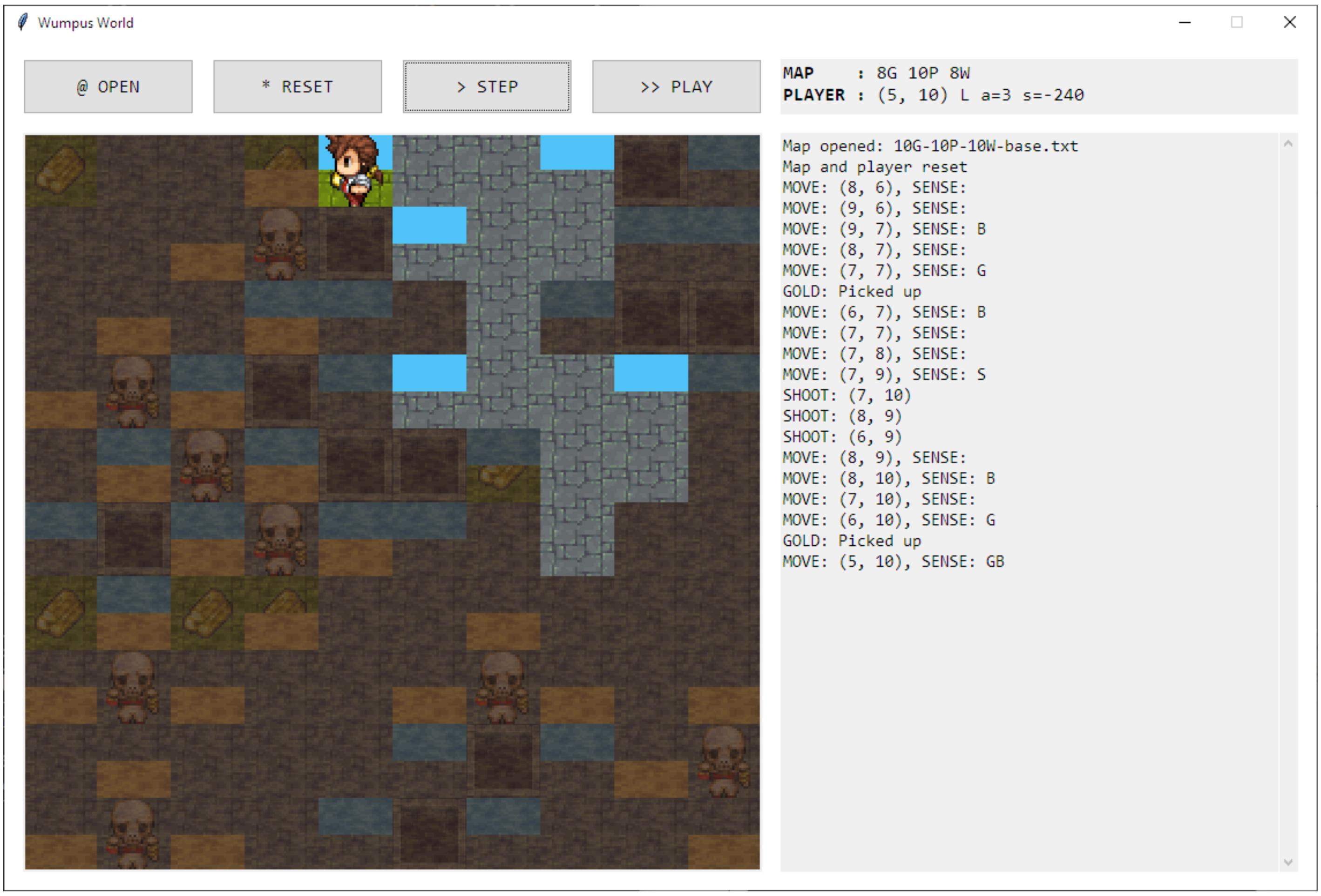


```
MAP    : 8G 10P 8W
PLAYER : (5, 10) L a=3 s=-240
```

```
Map opened: 10G-10P-10W-base.txt
Map and player reset
MOVE: (8, 6), SENSE:
MOVE: (9, 6), SENSE:
MOVE: (9, 7), SENSE: B
MOVE: (8, 7), SENSE:
MOVE: (7, 7), SENSE: G
GOLD: Picked up
MOVE: (6, 7), SENSE: B
MOVE: (7, 7), SENSE:
MOVE: (7, 8), SENSE:
MOVE: (7, 9), SENSE: S
SHOOT: (7, 10)
SHOOT: (8, 9)
SHOOT: (6, 9)
MOVE: (8, 9), SENSE:
MOVE: (8, 10), SENSE: B
MOVE: (7, 10), SENSE:
MOVE: (6, 10), SENSE: G
GOLD: Picked up
MOVE: (5, 10), SENSE: GB
```

