

```
Python3 -P SimpleSurvivor Problem -f tinyHouseSearch -l tinyHouse -x 1.0 -z 2.0
```



readCommand(argv)

(Always returns the sequence to solve tinyHouse layout)

↳ other layouts fail

PROBLEM_CHOICES → ① Simple Survivor
→ ② Multi Survivor

mandatory args

Flags =

- P Problem (one of the problem choices)
- f function (search: BFS, tinyHouse, A*)
- h heuristic (nullHeuristic, manhattanHeuristic)
- l layout (layout file: burning office, power plant, etc)
- t text graphics (display steps text output)
- q quiet graphics (no graphics, only final output)
- z zoom (float multiplier for window zoom)
- r recordActions (writes actions to a file)
- C catchExceptions (turns on exception handling during mission)

- Loads layout, identifies rescuer, and passes args to runMission.



runMission(**args)

args = {**} → layout
rescuer
display
record
catchException

This calls newMission on the RescueMission instance (RescueMission)

newMission(layout, rescuer, display, False, catchExceptions)



Initializes a new RescueState (initState) that contains:

- Rescuer position
- Survivors remaining
- Terrain layout

A new Game is created (mission)

Game(agents, display, self, catchExceptions)



mission contains the following data

Game

- agentCrashed: bool
- agents: Agent[]
- display: RescueGraphics
- rules: RescueMission
- startingIndex: int
- gameOver: bool
- muteAgents: bool
- catchExpressions: bool
- moveHistory: Array[]
- totalAgentTimes: Array[]
- totalAgentTimeWarnings: Array[]
- agentTimeout: bool

We set `mission's state` to `initState`, then set `rescueMission's` `initialState` to a deepcopy of `initState`, and set `rescueMission's` `quiet` value.

This object is now called `episode`.

We now call `run()` on `episode` object.

`run()`

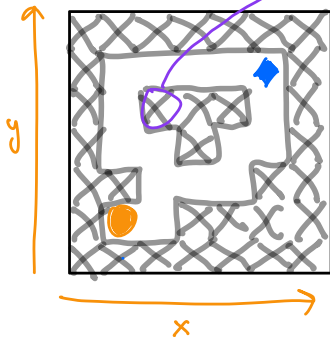


① The game's display is initialized with the layout and all attributes about terrain, survivors, etc and their locations.

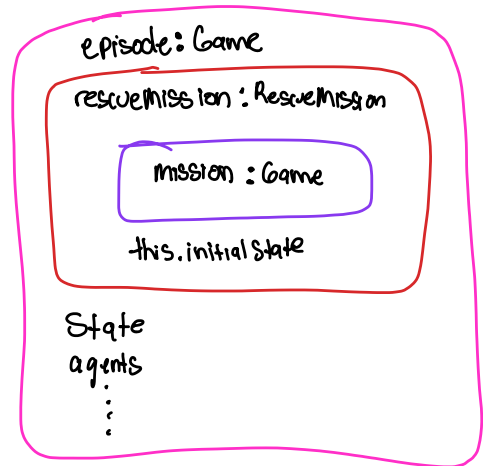
② The agents are initialized with `initialState` via `registerInitialState()`.

③ The agent (`SearchAgent`) sets the problem (`Search problem`) by calling the `init` from either `SimpleSearch`.

This `problem` contains locations of walls



```
> data = [[True, True, True, True, True, True, True], [True, False, Tr
> special variables
> function variables
> 0 = [True, True, True, True, True, True, True]
> 1 = [True, False, True, False, False, False, True]
> 2 = [True, False, False, False, True, False, True]
> 3 = [True, True, False, True, True, False, True]
> 4 = [True, True, False, False, True, False, True]
> 5 = [True, True, True, False, False, False, True]
> 6 = [True, True, True, True, True, True, True]
```



Only A* uses heuristics

Using the `problem`, we pass it to the `searchFunction()` so we can get the sequence of moves to solve the given layout.

[this is where we must add the search algorithms!] (`Search.py`)

This is when we get the cost based on the terrain conditions.

In `getCostOfActions`, using the differentials `dx` and `dy` and the terrain type crossed over, we add up the cost of the path taken.

We then draw each step on the map in sequence.