INFO1110 / COMP9001 Acorn Runner

Acorn Runner: An escape to survive

Milestone Deadline: Friday 15th May, 11:59 pm

Final Deadline: Friday 29th May, 11:59 pm

Weighting: 5% Milestone, 10% Final

> You are an acorn, heir to the Honourable Furious Forest Throne. Your beauty shines like no other, reflecting the colourful solar rays into the eyes of all spectators. But instead of your usual coat of beautiful acorn shell, you find yourself covered in ash. Not only that, but you've fallen great depths from the heights of the towering trees of the Honourable Furious Forest that once cared for you.
>
> You shed a tear of loss. The horrid memories of the Fire Nation's invasion flash before you as you relive the moments your hometown, the Honourable Furious Forest, was burnt to ashes. Your friends, the koalas and kangaroos, your family, the Honourable Furious Forest members, all burnt to a crisp.
>
> Mustering up your motivation for revenge, you, the acorn, heir to the Honourable Furious Forest Throne, stumble forward and find yourself in a maze. You observe walls of fire, helicopter search lights and teleporting pads within.
>
> You cry out and slam your little acorn fist on the greyed Honourable Furious Forest floor of dried up leaves. You swear upon your Father's name, Lord Scarlet Oak of the Honourable Furious Forest, that you will conquer this maze and restore the Honourable Furious Forest back to its former glory of rainbow and sunshine.

Check the Ed pinned posts for a video demonstrating this assignment in action!

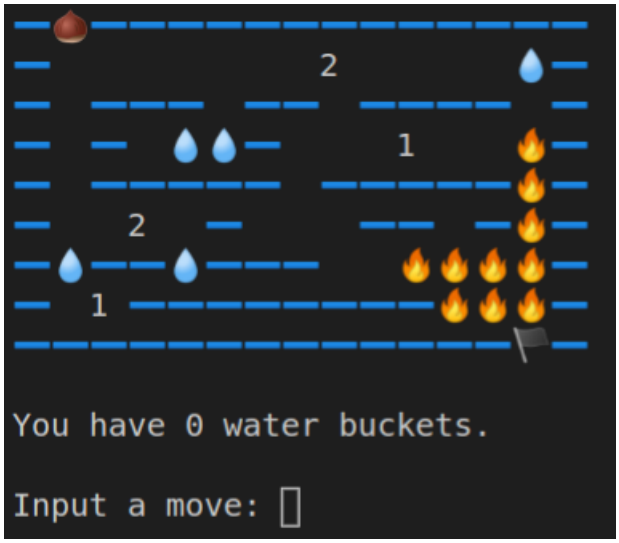
This document may receive updates. Please keep an eye on Ed announcements for any amendments.

Description

In this assignment there will be three parts:

- A game component. You must be able to play the game yourself.
  - The game will be a 2D maze with the objective of moving from start to end.
- A solver component.
  - It should play the game as many times as it needs to generate a successful path.
  - More on this in the Solver section of this specification.
- A report.
  - 3 questions on testing and a short analysis on the solver algorithms involved.

Example screenshot with emojis:



Legend:

- 🌰: The acorn (start)!
- 🏴: Ending/Goal cell
- ➖: Wall cell
- 💧: Water bucket cell
- 🔥: Fire cell
- 1/2: Teleport cell

Unfortunately as Ed doesn't support emojis, your game will have to be in ASCII letters as shown in the sample outputs below!

Cells

Cell character	Meaning
A	Player cell (stands for Acorn)
' '	Air cell (space bar)
X	Starting cell
Y	Ending/Goal cell
*	Wall cells
1, 2, 3, 4, 5, 6, 7, 8, 9	Teleport cells. These numbers will come in pairs. On stepping onto the cell, you enter the cell '1', you teleport to the other '1'. Values greater than 9 will not be given. Note: 0 is not a valid teleport pad!
W	A water bucket cell. On stepping onto the cell, the player gains a water bucket.
F	A fire obstacle that you cannot pass unless you have a water bucket.

Configuration

There will be one txt file which contains an ASCII representation of the maze. The maze may have more than one viable solution. The symbols correspond to the cell characters outlined above. All letters shall be in upper case.

