This is the final report on your implementation of project 2. Again, retain these gray parts. Keep in mind the evaluation criteria (at the end). For voluminous material, reference appendices (at the end). These will be read on an as-needed basis. Excluding appendices and figures, this response should not exceed 5 pages of 12-point Times New Roman text.

Maze Runner

by Jan Allemann

# PROJECT PROPOSAL WITH YOUR RESPONSES

Paste Assignments 2.1 here *as is*, but with a response to each of the comments, inserted within each of them. Please insert your

# Assignment 1 3/13/20

## 1.1 SUMMARY DESCRIPTION

One-paragraph overall description of the purpose, inputs and outputs for your proposed semester project. Do not go into details because the next section does that.

The main goal of this project is, to get familiar with PDDL. Having no experience in this language at all, I find it highly interesting. Therefore, a big part of the project will be to learn the PDDL basics. The input will be a maze containing empty and blocked fields that gets translated into initial states that describe the world. Finally, the PDDL scripts shall find a plan to escape the maze.

## 1.2 Requirements

State your requirements for a project. Number the requirements D1, D2, … and N1, N2, … where D means “Definite” and N means “Nice to do.”

i = 1, 2, 3, …

There should be 3-5 items in the “Definite” list and at least 2 in the “Nice to do” list. You will reference these numbered requirements in Phase 2 when you will be asked to show what the project accomplished.

Requirements are *declarative* statements of the application’s intended functionality such as “the expected number of years to graduate college shall appear on the console.” (A statement such as “Find an agent tool” is a procedure step, not a requirement.)

### D1 Maze translation into initial state

The python code translates the maze into a set of initial states suitable for PDDL

### D2 PDDL scripts finds a plan to reach the goal

The PDDL runs successfully to find a plan to satisfy a set goal (escape the maze).

### D3 Validate the plan using A\*

An A\* algorithm can find a way out of the maze. The PDDL output is compared to the output of the A\*.

### N1 Display the plan

The output of PDDL is a set of actions as a txt file. This output shall be visualized.

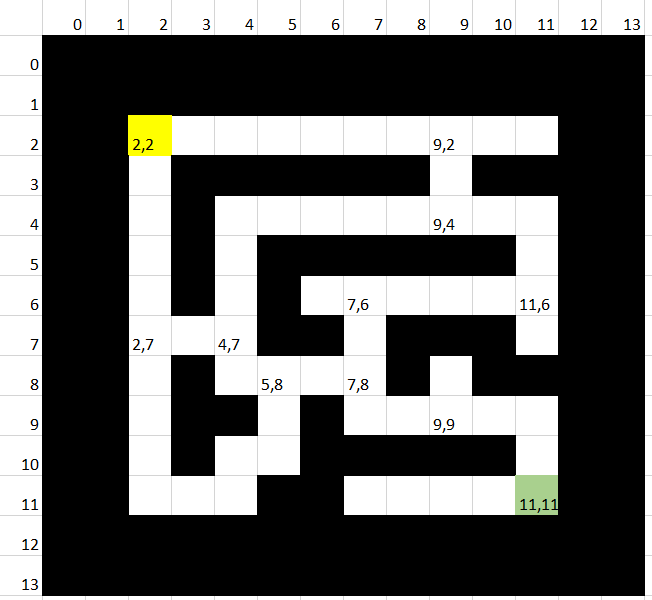
### N2 Scale up to more a more complex maze

A more complex, bigger maze is created and the PDDL script is tested in order to scale up.

