

# Project Report - "The Magoni Adventure"

## Introduction

Welcome to the most thrilling project report you'll read today! Our project, codenamed "The Magoni Adventure" is a masterpiece in the making. We hope you enjoy reading it as much as we enjoyed writing it. We are **Paul Magos & Stefano Zanoni**, and we're here to take you on a journey of algorithms comparison (with some strange approaches). Our aim is nothing short of finding something interesting in the field of **Efficiently Solving Mazes in Minihack**.

## Start

Now that you're all set, let's dive into the good stuff. We'll start with a brief standard algorithm comparison, then we'll move on to the more interesting stuff. First we'll compare many of the algorithms seen in the course, implemented by us.

Results:  
Algorithm: AStarManhattan Time:0.2654862403869629 Visited:49 Path:27  
Algorithm: AStarEuclidean Time:0.14666199684143066 Visited:51 Path:27  
Algorithm: AStarChebysev Time:0.11014294624328613 Visited:52 Path:27  
Algorithm: DFS Time:0.10768485069274902 Visited:96 Path:27  
Algorithm: BFS Time:0.11055302619934082 Visited:65 Path:27  
Algorithm: Dijkstra Time:0.10345911979675293 Visited:65 Path:27  
Algorithm: GreedyManhattan Time:0.12991118431091309 Visited:39 Path:28  
Algorithm: GreedyEuclidean Time:0.11697793006896973 Visited:38 Path:28

The results are not overwhelming. We can see also the results of the comparison in the Appendix [3].

## Genetic Algorithms

Results:  
Algorithm: Genetic Time:377.514986038208 Visited:1001 Path:1001

The brains are not braining today, but we know someone else have done it better than us, so we'll skip it.

## QLearning

### QTable

Results:  
Algorithm: QTable Time:0.6680302619934082 Visited:71 Path:71

The question is, does it work? No it doesn't work well, but it was a nice try which we retained not to be explored in a deeper way for the purpose of this project. See also the Appendix [4].

### QNN and QLSTM

Results:  
Algorithm: QNN Time:18.191520929336548 Visited:11 Path:11  
Algorithm: QLSTM Time:154.4595320224762 Visited:18 Path:18

For a better understanding of the results we suggest to read the Appendix [5].

## Here comes the strange part

We decided to create a Neural Network that approximates the Manhattan distance between a starting point a the target point. No other information is given to the NN, just the starting point and the target point. We don't want it to learn something about the maze, we just want it to learn the distance between the two points. We trained it with a training dataset of ~33000 mazes, a dataset made by us. We are not gonna show the CNN, LSTM, NN, that try to find the real path between the starting point and the target point. This is to be more brief about the report, but you can find them in the **pseudo\_heuristics** folder, and you can run them by yourself with some specifications, they are not that interesting, but they are there. It took some time to train them, but it was a pretty "not involving" task, so we did it.

## We present you the NNHeuristic, our pseudo heuristic approach for solving mazes with A\*

WARNING:NNHeuristic:This is a pseudo-heuristic. It is not a real heuristic.  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.  
WARNING:absl:Skipping variable loading for optimizer 'SGD', because it has 11 variables whereas the saved optimizer has 1 variables.  
WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.  
WARNING:absl:Skipping variable loading for optimizer 'SGD', because it has 11 variables whereas the saved optimizer has 1 variables.  
WARNING:NNHeuristic:This is a pseudo-heuristic. It is not a real heuristic.

Results:  
Algorithm: AStarNNHeuristic Time:1.4738359451293945 Visited:37 Path:27  
Algorithm: GreedyNNHeuristic Time:2.8895130157470703 Visited:39 Path:28

As we can clearly see from the results above, our pseudo heuristic outperforms all the other heuristics in A\*, and it's really pushing to the real path, so we can say that it's a really good approximation of the real distance. Let's see how it performs on bigger mazes

For the bigger mazes we'll use the a new seed, so we can compare the results with the previous ones. A brief comparison of these two seeds can be found in the Appendix [6] and [7].