Project 02: Wumpus World

A project for `CSC14003` "Introduction to Artificial Intelligence" @ 18CLC6

About this project

- **Collaborators**:
    - `18127221` **Bùi Văn Thiện** (@84436):
      GUI, Maps, Game controller
    - `18127231` **Đoàn Đình Toàn** (@t3bol90):
      Agent/Player/Logic core, Game controller, Bugfixes
- **Software stack** used in this project:
    - Language: Python3
    - External libraries: `Tkinter` (GUI), `python-SAT/Glucose3` (Logic)
    - IDE: VSCode
- **Progress** (on the scale of `1.00`):
    - `1.00` Main + GUI (Tkinter)
    - `1.00` Map
    - `1.00` Map utilities (generator, checker)
    - `1.00` Game controller
    - `1.00` Player/Logic core

Note: the word "Agent" and "Player" will be used interchangably in this report from this point onward.

Quick start

1. **Prepare**
   Make sure all external libraries are installed.
2. **Run**
   Run `main.py`
3. **Load map**
   The main GUI window will open with no maps preloaded. To load a map, click `@ OPEN`.
4. **Control**
   `> STEP` or `>> PLAY` through the map, then `* RESET` or open another map once the game is over.

`main.py`

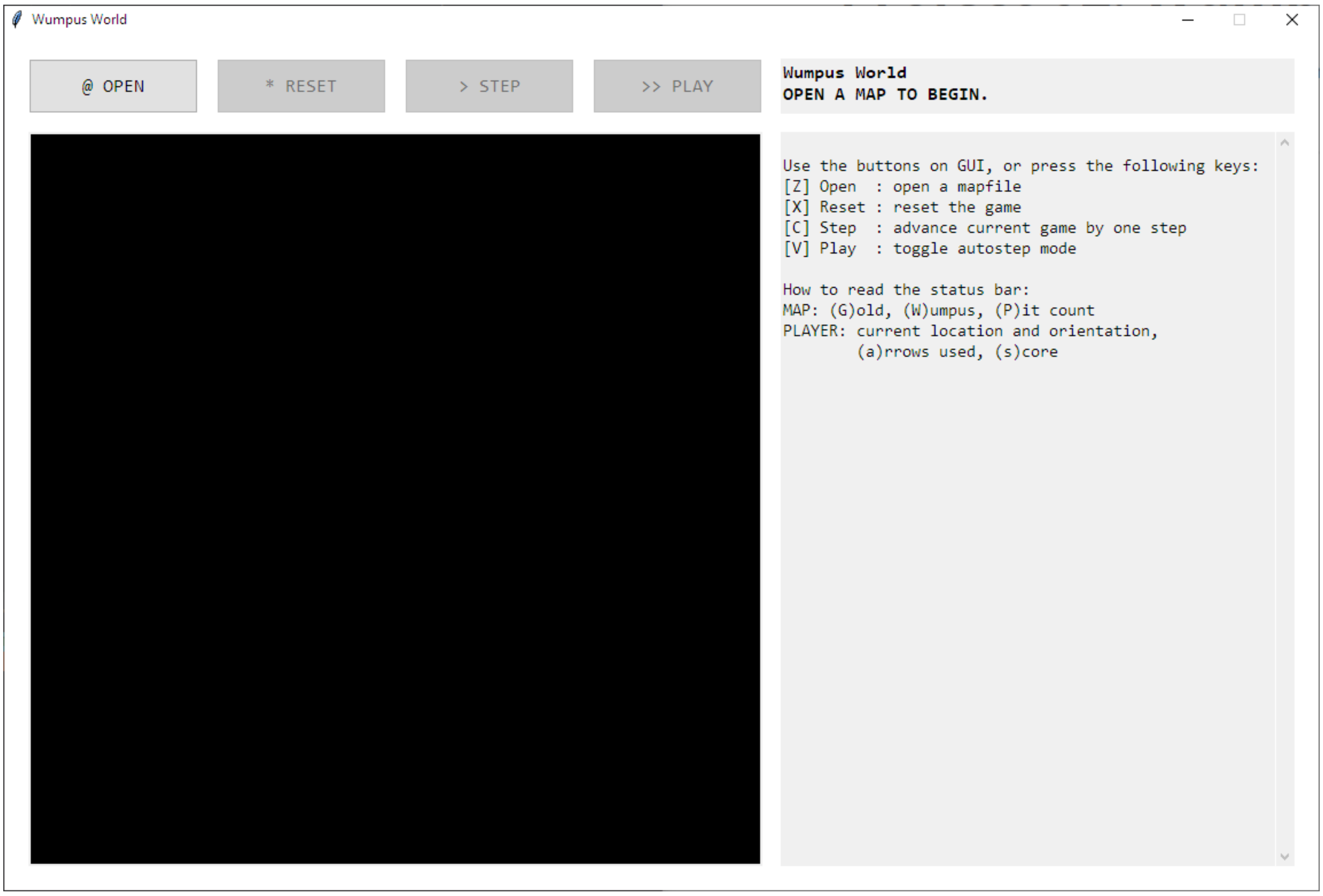
This file...

