JiggaJigga 2025-26: Rising Tide

SUMMARY

In *Rising Tide*, you protect your settlement on an island from rising sea levels by pushing land around to construct seawalls. How long can you stay above the waves?

GAMEPLAY

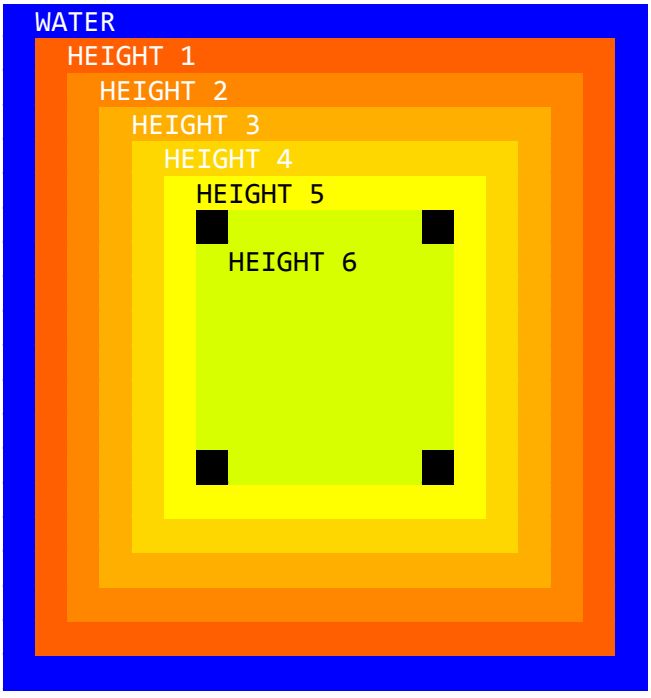
The game takes place on a 18x18 square grid, representing an island. Each of the 4 players has a settlement on this island. Over time, parts of the island may flood. If your settlement floods, you are eliminated.

Each location on the island, as long as it is not flooded, has an integer height between 0 and 8, inclusive. Each move consists of pushing land from one cell to a neighboring cell, including diagonally.

GAME START

At the start of the game, the terrain heights are as follows: All cells directly touching the edge have height 1, all cells 1 unit inland have height 2, and so on, until the tiles 5 units or more inland all have height 6. However, any settlement has height 0.

This map shows the starting terrain heights. The black squares are the settlements.



SETTLEMENTS

Each player has a single settlement on a cell near a corner of the grid, 5 units from the edge. This means they are at the locations (5, 5), (5, 12), (12, 12), and (12, 5).

Note that these coordinates, and all other coordinates used in Rising Tide, are given in zero-indexed (i, j) formats, meaning that they start counting at 0. The first coordinate is how many units down you are from the top edge, and the second coordinate is how many units right you are from the left edge.

MOVING

Each round, each player submits a move, which consists of a source cell and a destination cell. 1 height-unit of land on the source will be moved to the destination, effectively adding 1 height unit to the destination and subtracting 1 from the source.

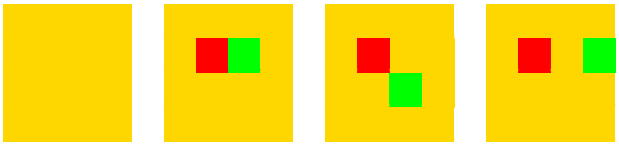
A move must satisfy the following requirements to be legal:

1. Neither the source nor the destination may be flooded.
2. The source may not already be at height 0, nor may the destination already be at height 8.
3. The source and destination must be neighbors, including diagonally.
4. Neither the source nor the destination may be the location of any player's settlement.

Each round, all players submit their moves, which are all checked for legality, and then executed in random order. If, at the time of execution, a move is no longer legal, it is skipped. A player may also pass their turn.

Here are some examples of legal and illegal moves. Yellower/greener cells are higher, so after a move, the source gets redder and the destination gets yellower/greener.

START LEGAL LEGAL ILLEGAL



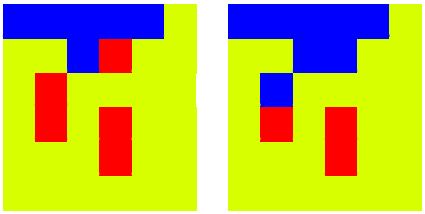
FLOODING

After all players have made their move, flooding occurs.

The water level starts at a height of 0. Each round, it increases by 0.01. During flooding, any cells that have a height below the current water level and are next to (including diagonally) any flooded cells (or the edge of the grid) will flood.

Here is an example of flooding. In this case, red cells are below the water level, green cells are above the water level, and blue cells are flooded. Note that only the low cells adjacent to flooded cells become flooded.

BEFORE AFTER



ELIMINATION

If the cell containing a player's settlement floods, they are immediately eliminated from the game.