Example configuration file:

```
*X*************
*       2 *   *
* *** ** **** *
* *  W*   1   *
* ***** ***** *
*  2 *   ** *F*
* ** ***  F   *
* 1********   *
*************Y*
```

Commands

Command	Meaning
w	Move up
a	Move left
s	Move down
d	Move right
e	Wait a turn
q	Quit the game

If the user enters an invalid move, print `Please enter a valid move (w, a, s, d, e, q).`.

These commands are case-insensitive!

See the sample outputs below for usage!

Implementation details

Your program will be written in Python 3. The only in-built module methods and attributes you are allowed to use are:

- `sys.argv`
- `sys.exit()`
- `os.system("clear")` <--- (More on this later)

You may not import any other modules.

To help you, a scaffold of a suggested implementation structure is provided. Some test cases require certain features of your code and cannot be modified.

Things you CANNOT modify (these are tested):

File	Function/Attribute	Why it must not be modified
game_parser.py	`parse()`	`parse(lines)` must take in a list of strings and must return a list of lists of cells. This will be tested.
grid.py	`grid_to_string()`	`grid_to_string(grid, player)` must take in a list of list of cells and a player and must return a single string representing the grid & water buckets. This will be tested
cells.py	`display`	The `display` attribute will be used to test your `grid_to_string()` function.
player.py	`display`	Same as above.
player.py	`num_water_buckets`	The `num_water_buckets` attribute will be used to test your `grid_to_string()` function.
player.py	`row`	Specifies their row in the grid.
player.py	`col`	Specifies their col in the grid.

You may modify any other files to your liking. The rest of the scaffold is to help you with some basic structure! You may, of course, import your own modules...! You are also encouraged to write helper functions appropriately.

Note: No file other than `run.py` and `solver.py` should print to the screen! Only `run.py` should print to the screen while playing the game. This is good programming practice, so get used to it early! If needed, you may use it sparingly to debug your program!

game.py

It is recommended you write your game engine here. By writing a `Game` class, you can easily run and delete game instances for your solver. The game engine should hold all the relevant data regarding the game's state. This includes the moves made, the player's position, the cells, etc.

You should call `read_lines()` which uses the `parse()` function to parse the lines in the file. Your `read_lines()` function should return the grid as well. If you decide to do this differently however, that is also ok!

cells.py

It is recommended that you write your cells here. All cells must have a `display` attribute. We recommend your cells also have a `step()` method. The `step()` method should take in a `Game` object (defined above) and make modifications to the game depending on the cell.

If you are a strong programmer and know how to use inheritance, you may use it. Using inheritance will not advantage you much however, and the assignment is completely doable without it! We will not teach it in this course, so it is up to you if you want to use it!

Notable cells:

`Wall` Cell

If a user steps onto a `Wall` cell, the user should be pushed back to their original cell and the message `You walked into a wall. Oof!` should be returned. The game should not record illegal user moves onto walls or out of bounds!

`Water` Cell

If a user steps onto a `Water` cell, it should increment the player's water bucket count and return the message `Thank the Honourable Furious Forest, you've found a bucket of water!`. It should then behave like an `Air` block.

`Fire` Cell

If a user steps onto a `Fire` cell, two things can happen:

- If the user has at least one water bucket, it should return the message `With your strong acorn arms, you throw a water bucket at the fire. You acorn roll your way through the extinguished flames!` and reduce the player's water bucket count by one. The fire block should then behave like an `Air` block.
- If the user does not have a water bucket, it should return the message `You step into the fires and watch your dreams disappear :(.` and end the game.

`Teleport` Cell

If a user steps onto a `Teleport` cell, it should teleport the user to the destination and return the message `Whoosh! The magical gates break Physics as we know it and opens a wormhole through space and time.`.

If a user waits (by inputting `'e'`) on a teleport cell after stepping on one, they will be teleported again.

If the Acorn is currently on a teleport pad and walk into a wall, when the Acorn is pushed back, it does not re-trigger the teleport pad.

player.py

It is recommend that you write your `Player` class here. It must contain the attribute `num_water_buckets`. The `Player` class should also contain a method called `move(...)` which will receive a move command and move the player.

