Dungeon Resolver

Davide Leone - Mattia Zavaglio

May 07, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION

Dungeon Resolver is a project developed initially as a university assignment for the “Intelligent Systems” course, focusing on Automated Planning. The project utilizes the unified-planning library to implement a planner and simulator for resolving randomly generated dungeon instances.

1.1 Why Automated Planning?

Automated Planning is a field within Artificial Intelligence (AI) that deals with designing algorithms and systems capable of generating plans to achieve specific goals. In the context of Dungeon Resolver, the objective is to create a system that can navigate through a randomly generated dungeon, overcoming obstacles, enemies, and collecting resources along the way.

1.2 Features

- **Random Dungeon Generation**: The program creates random dungeon instances with various rooms containing weapons, enemies, potions, and loot. This ensures each run provides a unique challenge.
- **Unified-Planning Planner**: Dungeon Resolver employs the unified-planning library to implement the planner. The planner, named “enhsp”, is responsible for generating a plan to navigate through the dungeon efficiently.
- **Plan Representation**: Once a plan is generated, it is represented visually using a graphical user interface (GUI). The GUI simulates a 2D top-down game, providing a clear visualization of the plan in action.

1.3 Usage

To use Dungeon Resolver, follow these steps:

1. Clone the repository to your local machine.
2. Install the necessary dependencies, including the unified-planning library
3. Run the program and specify the parameters for dungeon generation.
4. The planner will then generate a plan (if exits) for navigating through the dungeon.
5. At the user’s choice plan will be displayed in the GUI, allowing the user to visualize the execution steps.
6. At the user’s choice the dungeon structure will be plotted using networkx graph.

1.4 Contributors

- Davide Leone
- Mattia Zavaglio

1.5 Licenses

Everything used inside the project is free license!

1.6 Acknowledgments

The unified-planning library contributors for providing the planner.

CHAPTER

TWO

GETTING STARTED

In this guide we present the main functionalities offered by Dungeon Resolver.

2.1 What is Dungeon Resolver?

Dungeon Resolver is a project developed initially as a university assignment for the "Intelligent Systems" course, focusing on Automated Planning. The project utilizes the unified-planning library to implement a planner and simulator for resolving randomly generated dungeon instances.

Unified-planning reads the pddl dungeon domain file, that define the dungeon's structure, specifying the problem types (object), predicates (fluents), functions (numerics) and the actions that a hero can performe inside the dungeon.

Note: A dungeon is made up of rooms, connected to each other, sometimes separated by closed doors: inside them it's possible to find keys to open doors, treasures to collect, enemies to defeat, weapons to fight them and healing potions. All these items (except keys) can have different values (E.g. different strength of enemies or more valuable treasures). In an abstract way it's possible to represent the dungeon as an undirected graph.

Instead, for the pddl dungeon instance file is possible to choose one of the following option:

- **Generate and solve a random dungeon instance**: Dungeon Resolver generates a random pddl dungeon instance file after the user has specified the desired number of rooms and the seed for the random functions. Then calls a unified-planning function to solve the problem. Finally it's possible to run the dungeon GUI and to view the abstract graphical representation of the dungeon.
- **Solve an already existing dungeon instance**: It's also possibile to read an already existing pddl dungeon instance file and to call the unified-planning function to solve the problem, printing the result. In this case the GUI is not available.

Note:

To solve the problem hero needs to:

- survive (hero life always greater than zero)
- collect at least a pre-established percentage of treasures
- defeat at least a pre-established percentage of enemies
- reach the exit room and escape from dungeon

2.2 Installation

To get started with Dungeon Resolver, you can follow these steps:

1. Clone the repository:

```
git clone https://github.com/scrapanzano/dungeon_resolver.git
```

2. Install dependencies:

```
cd .\dungeon_resolver
pip install -r requirements.txt [Windows OS]
pip3 install -r requirements.txt [Mac OS]
```

3. Run the program:

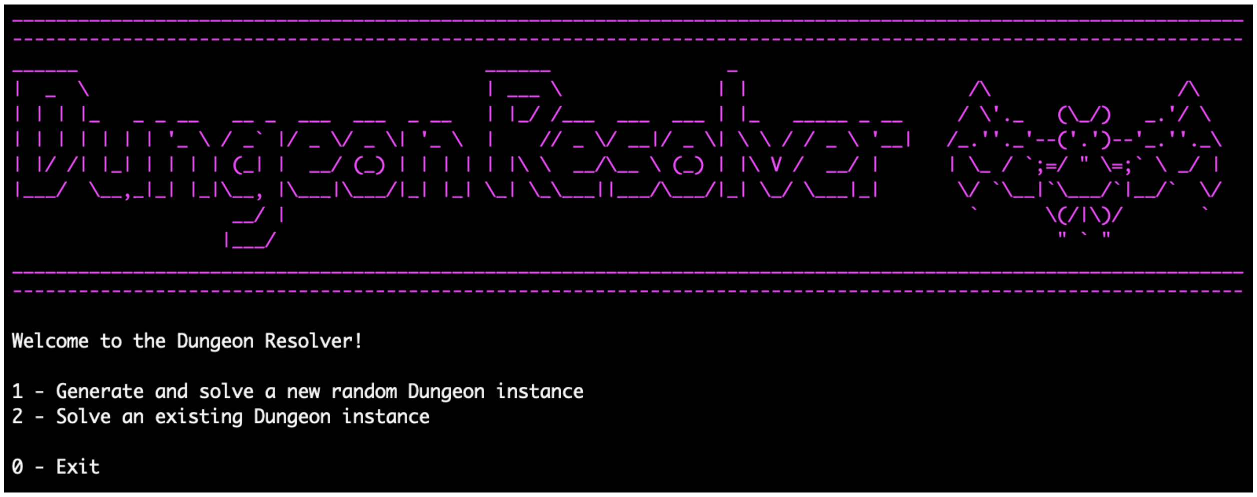
```
python3 .\dungeon_resolver\generate_dungeon_problem.py
```