## 1.3 Design and Theory

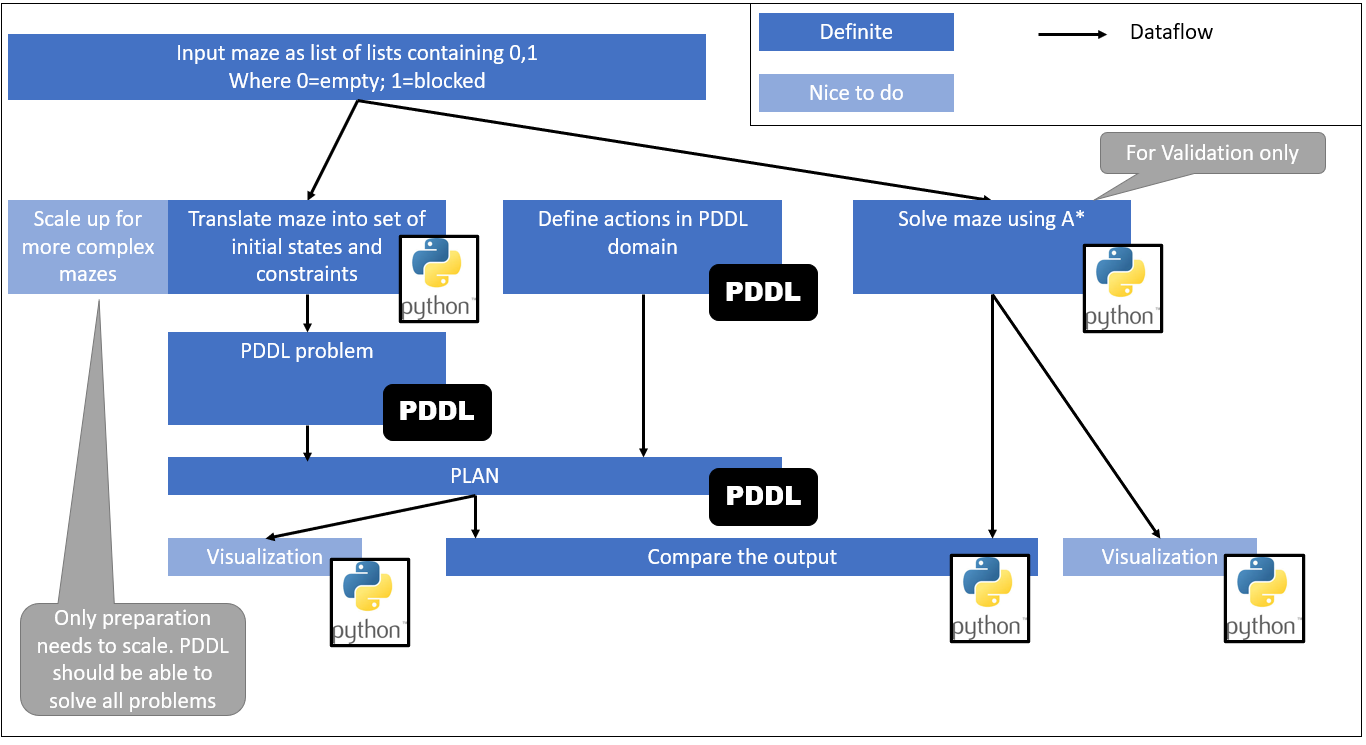
Describe the design of your proposed system. The reader should understand how you plan to fit the pieces together. Show this at a high level, as well as providing as much relevant detail as you can. Include at least one (meaningful) annotated figure. An example of a figure form is shown below, in which data flows from one process to the next.

The contains out of a list of list, where each entry is a field. The field can either be empty (=0) or blocked (=1). The field can be identified via x- and y-coordinates according the following picture. The yellow field (2,2) is the players initial position while the green field (11,11) is the exit and therefore the goal to reach.



To set the world up for PDDL, all possible paths for each fork need to be calculated in python. This information will then be used in PDDL to build the initial states and constraints.

The output of the PDDL will be a plan as txt file, and an A\* algorithm solves the maze as well. The results shall be compared to validate both results.



## 1.4 Tools

Unless you intend to build from scratch (i.e., with a higher-level language), describe the tool(s) you will use.

The translation of the input into a set of initial state is made using the web application ‘Jupyter Notebook’ developed by project jupyter.

To run and develop the PDDL scripts, the web application [web-planner](https://web-planner.herokuapp.com/) [1] developed by the School of Computer Science (FACIN) is used.

