About

Author: Claudio Scheer

Title: RPG formalization in PDDL

Paper

Introduction

RPG is the problem of a hero who needs to go through rooms to escape from a dungeon. Each room can have an obstacle, a weapon or nothing. There are two obstacles, a monster and a trap, and a sword as a weapon. To fight against the monster, the hero must have the sword in his hands. Otherwise, the monster kills the hero. To disarm the trap, the hero must be empty-handed. Therefore, if the hero has the sword in his hands, he will need to drop it. In this paper, the proposed formalization, described in details in the section **Problem Formalization**, allows n heroes to escape the dungeon in different ways. The **Performance** section presents the results and the **Conclusion** section a discussion of the results and limitations of this formalization.

Problem Formalization

The requirements section in the PDDL file will indicate the features that the current domain is using. In this formalization, three requirements were used, as shown below:

- `:strips` - indicates that the domain is of the simplest form (Haslum et al. 2019)
- `:negative-preconditions` - enables the use of the keyword `not` on preconditions
- `:typing` - enables to define a type for the objects

Eight predicates were used to formalize the problem. Five of them, `hero_at`, `goal_at`, `monster_angry`, `trap_armed` and `sword_at`, are used to define the rooms where the heroes, goals, monsters, traps and weapons are. The `visited_room` predicate stores all rooms that have already been visited by one of the heroes. This predicate ensures that, as soon as the hero leaves, the room will be destroyed. The `valid_move` and `empty_handed` predicates store the corridors that connect the rooms and whether the hero has a sword in his hands, respectively.

As it is possible to observe, the parameters shared by the predicates are the hero and the room. Therefore, the types `hero` and `room` were defined in the `:types` section of the PDDL domain file. According to **(Haslum et al. 2019)**, `:types` restricts the parameters to the specified type. However, declaring types for predicates is only useful for validators. The actual type of the parameters will be defined in the initial state declared in the problem file. Below are the types declared in the domain file.

```
(:types
    room
    hero
)
```

The actions that heroes can perform are listed and explained below. It is important to note that the preconditions are described in a textual language, followed by the precondition as placed in the domain file. In addition, in some actions I mention some specific problems that I faced when planning the problem. Details to these problems were not presented. However, if necessary, I can describe them.

`:action move`

The purpose of this action is to move the hero to the desired room, ensuring that it is a valid move. As parameters, this action receives the hero who wants to perform the action, and the rooms the hero is in and wants to go, as shown below.

```
:parameters (?hero - hero ?from - room ?to - room)
```

This precondition ensures that the hero is in the room from which he is trying to move.

```
(hero_at ?hero ?from)
```

This precondition ensures that the rooms, where the hero is and where he wants to go, have a corridor between them.

```
(valid_move ?from ?to)
```

The next two precondition ensures that none of the heroes has visited these rooms before. It is necessary to check the room where the hero is, because in problem 2 the hero arrives in room 6, takes the sword and then has two options: $room\ 2 \rightarrow room\ 5$ or $room\ 6 \rightarrow room\ 5$. However, the hero has already visited room 2. Therefore, the $room\ 2 \rightarrow room\ 5$ move must be blocked.

```
(not (visited_room ?from))
(not (visited_room ?to))
```

This action ensures that the hero does not leave the room with a trap armed.

```
(not (trap_armed ?from))
```

This action allows the hero to move if the target room does not have a monster.

```
(not (monster_angry ?from))
```

If the target room has a monster, the hero can only move if he hold a sword. This precondition is best explained by the truth table below, where M indicates that the target room has a monster and E indicates that the hero is empty-handed.

M	E	$\neg(M \wedge E)$
1	1	0
1	0	1
0	1	1
0	0	1

As shown in the truth table above, the only case where the hero cannot move to the target room is when there is a monster in the target room and the hero is empty-handed. This truth table is represented as the following precondition:

```
(not (and
    (monster_angry ?to)
    (empty_handed ?hero)
))
```

If all of these precondition are true, the hero can move to the target room. In doing so, this action has two effects on the state: the hero moves to the target location and the room where the hero was in is marked as visited. Therefore, no heroes can pass through this room again.

```
:effect (and
    (hero_at ?hero ?to)
    (visited_room ?from)
)
```