- **is the starting point of the program.** It creates an instance of **Controller** and `.start()` it.
- provides parameters ("configurables") for tweaking the program, constants (for layout/geometry) and messages for GUI, and directory paths, as following

Parameter	Default value	Description
`ASSET_THEME`	`thiagodnf`	Theme/Set of textures used for map drawing. This should be a folder name in the asset directory.
`TILES_SHOW_HIDDEN`	`True`	If `True`, the "hidden tile" texture with partial transparency (`tile_hidden_alpha80.png`) will be loaded instead of the usual hidden tile. This helps show
`WINDOW_TITLE`	`Wumpus World`	Window title
`FONT`	`Consolas`	Font used in the whole GUI. `Consolas` is a default monospace font available in Windows.
`AUTOSTEP_DELAY`	`100`	Delay between steps in "Autoplay mode", activated by `>> PLAY` button.
`KEYB_OPEN`	`z`	Keyboard shortcut for `@ OPEN`
`KEYB_RESET`	`x`	Keyboard shortcut for `* RESET`
`KEYB_STEP`	`c`	Keyboard shortcut for `> STEP`
`KEYB_AUTOSTEP`	`v`	Keyboard shortcut for `>> PLAY`
`BASE_DIR`	{see `main.py`}	Base directory. This directory is expected to contain both `MAPS_DIR` and `ASSETS_DIR`.
`MAPS_DIR`	`maps/` from `BASE_DIR`	Directory for containing mapfiles.
`ASSETS_DIR`	`assets/{theme}` from `BASE_DIR`	Directory for containing assets for a theme (specified by `ASSET_THEME`) used in map drawing.
`ONBOARDING_MSG`	{see `main.py`}	Tuple of 2 strings used as welcome message on the GUI on program's first launch.

`gui.py`: GUI

This module provides a GUI for this program.



```
Wumpus World
OPEN A MAP TO BEGIN.
```

```
Use the buttons on GUI, or press the following keys:
[Z] Open  : open a mapfile
[X] Reset : reset the game
[C] Step  : advance current game by one step
[V] Play  : toggle autostep mode

How to read the status bar:
MAP: (G)old, (W)umpus, (P)it count
PLAYER: current location and orientation,
        (a)rrows used, (s)core
```

> Figure: Wumpus World GUI on first launch

The GUI is divided into 4 main parts. In clockwise order:

- **Status**: Show the current status of map and player.
  - Status of map: number of Gold, Wumpus and Pit left
  - Status of player: current location, orientation, number of arrows used, and score
- **Log**: Log simple map- and player-related events.
  - This log will be cleared on every map open.
- **Canvas**: Show a graphical representation of the current map.
  - Currently it only shows maps of size 10×10 maximum.
- **Buttons**: Buttons for controlling the game.
  - `@ OPEN` (default shortcut key: `Z`): open a mapfile.
  - `* RESET` (default shortcut key: `X`): reset the current map and player.

    - `> STEP` (default shortcut key: `C`): advance the current game state by one step.
    - `>> PLAY` (default shortcut key: `V`): toggle autostep mode.

This module also provides methods for manipulating part of the GUI:

- `map_open_dialog` for calling the system native "Open file" dialog to choose a mapfile
- `status_update` and `canvas_update` for updating the status and canvas based on the current map.
- `log_write` and `log_clear` for writing/clearing the log.
- `game_*` for reflecting the current/corresponding game state onto the GUI.

A number of callbacks and a "location invert helper" (`loc_invert_helper`) is needed to properly initialize this module.

- The list of callbacks are mentioned in the **Controller** description below.
- The coordinates of items in map is represented as an index of a 2D array, with `(0,0)` as the top–left corner, and `(9,9)` as the bottom–right corner (assuming a 10×10 map.) In order to be compliant to the program requirements given in the original problem statement (`(1,1)` at **bottom**-left and `(10,10)` at **top**-right), `loc_invert_helper` is needed to "invert" the coordinates shown on the screen.

One notable thing is that the Log *does not* provide KB deltas and everything underneath the logic core; they are instead shown in a separate terminal.

`map*.py`: Map & map utilities

About `map.py`

This module (`map`) provides methods for...

