PAR Assignment 1 Report

Authors: Bruno Sánchez & Jean Dié

Introduction to the problem

The problem to solve in this assignment consists of modeling a rescue drone in an emergency site, which has to rescue stranded people and carry them to a safe zone while avoiding some obstacles.

In the model, the emergency site is represented by a grid (which we assume to be squared) where some of the cells contain obstacles, while others contain the safe zone, drone and stranded people. The drone will move horizontally or vertically between adjacent cells – as long as the target cell is not an obstacle – and will pick up the people stranded throughout the grid and carry them to the safe zone (one by one), where it will drop them off.

Additionally, there is a condition which the model must satisfy: the maximum safe zone capacity (how many people it can hold at once). For a grid of dimensions *N*x*N*, the safe zone has a capacity of $M = N - 1$. Once the drone has rescued *M* people from the grid and dropped them off at the safe zone, if there are more stranded people left to rescue we must simulate the process of people being evacuated from the safe zone in order to make room for the remaining ones that still need rescue.

The objective of the model is to find a series of steps (i.e. *a plan*) which allows us to rescue all of the stranded people in the emergency zone. For this, we will employ the PDDL language to build the model (*domain*), set the initial conditions and goal (*problem*), and finally calculate a solution (*plan*).

Problem analysis

First of all, we need to establish how we will construct each of the parts of the model in order to be able to implement it into PDDL. For this, we need to define the *predicates* and *actions* that we will consider.

Note: The default model proposed in the assignment considers one location per cell of the grid and expects an explicit specification of all the adjacencies of locations in order to establish the layout of the grid. Then, the movement of the drone is based on this concept of adjacency between locations. While this implementation simplifies the movement action of the drone (by unifying all of the movement directions into one single action), it requires a significant effort of initialization in order to express the adjacencies of all the locations. Because of this, we have opted for a different

approach, based on coordinates (X, Y). This approach requires one action for every direction of movement, but simplifies the initialization considerably, since it is sufficient to establish the order of the coordinate values.

The model based on adjacencies was also implemented in PDDL as extra work (and can be found in the attached ZIP folder), but in this report we will only comment and analyze the model based on coordinates.

Predicates:

- Position(P): P is a valid coordinate position of the grid.
- Inc(P, PP): PP represents an increment in position with respect to P.
- Safe-zone(X, Y): The safe zone is at coordinates (X, Y) of the grid.
- Spot(S): S is a valid spot in the safe zone (to hold a person).
- Free-spot(S): Spot S of the safe zone is free at the moment.
- Obstacle(X, Y): There is an obstacle at coordinates (X, Y).
- Person(X, Y): There is a person at coordinates (X, Y).
- Drone(X, Y): The drone is at coordinates (X, Y) at the moment.
- Empty-drone(): The drone is not carrying a person at the moment.

Note: Since the grid is squared (*N*x*N*), we only need *N* coordinate positions *P1*, …, *PN* to build a coordinate system that fully describes the grid. Then, both coordinates *X* and *Y* take values among those coordinate positions.

Actions:

- Up(X, Y, NY): The drone moves up from coordinates (X, Y) to (X, NY).
- Down(X, Y, NY): The drone moves down from coordinates (X, Y) to (X, NY).
- Right(X, Y, NX): The drone moves right from coordinates (X, Y) to (NX, Y).
- Left(X, Y, NX): The drone moves left from coordinates (X, Y) to (NX, Y).
- Pick-up(X, Y): The drone picks a person up from coordinates (X, Y).
- Drop-off(X, Y, S): The drone drops a person off at coordinates (X, Y), as long as the safe zone is there and the spot S is free.
- Evacuate-safe-zone(S): A person is evacuated from spot S of the safe zone, in order to free it.

Note: Before taking the moving actions (*up*, *down*, *right* and *left*), it is checked whether the target coordinates hold an obstacle or not. If they do, the action cannot be taken. SImilarly, the *pick-up* action checks that a person is at position (X, Y) before being taken.

PDDL implementation

The PDDL implementation essentially consists of a *domain.pddl* file that establishes the model's conditions (predicates and actions), and one or more *problem.pddl* files

that establish a set of initial conditions for the predicates and what the goal state is. In this section, we will only display the *domain.pddl* implementation. In the next section (*Test cases and results*), we will show the multiple *problem.pddl* configurations that we have tested and the results attained with each of them.

Domain.pddl

```
Unset
(define (domain rescue-drone)

    (:requirements :strips :equality :negative-preconditions)

    (:predicates (position ?P)
                 (inc ?P ?PP)
                 (safe-zone ?X ?Y)
                 (obstacle ?X ?Y)
                 (person ?X ?Y)
                 (drone ?X ?Y)
                 (empty-drone)
                 (spot ?S)
                 (free-spot ?S)
    )

    (:action up
        :parameters (?X ?Y ?NY)

        :precondition (and
                      (position ?X) (position ?Y) (position ?NY)
                      (inc ?NY ?Y)
                      (drone ?X ?Y)
                      (not (obstacle ?X ?NY))
        )

        :effect (and
                (drone ?X ?NY)
                (not (drone ?X ?Y))
        )
    )

    (:action down
        :parameters (?X ?Y ?NY)

        :precondition (and
                      (position ?X) (position ?Y) (position ?NY)
                      (inc ?Y ?NY)
                      (drone ?X ?Y)
                      (not (obstacle ?X ?NY))
```

```
        )

        :effect (and
                (drone ?X ?NY)
                (not (drone ?X ?Y))
        )
    )

    (:action right
        :parameters (?X ?Y ?NX)

        :precondition (and
                    (position ?X) (position ?Y) (position ?NX)
                    (inc ?X ?NX)
                    (drone ?X ?Y)
                    (not (obstacle ?NX ?Y))
        )

        :effect (and
                (drone ?NX ?Y)
                (not (drone ?X ?Y))
        )
    )

    (:action left
        :parameters (?X ?Y ?NX)

        :precondition (and
                    (position ?X) (position ?Y) (position ?NX)
                    (inc ?NX ?X)
                    (drone ?X ?Y)
                    (not (obstacle ?NX ?Y))
        )

        :effect (and
                (drone ?NX ?Y)
                (not (drone ?X ?Y))
        )
    )

    (:action pick-up
        :parameters (?X ?Y)

        :precondition (and (drone ?X ?Y) (person ?X ?Y) (empty-drone))

        :effect (and (not (person ?X ?Y)) (not (empty-drone)))
    )
```

```
    (:action drop-off
        :parameters (?X ?Y ?S)

        :precondition (and
                    (drone ?X ?Y) (not (empty-drone))
                    (safe-zone ?X ?Y) (spot ?S) (free-spot ?S)
        )

        :effect (and (empty-drone) (person ?X ?Y) (not (free-spot ?S)))
    )

    (:action evacuate-safe-zone
        :parameters (?S)
        :precondition (and (spot ?S) (not (free-spot ?S)))
        :effect (free-spot ?S)
    )
)
```

Test cases and results

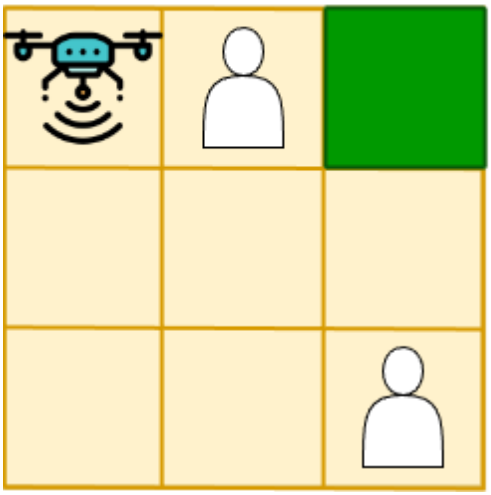
We have tested a total of 7 cases of increasing complexity, starting with an extremely basic scenario to test the core functionality of the model, and ending with an impossible case which is unsolvable with the established rules and limitations.