`:action get_sword`

This action will be triggered when a hero is empty-handed and enters a room that has a sword. The sword is needed to cross the room with the monster. Therefore, as I ensure that the hero will not enter a room with a monster and empty-handed, at some point the hero will need to get the sword. This action receives as parameters the hero who wants to get the sword and the hero's location, as shown below.

```
:parameters (?hero - hero ?location - room)
```

To perform this action the hero must to be at the location of the sword, as shown in the two preconditions below.

```
(hero_at ?hero ?location)
(sword_at ?location)
```

The hero is not allowed to hold two swords at the same time. Therefore, the hero can only get the sword if he is empty-handed. For the problem, this precondition is useless. However, I used it to maintain a consistency in the problem solution.

```
(empty_handed ?hero)
```

In the problem 4, there is a point where the hero has two options: get the sword in room 3 or room 6. However, the hero has already visited room 6. He cannot go there again. This violates the rules of the problem. Therefore, it is necessary to test in this action whether the hero has already visited the room or not.

```
(not (visited_room ?location))
```

If all these precondition are true, the hero can get the sword to face the monster. The only effect of this action is that the hero is no longer empty-handed.

```
:effect (and
    (not (empty_handed ?hero))
)
```

`:action drop_sword`

The hero cannot disarm a trap if he is not empty-handed. Therefore, to perform this action, it is necessary to know which hero will drop the sword and the location the trap is. By the way, these are the two parameters that the action receives.

```
:parameters (?hero - hero ?location - room)
```

Therefore, if the hero is where the trap is and is not empty-handed, he will need to drop the sword. The effect of this action is that the hero is empty-handed at the end. The two parts of the problem formalization below show the preconditions and effects of this action, respectively.

```
(hero_at ?hero ?location)
(not (empty_handed ?hero))
(trap_armed ?location)
```

```
:effect (and
    (empty_handed ?hero)
)
```

`:action hug_monster`

This is the most paradoxical action in the formalization of the problem and I explain why. The purpose of this action is to calm the angry monster. Since the hero always faces the monster with a sword in hands, he could just kill the monster and move on. However, with the kindest attitude, the hero hugs the monster. Moments after the hero leaves the room, the room is destroyed with the monster inside, though. Go figure!

Leaving this little digression behind, this action receives as parameters the hero and the location of the monster. As preconditions, the hero must be at the monster's location, the monster must be angry, and the hero cannot be empty-handed.

```
:parameters (?hero - hero ?location - room)
```

```
(hero_at ?hero ?location)
(monster_angry ?location)
(not (empty_handed ?hero))
```

As mentioned earlier, the effect of this action is to calm the angry monster.

```
:effect (and
    (not (monster_angry ?location))
)
```

`:action disarm_trap`

The purpose of this action is to disarm the trap. To do this, the hero must be at the trap location, the trap must be armed, and the hero must be empty-handed. If the hero is holding a sword, he will first perform the action of dropping the sword and then disarming the trap.

```
:parameters (?hero - hero ?location - room)
```

```
(hero_at ?hero ?location)
(trap_armed ?location)
(empty_handed ?hero)
```

As effect of this action, the trap is disarmed and the hero can move on.

```
:effect (and
    (not (trap_armed ?location))
)
```

Scenarios

The domain formalization was tested in five scenarios. Four scenarios have one hero trying get out of the dungeon, and only the scenario five has two heroes trying to achieve different goals. In the simulation with two heroes, I assume that two rooms can have an exit. However, scenario five will also work if the two heroes need to leave on a common exit.

There is not much to explain about each scenario. Therefore, I just show the scenarios as tables, where each cell has the room number and the object that is there.

Below is the legend of the symbols used in the tables and, following, the five scenarios.

Symbol	Description
G	Goal. G can be followed by a number to indicate the hero who needs to reach that goal.
M	Monster.
T	Trap.
H	Hero's starting position. H can be followed by a number to indicate the hero in that room.
S	Sword.
E	Empty.
	Inaccessible room.

Scenario 1

1 - H	2 - E	3 - M
4 - M	5 - E	6 - G

Scenario 2

1 - M	2 - H	3 - E
4 - G	5 - M	6 - S

Scenario 3

1 - H	2 - M	3 - E	
4 - S	5 - M	6 - T	7 - G

Scenario 4

1 - H		3 - S	4 - M	5 - T
6 - S	7 - M	8 - T		10 - G

Scenario 5

1 - H1	2 - S	3 - M	4 - S	5 - M
6 - M		8 - M	9 - G1	10 - T
11 - G2	12 - T	13 - M	14 - M	15 - M
16 - M	17 - M	18 - S	19 - E	20 - H2

Performance

To assess the performance of the proposed formalization, I used the WEB PLANNER, available at https://web-planner.herokuapp.com. The five different scenarios were evaluated considering the time to solve the plan, the tree height, and the number of states visited until reaching the goal. The solution time averages five plans.

