Project Report - "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) in the field of **Efficiently Solving Mazes in Minihack**.

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

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

Our Goal

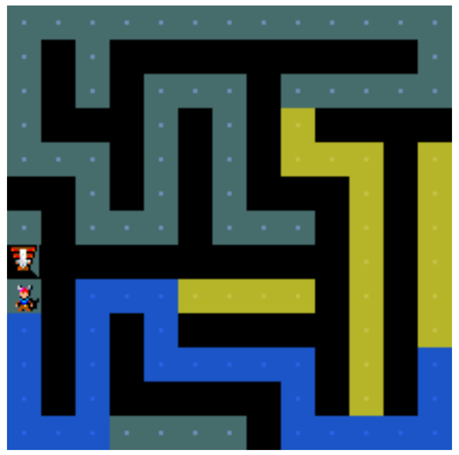
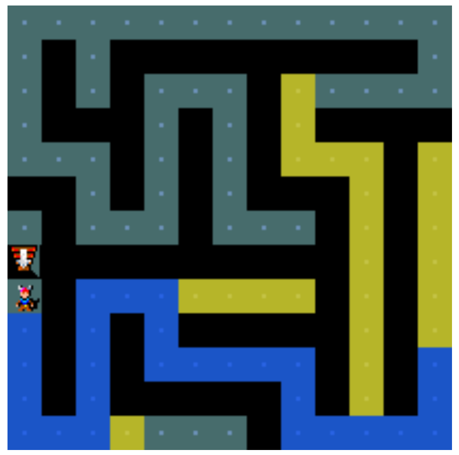
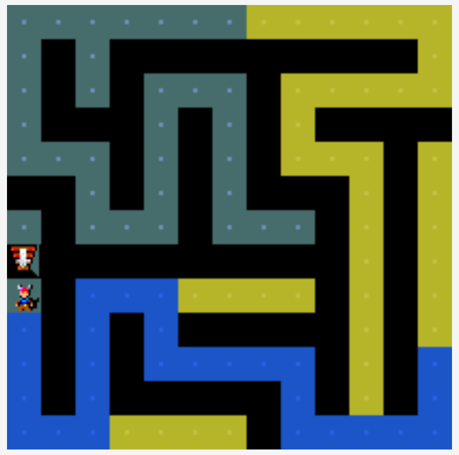
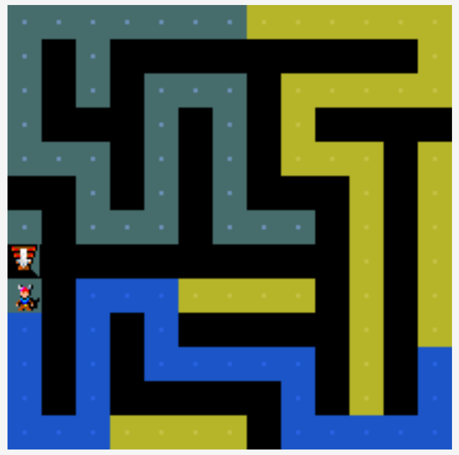
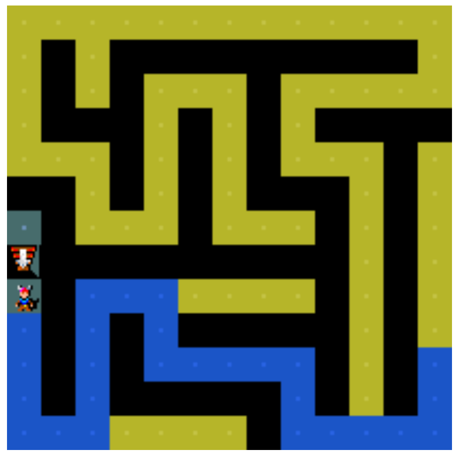
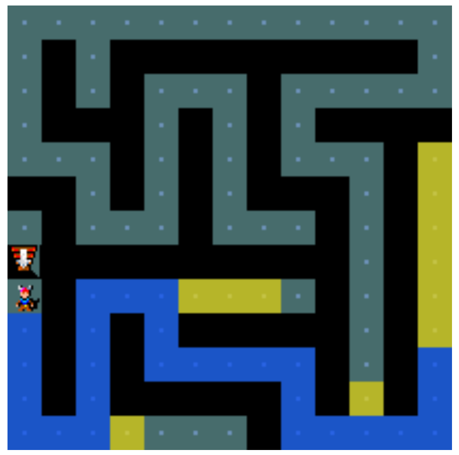
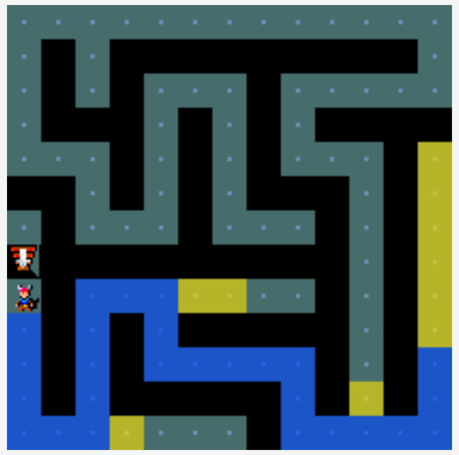
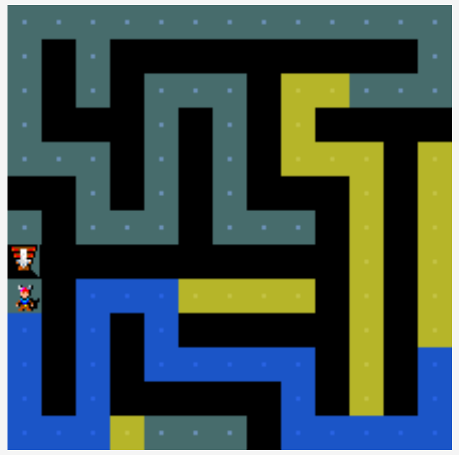
Our aim is nothing short of finding something interesting in the field of Efficiently Solving Mazes in Minihack.