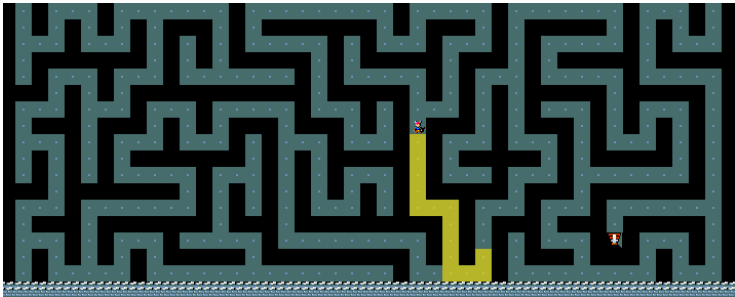
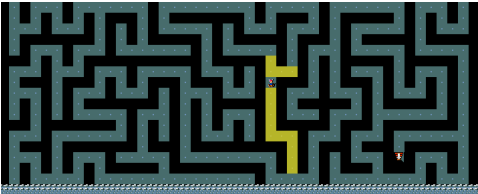
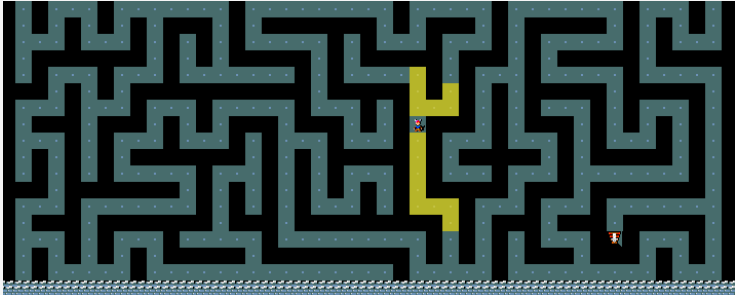
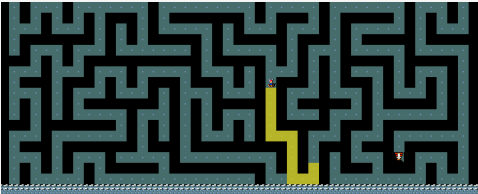
Results:  
Algorithm: AStarNNHeuristicLarge Time:6.423929691314697 Visited:200 Path:107  
Algorithm: AStarManhattanLarge Time:0.11173200607299805 Visited:287 Path:107  
Algorithm: AStarEuclideanLarge Time:0.21919488906860352 Visited:288 Path:107

It's actually better, furtherly, after some research and comparison of the manhattan distance and the euclidean distance, we found that the distance that our pseudo heuristic approximates is not a really known distance.  
To have a better understanding of the capabilities of our pseudo heuristic take a look at the Appendix [\[8\]](#).  
At the best of our knowledge there was no similar approaches in the literature and we found out it is trying to aproximate the **SManhattan distance** (S for strange).  
The SManhattan distance is the Manhattan distance with a little twist, it's the Manhattan distance mutliplied by a factor that is a costant, defined by us.

We took our shot and made the SManhattan distance, defined as follows:

$$h(start, end) = abs(start.x - end.x) + abs(start.y - end.y) * 3$$

Results:  
Algorithm: AStarSManhattanLarge Time:0.10812211036682129 Visited:201 Path:107

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* SManhattan	0.108	201	107		A* Manhattan	0.112	287	107	
A* Euclidean	0.219	288	107		A* NNHeuristic	6.424	200	107	

As we can see, the SManhattan distance is really close to the NN Pseudo Heuristic for A\* in this case, but we can't say that it's the best heuristic for all the mazes.  
Also we can't say that it's the best heuristic for Greedy, because it's not, by our tests it's equivalent to the Manhattan distance.  
We added for a better understanding of our approach all the tests in the Appendix [\[10\]](#).

## Methodologies

We wanted to compare our algorithms in terms of visited cells and path length  
We can run each algorithm (that is worth trying) on many mazes and then comparing the mean of the time it took, the visited cells and the path length.

WARNING:absl:At this time, the v2.11+ optimizer `tf.keras.optimizers.SGD` runs slowly on M1/M2 Macs, please use the legacy Keras optimizer instead, located at `tf.keras.optimizers.legacy.SGD`.  
WARNING:absl:Skipping variable loading for optimizer 'SGD', because it has 11 variables whereas the saved optimizer has 1 variables.  
WARNING:NNHeuristic:This is a pseudo-heuristic. It is not a real heuristic.

