# Asrael Documentation

Alexander Pagonis (0931058)

Graz, May 7, 2016

# Contents

# List of Figures

# 1 Universal Behaviour

This section briefly covers the behaviour of the simulator, that is equivalent in every simulations (Kitchen, Wumpus and Labyrinth).

## 1.1 Loading Levels

Every simulation starts with an empty world. The world is then customised during the initialisation phase. For this there are distinct Folders in the data folder of the deliverables (Windows and Linux) or within the Mac binary package. Within these folders, the user may define levels in simple plaintext-files. For the Kitchen, the folder is called **"KitchenSettings"**. The Wumpus Simulator loads levels form the **"WumpusLevels"** file and the Labyrinth Simulator from the **"LabyrinthLevels"** folder. The user may define as many levels as desired. The simulator will then choose one level randomly by default. If the user wants a specific level to be loaded, the name of the desired level has to be entered in the **"level.config"** file, in the same folder. If this file is empty, random levels will be loaded. Additionally, there is a **"quality.config"** file, which may be used to define the graphical appearance of the simulator. It is also just a plaintext file that only includes a number from 0 to 5. Here 0 is the weakest graphic setting, which should run smoothly on weak hardware. If the quality level is set to 5 the simulator will feature dynamic shadows, high resolution textures and other stuff that is not necessary for the simulation to work, but improves the graphical appearance. The Kitchen and Labyrinth also feature feedback to the user if the configuration is invalid or ambiguous. For the Kitchen, every move if logged in the **"log.out"** file. The Labyrinth will only inform the user about invalid or ambiguous configuration settings in the **"compilation_warnings.out"** file. Due to the simplicity of the config files in the rumpus simulation, a log file is not created. Problems will be communicated to the user within the XML-Feedback only.

# 2 Kitchen

## 2.1 General Information

The Kitchen scene features a static base environment with a kitchen unit and a table. This static environment includes pre-specified disposals, that can be used to place objects on them. The base environment does not feature any portable objects, that can be manipulated by the robot. These objects are loaded from txt-files when the scene is initialised. These portable objects (spoons, plates, pans...) may again have disposals in order to allow them to be stacked onto each other. Objects can have only one, many or no disposal. These disposals also have a temperature. When now an object is placed onto the disposal, it will adopt its temperature. When the objects is being carried by the robot, it will adopt to the rooms temperature, which is set to 20°C.

## 2.2 Defining Portable Objects

This section will explain how to customise the kitchen scene with portable objects. Portable objects can be defined in txt-Files within the "KitchenLevels"-Folder. This way, the user can place objects of predefined classes with a self-chosen name, on specific locations within the scene. There is no limit to how many objects the user can define.

Following syntax applies for the level config files, shown on a example:

- po>LargePot1;LargePot;Cabinet5_Top;0

  **po>**: Command ("place object")

  **LargePot1**: *Name* of new object to be placed

  **LargePot**: *Class* of object to be placed

  **Cabinet5_Top**: *Disposal*, where to place object

  **0**: specific position on disposal, where to place (This attribute is optional. When not given, the first free position will be used)

### 2.2.1 Object Names

The chosen name can be any arbitrary UNIQUE identifier. Objects with identifiers that are already used will be ignored. For that reason, a log is created within the "KitchenLevels" folder to keep track of any problems and decisions taken for the user.

### 2.2.2 Available Classes

There are a few classes of placeable objects to choose from. These objects are all portable (can be carried by the robot) and may or may not also have a disposal (disposals are places where other objects can be placed on). Following classes exist:

- **Plates**

  BeigePlate

  BrownPlate

  GreenPlate

- **Food**

  Bread

  Cheese

  Ham

  Tomatoes

- **Silverware**

  Fork

Knife

LargePot

Spoon

- **Kitchen Utils**

  Pan

  SmallPot

  WoodenBowl

### 2.2.3 Available Disposals

There are many static disposals placed all over the scene. Most of the objects loaded from the txt-file also include a disposal. A scene overview is given in figure 1.

