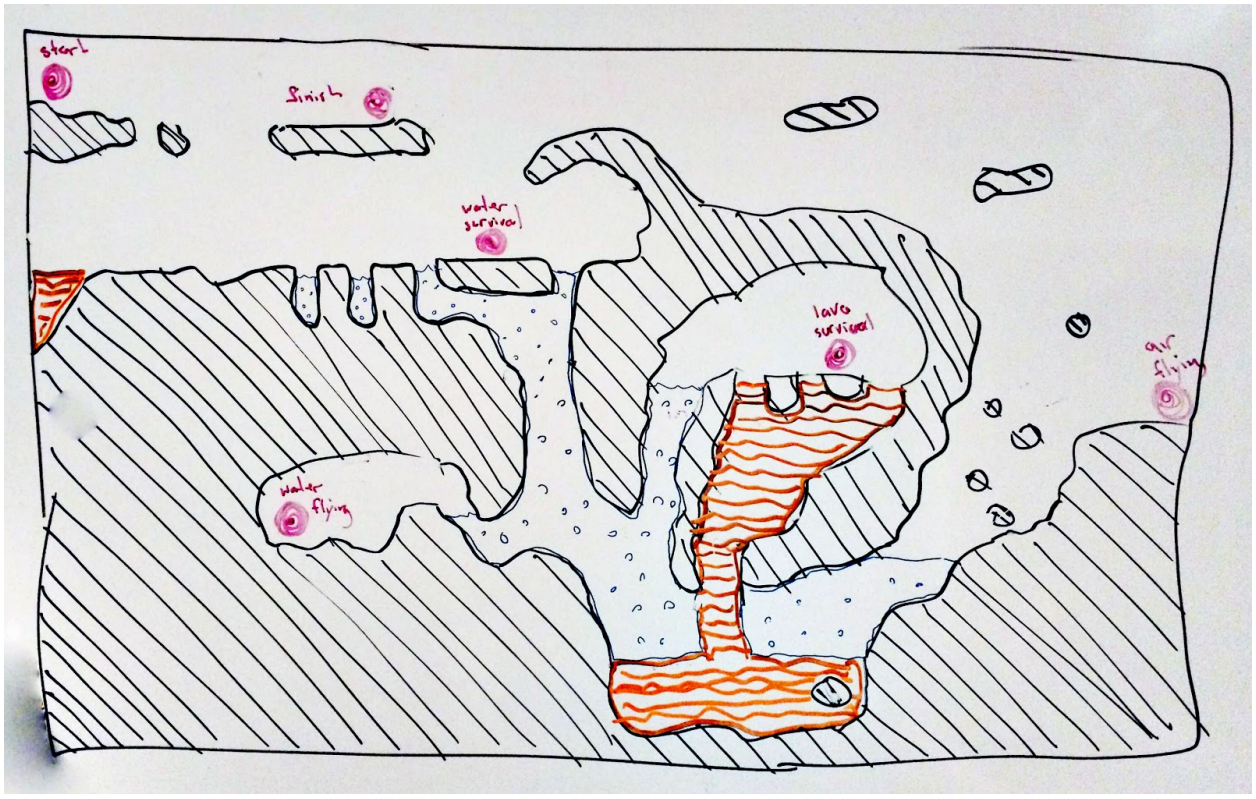
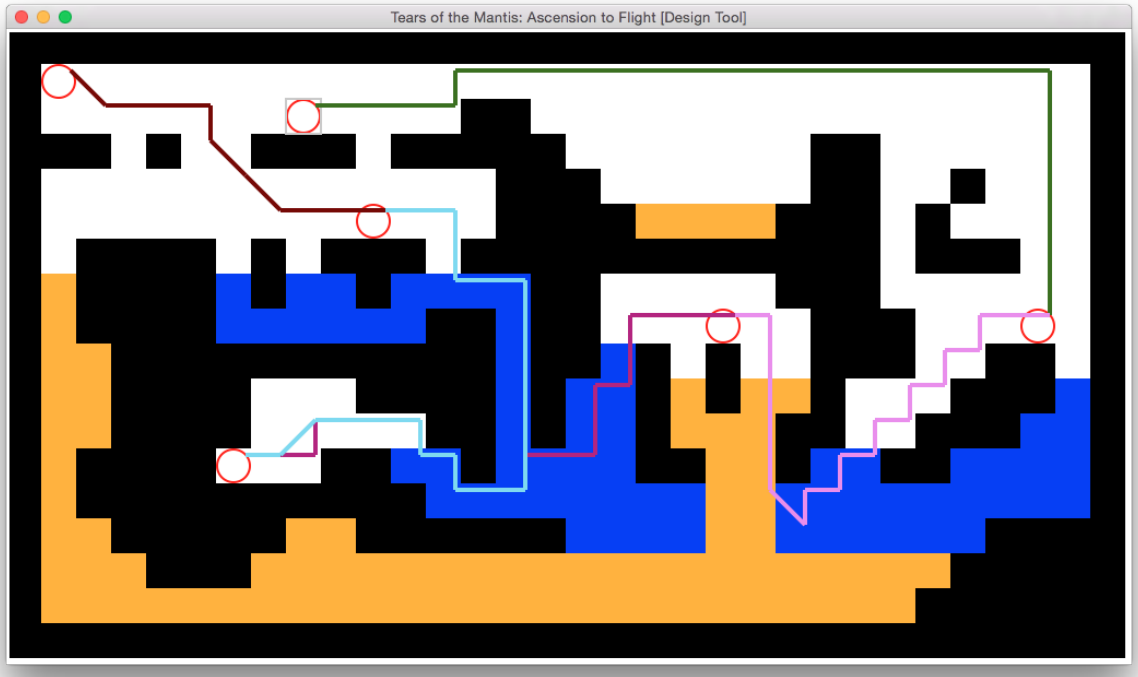
P6: Reachability Analysis Visualization