We computed the results of each algorithm (*A, Greedy, Dijkstra, BFS, DFS*)  
*with all the heuristics and pseudo heuristics for A* and Greedy.  
We also created a Scoring System for the Assessment phase.

Scoring System (Assessment): Defined as the ratio between the path length and the visited cells length.

## Results

Drumroll, please! The moment of truth has arrived.

### Large Maze Results

Algorithm	Time_mean	Visited_Length_mean	Path_Length_mean	Path_Score_mean
GREEDY-Manhattan-Env1	0.011785	122.03	77.94	0.686225
GREEDY-SManhattan-Env1	0.012042	122.03	77.94	0.686225
ASTAR-NNManhattan-Env1	3.822897	129.22	75.84	0.678001
ASTAR-SManhattan-Env1	0.012503	131.24	75.84	0.664671

### Medium Maze Results

Algorithm	Time_mean	Visited_Length_mean	Path_Length_mean	Path_Score_mean
ASTAR-NNManhattan-Env2	1.047521	34.13	23.47	0.772143
ASTAR-SManhattan-Env2	0.011277	34.55	23.47	0.765570

GREEDY-Manhattan-Env2	0.011131	35.11	24.21	0.725002
GREEDY-SManhattan-Env2	0.011163	35.11	24.21	0.725002

## Small Maze Results

Algorithm	Time_mean	Visited_Length_mean	Path_Length_mean	Path_Score_mean
ASTAR-NNManhattan-Env3	0.399369	12.00	9.63	0.878512
ASTAR-SManhattan-Env3	0.011111	11.95	9.59	0.875009
ASTAR-Manhattan-Env3	0.011035	12.36	9.49	0.845445
ASTAR-EUCLIDEAN-Env3	0.011041	13.01	9.49	0.811554

## Our Considerations

We can see, our Pseudo Heuristic is doing well, being in the top scores almost all the time. It certainly takes time since it's a NN, it will be better to use in batch by giving it all the possible starting points and target points, and then use the output. Also our SManhattan is really good. To be fair we have to say that there are some cases as we can see from the mean scores in which there is a negligible discrepancies, but since our are Pseudo Heuristics we can't obviously guarantee the properties of an Heuristic.

## Conclusion

In conclusion we are proud of this project, and the nice results that we got. We have achieved better performance than the state of the art by just using some of our basic skills in the AI field. The pseudo heuristic is interesting, and it took inspiration from some of the papers that we read in which they used a so called Differential Heuristic. Our contribution is to create a Neural Network that approximates the Manhattan distance, and then use it as a heuristic for A\* and Greedy. Also we further propose a pseudo heuristic that we called SManhattan, which is the Manhattan distance multiplied by a constant. We would really like to know why the weighting of the Manhattan Distance by a constant was achieving better performance that the normal Heuristic.

## Dataset

We provide with this project a dataset of mazes, that we created with a script that we wrote. We have both images and chars mazes, and we also have the solution for each maze. We created the dataset to train our models. The dataset will not be provided with the project, but you can download it by yourself, it's a really small dataset, so it's not a problem. Read the README

## Contribution

We created basically our heuristic for maze solving, and we wanted to see how it performs. It was a really interesting journey, and we are really proud of our work. We also created a dataset of mazes, and this project itself which is a library, so we can use it in the future for other projects, or **someone else can use it** to create something even better.

## Challenges Faced

We had a lot of challenging moments on trying to use the minihack enviornment... It is not well documented, and it's not easy to use. Also it doesn't provide the Animation that we provided... We took our shot and we created it by ourselves, it was a really nice to see all the algorithms outcome.

## Future Work

Finally we want to propose the dataset we created as well as the library itself as a starting point for future generations of AIF students that can extend our work and improve it. Our journey doesn't end here. Many tools such as Prolog or algorithms like Monte-Carlo can be applied to this field, also many of the not informed practical algorithms can be implemented. We also wanted to try to implement a new algorithm that we called **QStar** which is a mix between QLearning and A\*, but this is another story.