- Maps: parsing, storing (in simple 2D array of strings) and manipulating based on player's action
- Players: stats (current location, numbers of arrows used and scores), scoring rules and actions to manipulate map (look around, move, shoot, grab gold, and leave the cave)

About utilites

Two utilities are provided to ease map designing and fixing. They are standalone and do not require any modules in the main program (not even Map.)

- Map generator (`map_generator.py`) randomly generates maps based on a given map size (currently fixed to 10) and number of items (pit/wumpus/gold) in the map.
- Map checker (`map_checker.py`) fix a given mapfile to make it readable to the Map module.

Map format

There are two map formats (actually three) supported in this program.

- The original format as given by the problem description: each empty tile is denoted by a hyphen `-`.
- `compact`: each empty tile has nothing in between.
- `readable`: paddings using hyphens `-` are added to each empty tiles so the width of each tile appeared in the text file is the same. This makes the mapfile itself more human-readable.

Map generator provides a parameter, `gen_format`, for choosing between this "new" format (`readable`) and the original format (`compact`). Map checker, however, currently enforces the new `readable` map, with no overriding parameters.

To demonstrate, here's a sample map (`Figure1.txt`) in 3 format:

`10`	`10`	`10`
`G.G..B.P.B....`	`G.G.-.B.P.B.-.-.-.`	`-G.-G.--.-B.-P.-B.--.--.--.--`
`G.G.S.B.P.B..G..`	`G.G.S.B.P.B.-.G.-.`	`-G.-G.-S.-B.-P.-B.--.-G.--.--`
`.SB.W.SB.P.B....`	`.SB.W.SB.P.B.-.-.-.`	`--.SB.-W.SB.-P.-B.--.--.--.--`
`B.P.SB..B.....`	`B.P.SB.-.B.-.-.-.-.`	`-B.-P.SB.--.-B.--.--.--.--.--`
`P.B........`	`P.B.-.-.-.-.-.-.-.`	`-P.-B.--.--.--.--.--.--.--.--`
`B.....B....`	`B.-.-.-.-.B.-.-.-.`	`-B.--.--.--.--.-B.--.--.--.--`
`....B.P.B...`	`.-.-.-.B.P.B.-.-.`	`--.--.--.--.-B.-P.-B.--.--.--`
`.....B.G.B..`	`.-.-.-.-.B.G.B.-.`	`--.--.--.--.--.-B.-G.-B.--.--`
`......B.P.B.`	`.-.-.-.-.-.B.P.B.`	`--.--.--.--.--.--.-B.-P.-B.--`
`A.......B..`	`A.-.-.-.-.-.-.B.-.`	`-A.--.--.--.--.--.--.-B.--.--`
Compact	Original	Readable

All maps provided in `maps/` are in the `readable` format.

`controller.py`: Controller

This module acts as a middle layer gluing **GUI**, **Map** and **Agent** together. It does the following things:

- Create an instance of GUI, Map and Agent
- Provide callbacks for GUI so they can be bound to certain buttons and subroutines invoked by the GUI. These callbacks include:
  - `cb_map_get`: Provide the current Map object for GUI (to redraw screen, update status and logs)
  - `cb_map_open`: Handle map opening (by replacing the current Map object with a new one, with path to mapfile given by the system's "Open file" dialog.)
  - `cb_reset`: Reset Map and Agent (by calling their corresponding `.reset()` methods)
  - `cb_step`: Forward current game state by one step, then handle Agent's request and game-ending condition checks
- Handle the game flow (as hinted by the `cb_step()` method.)

`logic_random.py`: Agent

***NOTE**: The name of this module is a misnomer; this module was originally intended to provide only a dumb agent that can only move randomly (hence the name `logic_random`) to assist in writing the interface of Controller and debugging GUI.*

This module provides two agents (called "Players") for solving the game.

"Dumb" agent

This is a simplistic agent that has limited memory (and only for a specific purpose; see below) and limited capability of reacting to the environment ("simple reflex").

- Whenever it sees gold, it will pick up (obviously);
- Whenever it senses stench, it will remember the adjacent locations (maximum of 4) and shoot at each location on each agent's call;
- Otherwise, it will pick a random direction to move to.

"Smart" agent (that actually works)

The Logic core is built using **Glucose3** solver provided by `python-SAT`.

In this project, the Agent can be implement by using First Order Logic (FOL)– or Propositional Logic (PL)– Resolution. In a quick survey, we understand the concept of FOL and PL and it's advantages/disadvantages.

Properties	**FOL**	**PL-Resolution**
Implementation	Hard (Re-implementation of clause parser, entailing, etc. is required.)	Quick and easy
Performace	Normal	Normal — but it's will be slower than FOL in large space
Correctness	✅ Yes	✅ Yes
Space Complexity	Small	Large

Since we are on a tight time budget, we decided to build the agent's logic core/KB following the PL–Resolution method.