Project

Let's dive into the details of our masterpiece. Brace yourself; it's going to be an interesting journey.

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.

```
100%|██████████| 26/26 [00:00<00:00, 188.40it/s]
100%|██████████| 26/26 [00:00<00:00, 178.98it/s]
100%|██████████| 26/26 [00:00<00:00, 185.48it/s]
100%|██████████| 26/26 [00:00<00:00, 186.46it/s]
100%|██████████| 26/26 [00:00<00:00, 188.55it/s]
100%|██████████| 26/26 [00:00<00:00, 190.19it/s]
100%|██████████| 26/26 [00:00<00:00, 185.87it/s]
100%|██████████| 26/26 [00:00<00:00, 193.57it/s]
```

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* Manhattan	0.166	49	27		A* Euclidean	0.114	51	27	
BFS	0.116	65	27		Dijkstra	0.103	65	27	
DFS	0.108	96	27		Greedy Manhattan	0.11	39	28	
Greedy Euclidean	0.112	38	28		A* Chebysev	0.109	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.

Genetic Algorithms

```
Generation: 2 | Best Fitness: 0: 100%|██████████| 100/100 [06:05<00:00,  3.65s/it]
```

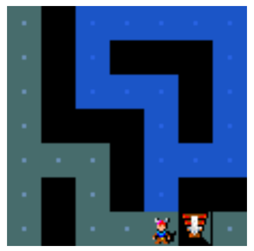
Algorithm	Time	Visited	Path	Image
Genetic	371.261	1001	1001	

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

Let's start with QLearning

QTable

```
Training:  17%|█▋        | 167/1000 [00:00<00:03, 272.12steps/s]
100%|██████████| 81/81 [00:00<00:00, 338.73it/s]
```

Algorithm	Time	Visited	Path	Image
QTable	1.248	167	167	

The question is, does it work?

No it doesn't, but it was a nice try which we retained not to be explored in a deeper way for the purpose of this project.