4. Follow the on-screen instructions to specify dungeon parameters and visualize the plan.

2.3 Quickstart

This guide shows the usage of Dungeon Resolver.

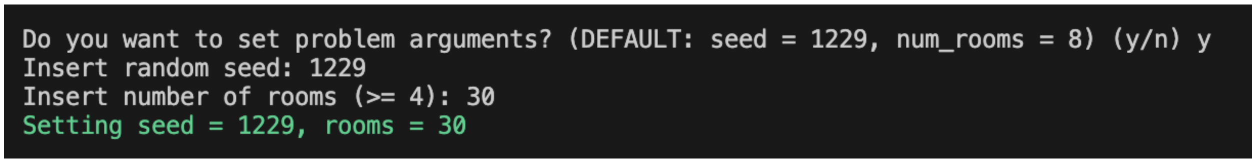
2.3.1 Main Menu



```
————————————————————————————————————————————————————————————————————————————————————————————————————————————————
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

______                                    ______                _
|  _  \                                   | ___ \              | |                   /\                 /\
| | | |_   _ _ __   __ _  ___  ___  _ __  | |_/ /___  ___  ___ | |_   _____ _ __       / \'._   (\_/)   _.'/ \
| | | | | | | '_ \ / _` |/ _ \/ _ \| '_ \ |    // _ \/ __|/ _ \| \ \ / / _ \ '__|     /_.''._'--('.')--'_.''._\
| |/ /| |_| | | | | (_| |  __/ (_) | | | || |\ \  __/\__ \ (_) | |\ V /  __/ |        | \_ / `;=/ " \=;` \ _/ |
|___/  \__,_|_| |_|\__, |\___|\___/|_| |_|\_| \_\___||___/\___/|_| \_/ \___|_|         \/ `\__|`\___/`|__/`  \/
                    __/ |                                                                `      \(/|\)/       `
                   |___/                                                                          " ` "

————————————————————————————————————————————————————————————————————————————————————————————————————————————————
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Welcome to the Dungeon Resolver!

1 - Generate and solve a new random Dungeon instance
2 - Solve an existing Dungeon instance

0 - Exit
```

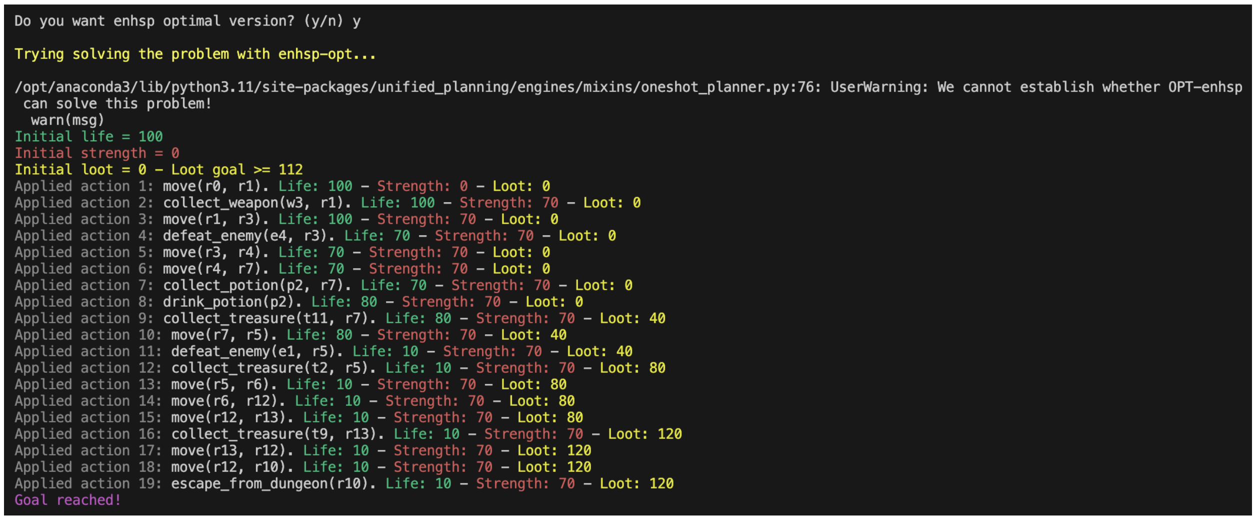
1 - Generate and solve a new random Dungeon instance