## Acknowledgments (Related Works)

The course matherials were really useful for this project, obviously.

In the field of our project we found some interesting papers that we used as inspiration for our work:

- [The Compressed Differential Heuristic](#)
- [Reinforcement Learning with A\\* and a Deep Heuristic](#)
- [Structured World Representations in Maze-Solving Transformers](#)

Also a project for probably University of Petr Posik

- [Robot localization in a maze](#)

We'd like to aknowledge the following resources that contributed to our ideas for this project:

- [Minihack Documentation](#)
- [Minihack Github](#)
- [StackOverflow](#)
- [Github Community](#)

# Appendix

We both contributed to the project in the same way. We split some parts of the project such as Paul written some of basic algorithms and the QTable, Stefano contributed to the part of implementing the QLearning agents, the QLSTM and the QNN. We both contributed to this final notebook. Then the utils where actually taken from the course Hands-On, and we modified them to fit our needs. The Animator class was made from scratch, as well as almost all the code in our library and the dataset.

The project is available on [Github](#) as well as all the metrics which are not truly representative since we worked many times from the University on the same pc.

We loved trying to figure out how some of the arguments seen in the Search Chapter of the course could be applied to the Minihack environment. We also loved trying to figure out how to create a library that integrates part of the algorithms seen, even if we wanted to add more of them. Also we further proposed a new heuristic that we called SManhattan.

We are really proud of our work, and we hope that it is actually enjoyable.

## [1] Technologies Used

We harnessed the power of many libraries to create our project. Before we begin, let's make sure you have everything you need to follow along. Here's a list of used libraries:

- **numpy**
- **matplotlib**
- **minihack**
- **scipy**
- **nethack**
- **nle**
- **keras**
- **gym**
- **tensorflow**
- **pytorch**
- **pickle**
- **Pillow**
- **imageio**
- **pandas**

And here's a list of the other optional libraries:

- **wandb** (not necessary, but useful if you want to perform again the experiments)

Uncomment the following cell if you need to install the libraries









## [2] Project Structure

Here's a quick overview of the project structure. We've included a brief description of each file to help you navigate:

- **nextdataAI** The root folder of the project:
  - **algorithms** This folder contains all the algorithms that we implemented  
*Algorithm.py* Is the superclass of all algorithms, which creates the env, sets the seed so that all are working on the same maze, and also contains the method to run the algorithm.
    - Standard:
      - *AStart, Greedy, Random, Genetic, Dijkstra, BFS (in FS), DFS (in FS)*
    - Not Standard
      - *QLSTM, QNN, QLearning*
  - **heuristics** This folder contains all the heuristics that we implemented for the A\* algorithm and also for Greedy  
*Heuristic.py* Is the superclass of all heuristics
    - *Manhattan, Euclidean, SManhattan (later we'll explain this one), Chebysev*
  - **pseudo\_heuristics** This folder contains all the so called by us pseudo heuristics that we implemented for the A\* algorithm and also for Greedy, this part of our real contribution to the "field" *PseudoHeuristic.py* Is the superclass of all pseudo heuristics
    - *CNN & LSTM are pseudo heuristic that uses a CNN or an LSTM to predict the next action, takes the whole map pixels as input and it has to return the distance from the goal (spoiler: both doesn't work well, but we din't won't to waste time on it)*
    - *NN Same as CNN and LSTM but it takes the chars map, and still doesn't work well*
    - *NNHeuristic **THIS** is the pseudo heuristic that uses a NN which we trained to approximate the Manhattan distance, it takes only the starting and ending position as input and it returns the distance between them. (It works really well, too much, outperforming in most cases all the other heuristics improving the performance of AStar by a lot)*
  - **QLearning** This folder contains all the Reinforcement Learning basics that we implemented
  - *AnimateGif.py* This class is used to create the gif of the maze solving process
  - *HeuristicUtils.py & utils.py* Just utils used by many of our algorithms
- **data** The folder that contains all the data, with a dedicated dataset class for reading the files, that we used for our experiments with NNs approaches (generated by us)
- **Papers** The folder that contains some papers that we used as inspiration for our work