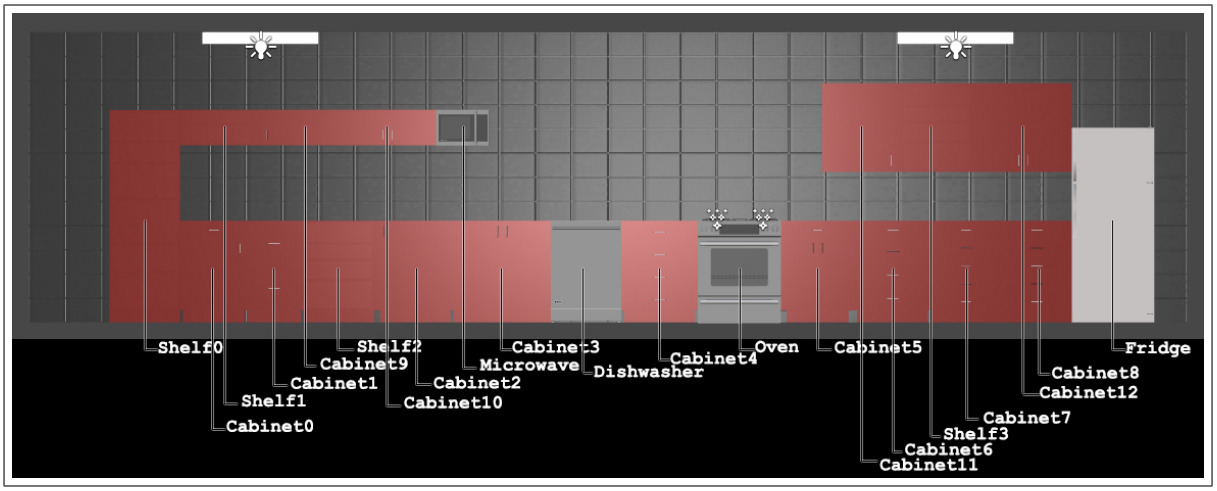


Figure 1: Overview kitchen unit

In addition, an overview over the tables disposals, including the available positions, is depicted in figure 2.

Bigger disposals like a long shelf, which has enough space to hold more than one object have multiple positions. Therefore, to identify an exact disposal position, the disposal can be followed by a number (eg. ";0"). This number denotes, where exactly the object is to be placed. Following disposals are available (the number in brackets denotes the amount of positions):