## 1.5 Implementation Fragments

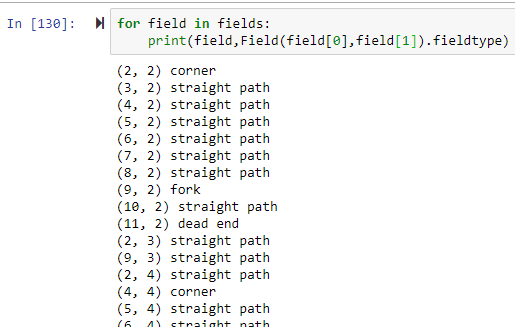
Show enough *parts* of an implementation—or a simplified form of it—to convince the reader that you will have the implementation of the definite requirements completed on time. These can be experimental or exploratory in nature. Cut and paste commented code, and explain its context.

Multiple functions are developed to move a player around the field via arrow keys. The player always moves until the next fork or wall is reached. While the input via arrow keys will not be needed in the project, the rest of the functionality will be very useful in order to find all the possible paths through the maze:

|  |  |
| --- | --- |
| Initial state | After ‘arrow right’ is pressed |
|  |  |

The called functions can be found in appendix A

To solve the maze in PDDL, all possible ways need to be known. The first step to do so, is recognizing all the forks, dead ends etc. as such. When creating an instance of the class Field (see Appendix B), the fieldtype is recognized by analyzing the surrounding fields. This can be done for each empty field:



## 1.6 References

Fill in below—and cite each of the following (e.g., “[2]“) within the text. References can include specific places in the notes and textbook.

|  |  |
| --- | --- |
| [1] | (PUCRS), P. C. (2020, April 10). *Web Planner*. Retrieved from https://web-planner.herokuapp.com/ |
| [2] | Dolejsi, J. (2020, April 10). *PDDL Samples*. Retrieved from https://github.com/jan-dolejsi/vscode-pddl-samples |
| [3] | Helmert, M. (2016). *An Introduction to PDDL.* Toronto: University of Toronto. |
| [4] | Maur´ıcio C. Magnaguagno, R. F. (2017). *Web Planner: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search.* Port Alegre: Pontifical Catholic University of Rio Grande do Sul (PUCRS). |

# FINAL DESIGN AND IMPLEMENTATION

## 2.1 Summary

In a paragraph or two, summarize the outcome of your project functionally and learning-wise but avoid duplication with Section 3.3 below. (Reminder: leave these gray sections in your paper.) 4/9/20

The main achievement of the project is the capability to develop a plan for escaping a maze using Planning Domain Definition Language (PDDL). The generation of the PDDL problem definition is partly automated using Python. The Python script can analyze a maze and print out the necessary code-lines to define the maze in PDDL. The output (plan) is compared with the solution of an A\* algorithm developed in Python in order to verify both algorithms.

I did not have any prior knowledge of PDDL and thus didn’t know how complex it is to develop such an algorithm. Throughout the project I gained valuable skills and learned that PDDL is a great way to solve planning problems. Thanks to this project, I will consider using the language in further projects if applicable.

## 2.2. Report on Requirements

Describe the extent to which you accomplished each definite requirement "Di" as well as any nice-to-do (Ni) requirement. For each requirement, provide a title of 5 or fewer words (e.g., “Sentiment Output”), 1-3 sentences, and screenshot of input/output. Your effectiveness depends largely on how much you demonstrate that you learned. Limit: 2 pages of 12-point Times New Roman.

### D1 Maze translation into initial state

The python code translates the maze into a set of initial states suitable for PDDL

* 100% fulfilled

The initial states describe the world as it stands initially. In an abstract way, a maze is a net of nodes that are connected to each other, where each node is an empty field. Depending on whether the empty field is a corner, fork, dead end or just a path, it has 1 to 4 neighbors. These neighborhood-relationships are the initial states in the PDDL problem definition.

As an example, the marked field (x=11, y=11) is a corner and has therefore 2 neighbors:

The neighbor above is the field x=11, y=10

The neighbor to the left is the field x=10, y=11

Therefore, the python script creates 2 relationships:

|  |  |
| --- | --- |
|  | **Screenshot:** |

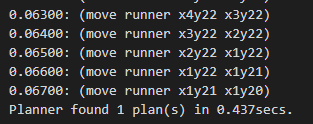
The Python script saves two .txt files containing objects (all empty fields in the maze) and all inits (all neighbor relationships).