With Glucose3 model, the KB is much simpler. KB is a set of clauses as a `CNF` form. The model will solve it and return the answer of a 200–variable equation (200 because – assuming the map is 10×10 – it's 100 tiles + 100 Wumpus' clusters + 100 Pits' clusters.)

Our strategies are:

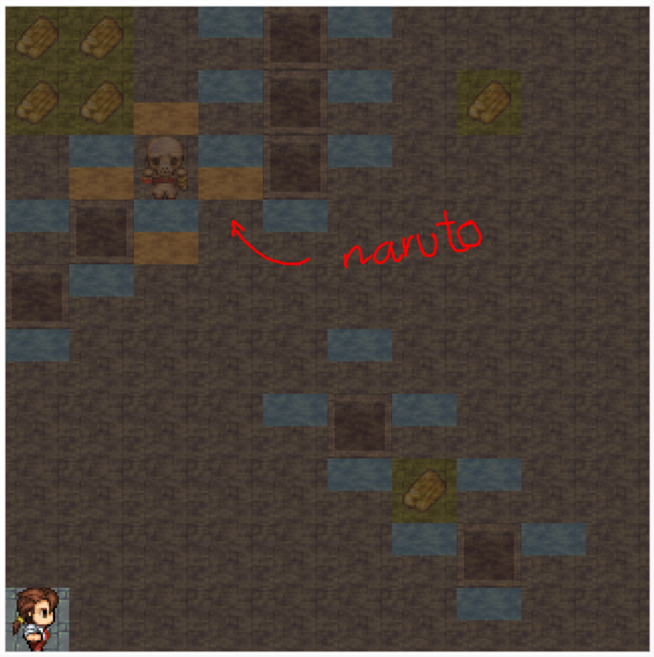
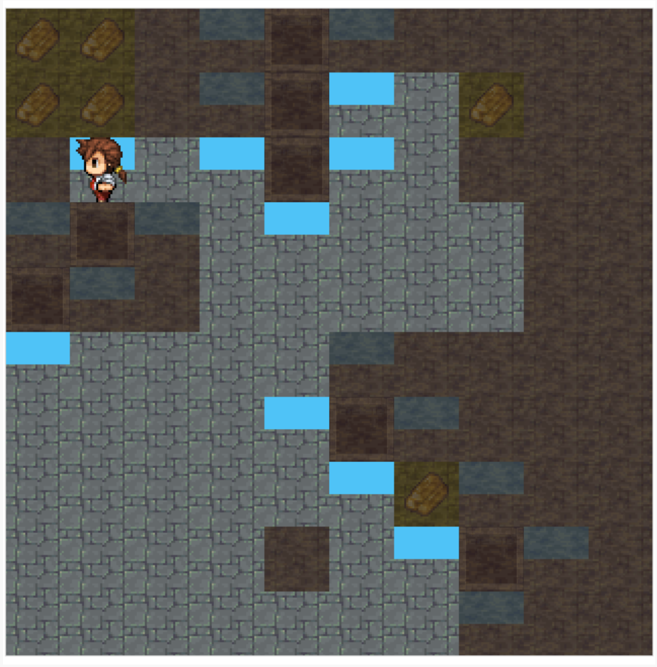
- Finish the game by going through all of safe cells in the map.
- Kill Wumpuses with the least shoots, depend on what KB entailed.
- Collect all the gold from the safe cells.

> There's nothing special here :>. It's just an agent, it finishes the game successfully. The most surprising thing to us is that our agent can think and returns the right decisions. We use the term 'right' but not 'good' because if you are a human, you can 'feel' that sometimes our score is enough to give up. But our agent will neva give up till it can not entail a safe cell to go; after all that it then climbs out of the cave.

There are some emergent behaviors we get from our agent:

