 2, 40%)**. These percentages will be scaled to the 15% course weighting for this assessment item.

To turn DRAGONGAME into a search problem, you have will have to first define the following agent design components:

- A problem state representation (state space),
- A successor function that indicates which states can be reached from a given state (action space and transition function), and
- A cost function (utility function); **the cost of each movement is given in Table 2**

Note that a goal-state test function is provided in the support code. Once you have defined the components above, you are to develop and submit code implementing two discrete search algorithms:

1. Uniform-Cost Search, and
2. A\* Search

Your submitted code should run A\* search or UCS search based on the 'mode' argument. Both UCS and A\* will be run separately by the autograder, and the result will be the best score out of both algorithms. Finally, after you have implemented and tested the algorithms above, you are to complete the questions listed in the section "Part 2 - The Report" and submit them as a written report.

More detail of what is required for the programming and report parts are given below. Under the grading rubric discussed below, the testcases used to assess the programming component will give a higher mark for A\* search, and you will not be able to answer some of the report questions without considering A\* or implementing it. These elements of the rubric give you an incentive to implement A\* search over the simpler UCS algorithm. *Hint: Start by implementing a working version of UCS, and then build your A\* search algorithm out of UCS using your own heuristics.*

## Part 1 — The programming task

Your program will be graded using the Gradescope autograder, using the testcases in the support code provided at <https://gitlab.com/3702-2021/a1-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). Your solution code only needs to interact with the autograder via the *output\_file* it generates. This is handled as follows:

- The file `solution.py`, supplied in the support code, is a template for you to write your solution. All of the code you write can go inside this file, or if you create your own additional python files they must be invoked from this file.
- Your program will: (i) take a testcase as the `input_filename` filename, an `output_filename`, and a mode (either 'ucs' or 'a\_star') as arguments, (ii) find a solution to the testcase, and (iii) write the solution to an output file with the given `output_filename`.
- Your code should generate a solution in the form of a comma-separated list of actions, taken from the set of *action symbols* defined in the supplied `game_env.py` file and in Table 2, which are:

- `WALK_LEFT` = 'wl'
- `WALK_RIGHT` = 'wr'
- `JUMP` = 'j'
- `GLIDE_LEFT_1` = 'gl1'
- `GLIDE_LEFT_2` = 'gl2'
- `GLIDE_LEFT_3` = 'gl3'
- `GLIDE_RIGHT_1` = 'gr1'
- `GLIDE_RIGHT_2` = 'gr2'
- `GLIDE_RIGHT_3` = 'gr3'
- `DROP_1` = 'd1'
- `DROP_2` = 'd2'
- `DROP_3` = 'd3'

- The `main()` method stub in `solution.py` makes it clear how to interact with the environment: (i) The `game_env` constructor (`__init__(filename)`) handles reading the input file, (ii) your code is called to solve the problem, with your solver's actions written to the `actions` variable, then (iii) `write_output_file(filename, actions)` function handles writing to the output file in the correct format, which is passed to the autograder.
- The script `tester.py` can be used to test individual testcases.
- The *autograder* (hidden to students) handles running your python program with all of the testcases. It will run the `tester` python program on your output file and assign a mark for each testcase based on the return code of `tester`.
- You can inspect the testcases in the support code, which each include information on their optimal solution path lengths and test time limits. Looking at the testcases might also help you develop heuristics using your human intelligence and intuition.
- To ensure your submission is graded correctly, do not rename any of the provided files or alter the methods in `game_env.py` or `game_state.py`.

More detailed information on the DragonGame implementation is provided in the Assignment 1 Support Code *README.md*, while a high-level description is provided in the DRAGONGAME AI Environment description document.

### Grading rubric for the programming component (total marks: 60/100)

For marking, we will use 8 different testcases of ascending level of difficulty to evaluate your solution. There will be a total of 60 code marks, consisting of:

- 20 Threshold Marks
  - Program runs without errors (+5 marks)
  - Program solves at least 1 testcase within 5x time limit (+7.5 marks)
  - Program solves at least 2 testcases within 5x time limit (+7.5 marks)
- 40 Testcase Marks
  - For each testcase, both UCS and A\* modes will be run
    - \* Score for the testcase is the maximum of the scores from the two modes
    - \* Full mark for higher difficulty testcases will only be possible with A\*
  - Maximum of 5 marks for each testcase, with deductions for taking more than the time limit or solution having higher than optimal cost ( $Cost_{Tgt}$ ), proportional to the amount exceeded
  - For each test case:
 
$$\text{mark} = 5.0 \times \left[ 1.0 - \frac{\text{clip}(\text{Cost} - \text{Cost}_{Tgt}, 0)}{\text{Cost}_{Tgt}} - 0.5 \frac{\text{clip}(\text{Time} - \text{TimeLimit}, 0)}{\text{TimeLimit}} \right]$$
  - Program will be terminated after 5x time limit has elapsed

## Part 2 — The report

The report tests your understanding of the methods you have used in your code, and contributes 40/100 of your assignment mark. **Please make use of the report templates provided on Blackboard**, because Gradescope makes use of a predefined assignment template. Submit your report via Gradescope, in .pdf format (i.e. use "save as pdf" or "print-to-file" functionality), and named according to the format `a1-[courseCode]-[SID].pdf`. Reports will be graded by the teaching team.

Your report task is to answer the questions below:

**Question 1. (5 marks)**

State the dimensions of complexity in DRAGONGAME, and explain your selection.

**Question 2. (5 marks)**

Describe the components of your agent design for DRAGONGAME.

**Question 3. (15 marks)**

Compare the performance of Uniform Cost Search and A\* search in terms of the following statistics:

- a) The number of nodes generated
- b) The number of nodes on the fringe when the search terminates
- c) The number of nodes on the explored list (if there is one) when the search terminates
- d) The run time of the algorithm (e.g. in units such as mins:secs). Note that you can report run-times from your own machine, not the Gradescope servers.
- e) Discuss and interpret these results. If you are unable to implement A\* search, please report and discuss the statistics above for UCS only.

**Question 4. (15 marks)**

Some challenging aspects of designing a DRAGONGAME agent are the asymmetric movement dynamics (moving up behaves differently to moving down), the problem of choosing the order in which to visit and collect each gem, and the large number of available actions.

Design and describe heuristics (or components of a combined heuristic function) that you have developed in the search task that account for these aspects (or any other challenging aspects you have identified) of the problem. Your documentation should provide a thorough explanation of the rationale for using your chosen heuristics (maximum of 5 marks per heuristic).

## 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, 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.
- Please also include a list of references indicating code you have drawn on in your `solution.py` docstring.

**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.

Late submission of the assignment will **not** be accepted. Unless advised, assessment items received after the due date will receive a zero mark unless you have been approved to submit the assessment item after the due date.

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 Progressive Assessment form at least 48 hours prior to the submission deadline.