### D2 PDDL scripts finds a plan to reach the goal

The PDDL runs successfully to find a plan to satisfy a set goal (escape the maze).

* 100% fulfilled

Note: The requirement, as defined, is 100% fulfilled. However, I’m not fully happy with the final solution. I was hoping to find a planner that can handle functions and metrics. Unfortunately, this update of the publicly available planners is not yet released.



### D3 Validate the plan using A\*

An A\* algorithm can find a way out of the maze. The PDDL output is compared to the output of the A\*.

* 100% fulfilled

The comparison is made by looking at the number of steps the algorithms took to escape the maze. Additionally, there is a visual comparison to see if the algorithms chose the same path or not.

|  |  |
| --- | --- |
|  |  |

### N1 Display the plan

The output of PDDL is a set of actions as a txt file. This output shall be visualized.

* 100% fulfilled

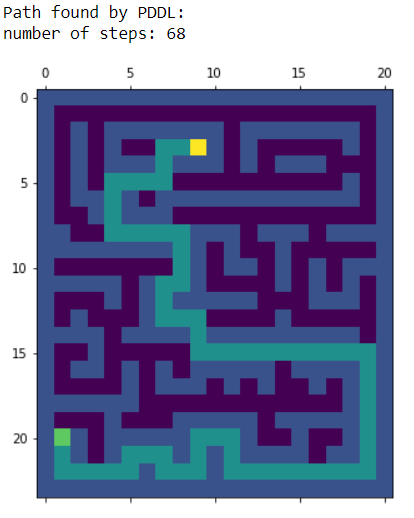
The output is visualized in python to see which way the algorithms has chosen (see Requirement D3)

### N2 Scale up to more a more complex maze

A more complex, bigger maze is created and the PDDL script is tested in order to scale up.