For each of the 7 cases, we briefly explain the objective of that test case, display a diagram of the initial and goal states, and then show the PDDL implementation of the problem.

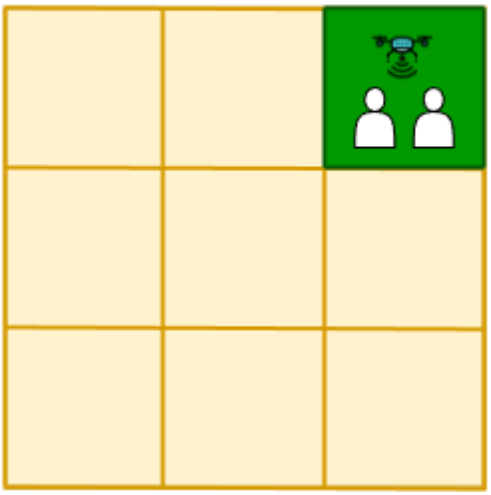
Case 1: Basic

First of all, let us test the basic functionalities of the model (moving and carrying actions) with a test case that has no obstacles and a number of people that fit into the safe zone, in a small (3x3) grid.

Initial state



Goal state



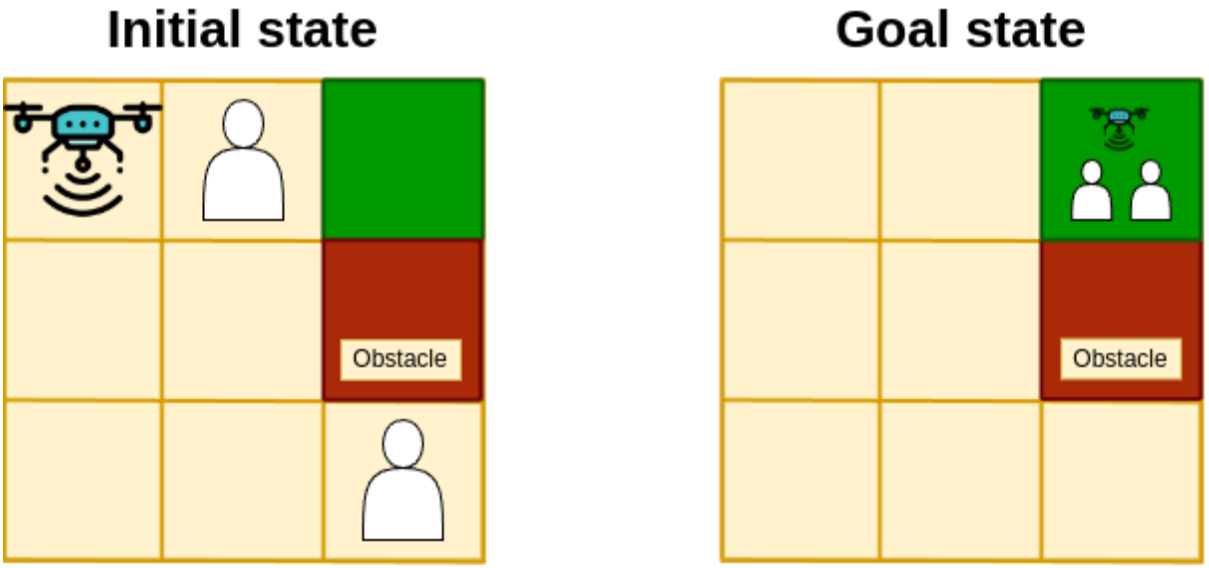
Unset

```
(define (problem pb1)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 S1 S2
    )

    (:init
        (position P1)
        (position P2)
        (position P3)

        (inc P1 P2)
        (inc P2 P3)

        (safe-zone P3 P1)

        (spot S1)
        (spot S2)

        (free-spot S1)
        (free-spot S2)

        (person-location P2 P1)
        (person-location P3 P3)

        (drone-location P1 P1)
        (empty-drone)
    )
```

```
    (:goal
        (and
            (not (person-location P2 P1))
            (not (person-location P3 P3))
            (person-location P3 P1)
            (empty-drone))
    )
)
```

Case 2: Basic with obstacles

Let us now introduce a new feature of the model: obstacles. In this case, we test the same scenario as in Case 1, with a small grid and a number of people that fit in the safe zone, but now introducing an obstacle in the direct path between one of the people and the safe zone, in order to test the capability of the drone to avoid obstacles.

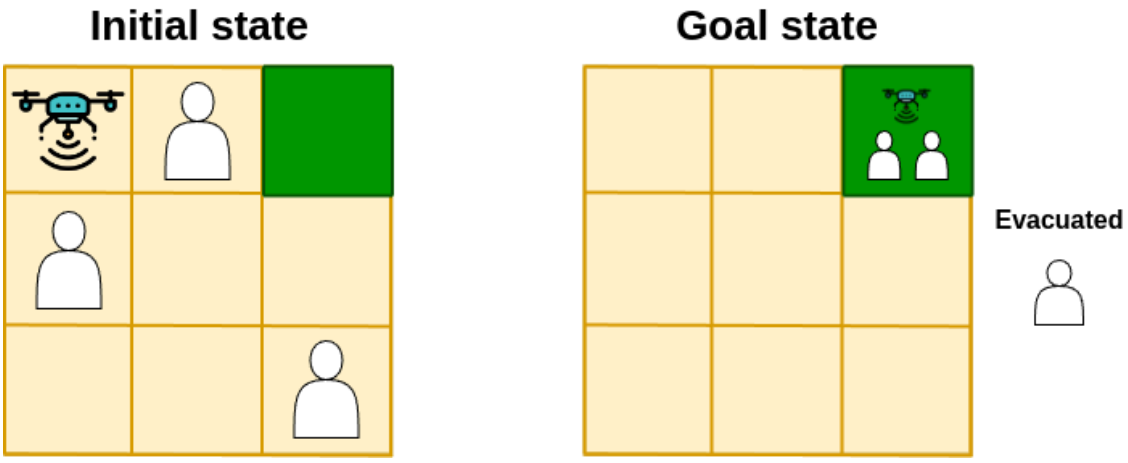



```
Unset
(define (problem pb2)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 S1 S2
    )

    (:init
```

```
        (position P1)
        (position P2)
        (position P3)

        (inc P1 P2)
        (inc P2 P3)

        (safe-zone P3 P1)

        (spot S1)
        (spot S2)

        (free-spot S1)
        (free-spot S2)

        (person-location P2 P1)
        (person-location P3 P3)

        (obstacle P3 P2)

        (drone-location P1 P1)
        (empty-drone)
    )

    (:goal
        (and
            (not (person-location P2 P1))
            (not (person-location P3 P3))
            (person-location P3 P1)
            (empty-drone))
    )
)
```