The `Player` class must have a `row` and `col` attribute which represents their location on the grid. The `grid_to_string()` function must use these attributes when drawing the player.

If a Player tries to leave the grid (say there was a hole in the perimeter), it should act as if the player walked into a wall.

game_parser.py

It is recommended that you write your parser functions here. A parser is a module which reads an input and processes it into something useful.

- You may assume that the input grid given in the configuration file is a rectangle (i.e. each row has the same number of entries). You may also assume that the border of the maze given in the configuration file will be surrounded with wall cells (except the starting and ending cells)!
- Your `parse()` function will be tested. It should receive a list of strings. The `parse()` function must be able to handle `\n` characters at the end of each string in the input list. There are certain error cases it must be able to handle. **In this specific order:**
    - If the configuration file contains an unknown letter, raise a `ValueError` with message `Bad letter in configuration file: <letter>.` . If there are multiple unknown letters, you only need to output a single unknown letter in the message. You do not need to find them all and concatenate the output!
    - If the configuration file does not contain exactly one `X`, raise a `ValueError` with message `Expected 1 starting position, got <number>.`, where number is the occurrence of `X`.
    - If the configuration file does not contain exactly one `Y`, raise a `ValueError` with message `Expected 1 ending position, got <number>.`, where number is the occurrence of `Y`.
    - If the teleport pads do not come in exact pairs, raise a `ValueError` with message `Teleport pad <number> does not have an exclusively matching pad.`, where number is the number of the teleport pad. If there are multiple pairs of non-matching pads, you only need to output a single pad number in the message. You do not need to find them all and concatenate the output!

The function `read_lines()` will not be tested. You will need to use this to prepare your data to pass into the `parse()` function. If the file doesn't exist, print `<filename> does not exist!` and exit gracefully.

The order of function calls in this file should look like:

- Call read_lines()
    - Call parse(lines)
    - Return grid
- Return grid

grid.py

It is recommended that you write helper functions for the grid here.

The `grid_to_string()` function must be implemented in this file as it is tested. It should take in a grid (list of lists of `Cells`) and a `Player`. It should **return** the grid and the number of water buckets the player has as a **single string**.

Note: Your returned string should include `\n` characters.

Example contents of the string:

```
*A************
*       2    W*
* *** ** **** *
* * WW*   1  F*
* ***** *****F*
*  2 *   ** *F*
*W**W***  FFFF*
* 1********FFF*
************Y*

You have 0 water buckets.
```

run.py

This file is the entry-point to your game program. It will be run given a single command line argument that represents the filename of the configuration file.

If no command line arguments are provided, print `Usage: python3 run.py <filename> [play]` and exit gracefully.

`run.py` should create a game object, handle user inputs, and handle messages from the game object to display to the user.

You may import `os` and use `os.system("clear")` to clear the screen after each user input. This creates a much better user experience. See the attached demonstration video to see it in action. Ensure that this mode is only activated if the user provides the optional `play` command line argument. This screen clearing functionality is not tested and you may skip this.

solver.py - Solver

This file is the entry-point to your solver program. Your solver should have two modes, Depth First Search (DFS) and Breadth First Search (BFS). These are two maze solving algorithms and the **pseudocode will be provided in the corresponding lab sheet**. Your solver should take in two command line arguments that represents the filename of the configuration file and the mode (DFS or BFS).

If no command line argument is provided, print `Usage: python3 solver.py <filename> <mode>` and exit gracefully.

It should print out the number of moves made as well as the path. There may be more than one path at the shortest path length. These are all accepted solutions. You only need to find and output one such path. The output shall follow the format:

```
$ python3 solver.py <filename> <mode>
Path has <number> moves.
Path: <moves comma delimited>
```

If there are no solutions:

```
$ python3 solver.py <filename> <mode>
There is no possible path.
```

Some tips:

- You should only need to change **one line** to move between DFS and BFS modes!
- Truly understand how these algorithms work. Their advantages, shortcomings, appropriateness, etc.
- Avoid recursion as it is not very efficient and your solves may time out!
- You may want to prohibit your solver from "waiting" more than three turns consecutively.
- You may ask your tutor for assistance in the solver component regarding your algorithm design. Not your code!
- Ed test cases will test both DFS and BFS modes. While the test cases will be as lenient as possible in terms of run-time, If your solver times out, you need to rethink how you approach the problem!
    - The DFS and BFS modes will be given different configuration files as they shine in different scenarios.
    - Your BFS solver must produce an optimal path. There may be multiple optimal paths and these will all be considered correct! (what do you think 'optimal' here means?)
    - Your DFS solver only needs to produce a valid path since there are many paths it can take.