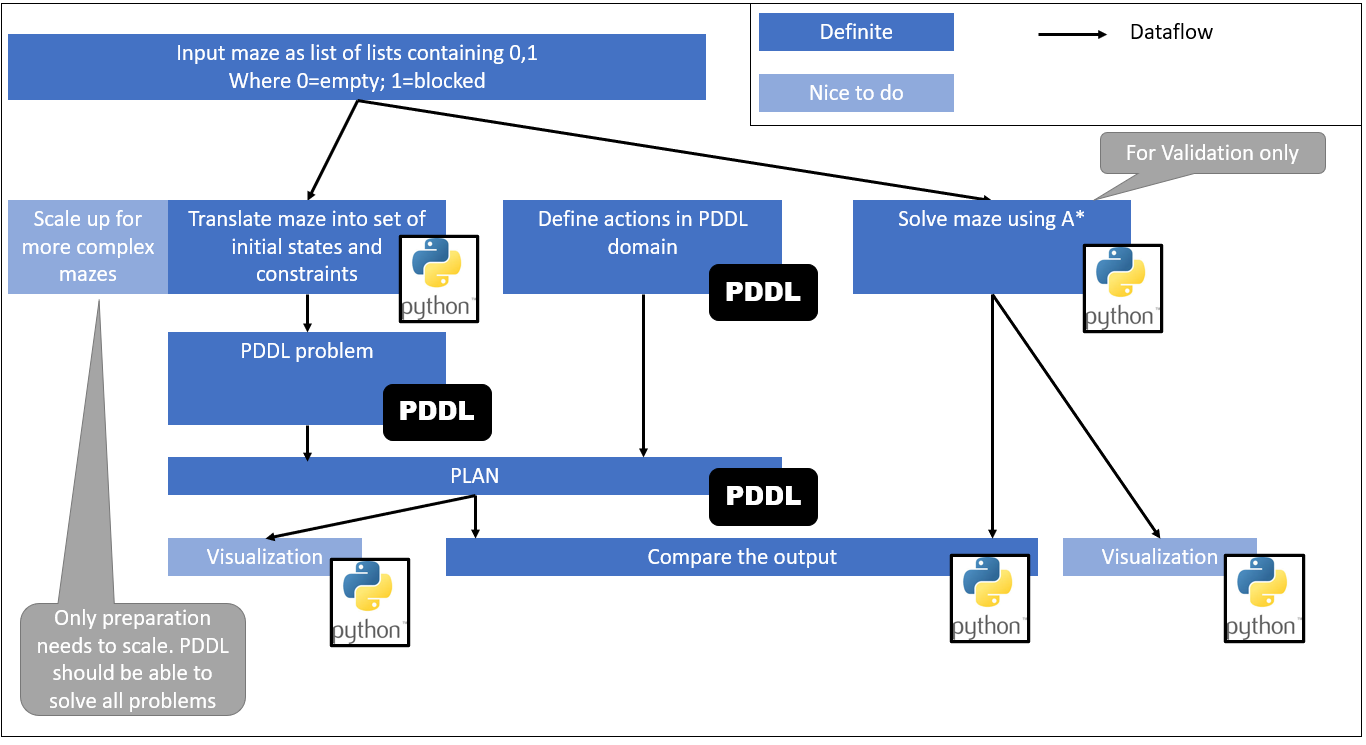
* 100% fulfilled

All described functionalities (Generating problem definition, solving in PDDL, solving in A\*) work with bigger, more complex mazes as well:



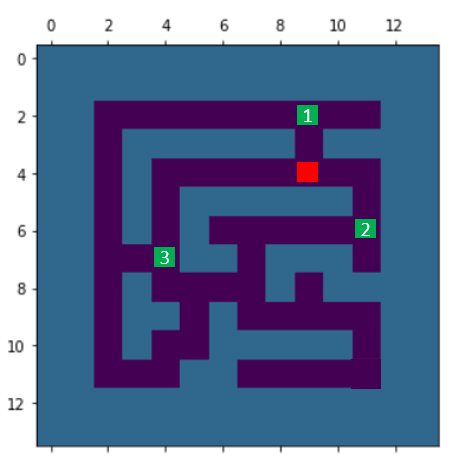
## 2.4. Report on Design

Describe the design that you finally used. Indicate how, where, and why it differed from your planned design. Describe its advantages and its shortcomings. Include a description of how the technologies you explored (not the tools—those are described below) leveraged each other. Include at least one diagram. Limit: 2 pages of 12-point Times New Roman.

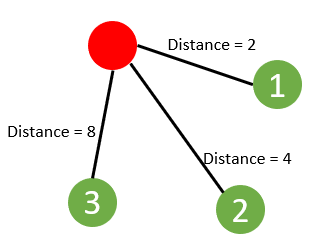


Eventually, there are no major changes in the final design compared to the initially planned design.

At the beginning of phase 2, I intended to develop a more elegant way to solve the problem in PDDL. Rather than looking at direct neighbors, a neighbor could also be the next cell that is not a straight path, corner or dead end:



In that way, a maze can be described as a net of neighbors and weighted connections, where the weight of the edge equals the distance between the nodes:



This approach requires fewer initial states and describes a maze in a simplified way. The PDDL problem and domain are developed to run this kind of maze-definition. Unfortunately, the publicly available solvers can’t handle planning with optimization of a function yet.

## 2.5. Tools

Describe the tool(s) that you used. Show samples. Describe their advantages and their shortcomings. (List other tools separately.) Limit: 1 page of 12-point Times New Roman.

## All Python implementations are developed in the web application ‘Jupyter Notebook’ by project jupyter.

To develop PDDL-scripts, the PDDL Extension [6] for Microsoft Visual Studio is used. This extension works with the open source solver from the planning.domain collection [5].

## 2.6 Contrast between approaches

You were to include two of (a) *Planning* (b) *Uncertainty* (c) *Natural language*. Explain how the two that you selected fit together.

A maze is described as a set of possible steps and action that can be taken. The action has a set of preconditions that need to be satisfied and effects that result from the action. Forward planning finds an optimal way for the mazerunner to escape the maze.