Case 3: Overcrowded

For the next case, we will test another of the features of the model: the safe zone capacity. In order to do this, we again utilize a simple layout with a small grid and no obstacles, but in this case we introduce one more person to rescue, which makes the total number of people bigger than the safe zone capacity. The goal of this test case is to study whether the “spots” feature of the safe zone works as expected.



Unset

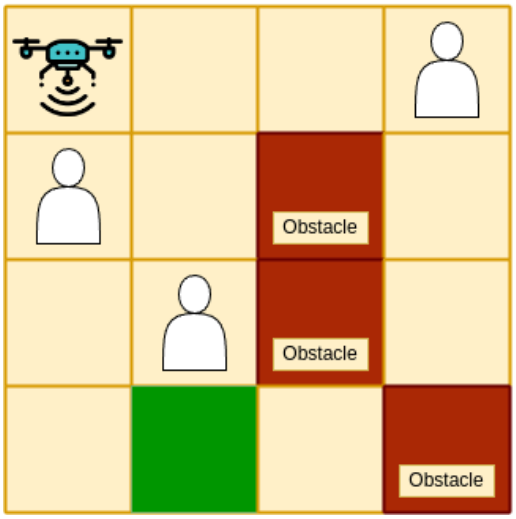
```
(define (problem pb3)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 S1 S2
    )

    (:init
        (position P1)
        (position P2)
        (position P3)

        (inc P1 P2)
        (inc P2 P3)

        (safe-zone P3 P1)

        (spot S1)
        (spot S2)

        (free-spot S1)
        (free-spot S2)

        (person-location P2 P1)
        (person-location P1 P2)
        (person-location P3 P3)

        (drone-location P1 P1)
        (empty-drone)
    )

    (:goal
```

```
        (and
            (not (person-location P2 P1))
            (not (person-location P3 P3))
            (not (person-location P1 P2))
            (person-location P3 P1)
            (empty-drone))
    )
)
```

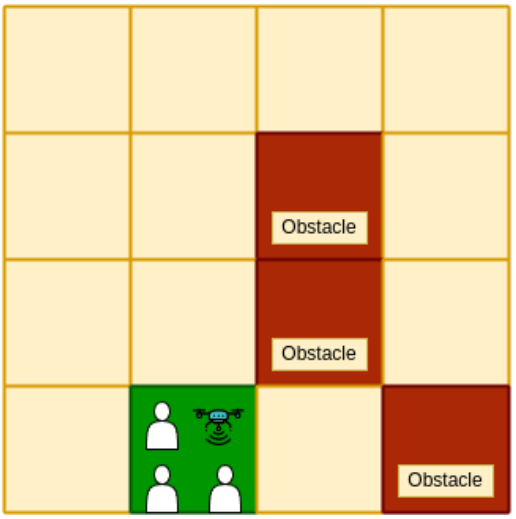
Case 4: Default

Let us now finally test the default case introduced as an example of the assignment. In this configuration, there are 3 people to rescue and 3 obstacles in a 4x4 grid, which tests almost all features of the model in a slightly bigger grid than before.

Initial state



Goal state

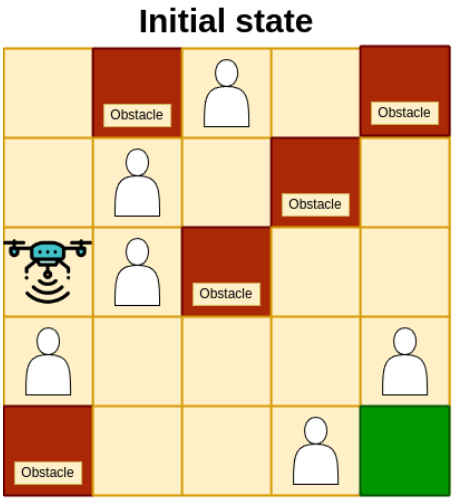


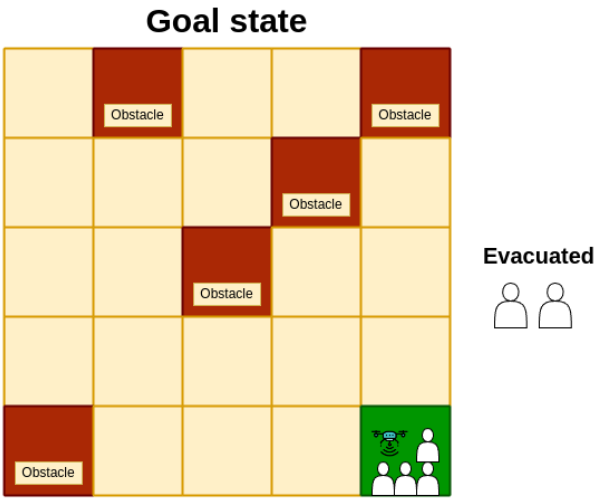

```
Unset
(define (problem pb4)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 P4 S1 S2 S3
    )

    (:init
```

```
        (position P1)
        (position P2)
        (position P3)
        (position P4)

        (inc P1 P2)
        (inc P2 P3)
        (inc P3 P4)

        (safe-zone P2 P4)

        (spot S1)
        (spot S2)
        (spot S3)

        (free-spot S1)
        (free-spot S2)
        (free-spot S3)

        (person-location P4 P1)
        (person-location P1 P2)
        (person-location P2 P3)

        (obstacle P3 P2)
        (obstacle P3 P3)
        (obstacle P4 P4)

        (drone-location P1 P1)
        (empty-drone)
    )

    (:goal
        (and
            (not (person-location P4 P1))
            (not (person-location P1 P2))
            (not (person-location P2 P3))
            (person-location P2 P4)
            (empty-drone))
    )
)
```

Case 5: Bigger grid

Now we now perform a test in a bigger grid. There are now 5 obstacles and 6 people in a 5x5 grid, which implies that there are more people to rescue than spots in the safe zone. This way, we test all of the functionalities of the model in a single case.