Game Finish

When the player finishes the game successfully by reaching the ending/goal cell, print:

```
You conquer the treacherous maze set up by the Fire Nation and reclaim the
Honourable Furious Forest Throne, restoring your hometown back to its former
glory of rainbow and sunshine! Peace reigns over the lands.

Your made <num of moves> moves.
Your moves: <moves comma delimited>

======================
====== YOU WIN! =====
======================
```

When the player finishes the game unsuccessfully, print:

```
The Fire Nation triumphs! The Honourable Furious Forest is reduced to a pile of
ash and is scattered to the winds by the next storm... You have been roasted.

Your made <number of moves> moves.
Your moves: <moves comma delimited>

======================
===== GAME OVER =====
======================
```

Submission

You will submit your code on two separate assessment pages on **Ed**, a milestone submission assessment page and a final submission assessment page. There will be some public and hidden test cases to guide you, however, be aware that there is a portion of marks allocated to **writing your own test cases**!

Here is a brief checklist of the things you will be required to submit:

Item	Where	Time
Game code Milestone	Ed Milestone submission page	Milestone Deadline
Game code	Ed Final submission page	Final deadline
Solver code	Ed Final submission page	Final deadline
Test code (unit tests and E2E tests)	Ed Final submission page	Final deadline
Report (no more than 500 words as specified above)	Ed Final submission page and **Canvas TurnItIn**	Final deadline

Milestone:

The following will be tested against automated test cases. There is no manual marking component.

- game_parser.py:
  - `parse()`
- grid.py:
  - `grid_to_string()`
- player.py:
  - `num_water_buckets` attribute (int)

Marking breakdown

This assignment is worth 15% of your final mark. The breakdown of this 15% is outlined below.

Criteria	Weighting
Milestone test cases	33%
Final submission test cases	27%
Final submission solver test cases	10%
Final submission student test cases	10%
Code style and structure	10%
Report	10%

All Ed test cases are case sensitive.

Student test cases - 10%

You will be required to write your own test cases. The public test cases are general end-to-end (E2E) tests and are not sufficient! You will be required to write unit tests and end to end tests using input vs output. Refer to the relevant tutorial sheet for more information regarding these techniques.

Further breakdown:

- Unit tests and edge case coverage - 5%
  - Tests should cover edge cases.
  - Unit tests should comprehensively test each module.
  - Tests should be justified.
- End to end tests covering most normal usage - 5%
  - Test cases should be general.
  - Test cases should represent normal usage by a player.
  - Test cases should cover program operation from start to end.
  - E2E test cases should be in a test folder with all the test cases labelled appropriately (e.g. `1.in`, `1.out`, `1.txt`) where the justification is in the corresponding text file.
- You do not need to write tests for your solver.

Code style and structure - 10%

Code style and structure is essential to the success of a coding project. The scaffold provides a basic structure for writing the program, however there will be parts where you need to decide where to write certain functionality. Does the player class make the player teleport? Should it have a `set_position()` method? etc.

Further breakdown:

- Self documenting code - 2.5%

    - The code should largely talk for itself.
    - Avoid complex one liners that others may not understand.
    - Ensure variable names are meaningful.
- Good comments - 2.5%
    - Not too many, not too few.
    - Should compliment code, not act as a band-aid for nonsensical code!
- Layout - 2.5%
    - Code should be spaced out vertically.
    - Code should be spaced out horizontally (`i = 0` is better than `i=0`).
    - `import`s are at the top of the file.
- Structure - 2.5%
    - Modularity: should this be in its own function/method?
    - Program flow is well thought out and makes sense.
    - Code and state are stored in relevant places.
    - Code should not be overly complex if there is a clearly easier way to do it.
    - Side effects should be kept to a minimum.