Initially it’s possible to choose whether to set the problem arguments or use the default parameters:



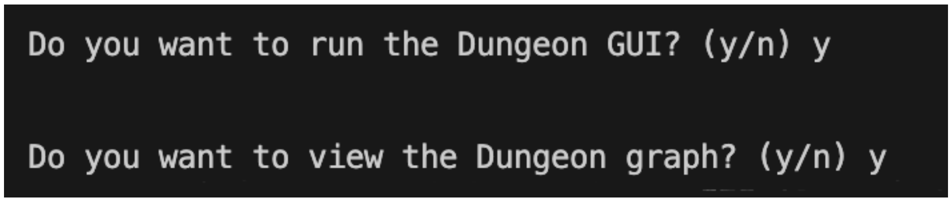
```
Do you want to set problem arguments? (DEFAULT: seed = 1229, num_rooms = 8) (y/n) y
Insert random seed: 1229
Insert number of rooms (>= 4): 30
Setting seed = 1229, rooms = 30
```

Then it’s possible to choose whether to run the optimal version of unified-planning planner, that will solve the problem and print the result:

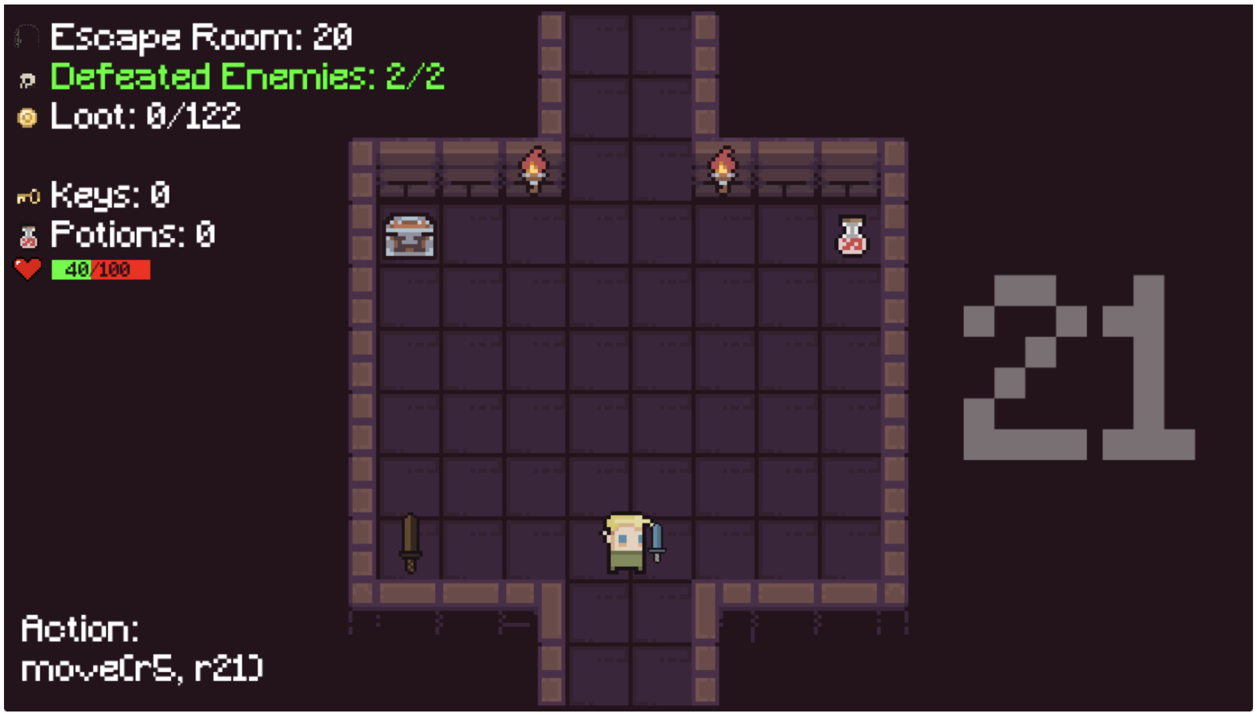


```
Do you want enhsp optimal version? (y/n) y

Trying solving the problem with enhsp-opt...

/opt/anaconda3/lib/python3.11/site-packages/unified_planning/engines/mixins/oneshot_planner.py:76: UserWarning: We cannot establish whether OPT-enhsp
 can solve this problem!
  warn(msg)
Initial life = 100
Initial strength = 0
Initial loot = 0 - Loot goal >= 112
Applied action 1: move(r0, r1). Life: 100 - Strength: 0 - Loot: 0
Applied action 2: collect_weapon(w3, r1). Life: 100 - Strength: 70 - Loot: 0
Applied action 3: move(r1, r3). Life: 100 - Strength: 70 - Loot: 0
Applied action 4: defeat_enemy(e4, r3). Life: 70 - Strength: 70 - Loot: 0
Applied action 5: move(r3, r4). Life: 70 - Strength: 70 - Loot: 0
Applied action 6: move(r4, r7). Life: 70 - Strength: 70 - Loot: 0
Applied action 7: collect_potion(p2, r7). Life: 70 - Strength: 70 - Loot: 0
Applied action 8: drink_potion(p2). Life: 80 - Strength: 70 - Loot: 0
Applied action 9: collect_treasure(t11, r7). Life: 80 - Strength: 70 - Loot: 40
Applied action 10: move(r7, r5). Life: 80 - Strength: 70 - Loot: 40
Applied action 11: defeat_enemy(e1, r5). Life: 10 - Strength: 70 - Loot: 40
Applied action 12: collect_treasure(t2, r5). Life: 10 - Strength: 70 - Loot: 80
Applied action 13: move(r5, r6). Life: 10 - Strength: 70 - Loot: 80
Applied action 14: move(r6, r12). Life: 10 - Strength: 70 - Loot: 80
Applied action 15: move(r12, r13). Life: 10 - Strength: 70 - Loot: 80
Applied action 16: collect_treasure(t9, r13). Life: 10 - Strength: 70 - Loot: 120
Applied action 17: move(r13, r12). Life: 10 - Strength: 70 - Loot: 120
Applied action 18: move(r12, r10). Life: 10 - Strength: 70 - Loot: 120
Applied action 19: escape_from_dungeon(r10). Life: 10 - Strength: 70 - Loot: 120
Goal reached!
```