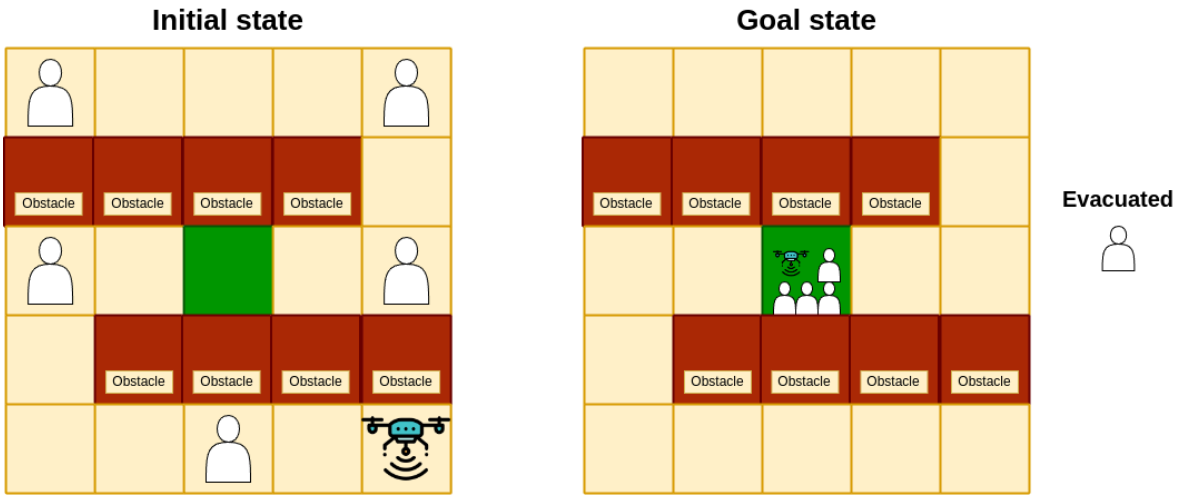
Unset

```
(define (problem pb5)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 P4 P5 S1 S2 S3 S4
    )

    (:init
        (position P1)
        (position P2)
        (position P3)
        (position P4)
        (position P5)

        (inc P1 P2)
        (inc P2 P3)
        (inc P3 P4)
        (inc P4 P5)

        (safe-zone P5 P5)

        (spot S1)
        (spot S2)
        (spot S3)
        (spot S4)

        (free-spot S1)
        (free-spot S2)
        (free-spot S3)
        (free-spot S4)
```

```
        (person-location P3 P1)
        (person-location P2 P2)
        (person-location P2 P3)
        (person-location P1 P4)
        (person-location P5 P4)
        (person-location P4 P5)

        (obstacle P2 P1)
        (obstacle P5 P1)
        (obstacle P4 P2)
        (obstacle P3 P3)
        (obstacle P1 P5)

        (drone-location P1 P3)
        (empty-drone)
    )

    (:goal
        (and
            (not (person-location P3 P1))
            (not (person-location P2 P2))
            (not (person-location P2 P3))
            (not (person-location P1 P4))
            (not (person-location P5 P4))
            (not (person-location P4 P5))
            (person-location P5 P5)
            (empty-drone))
    )
)
```

Case 6: Maze

With this test case we aim to again display all of the functionalities of the model, but this time in a neat and visual scenario that simulates a maze.



Unset

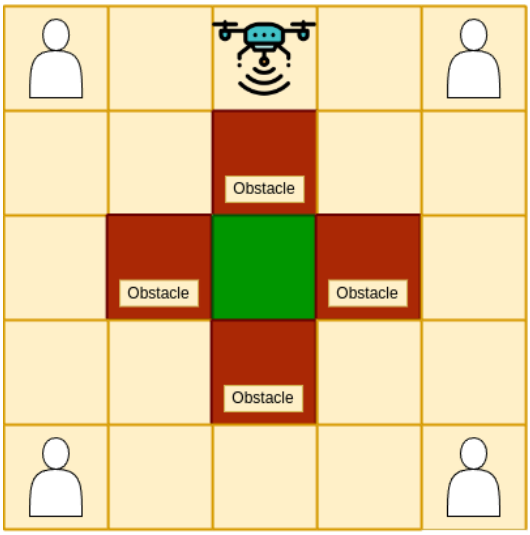
```
(define (problem pb6)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 P4 P5 S1 S2 S3 S4
    )

    (:init
        (position P1)
        (position P2)
        (position P3)
        (position P4)
        (position P5)

        (inc P1 P2)
        (inc P2 P3)
        (inc P3 P4)
        (inc P4 P5)

        (safe-zone P3 P3)

        (spot S1)
        (spot S2)
        (spot S3)
        (spot S4)

        (free-spot S1)
        (free-spot S2)
        (free-spot S3)
        (free-spot S4)
```

```
        (person-location P1 P1)
        (person-location P5 P1)
        (person-location P1 P3)
        (person-location P5 P3)
        (person-location P3 P5)

        (obstacle P1 P2)
        (obstacle P2 P2)
        (obstacle P3 P2)
        (obstacle P4 P2)

        (obstacle P2 P4)
        (obstacle P3 P4)
        (obstacle P4 P4)
        (obstacle P5 P4)

        (drone-location P5 P5)
        (empty-drone)
    )

    (:goal
        (and
            (not (person-location P1 P1))
            (not (person-location P5 P1))
            (not (person-location P1 P3))
            (not (person-location P5 P3))
            (not (person-location P3 P5))
            (person-location P3 P3)
            (empty-drone))
    )
)
```

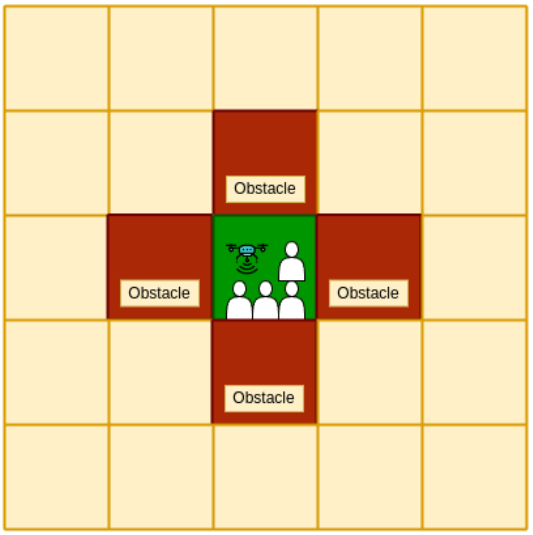
Case 7: Impossible

Finally, as a way to showcase the limits and restrictions of the model, we introduce a test case which is unsolvable. In this layout, the safe zone is fully surrounded by obstacles, and the only possible way to access it might be by moving diagonally, which is currently not an allowed type of movement.