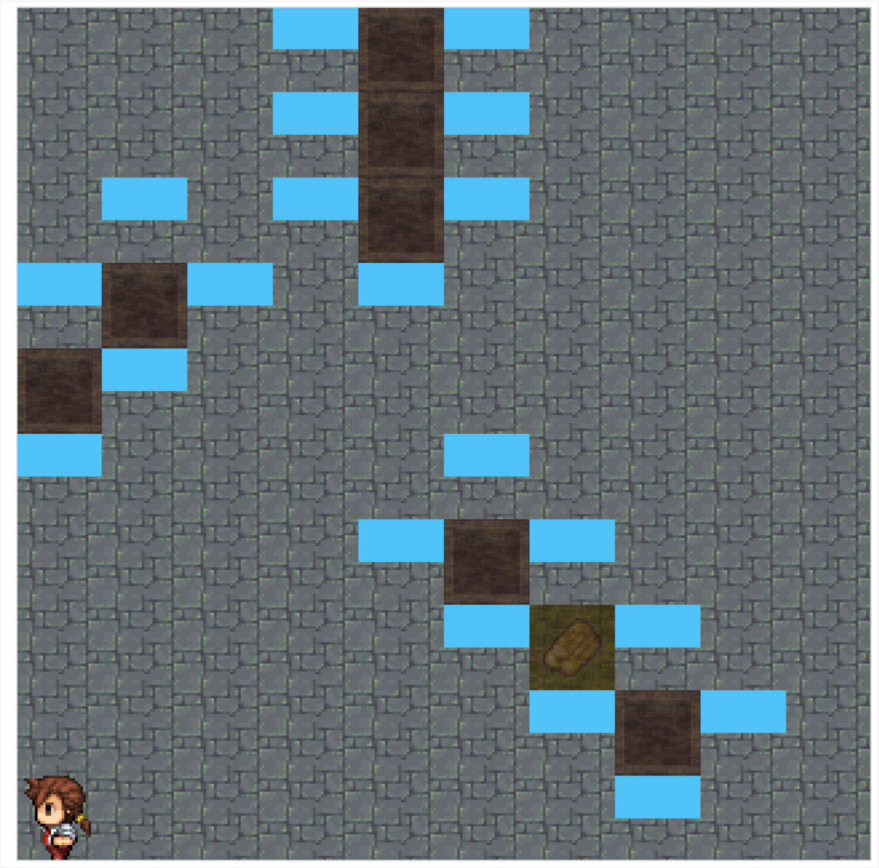
Naruto[1] case

If you shoot the Wumpus, it still unsafe because the Breeze tell us that there is a Pit. But, if you shoots the Wumpus successfully, it's a safe cell

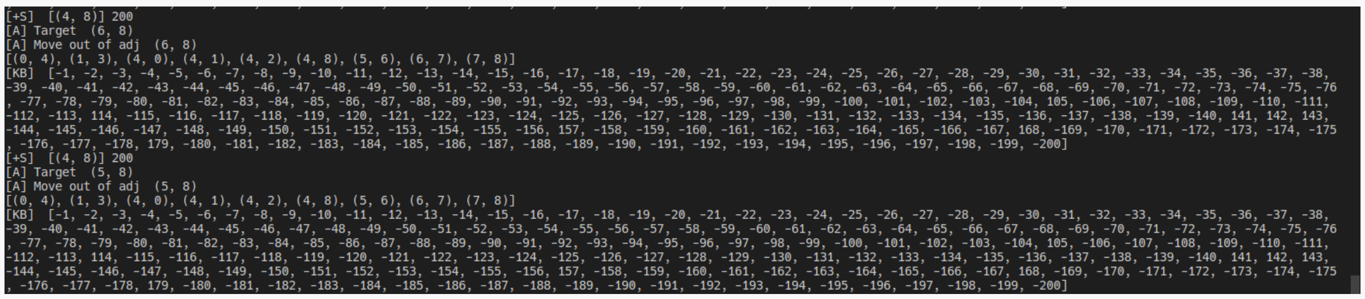
	
Naruto case	**Our agent pass Naruto and collect the golds**

Risky case

When there are no safe cells, the agent decide to get out the cave - it's a risky move if you try to get the remaining gold.