This process could be extended with an aspect of uncertainty in order to optimize the planner. As described, a maze can be seen as a net of nodes that are connected to each other. Each node has a certain amount of outgoing connections and has therefore an uncertainty to be the correct path to the target. Looking multiple steps ahead, I might be able to see that some of the outgoing connections have already been visited in previous steps. This might be a way to reduce uncertainty and eventually find a way to reduce computation time.

## 2.7 What did *not* work well

Explain the most important aspects of your project that fell short of your plans or desires.

It took me a long time to understand, that the available planner is not able to run optimization of a function. I also think that there is a way to successfully run it, but unfortunately, I couldn’t find one.

I didn’t really implement an aspect of uncertainty or natural language processing. I was trying to fit an uncertainty aspect into the project, but I couldn’t come up with good fit.

## 2.8 What *did* work well

In paragraph form, explain the most important aspects of your project that met or exceeded your plans or desires.

I didn’t have any prior knowledge of PDDL and the language is less documented than other major languages. I still managed to develop a running algorithm and could have even implemented a more advanced domain if the planner could handle it.

Working intensively on that project I realized, how easy a maze can be to solve. To describe a maze as a net of connected fields seems to be a very nice foundation for a planning problem.

## 2.9 Sample Source

Supply up to 1 page of key excerpts from your source code—or what comes closest to “source code.” Limit: 2 pages of 12-point Times New Roman. Include an explanation of where the excerpts fit in your implementation.

The following source code is the domain definition of the mazerunner. Having the correct problem definition, this simple domain can escape every maze.

; Domain description

; Describe the relations and transitions that can occur

; This one describes the Tower of Hanoi puzzle

(define (domain mazerunner) ; Domain name must match problem's

  ; Define what the planner must support to execute this domain

  ; Only domain requirements are currently supported

  (:requirements

    :strips                 ; basic preconditions and effects

    :negative-preconditions ; to use not in preconditions

    :equality               ; to use = in preconditions

    :typing

  )

  (:types

    field player

  )

  ; Define the relations

  ; Question mark prefix denotes free variables

  (:predicates

    (on ?player - player ?loc - field)      ; A player is on a field

    (neighbour ?from - field  ?to - field) ; A field ?end is a neighbor of a field ?start

  )

  ; Move from one field to a neighbor

  (:action move

    :parameters (?player - player ?from - field ?to - field)

    ; Only conjunction or atomic preconditions are supported

    :precondition (and

      (on ?player ?from)

      (neighbour ?from ?to)

    )

    ; Only conjunction or atomic effects are supported

    :effect (and

      ; Note that adding the new relations is not enough

      (on ?player ?to)

      (not (on ?player ?from))

    )

  ))

## 2.10 Source

Attach source code (or what comes closest to it) and input where possible. You may refer the reader to github for source if you prefer.

Please find all the source code on the public GitHub repository [cs-664-buallema-project2](https://github.com/jaal5/cs-664-buallema-project2)

## 2.11 References

Show that you used a wide variety of resources by listing them below and clearly indicating in the body above where you used them.

|  |  |
| --- | --- |
| [1] | (PUCRS), P. C. (2020, April 10). *Web Planner*. Retrieved from https://web-planner.herokuapp.com/ |
| [2] | Dolejsi, J. (2020, April 10). *PDDL Samples*. Retrieved from https://github.com/jan-dolejsi/vscode-pddl-samples |
| [3] | Helmert, M. (2016). *An Introduction to PDDL.* Toronto: University of Toronto. |
| [4] | Maur´ıcio C. Magnaguagno, R. F. (2017). *Web Planner: A Tool to Develop Classical Planning Domains and Visualize Heuristic State-Space Search.* Port Alegre: Pontifical Catholic University of Rio Grande do Sul (PUCRS). |
| [5] | Planning.domain (2015). *A collection of tools for working with planning domains.* |
| [6] | Dolejsi, J. (2020, April 20). *VS Marketplace*. Retrieved from https://marketplace.visualstudio.com/items?itemName=jan-dolejsi.pddl |
|  |  |

# Evaluation