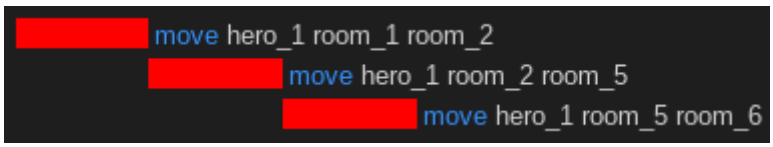
The state-space was explored using the Hamming distance. In resume, the Hamming distance heuristic indicates how many positions the current state differs from the goal state. WEB PLANNER also supports the breadth-first search algorithm. When comparing the Hamming distance heuristic and the breadth-first search, the number of states visited is different only for scenario five.

Therefore, the following table shows the time to solve the plan, the tree height, and the states visited for the first four scenarios, using the Hamming distance. The fifth scenario is discussed in more detail in the **Performance with Multiple Heroes** section.

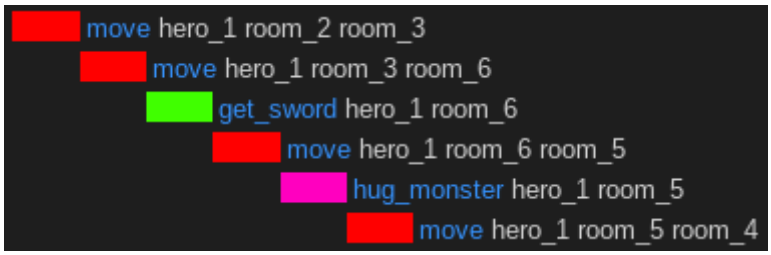
Scenario	Time (seconds)	Tree height	States visited
1	0.00126	3	7
2	0.00132	6	10
3	0.00168	8	18
4	0.0018	15	18

The movements needed to reach the goal are shown below.

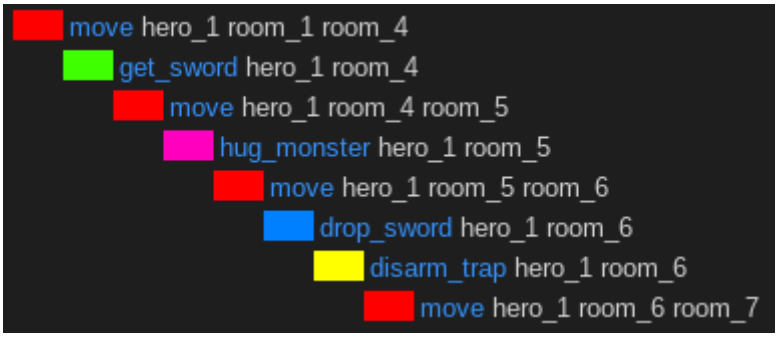
Movements in scenario 1



Movements in scenario 2



Movements in scenario 3



Movements in scenario 4

Performance with Multiple Heroes

As stated before, scenario 5 is the only one with several heroes trying to achieve different goals. As shown in the **Scenarios** section, the number of rooms is greater than in other scenarios. Hence, the state-space will also be larger. Unlike other scenarios, in this scenario, a hero can achieve his goal, but he continues to search a global state in which all heroes find their goal. For instance, to find the goal using the Hamming distance, it was necessary to visit 519 states. Using breadth-first search, it was necessary to visit 1581 states.