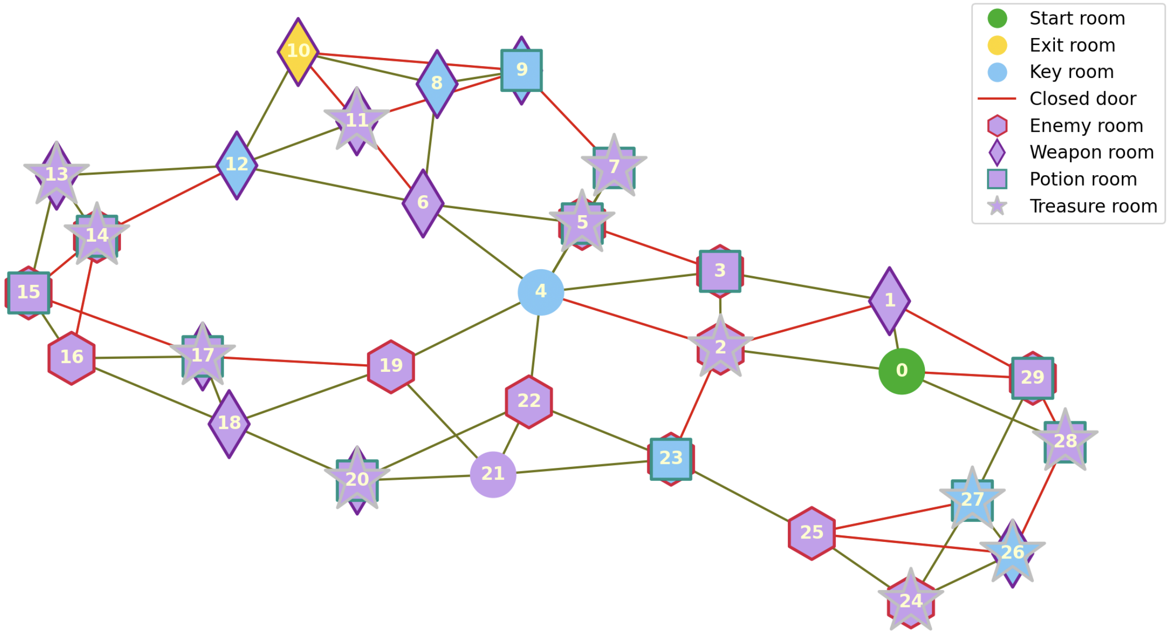
Finally it’s possible to choose whether to run the dungeon GUI and view the abstract graphical representation of the dungeon:



```
Do you want to run the Dungeon GUI? (y/n) y

Do you want to view the Dungeon graph? (y/n) y
```