Initial state



Goal state



(Impossible case)

Unset

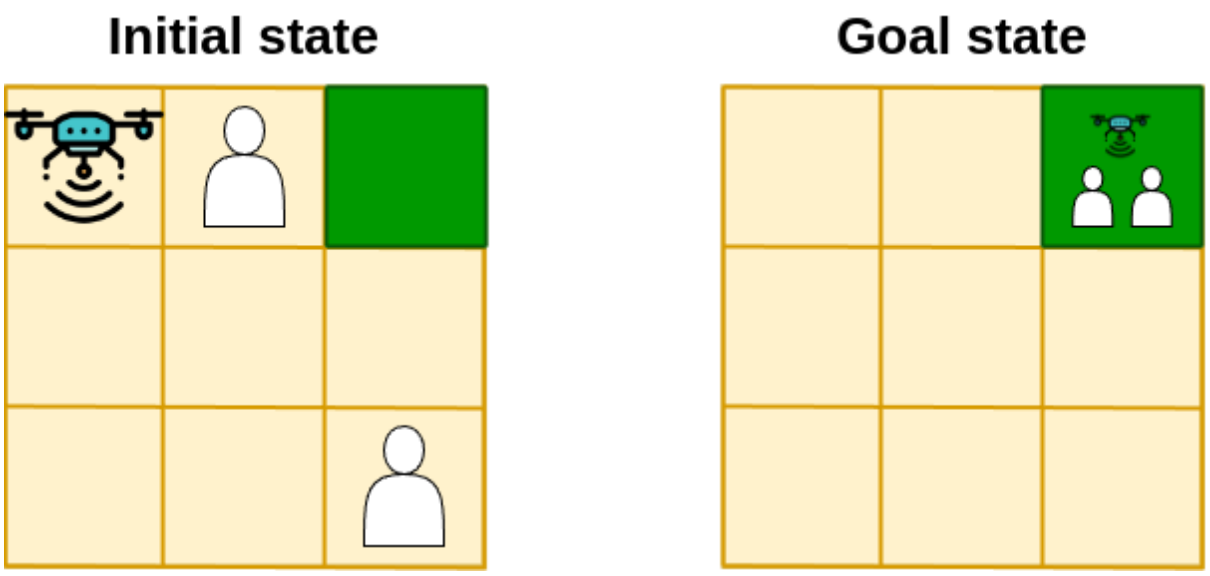
```
(define (problem pb7)

    (:domain rescue-drone)

    (:objects
        P1 P2 P3 P4 P5 S1 S2 S3 S4
    )

    (:init
        (position P1)
        (position P2)
        (position P3)
        (position P4)
        (position P5)

        (inc P1 P2)
        (inc P2 P3)
        (inc P3 P4)
        (inc P4 P5)

        (safe-zone P3 P3)

        (spot S1)
        (spot S2)
        (spot S3)
        (spot S4)

        (free-spot S1)
```

```
        (free-spot S2)
        (free-spot S3)
        (free-spot S4)

        (person-location P1 P1)
        (person-location P5 P1)
        (person-location P1 P5)
        (person-location P5 P5)

        (obstacle P3 P2)
        (obstacle P2 P3)
        (obstacle P4 P3)
        (obstacle P3 P4)

        (drone-location P3 P1)
        (empty-drone)
    )

    (:goal
        (and
            (not (person-location P1 P1))
            (not (person-location P5 P1))
            (not (person-location P1 P5))
            (not (person-location P5 P5))
            (person-location P3 P3)
            (empty-drone))
    )
)
```

Results analysis

In this section, we will display and analyze the results attained when solving the previously mentioned test cases. For the calculation of the plans we have utilized the *BFWS – FF-parser version* planner available through *Visual Studio Code*.

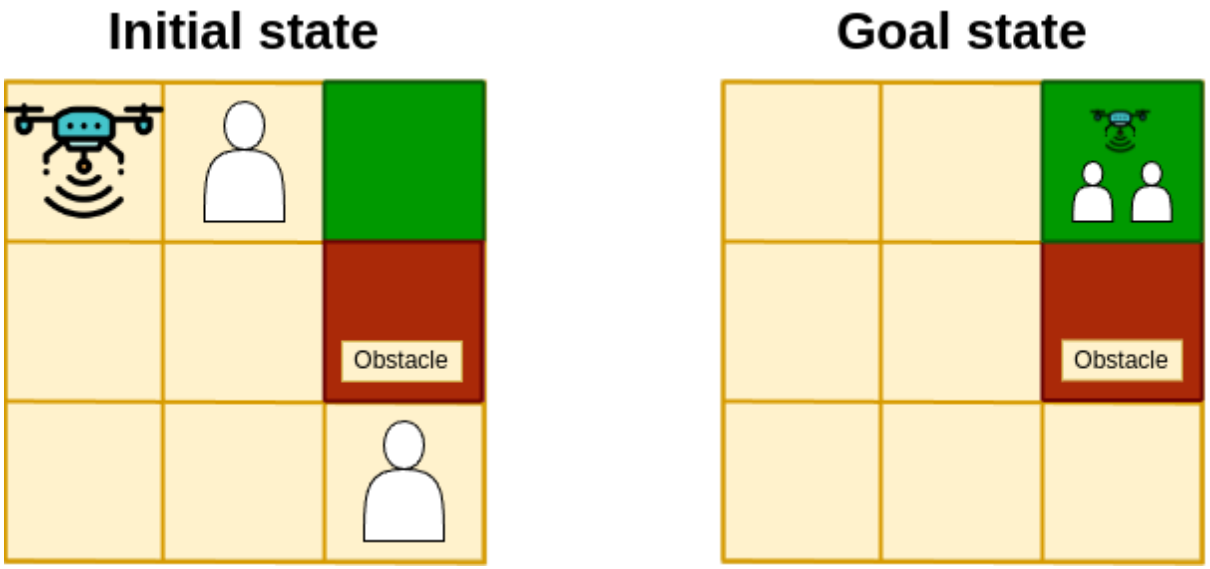
Case 1: Basic




```
Unset
;;!domain: rescue-drone
;;!problem: pb1

0.00000: (RIGHT P1 P1 P2)
0.00100: (PICK-UP P2 P1)
0.00200: (RIGHT P2 P1 P3)
0.00300: (DROP-OFF P3 P1 S2)
0.00400: (DOWN P3 P1 P2)
0.00500: (DOWN P3 P2 P3)
0.00600: (PICK-UP P3 P3)
0.00700: (UP P3 P3 P2)
0.00800: (UP P3 P2 P1)
0.00900: (DROP-OFF P3 P1 S1)

; Makespan: 0.009000000000000001
; Metric: 0.009000000000000001
```

We can see that the movement actions function correctly with the coordinate system we have built for the model, as well as the *pick-up* and *drop-off* actions.