Assignment Overview

In this assignment, you will build an **AI-assisted level design tool**. Two major components will be provided for you: the level editor (with pixel paint functionality), and the game simulator (exposing a Simulator class with get_initial_state() and get_next_state() methods). You will implement the back-end analysis component which inspects the current state of the level design and decides which lines to draw as the inspects different cells.

Game Design

The hypothetical game being modeled (*Tears of the Mantis: Ascension to Flight* -- you play as a mantis this time) captures the non-combat elements of a minimal Metroidvania game. In AtF, the world is composed of a grid of elemental tiles: earth, air, water, and fire. Initially, the player character may only traverse air tiles while supported below by an earth tile (allowing for jumps that span a gap size of one and horizontal air control). As more abilities are unlocked, the character gains resistance to water, the ability to swim in water, resistance to fire, and eventually the power of unrestricted flight through air tiles.

Base Code

Files:

- **p6_tool.py**: the level editor (provided, depends on p6_analysis.py)
- **p6_analysis.py**: the level analyzer (*not provided*, depends on p6_game.py)
- **p6_game.py**: the game simulator (provided)

Usage:

```
$ python p6_tool.py map.txt
```

Download:
https://drive.google.com/a/ucsc.edu/folderview?id=0B-PPiU3Ga8Z7fjlXVnBKMTZKYzRXN2VkeEdTNGdIeDlQenZwX0M3OWRzUnNhLXFJdlZscWM&usp=sharing

Analysis Engine

You should implement *p6_analysis.py*. This Python module should expose two functions: *analyze* and *inspect*. Here's a starter snippet you can copy:

```
from p6_game import Simulator

ANALYSIS = {}

def analyze(design):
    sim = Simulator(design)
    # TODO: fill in this function, populating the ANALYSIS dict
```

```
    raise NotImplementedError

def inspect((i,j), draw_line):
    # TODO: use ANALYSIS and (i,j) draw some lines
    raise NotImplementedError
```

p6_analysis.analyze(design)

This function will be called immediately after the user makes any change to the level design.

The only parameter, *design*, is a dict containing information loaded from the map file named on the command line. Your code does not need to inspect this object. Instead, use this dict to construct a *Simulator* object (implementation provided in p6_game.py).

Example usage of p6_game.Simulator:

```
from p6_game import Simulator

sim = Simulator(design)
init = sim.get_state_initial_state()
moves = sim.get_moves()
next_state = sim.get_next_state(init, moves[0])

position, abilities = next_state # or None if character dies
i, j = position
```

Use breadth-first search to exhaustively explore the state space of the game, recording information about which states are reachable and how they may be reached. Store this information in a global (module-level) variable. In the terminology we have used for past projects, saving the *prev* table from breadth-first search would be sufficient.

At the scale of the low-resolution example map provided exhaustive analysis should not incur any noticeable delay into the painting interaction.

p6_analysis.inspect((i,j), draw_line)

This function will be called as the user moves their mouse over different tiles in the editor. The i,j pair provided has already been mapped from pixel coordinates to map coordinates, so you may use this information to figure out which map location the user is trying to inspect.