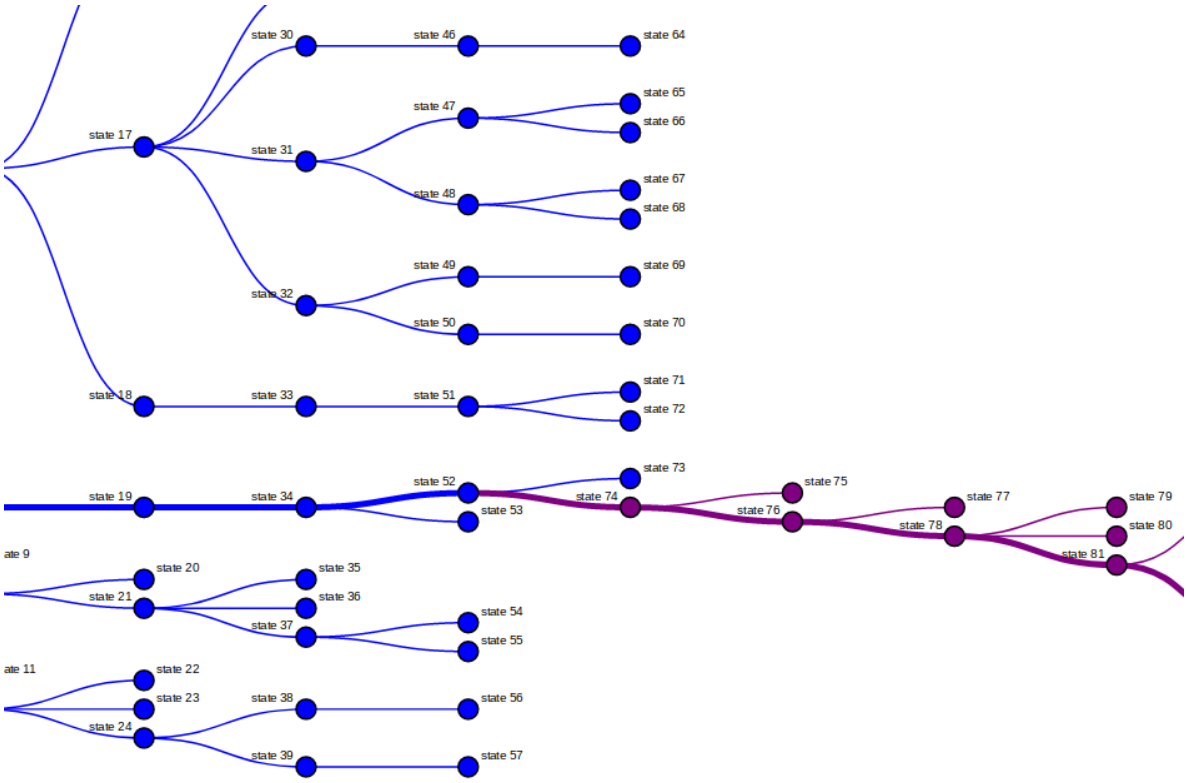
However, when a hero reaches his goal, he can interrupt his search and let only other heroes searching. Using this approach, it is possible to reduce the state-space. Therefore, to the action `move` the next precondition was added.

```
(not (goal_at ?hero ?from))
```

This precondition uses the `goal_at` predicate, which is instantiated in the problem description. Now, to move forward, the hero must not yet have reached his goal. With this approach, the time to solve the plan was of 0.0498 seconds, while with the approach without the goal precondition, the average time was 0.07344 seconds. A reduction of 32.19%. Other results are as follows:

	Tree height	States visited	Reduction in states visited (%)
Breadth-first search	14	927	41.37
Hamming distance	14	102	80.35

In the tree of states below, it is possible to see the point (state 74) at which a hero achieves his goal, and the search focuses only on finding the goal of other heroes.



As a result, scenario 5 requires the following moves to achieve each hero's goal.

Conclusion

The above discussion showed that the proposed formalization can deal with multiple heroes, achieving the same or different goals, in the same problem. However, the problem addressed is simple. For example, we can imagine a scenario in which the distance between the rooms is different or until some rooms have stairs to go up and enter the room. In such cases, the hero's movements would have a cost, and instead of making naive moves, the hero would need to choose the best path.

Therefore, this formalization can be problematic in the sense of finding a local minimum and interrupting the search. The ideal would be to reach the global minimum. Hence, looking only at the scenarios covered in this work, the approach that uses the goal of each hero as a precondition can provide a reduction in the search space and time to solve the plan.

References

Haslum, P., Lipovetzky, N., Magazzeni D., Muise, and C. 2019. An Introduction to the Planning Domain Definition Language. Reading, Mass.: Morgan & Claypool.

Miscellaneous

- This project, including the PDDL files and this document, is available at https://github.com/claudioscheer/pddl-rpg-domain.
- As a development tool, Visual Studio Code with extension `vscode-pddl` was used.