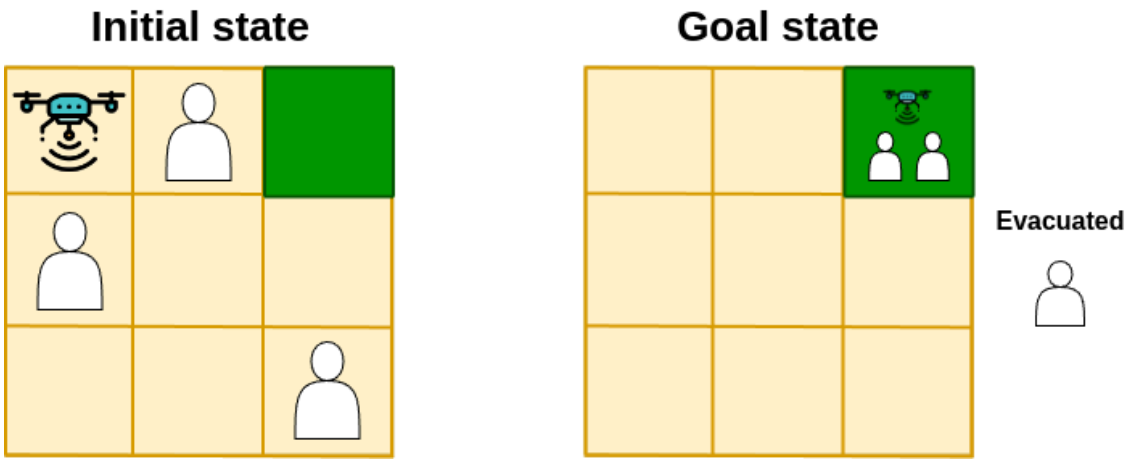
Case 2: Basic with obstacles




```
Unset
;;!domain: rescue-drone
;;!problem: pb2

0.00000: (RIGHT P1 P1 P2)
0.00100: (PICK-UP P2 P1)
0.00200: (RIGHT P2 P1 P3)
0.00300: (DROP-OFF P3 P1 S2)
0.00400: (LEFT P3 P1 P2)
0.00500: (DOWN P2 P1 P2)
0.00600: (DOWN P2 P2 P3)
0.00700: (RIGHT P2 P3 P3)
0.00800: (PICK-UP P3 P3)
0.00900: (LEFT P3 P3 P2)
0.01000: (UP P2 P3 P2)
0.01100: (UP P2 P2 P1)
0.01200: (RIGHT P2 P1 P3)
0.01300: (DROP-OFF P3 P1 S1)

; Makespan: 0.013000000000000005
; Metric: 0.013000000000000005
```

We now see that the obstacles also act as expected, impeding the movement actions through the cells they occupy. This makes the drone avoid the obstacle by moving around it.

Case 3: Overcrowded

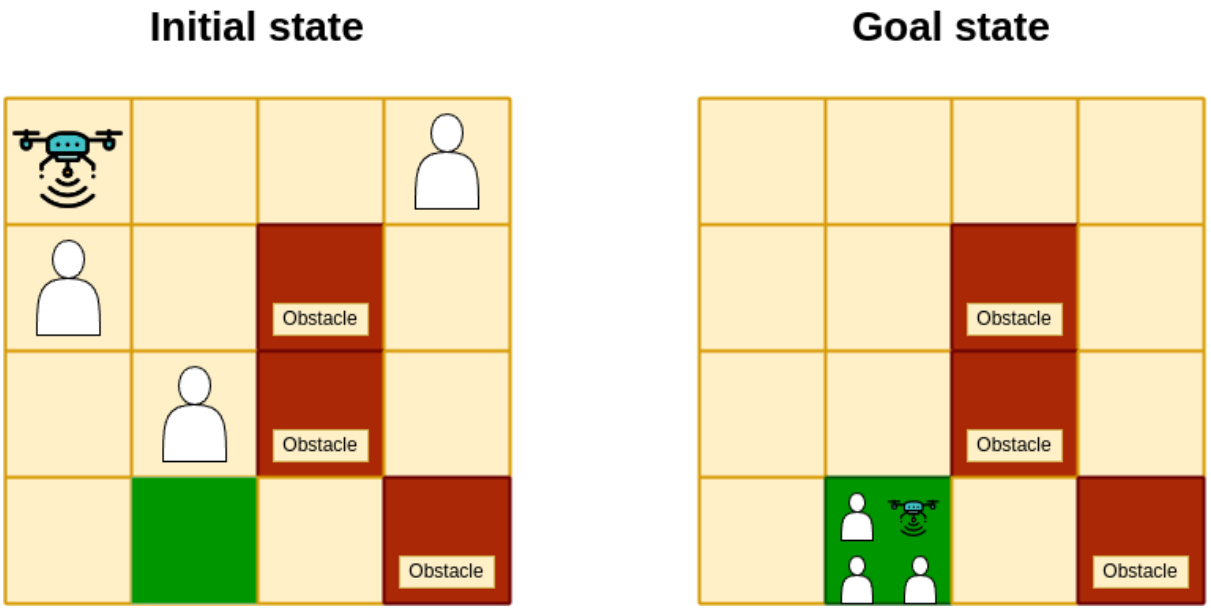



```
Unset
;;!domain: rescue-drone
;;!problem: pb3

0.00000: (RIGHT P1 P1 P2)
0.00100: (PICK-UP P2 P1)
0.00200: (RIGHT P2 P1 P3)
0.00300: (DROP-OFF P3 P1 S2)
0.00400: (DOWN P3 P1 P2)
0.00500: (DOWN P3 P2 P3)
0.00600: (PICK-UP P3 P3)
0.00700: (UP P3 P3 P2)
0.00800: (UP P3 P2 P1)
0.00900: (DROP-OFF P3 P1 S1)
0.01000: (LEFT P3 P1 P2)
0.01100: (LEFT P2 P1 P1)
0.01200: (DOWN P1 P1 P2)
0.01300: (PICK-UP P1 P2)
0.01400: (UP P1 P2 P1)
0.01500: (EVACUATE-SAFE-ZONE S2)
0.01600: (RIGHT P1 P1 P2)
0.01700: (RIGHT P2 P1 P3)
0.01800: (DROP-OFF P3 P1 S2)

; Makespan: 0.018000000000000001
; Metric: 0.018000000000000001
```

In this case, we prove that the safe zone capacity is being taken into account as it should, having to evacuate one of the spots before dropping off the last person at the safe zone.

Note: On earlier iterations of the model, we tried to implement the safe zone capacity with numeric functions (*:fluents* requirement of PDDL), but the available free online planners have trouble supporting them. Therefore, the concept of the *spots* of the free zone was implemented as a clean way to keep a count of the amount of people in the safe zone by only using predicates and actions (no functions).