- **Shelfs**

  *Shelf0*

  Shelf0_0

  Shelf0_1

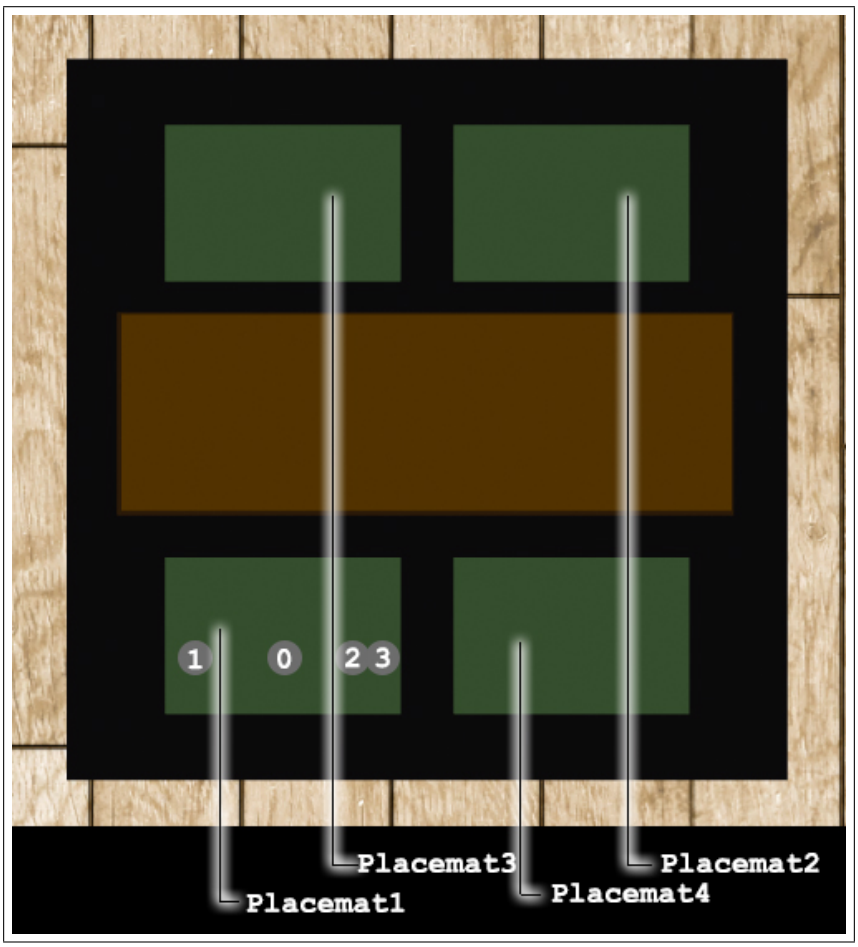


Figure 2: Overview table

Shelf0_2

Shelf0_3

Shelf0_4

Shelf0_5

Shelf0_Top

*Shelf1*

Shelf1_0

Shelf1_1

Shelf1_Top

*Shelf2*

Shelf2_0

Shelf2_1

Shelf2_2

Shelf2_3

Shelf2_Top

*Shelf3*

Shelf3_0

Shelf3_1

Shelf3_2

Shelf3_3

Shelf3_Top

- **Cabinets**

*Cabinet0*

Cabinet0_0

Cabinet0_1

Cabinet0_2

Cabinet0_Top

Cabinet0_Drawer0_Bottom (4x)

*Cabinet1*

Cabinet1_Drawer0_Bottom (4x)

Cabinet1_Drawer1_Bottom (4x)

Cabinet1_Top

*Cabinet2*

Cabinet2_0

Cabinet2_1

Cabinet2_2

Cabinet2_3

Cabinet2_Top

*Cabinet3*

Cabinet3_0 (2x)

Cabinet3_1 (2x)

Cabinet3_2 (2x)

Cabinet3_3 (2x)

Cabinet3_Top (2x)

*Cabinet4*

Cabinet4_Drawer0_Bottom (4x)

Cabinet4_Drawer1_Bottom (4x)

Cabinet4_Drawer2_Bottom (4x)

Cabinet4_Drawer3_Bottom (4x)

Cabinet4_Top

*Cabinet5*

Cabinet5_0

Cabinet5_1

Cabinet5_2

Cabinet5_Top

*Cabinet6*

Cabinet6_Drawer0_Bottom (4x)

Cabinet6_Drawer1_Bottom (4x)

Cabinet6_Drawer2_Bottom (4x)

Cabinet6_Drawer3_Bottom (4x)

Cabinet6_Top

*Cabinet7*

Cabinet7_Drawer0_Bottom (4x)

Cabinet7_Drawer1_Bottom (4x)

Cabinet7_Drawer2_Bottom (4x)

Cabinet7_Drawer3_Bottom (4x)

Cabinet7_Drawer4_Bottom (4x)

Cabinet7_Top

*Cabinet8*

Cabinet8_Drawer0_Bottom (4x)

Cabinet8_Drawer1_Bottom (4x)

Cabinet8_Drawer2_Bottom (4x)

Cabinet8_Drawer3_Bottom (4x)

Cabinet8_Drawer4_Bottom (4x)

Cabinet8_Top

*Cabinet9*

Cabinet9_0

Cabinet9_1

Cabinet9_Top

*Cabinet10*

Cabinet10_0 (2x)