Your implementation of inspect() should make use of the provided draw_line() function to draw annotations on top of the level design. This function takes between two and four arguments:

- **draw_line(src, dst, offset_obj=None, color_obj=None)**
    - src: an (i1,j1) tile location

    - dst: an (i2,j2) tile location
    - offset_obj: (optional) If provided, the hash of this object will be used to draw the line with a random position offset (this makes multiple overlapping lines visible). In the reference solution, the player character's abilities at the end of a solution are used as the offset_obj.
    - color_obj: (optional) If provided, the hash of this object will be used to draw the line with a random color (this makes identifying common segments of distantly lines easier). In the reference solution, the player character's current state (at this point in the trace) is used as the color_obj.

The lines drawn by your implementation of inspect should visualize the different cases in which the player character may reach the inspected tile. We are not interested all possible geometric paths to a cell (this can be explored by mousing around the map). Instead, we are interested in the sets of abilities the player character may have when reaching that point (analyzing sequence breaking).

If your analyze() function is based on breadth-first search, your visualization will naturally show shortests paths. Shortest paths are not required (it would be acceptable to use depth-first search in analyze() as well).

Your inspect() function should respond quickly enough to allow natural painting interactions without noticeably waiting for analysis or visualization to

Requirements:

(Equal points for each requirement.)

- **Get reachability checking working:** After each design edit, determine if the the point marked by Special 5 (see details in the level file format below) is reachable with any combination of character abilities, and display this information somewhere (console text is acceptable). This report may be produced in analyze() if you want. You may skip this requirement if your tool provides this same information for any inspected tile.
- **Get inspection working:** As the user mouses over each tile, report whether the inspected tile is reachable with any combination of character abilities. The report may be done in text or with on-screen lines.
- **Get single-line drawing working:** As the user mouses over each tile, visually report how the inspected tile may be reached with any combination of character abilities.
- **Get multiple-line drawing working:** As the user mouses over each tile, visually report how the inspected tile may be reached for *every feasible combination* of character abilities.
- **Design a custom level:** Invent a new level design in which *fire_survival* must be acquired before *water_survival* and that (like in the example map) all abilities are required to reach the end. Be prepared to use your visualization to demonstrate that these emergent properties are indeed the case. (If you want to move the Special tiles, you'll need to edit the map file by hand.)

Level File Format

This format is parsed automatically by the provided level editor code. However, you may wish to modify the example map (map.txt).

Whitespace within a line is ignored. The design is assumed to be rectangular, with the number of cells on the first line representative of every subsequent line.

Most cells are either E, W, A, or F, signifying *earth*, *water*, *air* or *fire* tiles.

Cells with digit characters (0-9) are considered *air* tiles with special properties (quintessence?). Here is how the special numbers are interpreted in the game mechanics:

- 0: initial location of the player character with no special abilities
- 1: touch this tile to gain *water_survival*
- 2: touch this tile to gain *water_flying* (swimming)
- 3: touch this tile to gain *fire_survival*
- 4: touch this tile to gain *air_flying*
- 5: the nominal goal for the player character (not used, but you can test with it)
- 6-9: not used

```
E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E
E 0 A A A A A A A A A A A A A A A A A A A A A A A A A A A A A A E
E A A A A A A A 5 A A A A E E A A A A A A A A A A A A A A A A A E
E E E A E A A E E E A E E E E E A A A A A A A E E A A A A A A E
E A A A A A A A A A A A A A E E E A A A A A A E E A A E A A A E
E A A A A A A A A A 1 A A A E E E E F F F F E E E A E A A A A E
E A E E E E A E A E E E A E E E E E E E E E E E E E A E E E A A E
E F E E E E W E W W E W W W W E E A A A A A E E E A A A A A A E
E F E E E E W W W W W W E E W E E A A A 3 A A E E E A A A 4 A E
E F F E E E E E E E E E E E W E E W E A E A A E E E A A E E A E
E F F E E E E A A A E E E E W E W W E F E F F E A A A E E E W E
E F F E E E E A A A A A E E W E W W E F F F E E A A E E E W W E
E F E E E E 2 A A E E W W E W W W W E E F F E W W E E W W W W E
E F E E E E E E E E E E E W W W W W W W W F F W W W W W W W W E
E F F E E E E E F F E E E E E E W W W W F F W W W W W W E E E E
E F F F E E E F F F F F F F F F F F F F F F F F F F F F E E E E E
E F F F F F F F F F F F F F F F F F F F F F F F F F F F E E E E E
E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E E
```