SCORING

After all but one player has been eliminated, or 800 rounds have passed, the game will end. Each player is assigned a score as follows:

1. If a player has been eliminated, their score is the number of rounds they survived.
2. If a player survives all 800 rounds, they get a score of 800.
3. If they are the last player standing, they get a score of 800.

During the tournament, players will be ranked by their average score across all games they played in.

MAKING A BOT

To make a bot, you first write the bot's code, and then specify its other details.

Your bot's code will be a function taking in two arguments and returning its move. The first line will look something like this:

```
def my_bot_fn(terrain, round_num):
    ...
```

`terrain` will be a list of lists of integers. `terrain[i][j]` will be -1 if cell (i, j) is flooded, otherwise it will be the height of the island's terrain at that cell. Note that `terrain` will be

rotated such that you are always the player in the upper left corner, with your settlement at (5, 5). `round_num` is the current round number.

Your bot should return a tuple of tuples of integers ((i1, j1), (i2, j2)) representing pushing land from (i1, j1) to (i2, j2). You may also return `None` to pass your turn.

To specify the rest of your bot, import the `make_bot()` function from `util.py` and call it on a name, two-letter-code, and your function, like this:

```
import util
my_bot = util.make_bot(“MyBot”, “MB”, my_bot_fn)
```

Finally, to run a game, import the `run_game()` function from `display.py` and call it on the bots in the game, like this:

```
import display
display.run_game(
    [my_bot, random_bot, random_bot, random_bot],
    mode=“terminal”,
)
```

The tournament configuration will be pretty much the exact same. It’ll import all your bots, import `display`, and call `display.run_game()`.

It’s totally fine if your bot has some form of memory, such as global variables.

LOGGING

Because the display is in the terminal by default, logged events will be written to a file called `log.txt`, in the same folder as all the other code. Logged events include bots throwing errors, taking too long, or returning invalid moves. The log will also record bots being eliminated or the game ending.

To see `log.txt` updated in real-time, open a new terminal, navigate to the folder, and enter “`tail -f log.txt`”. When the game ends, exit by hitting ctrl-c.

HELPER FUNCTIONS

We have created a few helper functions to assist you in making your bot. Find them in `util.py`.

SAMPLE BOTS

We've provided for you 6 sample bots this time! Each one is named after a coastal city for absolutely no reason whatsoever.

`pass_perth`: always does nothing
`random_rizhao`: selects a random legal move
`nearby_new_orleans`: builds a small mound near itself
`trench_tunis`: digs trenches through other bots' walls
`border_barcelona`: raises walls on existing coastlines
`level_los_angeles`: flattens the region just around itself

COMPATIBILITY & DISPLAY

The `NumPy`, `urwid`, and `Pygame` libraries are required for the game to run.

Rising Tide can run in one of two display modes: Terminal and Pygame. To set the display mode, change the mode parameter in `display.run_game()`. Selecting "terminal" will use the terminal mode and selecting "pygame" will use the pygame mode.

Terminal mode will display the game in the terminal. This may not work in every terminal/IDE, but the built-in MacOS terminal (aka terminal.app) works.

Pygame mode will open a pygame window and display the game in that window.

LIBRARIES

Feel free to use any libraries which can be installed using PIP. Examples:

1. `NumPy`
2. `SciPy`
3. `PyTorch` (if anyone pulls this off, we'll be very impressed)

SOME FINAL WORDS

There will be two divisions, a standard division and a hacker division. The standard division will be the normal main competition. The hacker division only has two rules:

1. Don't damage my computer. E.g. consume extreme amounts of CPU, mess with the configuration or filesystem, hack it, etc.
    a. Specifically, all your actions must be cleared by the time Python shuts down, and none of your actions may make the Python shutdown harder.
2. Win.

Do note that we have intentionally left a security flaw in the game. See if you can find it and come up with a creative way to make use of it!

Download the game files from this github repository.

A preliminary round will be held towards the end of block 2 so participants get to practice and see what other strategies people have, and the final competition will be held during the first week of the January intersession. To submit bots, just send 'em by email to lemonkat & admiralpav

In terms of strategy, note that you aren't required to cooperate. But no one will mess up your plans if they are gone, so you may want to eliminate other players to build your walls in peace.

This game is a metaphor, alright.

Have fun!

- lemonkat & admiralpav

APPENDIX: DisplayGrid

Rising Tide is built off of the DisplayGrid framework, which I (lemonkat) have been working on for the past two years, and with the release of Rising Tide, I'm putting it out there if anyone else might want to make use of it—just copy the display_grid folder into your own projects.

DisplayGrid is a tool for making terminal UI applications—not just displays like JiggaJigga, but also interactive tools and games. Literally anything you can make from ASCII art, you can draw with DisplayGrid. DisplayGrid can also detect key presses and mouse clicks, even scrolling, and lets you format your display text with bolding, italicization, or underlining. I won't go into all the details here, but the code is all very well-documented, so if you have some coding experience you should be able to figure it out.

You can find a GitHub repository for DisplayGrid, including testing code, here.