Risky case

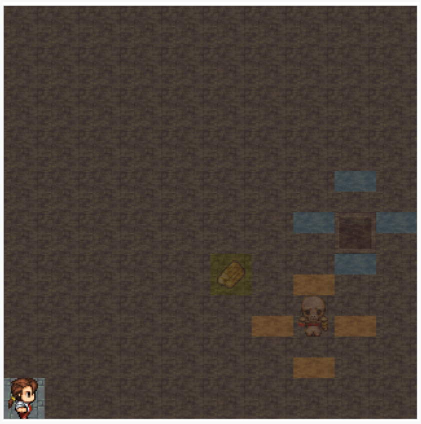
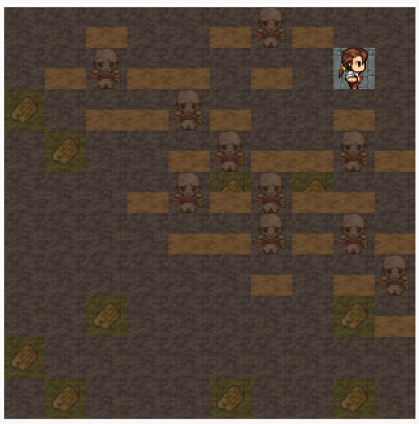
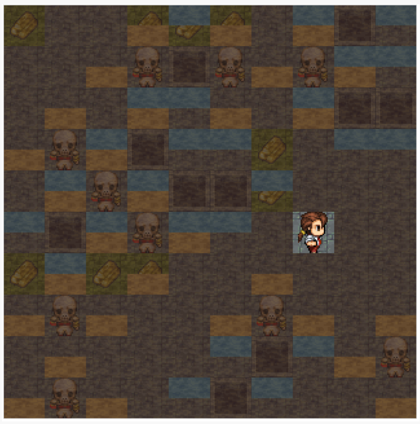
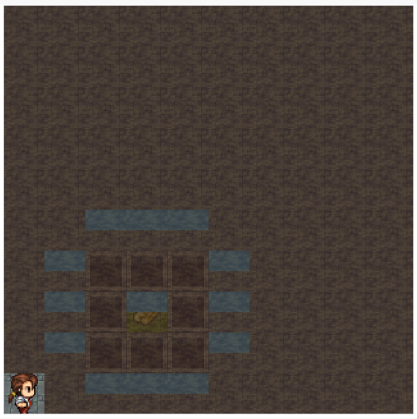
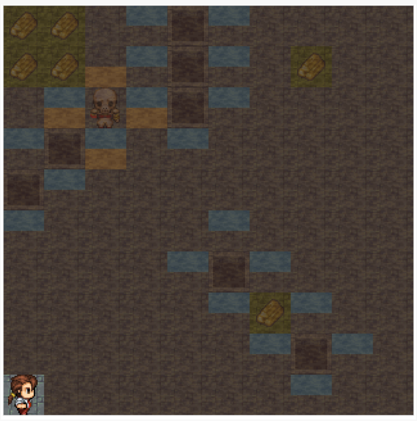
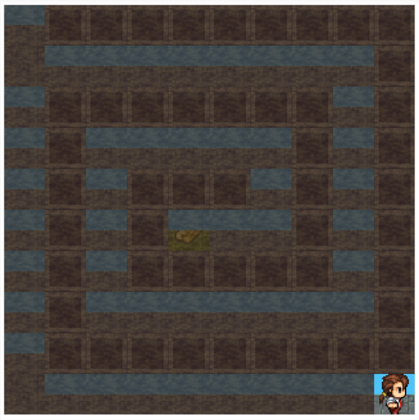
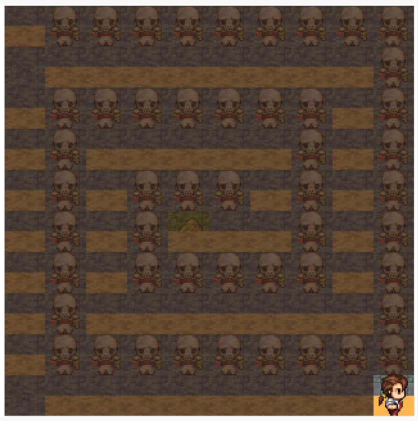
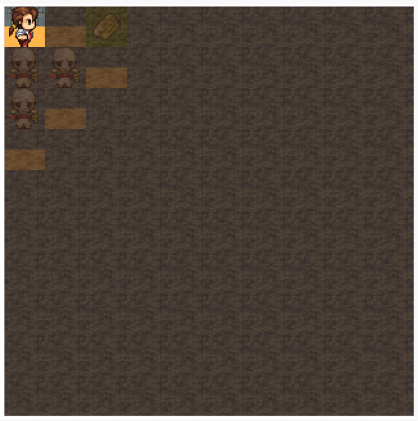
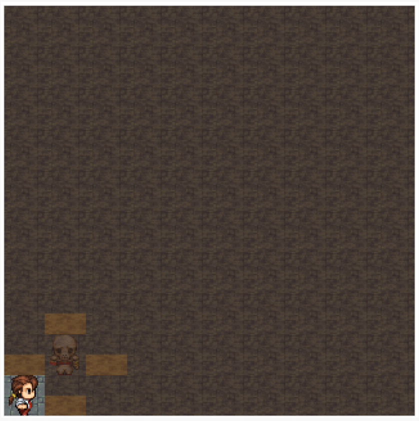
Our program will output the KB as a list of variables' cluster value (there's 200 of them):



```
[+S]  [(4, 8)] 200
[A] Target  (6, 8)
[A] Move out of adj  (6, 8)
[(0, 4), (1, 3), (4, 0), (4, 1), (4, 2), (4, 8), (5, 6), (6, 7), (7, 8)]
[KB]  [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, -24, -25, -26, -27, -28, -29, -30, -31, -32, -33, -34, -35, -36, -37, -38,
-39, -40, -41, -42, -43, -44, -45, -46, -47, -48, -49, -50, -51, -52, -53, -54, -55, -56, -57, -58, -59, -60, -61, -62, -63, -64, -65, -66, -67, -68, -69, -70, -71, -72, -73, -74, -75, -76
, -77, -78, -79, -80, -81, -82, -83, -84, -85, -86, -87, -88, -89, -90, -91, -92, -93, -94, -95, -96, -97, -98, -99, -100, -101, -102, -103, -104, 105, -106, -107, -108, -109, -110, -111,
-112, -113, 114, -115, -116, -117, -118, -119, -120, -121, -122, -123, -124, -125, -126, -127, -128, -129, -130, -131, -132, -133, -134, -135, -136, -137, -138, -139, -140, 141, 142, 143,
-144, -145, -146, -147, -148, -149, -150, -151, -152, -153, -154, -155, -156, 157, -158, -159, -160, -161, -162, -163, -164, -165, -166, -167, 168, -169, -170, -171, -172, -173, -174, -175
, -176, -177, -178, 179, -180, -181, -182, -183, -184, -185, -186, -187, -188, -189, -190, -191, -192, -193, -194, -195, -196, -197, -198, -199, -200]
[+S]  [(4, 8)] 200
[A] Target  (5, 8)
[A] Move out of adj  (5, 8)
[(0, 4), (1, 3), (4, 0), (4, 1), (4, 2), (4, 8), (5, 6), (6, 7), (7, 8)]
[KB]  [-1, -2, -3, -4, -5, -6, -7, -8, -9, -10, -11, -12, -13, -14, -15, -16, -17, -18, -19, -20, -21, -22, -23, -24, -25, -26, -27, -28, -29, -30, -31, -32, -33, -34, -35, -36, -37, -38,
-39, -40, -41, -42, -43, -44, -45, -46, -47, -48, -49, -50, -51, -52, -53, -54, -55, -56, -57, -58, -59, -60, -61, -62, -63, -64, -65, -66, -67, -68, -69, -70, -71, -72, -73, -74, -75, -76
, -77, -78, -79, -80, -81, -82, -83, -84, -85, -86, -87, -88, -89, -90, -91, -92, -93, -94, -95, -96, -97, -98, -99, -100, -101, -102, -103, -104, 105, -106, -107, -108, -109, -110, -111,
-112, -113, 114, -115, -116, -117, -118, -119, -120, -121, -122, -123, -124, -125, -126, -127, -128, -129, -130, -131, -132, -133, -134, -135, -136, -137, -138, -139, -140, 141, 142, 143,
-144, -145, -146, -147, -148, -149, -150, -151, -152, -153, -154, -155, -156, 157, -158, -159, -160, -161, -162, -163, -164, -165, -166, -167, 168, -169, -170, -171, -172, -173, -174, -175
, -176, -177, -178, 179, -180, -181, -182, -183, -184, -185, -186, -187, -188, -189, -190, -191, -192, -193, -194, -195, -196, -197, -198, -199, -200]
```