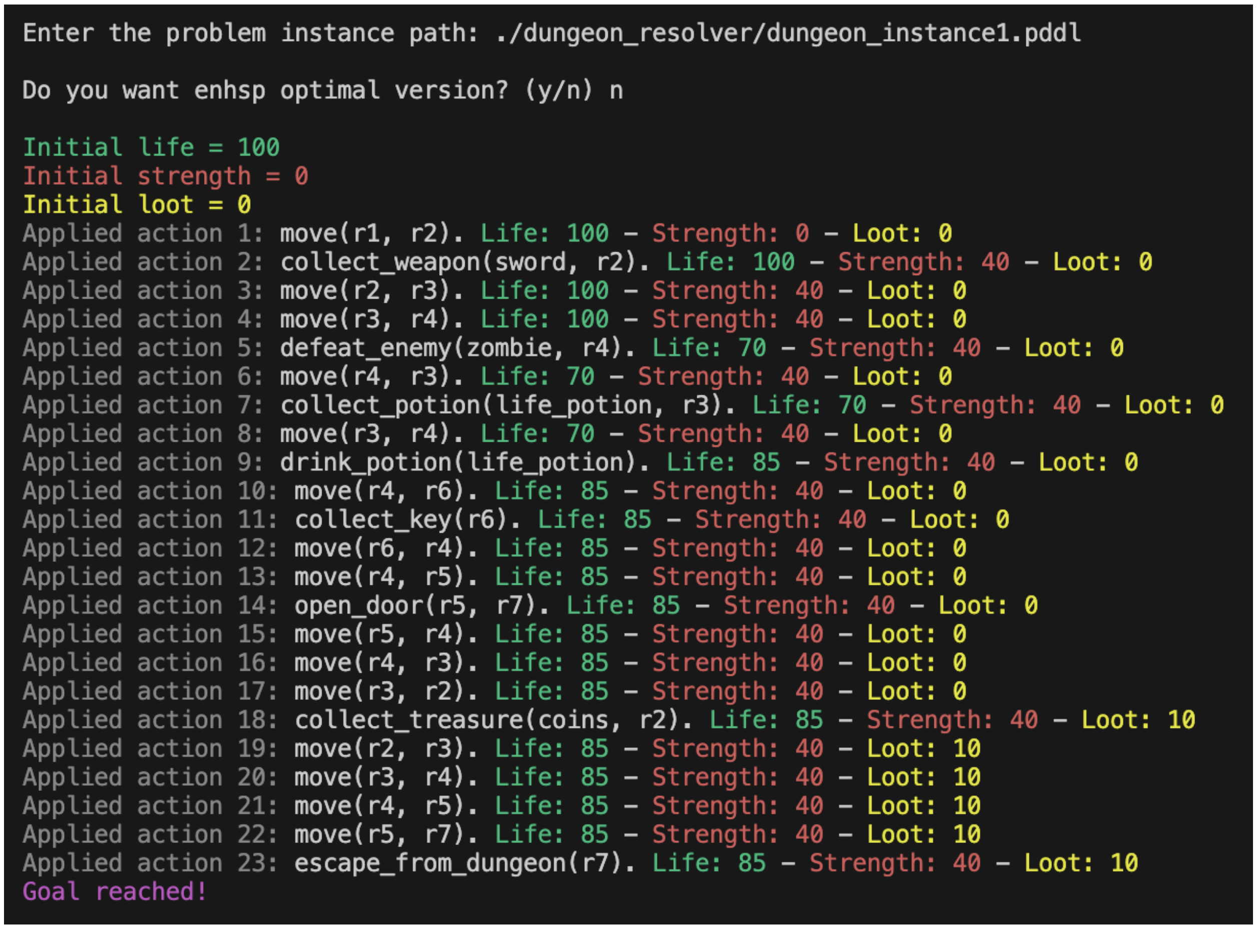


Escape Room: 20
Defeated Enemies: 2/2
Loot: 0/122
Keys: 0
Potions: 0
40/100
Action:
move(r5, r21)



2 - Solve an existing Dungeon

It's also possibile to read an already existing pddl dungeon instance file, specifying its path, and to call the unified-planning function to solve the problem, printing the result (in this case the GUI is not available):



```
Enter the problem instance path: ./dungeon_resolver/dungeon_instance1.pddl

Do you want enhsp optimal version? (y/n) n

Initial life = 100
Initial strength = 0
Initial loot = 0
Applied action 1: move(r1, r2). Life: 100 - Strength: 0 - Loot: 0
Applied action 2: collect_weapon(sword, r2). Life: 100 - Strength: 40 - Loot: 0
Applied action 3: move(r2, r3). Life: 100 - Strength: 40 - Loot: 0
Applied action 4: move(r3, r4). Life: 100 - Strength: 40 - Loot: 0
Applied action 5: defeat_enemy(zombie, r4). Life: 70 - Strength: 40 - Loot: 0
Applied action 6: move(r4, r3). Life: 70 - Strength: 40 - Loot: 0
Applied action 7: collect_potion(life_potion, r3). Life: 70 - Strength: 40 - Loot: 0
Applied action 8: move(r3, r4). Life: 70 - Strength: 40 - Loot: 0
Applied action 9: drink_potion(life_potion). Life: 85 - Strength: 40 - Loot: 0
Applied action 10: move(r4, r6). Life: 85 - Strength: 40 - Loot: 0
Applied action 11: collect_key(r6). Life: 85 - Strength: 40 - Loot: 0
Applied action 12: move(r6, r4). Life: 85 - Strength: 40 - Loot: 0
Applied action 13: move(r4, r5). Life: 85 - Strength: 40 - Loot: 0
Applied action 14: open_door(r5, r7). Life: 85 - Strength: 40 - Loot: 0
Applied action 15: move(r5, r4). Life: 85 - Strength: 40 - Loot: 0
Applied action 16: move(r4, r3). Life: 85 - Strength: 40 - Loot: 0
Applied action 17: move(r3, r2). Life: 85 - Strength: 40 - Loot: 0
Applied action 18: collect_treasure(coins, r2). Life: 85 - Strength: 40 - Loot: 10
Applied action 19: move(r2, r3). Life: 85 - Strength: 40 - Loot: 10
Applied action 20: move(r3, r4). Life: 85 - Strength: 40 - Loot: 10
Applied action 21: move(r4, r5). Life: 85 - Strength: 40 - Loot: 10
Applied action 22: move(r5, r7). Life: 85 - Strength: 40 - Loot: 10
Applied action 23: escape_from_dungeon(r7). Life: 85 - Strength: 40 - Loot: 10
Goal reached!
```