Cabinet10_1 (2x)

Cabinet10_Top (2x)

*Cabinet11*

Cabinet11_0

Cabinet11_1

Cabinet11_2

Cabinet11_3

Cabinet11_Top

*Cabinet12*

Cabinet12_0 (2x)

Cabinet12_1 (2x)

Cabinet12_2 (2x)

Cabinet12_3 (2x)

Cabinet12_Top (2x)

- **Placemats**

  Placemat1 (4x)

  Placemat2 (4x)

  Placemat3 (4x)

  Placemat4 (4x)

- **Appliances**

  *Microwave*

  Microwave_Base

  *Oven*

  Oven_Grill0 (4x) [On Top...]

  Oven_Grill1 [Inside...]

  *Dishwasher*

  Dishwasher_Base

  *Fridge*

  Fridge_0 (2x)

  Fridge_1 (2x)

- **Misc**

  Any Plate, Pan, Food where it makes sense

## 2.3 Openables

A few cabinets have doors or drawers, that can break the line of sight between the robot and the objects inside them. In order make them visible for the robot, it has to open them. The openable objects are:

- **Cabinets**

  Cabinet0_Drawer0

  Cabinet0_Door0

  Cabinet2_Door0

  Cabinet3_Door0

  Cabinet3_Door1

  Cabinet1_Drawer0

  Cabinet1_Drawer1

  Cabinet4_Drawer0

  Cabinet4_Drawer1

  Cabinet4_Drawer2

  Cabinet4_Drawer3

  Cabinet5_Door0

  Cabinet5_Door1

  Cabinet6_Drawer0

  Cabinet6_Drawer1

  Cabinet6_Drawer2

  Cabinet6_Drawer3

  Cabinet7_Drawer0

  Cabinet7_Drawer1

  Cabinet7_Drawer2

  Cabinet7_Drawer3

  Cabinet7_Drawer4

  Cabinet8_Drawer0

  Cabinet8_Drawer1

  Cabinet8_Drawer2

Cabinet8_Drawer3

Cabinet8_Drawer4

Cabinet9_Door0

Cabinet10_Door0

Cabinet10_Door1

Cabinet11_Door0

Cabinet12_Door0

Cabinet12_Door1

- **Misc**

  Fridge_Door

  Oven_Door

  Microwave_Door

  Dishwasher_Door

## 2.4 Switchables

Switchables are objects that can be turned on and off. In the simulator, they are used to enable heating capability of the Oven and the microwave. Therefore the switchable objects are:

- Oven
- Microwave

## 2.5 Commands

Following XML-Commands are available:

- ”God”,”LoadLevel”,”KitchenBlank”

  *use:* loads the kitchen level

  *note:* This will load the kitchen with random settings, unless distinct level is defined in ”level.config” file

- ”Gargamel”,”Move”,[id]

  *use:* Move to object with given id (id...string)

  *note:* The id is a string, that denotes the na,e of the game object in the scene (e.g.: Fork1)

- ”Gargamel”,”Open”,[id]

  *use:* Open object with given id

  *note:* Can only be applied to openable objects!

- ”Gargamel”,”Close”,[id]

  *use:* Close object with given id

  *note:* Can only be applied to openable objects!

- ”Gargamel”,”Take”,[id]

  *use:* Take object with given id

  *note:* Can only be applied portable objects (e.g.: Pans, pots, forks...)

- ”Gargamel”,”Put”,[id]

  *use:* Put object to disposal with given id

  *note:* The id is again a string, that contains the name of the desired object. This Command will only work for objects with disposals, that are not occupied yet.