KB in the terminal

Sample runs

		
Simple map	Wumpus and Gold only	A complex map with lots of wumpuses, pits and Naruto cases
		
Gold blocked	**Figure 1**	Pit Maze
		
Wumpus Maze	Optimal shoots	Bias test

> This project challenge our patience and creating an environment where we aren't only allowed to succeed but also to fail. But our Agent won't fail 💯. Where there is a will, there is a way. A way for our Agent finish the game.

— Acknowledgments —

To all teacher assistants of this course,

We would like to thank you for this semester, especially in this course. For your patience and for creating an environment where we aren't only allowed to succeed but also to fail, for replying every questions of us. Students succeed when students feel the freedom to imagine and trust you give ★.

This class was challenging at times but there was value in being exposed to the material. And this project is really challenging at all. But with all of our efforts, teamwork spirit and all of your supports, we did it.

At the end, we would like to acknowledge support for this project by my teachers at University of Science — VNU. Special thanks to our seniors at University of Science for providing insights and expertise that helped us finish this project.

Thanks to our teammates, we did it! Together.

—— Toàn Đoàn (and Văn Thiện) (but mostly Toàn Đoàn)

References

- Standard Python docs for `Tkinter` (GUI) and `python-sat/Glucose3` (Logic)
- GitHub: `thiagodnf/wumpus-world-simulator`: assets for map drawing
- Countless StackOverflow threads about everything
- That star ★ in the **Acknowledgment**:
  Milrad, Marcelo. (1999). Designing an Interactive Learning Environment to Support Children's Understanding in Complex Domains.

---

1. We call that is Naruto case because there are a bunch of fake unsafe cells, but if you shoot the right cell, it will disappear – just like Naruto and his clones when he use 分身の術 (Kage Bunshin no Jutsu – "The Art of Doppelganger".) ↩