CHAPTER THREE

API REFERENCE

3.1 Dungeon Resolver

3.1.1 Generate_dungeon_problem

This module allows to create a random dungeon pddl problem file, that describe the specific instance of the dungeon problem, or to import an existing dungeon pddl problem file, and to resolve them with the unified_plannig library

`generate_dungeon_problem.`**`Main`**`()`

Main function: generates and manages a user menu. It's possible to:

- Generate and solve a random problem instance, also invoking the GUI
- Read and solve a specified problem instance

`generate_dungeon_problem.`**`farthest_node`**`(`*G*, *start_room*`)`

Returns the farthest node from start_room inside the graph G

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

param start_room
Selected starting room

type start_room
int

Returns

returns
Farthest node from start_room

rtype
int

`generate_dungeon_problem.`**`generate_doors`**`(`*G*`)`

Generates links between rooms, setting graph edges as normal or door link.

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

`generate_dungeon_problem.`**`generate_enemies`**`(`*G*, *start_room*, *num_enemy_rooms*`)`

Generates enemies in rooms and returns rooms with enemy

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

param start_room
Selected starting room

type start_room
int

param num_enemy_rooms
Desired number of rooms with enemy

type num_enemy_rooms
int

Returns

returns
Dict of rooms with enemy [format: {room : enemy_value(life/strength)}]

rtype
dict

`generate_dungeon_problem.`**`generate_instance`**`(`*instance_name*, *num_rooms*`)`

Generates a random instance of the dungeon problem, starting from a pddl template file. In this function there are:

- Creation of a graph with networkx representing the dungeon
- Generation of all elements inside the dungeon (doors, keys, treasures, enemies, weapons, potions)
- Population and writing of the pddl template file with the previous elements
- Invocation of unified-planning planner to solve the problem
- Running the dungeon GUI
- Drawing the schematic representation (graph) of the dungeon whit matplotlib

Parameters

param instance_name
Instance name for the pddl problem file

type instance_name
str

param num_rooms
Number of dungeon rooms

type num_rooms
int

`generate_dungeon_problem.`**`generate_keys`**`(`*G*, *start_room*, *exit_room*`)`

Generates keys in rooms and returns a list of rooms with key

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

param start_room
Selected starting room

type start_room
int

param exit_room
Selected exit room

type exit_room
int

Returns

returns
List of rooms with key

rtype
list

`generate_dungeon_problem.`**`generate_loot_goal`**`(`*treasure_rooms*, *loot_rate*`)`

Generates and returns loot goal

Parameters

param treasure_rooms
Dict of rooms with treasure

type treasure_rooms
dict

param loot_rate
Selected loot rate

type loot_rate
float

Returns

returns
Loot goal value

rtype
int

`generate_dungeon_problem.`**`generate_potions`**(*G*, *start_room*, *num_potion_rooms*)

Generates potions in rooms and returns rooms with potion

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

param start_room
Selected starting room

type start_room
int

param num_potion_rooms
Desired number of rooms with potion

type num_potion_rooms
int

Returns

returns
Dict of rooms with potion [format: {room : potion_value}]

rtype
dict

`generate_dungeon_problem.`**`generate_treasures`**(*G*, *start_room*, *num_treasure_rooms*)

Generates treasures in rooms and returns rooms with treasure

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

param start_room
Selected starting room

type start_room
int

param num_treasure_rooms
Desired number of rooms with treasure

type num_treasure_rooms
int

Returns

returns
Dict of rooms with treasure [format: {room : treasure_value}]

rtype
dict

`generate_dungeon_problem.`**`generate_weapons`**(*G*, *start_room*, *enemy_rooms*)

Generates weapons in rooms and returns rooms with weapon

Parameters

param G
Graph on which calculate farthest node from start_room

type G
networkx Graph

param start_room
Selected starting room

type start_room
int

param num_enemy_rooms
Number of rooms with enemy

type num_enemy_rooms
int

Returns

returns
Dict of rooms with weapon [format: {room : weapon_strength}]

rtype
dict

`generate_dungeon_problem.`**`invoke_unified_planning`**`(`*path*`)`

Invokes unified_planning to read and solve the instance file specified in path

Parameters

param path
Pddl problem instance file path

type path
str

`generate_dungeon_problem.`**`yes_or_no`**`(`*question*`)`

Choices between yes or not (y/n)