- ”Gargamel”,”TurnOn”,[id]

  *use:* Turn On object with given id

  *note:* Only works with switchables

- ”Gargamel”,”TurnOff”,[id]

  *use:* Turn Off object with given id

  *note:* Only works with switchables

- ”Gargamel”,”IsAt”,[id]

  *use:* Returns true when Gargamel is at given id

- ”Gargamel”,”See”,[id]

  *use:* Returns true when Gargamel sees the object, denoted by a given id

- ”Gargamel”,”Hot”,[id]

  *use:* Returns true when the given object is hot (>100 degree)

### 2.5.1 Available IDs

The ids that can be used to identify targets for specific commands, are strings that describe the target object. Note, that an object can have multiple ids, that describe it's sub-components. For instance, Cabinet0, features a door and drawers, as well as a shelf space on top. Therefore it features following ids:

- *shelfs*

  Cabinet0_0

  Cabinet0_1

  Cabinet0_2

  Cabinet0_Top

  Cabinet0_Drawer0_Bottom (4x)

- *openables*

  Cabinet0_Drawer0

  Cabinet0_Door0

All of these ids may be used anytime. Some commands however, may not work on every id. The See Command for instance will always deliver a result, as long as the id is valid. A TurnOff or TurnOn Command however will always fail on every object that is not switchable. The Move Command will always approach the position of the given id. Therefore, it makes no difference wether the robot is told to move to id "Cabinet0" or "Cabinet0_Door0", as long as the see command succeeds for both ids.

# 3 Wumpus

## 3.1 General Information

Without the config files the Wumpus Scene is completely empty. The user has to define the shape of the level, as well as the position of the traps, Wumpus and the gold. Once the level has been loaded, the character can be controlled using XML-RPCs. Also information about the games state and score can be prompted any time, using the commands explained in the command section 3.3.

## 3.2 Defining Levels

Defining levels is very easy. Following Syntax applies:

- H ... Hole
- G ... Gold
- W ... Wumpus
- # ... Empty tile

An example file would look like this:

```
#######
###W#H#
#######
####G##
##H####
```

Note: The position of the player is always on the bottom left tile. Therefore make sure to always leave it empty ("#"). The output of the file from above is depicted in figure 3. The geometry of the levels has to be a square. This means that the **length of the lines and columns have to be equal!**

## 3.3 Commands

Following XML-Commands are available:

- "God","LoadLevel","Wumpus"

  *use:* loads the level

  *returns:*

  true: on success

  false: on fail

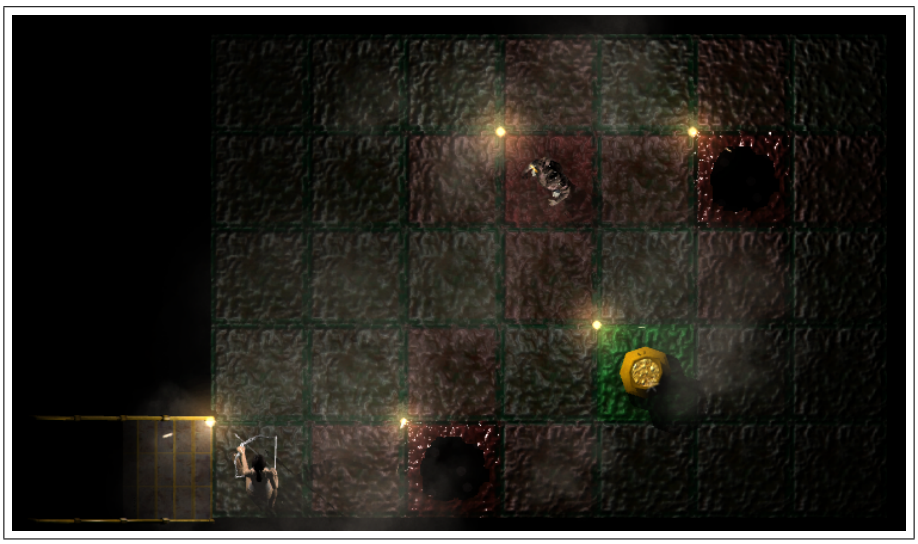

Figure 3: Wumpus example level

- ”God”,”wumpusState”

  *use:* returns the current game state (running/won/lost)