Hint: You may directly ask your tutor during the lab sessions what they like, and what they don't like to see! They will be the ones manually marking your assignment.

Report - 10%

Please attach your report as a pdf file into the same workspace as your final submission titled `Report.pdf`. There will be a strict word limit of 500 words. Your answers do not need to be long. You may use dot points to list your responses where appropriate. We want content in your answers, not word fluff!

Please also submit your report to TurnItIn via Canvas.

Section 1 - Testing (3% total, 1% each question):

1. List 3 reasons why it is good to write test cases.
2. We have not discussed mocks. Research and briefly explain the use of mocks in testing. List two advantages and two disadvantages of mocking. Where should mocks generally be used? (unit tests or E2E tests?)
3. Give a real life example of where insufficient code testing led to problems. Cite your resource appropriately. If you are unsure how to cite appropriately, use the Harvard referencing standard.

Section 2 - Solver (7%):

You may assume that the starting cell is on the top row and the ending cell is on the bottom row of the map. Qualitatively consider run-time and correctness.

1. What are the strengths and weaknesses of the BFS algorithm? When would you want to use BFS? (2%)
2. What are the strengths and weaknesses of the DFS algorithm? When would you want to use DFS? (2%)

3. What happens if the ending cell is very close to the starting cell? Is DFS guaranteed to be faster than BFS? (1%)
4. In normal BFS and DFS algorithms, there is usually a list of visited cells that help the algorithm perform faster by not re-visiting a cell. However, this game setup has a certain feature in it that does not allow you to use this. What feature of the game stops you from using a list of visited cells to improve run-time? Justify briefly by giving a simple example. (Hint: Think scenarios where you have to re-visit a cell to complete the game. What about you changed since the previous time you were there?) (2%)

FAQ

Where do I start!?

After reading through this sheet, you are recommended to start by tackling the milestone components!

I need help!

You are strongly encouraged to ask questions about the assignment to your tutors and on the discussion board, Ed. Tutors will be able to support you through the difficult sections of this assignment. Please read this assignment description carefully before you ask questions though!

DO NOT POST YOUR CODE (including small snippets) PUBLICLY. You will be penalised. You can post some code if it is made private and only view-able by tutors. However, limited help will be offered for the implementation of this assignment.

The solver looks too difficult! How can we do this as first years!?

- You can easily achieve a distinction this assignment if you completely ignore the solver component. If you want an HD, you have to earn it!
- The solver component was designed to be do-able for first year students who thoroughly understand procedural programming. Believe in yourself! You can do it!

Advice?

- Start early. You can complete 90% of this assignment once you take the week 9 lab.
- Develop on your own local environment. It's much faster and efficient to do so!
- Write tests as you write your modules (or even before)!

Can I use `for` and `in` in the assignment?

Yes!

Sample outputs

Simple Win

```
### Config ###
```

```
**X**
*   *
**Y**

### Output ###
**A**
*   *
**Y**

You have 0 water buckets.

Input a move: s
**X**
* A *
**Y**

You have 0 water buckets.

Input a move: s
**X**
*   *
**A**

You have 0 water buckets.


You conquer the treacherous maze set up by the Fire Nation and reclaim the
Honourable Furious Forest Throne, restoring your hometown back to its former
glory of rainbow and sunshine! Peace reigns over the lands.

Your made 2 moves.
Your moves: s, s

=====================
====== YOU WIN! =====
=====================
```

Simple Lose

```
### Config ###
*X*
*F*
*Y*

### Output ###
*A*
*F*
*Y*

You have 0 water buckets.

Input a move: s
*X*
*A*
*Y*
```

```
You have 0 water buckets.


You step into the fires and watch your dreams disappear :(.

The Fire Nation triumphs! The Honourable Furious Forest is reduced to a pile of
ash and is scattered to the winds by the next storm... You have been roasted.

Your made 1 moves.
Your moves: s

=====================
===== GAME OVER =====
=====================
```