Case 4: Default

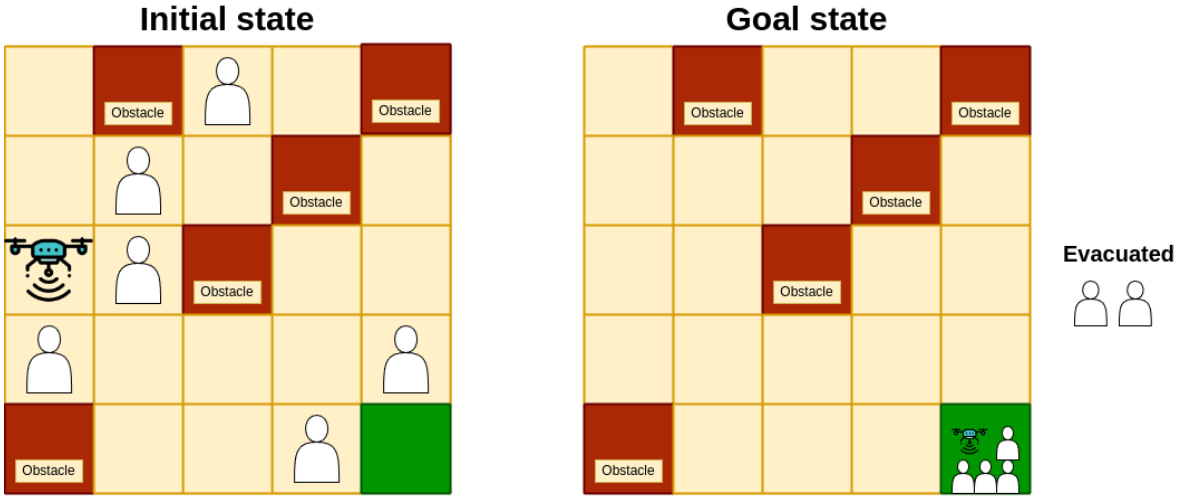



```
Unset
;;!domain: rescue-drone
;;!problem: pb4

0.00000: (DOWN P1 P1 P2)
0.00100: (PICK-UP P1 P2)
0.00200: (RIGHT P1 P2 P2)
0.00300: (DOWN P2 P2 P3)
0.00400: (DOWN P2 P3 P4)
0.00500: (DROP-OFF P2 P4 S1)
0.00600: (UP P2 P4 P3)
0.00700: (PICK-UP P2 P3)
0.00800: (DOWN P2 P3 P4)
0.00900: (DROP-OFF P2 P4 S2)
0.01000: (UP P2 P4 P3)
0.01100: (UP P2 P3 P2)
0.01200: (UP P2 P2 P1)
0.01300: (RIGHT P2 P1 P3)
0.01400: (RIGHT P3 P1 P4)
0.01500: (PICK-UP P4 P1)
0.01600: (LEFT P4 P1 P3)
```

```
0.01700: (LEFT P3 P1 P2)
0.01800: (DOWN P2 P1 P2)
0.01900: (DOWN P2 P2 P3)
0.02000: (DOWN P2 P3 P4)
0.02100: (DROP-OFF P2 P4 S3)

; Makespan: 0.02100000000000001
; Metric: 0.02100000000000001
```

It is now shown that we can easily enlarge the problem from a 3x3 to a 4x4 grid and the model still functions as expected. On top of this, we see that the planner can indeed find the most optimal solution (in terms of number of steps) and does not take any extra unnecessary steps.

Case 5: Bigger grid



Unset

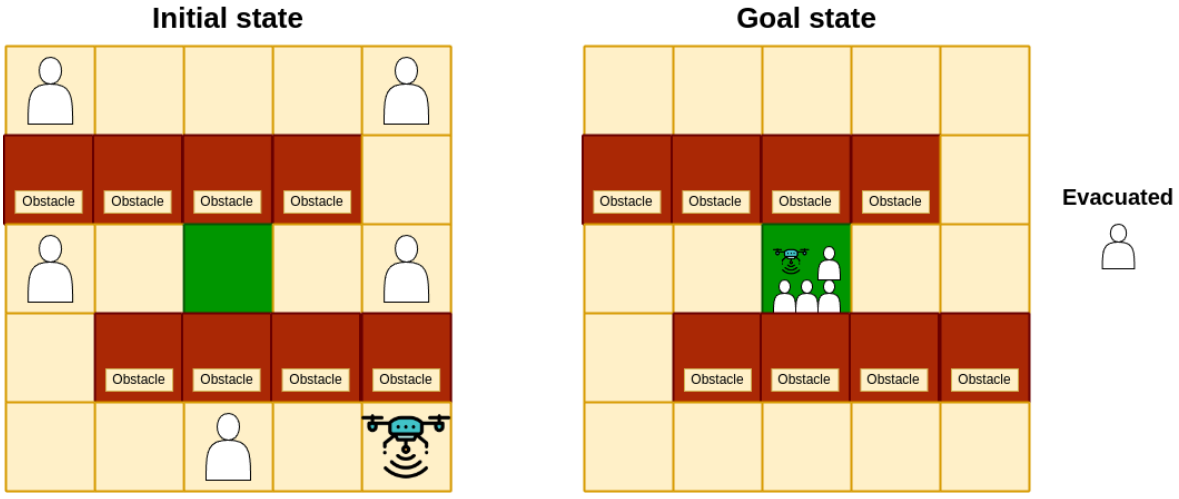
```
;;!domain: rescue-drone
;;!problem: pb5

0.00000: (RIGHT P1 P3 P2)
0.00100: (PICK-UP P2 P3)
0.00200: (DOWN P2 P3 P4)
0.00300: (RIGHT P2 P4 P3)
0.00400: (DOWN P3 P4 P5)
0.00500: (RIGHT P3 P5 P4)
0.00600: (RIGHT P4 P5 P5)
0.00700: (DROP-OFF P5 P5 S1)
0.00800: (UP P5 P5 P4)
0.00900: (PICK-UP P5 P4)
```

```
0.01000: (DOWN P5 P4 P5)
0.01100: (DROP-OFF P5 P5 S4)
0.01200: (LEFT P5 P5 P4)
0.01300: (PICK-UP P4 P5)
0.01400: (RIGHT P4 P5 P5)
0.01500: (DROP-OFF P5 P5 S3)
0.01600: (LEFT P5 P5 P4)
0.01700: (LEFT P4 P5 P3)
0.01800: (LEFT P3 P5 P2)
0.01900: (UP P2 P5 P4)
0.02000: (LEFT P2 P4 P1)
0.02100: (PICK-UP P1 P4)
0.02200: (RIGHT P1 P4 P2)
0.02300: (RIGHT P2 P4 P3)
0.02400: (RIGHT P3 P4 P4)
0.02500: (RIGHT P4 P4 P5)
0.02600: (DOWN P5 P4 P5)
0.02700: (DROP-OFF P5 P5 S2)
0.02800: (LEFT P5 P5 P4)
0.02900: (UP P4 P5 P4)
0.03000: (LEFT P4 P4 P3)
0.03100: (LEFT P3 P4 P2)
0.03200: (UP P2 P4 P3)
0.03300: (UP P2 P3 P2)
0.03400: (PICK-UP P2 P2)
0.03500: (DOWN P2 P2 P3)
0.03600: (EVACUATE-SAFE-ZONE S3)
0.03700: (DOWN P2 P3 P4)
0.03800: (DOWN P2 P4 P5)
0.03900: (RIGHT P2 P5 P3)
0.04000: (RIGHT P3 P5 P4)
0.04100: (RIGHT P4 P5 P5)
0.04200: (DROP-OFF P5 P5 S3)
0.04300: (LEFT P5 P5 P4)
0.04400: (LEFT P4 P5 P3)
0.04500: (LEFT P3 P5 P2)
0.04600: (UP P2 P5 P4)
0.04700: (UP P2 P4 P3)
0.04800: (UP P2 P3 P2)
0.04900: (RIGHT P2 P2 P3)
0.05000: (UP P3 P2 P1)
0.05100: (PICK-UP P3 P1)
0.05200: (DOWN P3 P1 P2)
0.05300: (LEFT P3 P2 P2)
0.05400: (DOWN P2 P2 P3)
0.05500: (DOWN P2 P3 P4)
0.05600: (EVACUATE-SAFE-ZONE S1)
0.05700: (RIGHT P2 P4 P3)
```

```
0.05800: (RIGHT P3 P4 P4)
0.05900: (RIGHT P4 P4 P5)
0.06000: (DOWN P5 P4 P5)
0.06100: (DROP-OFF P5 P5 S1)

; Makespan: 0.06100000000000005
; Metric: 0.06100000000000005
```

Here we can appreciate how our most comprehensive test case (which includes all of the functionalities of the model) is solved correctly, also in the lowest number of steps possible.