Parameters

param question
A yes or no question

type question
str

Returns

returns
True if yes chosen, False otherwise

rtype
bool

3.1.2 GUI

This module manages the project dungeon_gui

class `GUI.`**`GUI`**`(`*problem*, *result*, *rooms*`)`

Bases: `object`

This class manages the project GUI

`run``()`

Run the GUI

`GUI.`**`enter_room`**`(`*player*, *screen*, *room*, *hud*`)`

Updates room rendering when player enters the room

Parameters

param player
Player object

type player
Player

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param room
Room the player is entering

type room
Room

param hud
HUD object

type hud
HUD

`GUI.`**`exit_room`**`(`*player*, *screen*, *room*, *hud*`)`

Updates room rendering when player leaves room

Parameters

param player
Player object

type player
Player

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param room
Room the player is exiting

type room
Room

param hud
HUD object

type hud
HUD

`GUI.`**`fluent_to_int`**`(`*state*, *fluent*`)`

Converts unified_planning Fluent to int

Parameters

param state
Object representing the state of the problem

type state
unified_planning.shortcuts.State

param fluent
Object representing a fluent

type fluent
unified_planning.shortcuts.FluentExp

Returns

returns
The value of the fluent as an integer

rtype
int

`GUI.`**`update_hud`**(*hud*, *state*, *hero_loot*, *key_counter*, *potion_counter*, *actual_room_id*, *action*, *defeated_enemy_counter=None*, *is_exit=False*)

Updates all hud variables

Parameters

param hud
HUD object

type hud
HUD

param state
Object representing the state of the problem

type state
unified_planning.shortcuts.State

param hero_loot
Object representing the hero loot fluent

type hero_loot
unified_planning.shortcuts.FluentExp

param key_counter
Object representing the key counter fluent

type key_counter
unified_planning.shortcuts.FluentExp

param potion_counter
Object representing the potion counter fluent

type potion_counter
unified_planning.shortcuts.FluentExp

param actual_room_id
The id of the actual room

type actual_room_id
int

param action
The last action executed

type action
str

param defeated_enemy_counter
Object representing the defeated enemy counter fluent

type defeated_enemy_counter
unified_planning.shortcuts.FluentExp

param is_exit
Flag to check if the actual room is the exit room

type is_exit
bool

3.2 Dungeon GUI

3.2.1 Collectable

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

class `collectable.`**`Collectable`**

Bases: `object`

This is the superclass for all Collectable Objects

`collect()`

Sets the collected attribute for the Object to True

`render_collectable()`

Rendering of Collectable Object, implemented in all Collectable Class

3.2.2 Constants

This module is part of the dungeon_gui package, for the graphical representation of the dungeon. Collects some useful constants for other modules.

3.2.3 Enemy

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

`class Enemy.`**`Enemy`**`(`*damage*, *enemy_tileset=<Surface(512x256x32 SW)>*`)`

Bases: `object`

This class describes the representation of the Enemy Object

`kill``()`

Sets the killed attribute to True

`render_enemy``(`*screen*, *room_x*, *room_y*, *scale_factor*`)`

Rendering of the Enemy Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param room_x
X Room position on the screen

type room_x
int

param room_y
Y Room position on the screen

type room_y
int

param scale_factor
Object scale factor

type scale_factor
int

3.2.4 Health_bar

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

`class health_bar.`**`HealthBar`**`(`*blink_counter*, *x=50*, *y=260*, *max_health=100*, *current_health=100*`)`

Bases: `object`

This class describes the representation of the Health Bar Object

`draw``(`*screen*`)`

Draws the Health Bar Object on the screen

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

`update_health`(*health*)

Updates current_health, blinking and hp_text attributes

Parameters

param health
Hero health value

type health
int

3.2.5 Hud

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

`class hud.HUD`(*escape_room=0*, *hero_loot=0*, *hero_loot_goal=0*, *key_counter=0*, *potion_counter=0*, *room_id=0*, *defeated_enemy_counter=0*, *defeated_enemy_counter_goal=0*, *action=''*)

Bases: `object`

This class describes the representation of the HUD

`create_alpha_surface`(*text_surface*, *alpha_value*, *is_exit*)

Creates an alpha surface for HUD id representation

Parameters

param text_surface
Text Surface

type text_surface
pygame Surface

param alpha_value
Alpha value for the Surface

type alpha_value
int

Returns

returns
A Surface for the room id in HUD

rtype
pygame Surface

`render`(*screen*)
Rendering all HUD Object elements on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

`update_action`(*action*)
Updates action attribute and its HUD representation

Parameters

param action
Hero action performed

type action
str

`update_defeated_enemy_counter`(*defeated_enemy_counter*)
Updates defeated_enemy_counter attribute and its HUD representation

Parameters

param defeated_enemy_counter
Number of defeated enemies

type defeated_enemy_counter
int

`update_escape_room`()
Updates escape_room attribute and its HUD representation

`update_hero_loot`(*hero_loot*)
Updates hero_loot attribute and its HUD representation

Parameters

param hero_loot
Hero loot value

type hero_loot
int

`update_id`(*new_id*, *is_exit*)

Updates id attribute and its HUD representation

Parameters

param new_id
Number of room visited

type new_id
int

param is_exit
Boolean that indicates if the room is an exit

type is_exit
bool

`update_keys`(*keys*)

Updates keys attribute and its HUD representation

Parameters

param keys
Number of keys owned

type keys
int

`update_potions`(*potions*)