Solver - No path found

```
### Config ###
*X*
*F*
*Y*

### Output ###
There is no possible path.
```

Quit game

```
### Config ###
**X***
*    *
* ** *
*    *
*    *
****Y*

### Output ###
**A***
*    *
* ** *
*    *
*    *
****Y*

You have 0 water buckets.

Input a move: s
**X***
* A  *
* ** *
*    *
*    *
****Y*

You have 0 water buckets.
```

```
Input a move: q

Bye!
```

Walking into a wall

```
### Config ###
***
X Y
***

### Output ###
***
A Y
***

You have 0 water buckets.

Input a move: d
***
XAY
***

You have 0 water buckets.

Input a move: w
***
XAY
***

You have 0 water buckets.

You walked into a wall. Oof!

Input a move: s
***
XAY
***

You have 0 water buckets.

You walked into a wall. Oof!

Input a move: d
***
X A
***

You have 0 water buckets.


You conquer the treacherous maze set up by the Fire Nation and reclaim the
Honourable Furious Forest Throne, restoring your hometown back to its former
glory of rainbow and sunshine! Peace reigns over the lands.
```

```
Your made 2 moves.
Your moves: d, d

=====================
====== YOU WIN! =====
=====================
```

Water and Fire interactions

```
### Config ###
*****
XWFFY
*****

### Output ###
*****
AWFFY
*****

You have 0 water buckets.

Input a move: d
*****
XAFFY
*****

You have 1 water bucket.

Thank the Honourable Furious Forest, you've found a bucket of water!

Input a move: d
*****
X AFY
*****

You have 0 water buckets.

With your strong acorn arms, you throw a water bucket at the fire. You acorn
roll your way through the extinguished flames!

Input a move: d
*****
X  AY
*****

You have 0 water buckets.


You step into the fires and watch your dreams disappear :(.

The Fire Nation triumphs! The Honourable Furious Forest is reduced to a pile of
ash and is scattered to the winds by the next storm... You have been roasted.

Your made 3 moves.
Your moves: d, d, d
```

```
====================
===== GAME OVER =====
====================
```

Teleport cells

```
### Config ###
*****
X1 1Y
*****

### Output ###
*****
A1 1Y
*****

You have 0 water buckets.

Input a move: d
*****
X1 AY
*****

You have 0 water buckets.

Whoosh! The magical gates break Physics as we know it and opens a wormhole
through space and time.

Input a move: d
*****
X1 1A
*****

You have 0 water buckets.


You conquer the treacherous maze set up by the Fire Nation and reclaim the
Honourable Furious Forest Throne, restoring your hometown back to its former
glory of rainbow and sunshine! Peace reigns over the lands.

Your made 2 moves.
Your moves: d, d

====================
====== YOU WIN! =====
====================
```

Solver - BFS

```
### Config ###
*X**********
*       2 *   *
* *** ** **** *
* *  W*   1   *
```

```
* ***** ***** *
*  2 *   ** *F*
* ** ***  F   *
* 1********F  *
*************Y*

### Output ###
Path has 32 moves.
Path: s, d, d, d, d, s, s, a, d, w, w, d, d, d, d, a, s, s, d, d, a, d, d, d, d,
s, s, s, a, s, d, s
```

Solver - DFS

```
### Config ###
*X*************
*       2    W*
* *** ** **** *
* * WW*   1  F*
* ***** *****F*
*  2 *   ** *F*
*W**W***  FFFF*
* 1********FFF*
*************Y*

### Output ###
Path has 60 moves.
Path: s, d, d, d, d, d, d, d, d, a, d, d, d, d, d, a, d, s, s, a, a, a, a, w, s,
d, a, a, w, w, d, a, a, a, a, s, s, a, d, w, w, d, d, d, d, s, w, a, d, d, d, d,
d, s, s, s, s, s, s, s
```

Academic declaration

By submitting this assignment, you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the Student Plagiarism: Coursework Policy and Procedure can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgment from other sources, including published works, the Internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Computer Science, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and/or communicate a copy of this assignment to a plagiarism checking service or in-house computer program, and that a copy of the assignment may be maintained by the service or the School of Computer Science for the purpose of future plagiarism checking.