  *returns:*

  2: won

  3: lost

  4: running

- ”God”,”wumpusScore”

  *use:* returns the current game score

  *returns:*

  1: always and score in string

- ”Player”,”turnLeft”

  *use:* player character turns left

  *returns:*

  true: always (never fails)

- ”Player”,”turnRight”

  *use:* player character turns right

  *returns:*

  true: always (never fails)

- ”Player”,”move”

  *use:* player character moves ahead

  *returns:*

  true: when player moved to next tile

  false: when player could not move (hit the wall)

- ”Player”,”shoot”

  *use:* player character shoots an arrow

  *returns:*

  true: when player shot the arrow

  false: when player had no arrows left to shoot

- ”Player”,”senseWumpus”

  *use:* returns true if wumpus is close

  *returns:*

  true: when player is next to the wumpus tile and wumpus is still alive

  false: when player is not next to the wumpus tile or wumpus is dead

- ”Player”,”senseHole”

  *use:* returns true if hole is close

  *returns:*

  true: when player is next to a hole tile

  false: when player is not next to a hole tile

- ”Player”,”senseGold”

  *use:* used to check wether the player is on the gold tile

  *returns:*

  true: when player is on the gold tile

  false: when player is not on the gold tile or already carries the gold

- ”Player”,”takeGold”

  *use:* used to pick up the gold when on the gold tile

  *returns:*

  true: when player is on the gold tile and picked up the gold

  false: when player is not on the gold tile or already carries the gold

- ”Player”,”climb”

  *use:* used to escape the dungeon with the gold

  *returns:*

  true: when player has the gold and is on the entry tile

  false: when player is not on the entry tile or does not carry the gold when on the entry tile

- ”Player”,”hear”

  *use:* sense whether the wumpus was killed

  *returns:*

  true: only immediately after the wumpus was shot

  false: if there was another action between the shoot and the hear command or if the wumpus was not shot

# 4 Labyrinth

## 4.1 General Information

Just like the Wumpus simulation, the labyrinth simulation starts with an empty world. The user has to define the geometry of the labyrinth and the properties of the walls in two separate files.

## 4.2 "Gameplay"

The robot is placed somewhere in a labyrinth. It has the quest to correctly identify casualties that are placed in it. To identify casualties, it can sense heat and color of walls. If a wall has color AND heat coming from it, it holds a casualty. If not both of these criteria are met, it is not a casualty. When the robot thinks it has found a casualty, it can report it. When correctly reporting casualties, the player gets some points. For wrongly reported casualties, points are deducted. The robot only gets a certain period of time for exploring the labyrinth. When the time is over, it cannot be controlled anymore. Also it has the possibility to escape the labyrinth. If the robot explores the labyrinth and manages to find its way back to the entry point (color encoded: green tile) and escape, it is rewarded with a higher score.

## 4.3 Defining Levels

The geometry can be defined similar to the wumpus case. It is stored in the txt-file and follows the syntax:

- # ... Tiles to move on
- 0 ... No Tile (hole)
- N,E,S,W ... Tile including the robots starting position and rotation (North / East / South / West)

Additionally another File (extension: *.prop) has to be provided including the properties of the tiles. This file has to has the same name as the txt-File. For example: The level 1 consists of the files:

- 1.txt
- 1.prop

The prop-File follows the syntax: [target]>[Settings1],[Settings2],....
Following adjustments are possible (shown by examples):

- **define game parameters:**

  *example:* points_report>15

*use:* points for correctly reporting casualties (15 in this example)

*example:* points_leave>30

*use:* points for leaving the arena (30 in this example)

*example:* casualty_color>255;0;0

*use:* color for casualties r;g;b (red in this example)

*example:* max_time>120

*use:* The time (in seconds) the robot has for exploring the labyrinth

- **define tile properties (follows the syntax: [tile_id]>[Wall]:[Property])**

  *example:* 16>S:W,S:C

  *Tile_id:* number of tile (16 in this example)

  *Wall:* desired wall (N/E/S/W) (South Wall in this example)

  *Property:* W (Warm),C (Colored) or E (Empty) (Warm and Color)

  *meaning:* This means that the south wall of tile number 16 should be warm and have casualty color (so it holds a casualty)

### 4.3.1 Tile IDs

The tile-id is an implicit parameter that is calculated from the levels txt-file. The tiles are automatically numbered, starting with 0 and incrementing from left to right, top to bottom.