Updates potions attribute and its HUD representation

Parameters

param potions
Number of potions owned

type potions
int

3.2.6 Key

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

class `Key.`**`Key`**`(`*key_tileset=<Surface(160x160x32 SW)>*`)`

Bases: `Collectable`

This class describes the representation of the Key Object

`render_collectable``(`*screen*, *scale_factor*`)`

Rendering of the Key Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param scale_factor
Object scale factor

type scale_factor
int

3.2.7 Loot

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

class `Loot.`**`Loot`**`(`*loot_value: int*, *loot_tileset=<Surface(160x160x32 SW)>*`)`

Bases: `Collectable`

This class describes the representation of the Treasure Object

`render_collectable``(`*screen*, *scale_factor*`)`

Rendering of the Treasure Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param scale_factor
Object scale factor

type scale_factor
int

3.2.8 Player

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

`class Player.`**`Player`**`(`*current_health=100*, *max_health=100*, *pos_x=9.4*, *pos_y=8*, *character_tileset=<Surface(16x16x32 SW)>*, *weapon=None*, *potion=None*`)`

Bases: `object`

This class describes the representation of the Hero Player

`collect_potion``(`*potion*`)`

Calls the function to collect the potion

Parameters

param potion
Potion object

type potion
Potion

`get_damage``(`*damage*`)`

Manages the damage taken by the hero, updating his health

Parameters

param damage
Damage value

type damage
int

`get_heal``()`

Manages the hero’s health, when healing himself

Parameters

param heal
Heal value

type heal
int

`render_player``(`*screen*, *scale_factor*`)`

Rendering of the Player Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param scale_factor
Object scale factor

type scale_factor
int

`update_health`(*health*)

Updates current_health attribute and health_bar

Parameters

param health
Hero healt value

type health
int

`update_weapon`(*new_damage*)

Calls the function to update the weapon’s damage

Parameters

param new_damage
Weapon damage value

type new_damage
int

3.2.9 Potion

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

`class Potion.Potion`(*potion_value: int*, *potion_tileset=<Surface(512x256x32 SW)>*)

Bases: `Collectable`

This class describes the representation of the Potion Object

`render_collectable`(*screen*, *scale_factor*)

Rendering of the Potion Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param scale_factor
Object scale factor

type scale_factor
int

3.2.10 Room

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

`class Room.`**`Room`**`(`*id*, *key=None*, *loot=None*, *enemy=None*, *weapon=None*, *potion=None*, *width=160*, *height=176*, *has_door=False*, *is_exit=False*, *x=0*, *y=0*`)`

Bases: `object`

This class describes the representation of the Room

`collect_key``()`
Calls the function to set key's collected attribute

`collect_potion``()`
Calls the function to set potion's collected attribute

`collect_treasure``()`
Calls the function to set treasure's collected attribute

`collect_weapon``()`
Calls the function to set weapon's collected attribute

`defeat_enemy``()`
Calls the function to set enemy's killed attribute

`generate_tile_mapping``()`
Generates a tile mapping for the room representation

Returns

returns
A dict containing the tile mapping

rtype
dict

`render``(`*screen*`)`
Rendering of the Room Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

`set_enemy`(*enemy*)

Sets the value of enemy

Parameters

param enemy
Enemy Object to set

type enemy
Enemy

`set_key`(*key*)

Sets the value of key

Parameters

param key
Key Object to set

type key
Key

`set_loot`(*loot*)

Sets the value of loot

Parameters

param loot
Loot Object to set

type loot
Loot

`set_potion`(*potion*)

Sets the value of potion

Parameters

param potion
Potion Object to set

type potion
Potion

`set_weapon`(*weapon*)
Sets the value of weapon

Parameters

param weapon
Weapon Object to set

type weapon
Weapon

3.2.11 Weapon

This module is part of the dungeon_gui package, for the graphical representation of the dungeon

class `Weapon.Weapon`(*damage=0*, *weapon_tileset=<Surface(512x256x32 SW)>*, *weapon_pos_x=13.5*, *weapon_pos_y=18.5*)

Bases: `Collectable`

This class describes the representation of the Weapon Object

`render_collectable`(*screen*, *scale_factor*)
Rendering of the Weapon Object on the screen using a tile set

Parameters

param screen
Screen where dungeon_gui runs

type screen
pygame Surface

param scale_factor
Object scale factor

type scale_factor
int

`update_damage`(*damage*)
Updates weapon damage attribute and weapon representation for different values

Parameters

param damage
Weapon damage value

type damage
int

3.3 Utility

3.3.1 Menu

This module is part of the utility package, which contains some functions and utility classes

`class menu.`**`Menu`**`(`*title*, *menu_items*`)`

Bases: `object`

This class representat a Menu Object

`choose()`

Allows to enter a choice between different menu items

Returns

returns
Number of entered choice

rtype
int

`print_menu()`

Print a Menu composed by a title and different items

3.3.2 Title

This module is part of the utility package, which contains some functions and utility classes

`title.`**`print_title()`**

Print a title composed by several lines of string

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex

PYTHON MODULE INDEX

INDEX