### [3] Comparison Between: A\* (Manhattan, Euclidean, Chebysev), Greedy (Manhattan, Euclidean), BFS, DFS, Dijkstra



Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* Manhattan	0.265	49	27		A* Euclidean	0.147	51	27	
BFS	0.111	65	27		Dijkstra	0.103	65	27	
DFS	0.108	96	27		Greedy Manhattan	0.13	39	28	
Greedy Euclidean	0.117	38	28		A* Chebysev	0.11	52	27	

Nice, we can clearly see in **Yellow** the visited cells, in **Blue** the path found by the algorithm.  
**Also in the table we can clearly see the time taken by the algorithm, the number of visited cells and the path it found.**  
We can clearly see that in small mazes, the Greedy algorithm is the best one, but we'll later how A\* will be really improved by our contribution.

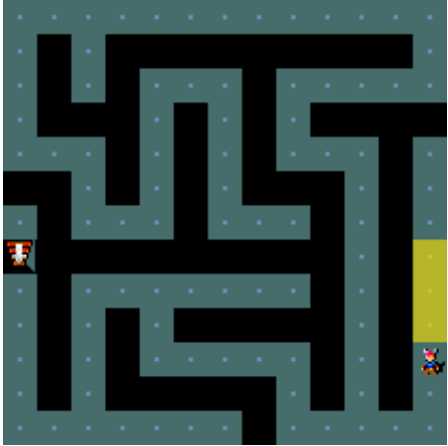

### [4] QTable Results

Algorithm	Time	Visited	Path	Image
QTable	0.668	71	71	





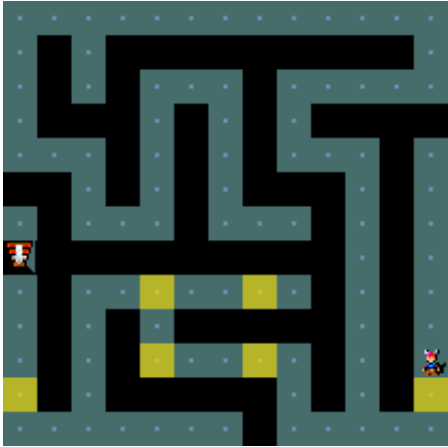
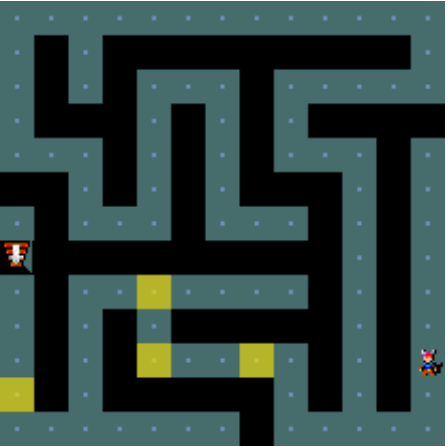
### [5] QNN and QLSTM Results 9x9

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
QNN	18.192	11	11		QLSTM	154.46	18	18	

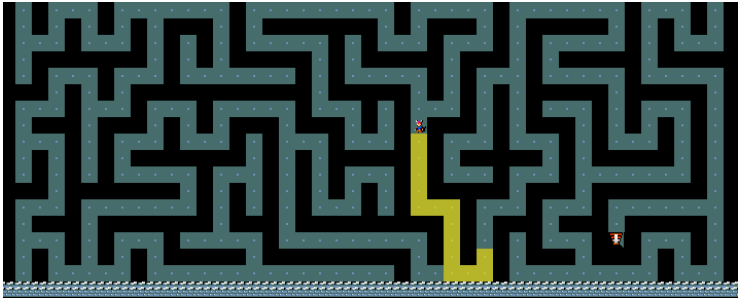
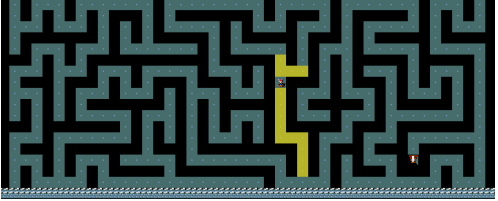
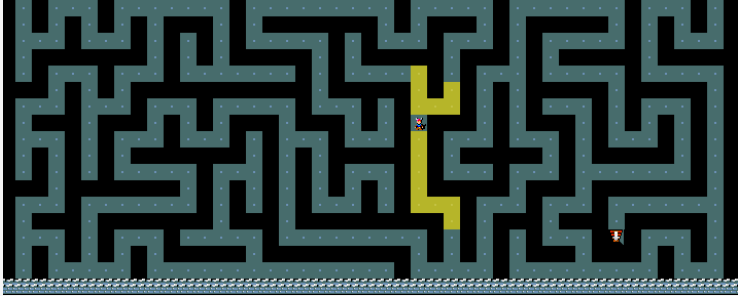
### [6] A\* and Greedy with NNPseudoHeuristic