### 4.3.2 Identifying the Walls of the Tiles

Every tile has 4 walls. The walls are identified by their cardinal directions, relative to the txt-file. This means that the north wall (N) of a tile, will always be the wall on the top.

### 4.3.3 Available Wall Properties

Every wall of a tile may have different properties. Following properties are available:

- W ... **W**arm
- C ... **C**olor
- E ... **E**nabled

Giving a wall the properties Warm AND Color implicitly denotes a casualty. The "Enabled" property is necessary because the **simulator automatically removes Walls between adjacent tiles, unless they are marked as Warm, Colored or Enabled.**

### 4.3.4 Example Level Definition

Following example should make it more clear. The following txt-file:

```
#00000
##0000
###00#
00#00#
E#####
#0#00#
```

combined with this prop-file:

```
points_report>15
points_leave>30
casualty_color>255;100;100
max_time>120
0>N:W
1>E:E
2>W:C
6>N:C,N:W,W:C
16>S:W,S:C
```

Creates the level, depicted in 4

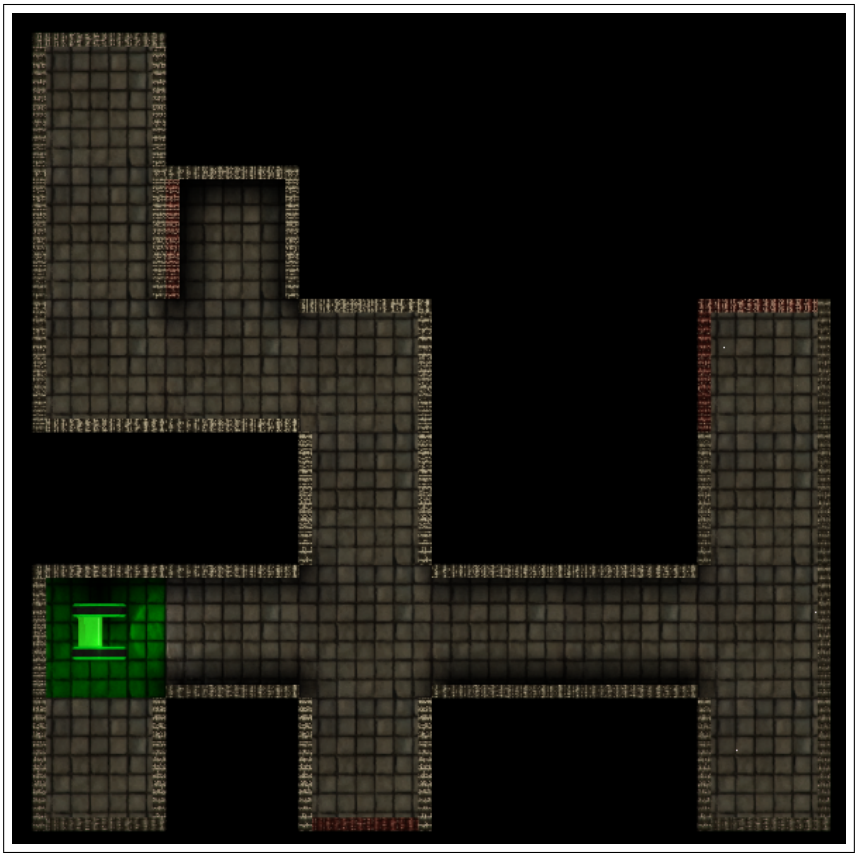

Figure 4: Labyrinth example level