Case 6: Maze



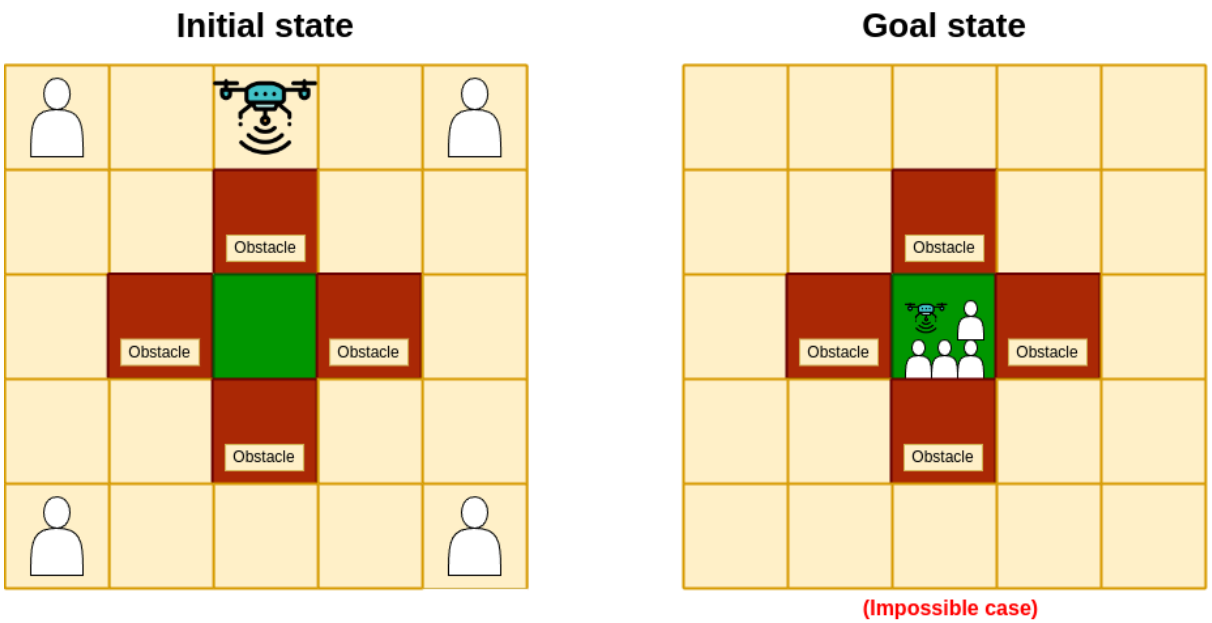
Unset

```
;;!domain: rescue-drone
;;!problem: pb6

0.00000: (LEFT P5 P5 P4)
0.00100: (LEFT P4 P5 P3)
0.00200: (PICK-UP P3 P5)
0.00300: (LEFT P3 P5 P2)
0.00400: (LEFT P2 P5 P1)
0.00500: (UP P1 P5 P4)
0.00600: (UP P1 P4 P3)
0.00700: (RIGHT P1 P3 P2)
0.00800: (RIGHT P2 P3 P3)
0.00900: (DROP-OFF P3 P3 S2)
0.01000: (LEFT P3 P3 P2)
0.01100: (LEFT P2 P3 P1)
```

```
0.01200: (PICK-UP P1 P3)
0.01300: (RIGHT P1 P3 P2)
0.01400: (RIGHT P2 P3 P3)
0.01500: (DROP-OFF P3 P3 S4)
0.01600: (RIGHT P3 P3 P4)
0.01700: (RIGHT P4 P3 P5)
0.01800: (PICK-UP P5 P3)
0.01900: (LEFT P5 P3 P4)
0.02000: (LEFT P4 P3 P3)
0.02100: (DROP-OFF P3 P3 S3)
0.02200: (RIGHT P3 P3 P4)
0.02300: (RIGHT P4 P3 P5)
0.02400: (UP P5 P3 P2)
0.02500: (UP P5 P2 P1)
0.02600: (PICK-UP P5 P1)
0.02700: (DOWN P5 P1 P2)
0.02800: (DOWN P5 P2 P3)
0.02900: (LEFT P5 P3 P4)
0.03000: (LEFT P4 P3 P3)
0.03100: (DROP-OFF P3 P3 S1)
0.03200: (RIGHT P3 P3 P4)
0.03300: (RIGHT P4 P3 P5)
0.03400: (UP P5 P3 P2)
0.03500: (UP P5 P2 P1)
0.03600: (LEFT P5 P1 P4)
0.03700: (LEFT P4 P1 P3)
0.03800: (LEFT P3 P1 P2)
0.03900: (LEFT P2 P1 P1)
0.04000: (PICK-UP P1 P1)
0.04100: (RIGHT P1 P1 P2)
0.04200: (EVACUATE-SAFE-ZONE S3)
0.04300: (RIGHT P2 P1 P3)
0.04400: (RIGHT P3 P1 P4)
0.04500: (RIGHT P4 P1 P5)
0.04600: (DOWN P5 P1 P2)
0.04700: (DOWN P5 P2 P3)
0.04800: (LEFT P5 P3 P4)
0.04900: (LEFT P4 P3 P3)
0.05000: (DROP-OFF P3 P3 S3)

; Makespan: 0.05000000000000004
; Metric: 0.05000000000000004
```

We check again with a different test case that all of the functionalities behave as they should.

Case 7: Impossible




```
Unset
;;!domain: rescue-drone
;;!problem: pb7

; ff: goal can be simplified to FALSE. No plan will solve it

; Makespan: 0
; Metric: 0
```

Finally, we see how the planner indeed cannot find a solution to the proposed problem, which was the expected outcome, given that it should be impossible to solve with the given set of rules of the model.

Conclusion

We declare the proposed model a success, with all of its functionalities correctly implemented and tested in PDDL, as well as its limitations.

Note: The model based on adjacencies was also implemented and tested. It can be found in the "*solution_adjacency*" folder of the attached ZIP. We preferred to showcase the model based on coordinates because it seemed cleaner, with a shorter initialization section in the *problem.pddl* files, even though it requires four moving actions instead of just one.