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* NNHeuristic	1.474	37	27		Greedy NNHeuristic	2.89	39	28	

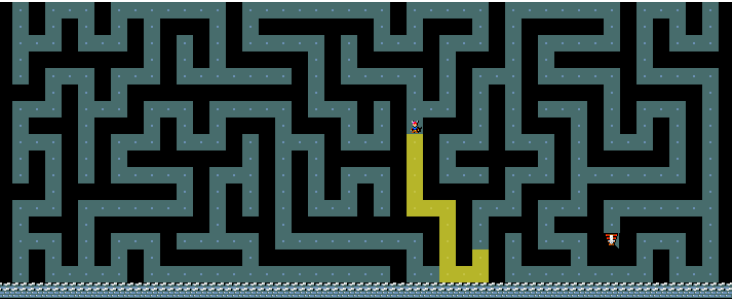
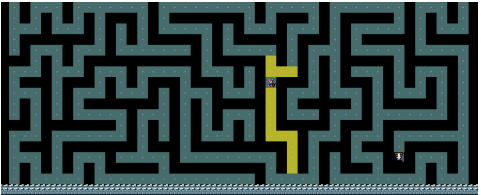

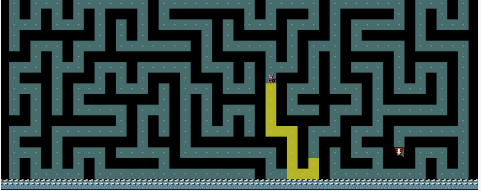
### [7] A\* (NNPseudoHeuristic, Manhattan, Euclidean), Greedy (NNPseudoHeuristic, Manhattan, Euclidean)

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* NNHeuristic	1.474	37	27		Greedy NNHeuristic	2.89	39	28	
A* Manhattan	0.265	49	27		A* Euclidean	0.147	51	27	
Greedy Manhattan	0.13	39	28		Greedy Euclidean	0.117	38	28	

### [8] Comparison between A\* (NNPseudoHeuristic, Manhattan, Euclidean) Large Maze

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* NNHeuristic	6.424	200	107		A* Manhattan	0.112	287	107	
A* Euclidean	0.219	288	107		—	—	—	—	—

## [9] Comparison between A\* (NNPseudoHeuristic, Manhattan, Euclidean, SManhattan) Large Maze

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* SManhattan	0.108	201	107		A* Manhattan	0.112	287	107	
A* Euclidean	0.219	288	107		A* NNHeuristic	6.424	200	107	

## [10] All the results on the mazes in the previous cells of the Appendix

All Results:

	Algorithm	Time	Visited	Path
0	AStarManhattan	0.265486	49	27
1	AStarEuclidean	0.146662	51	27
2	AStarChebysev	0.110143	52	27
3	DFS	0.107685	96	27
4	BFS	0.110553	65	27
5	Dijkstra	0.103459	65	27
6	GreedyManhattan	0.129911	39	28
7	GreedyEuclidean	0.116978	38	28
8	QTable	0.668030	71	71
9	AStarNNHeuristic	1.473836	37	27
10	GreedyNNHeuristic	2.889513	39	28
11	AStarNNHeuristicLarge	6.423930	200	107
12	AStarManhattanLarge	0.111732	287	107
13	AStarEuclideanLarge	0.219195	288	107
14	AStarSManhattanLarge	0.108122	201	107
15	Genetic	377.514986	1001	1001
16	QNN	18.191521	11	11
17	QLSTM	154.459532	18	18