QNN and QLSTM

We had a big trouble to run these algorithms on the notebook. But the results were discrete
so we decided to keep them even if they are not that good. We are not gonna show the final comparison for these.

Really Nice, it was a nice try... but we wanted to do a step further,
so 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.

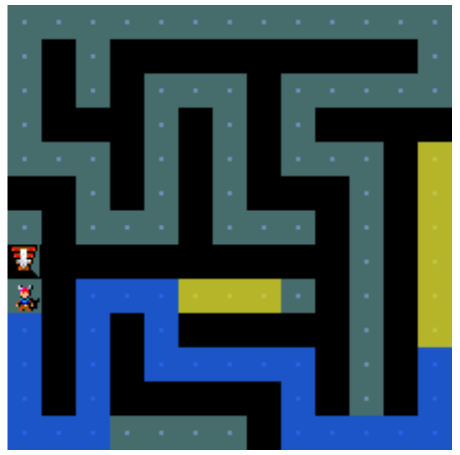
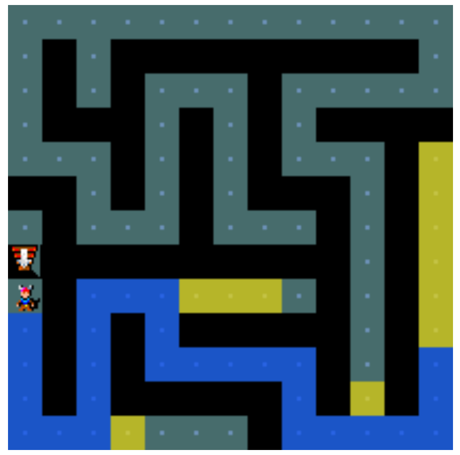
Here comes the strange part

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.

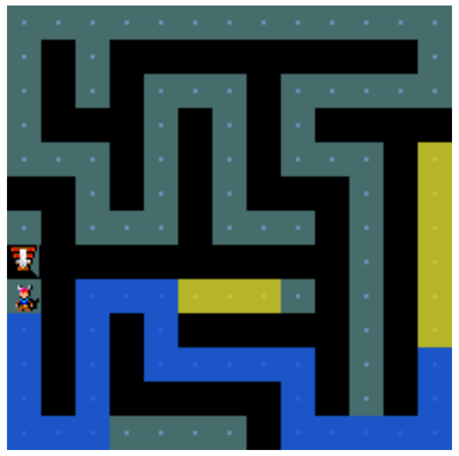
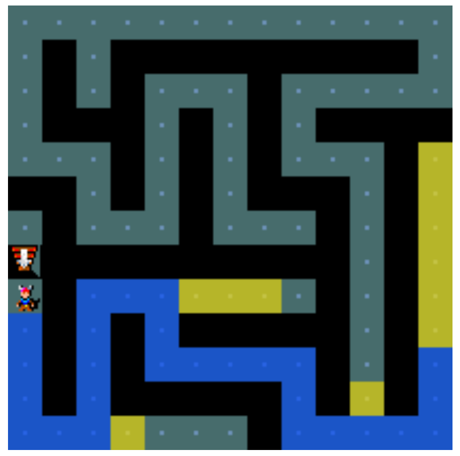
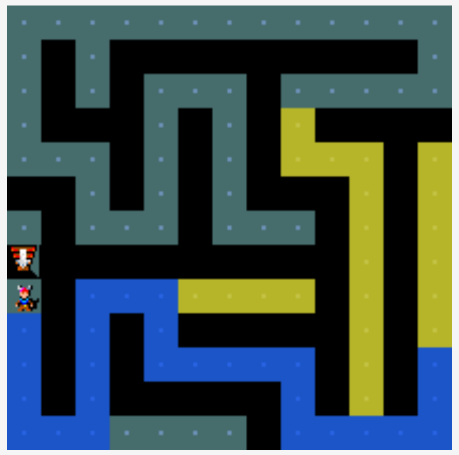
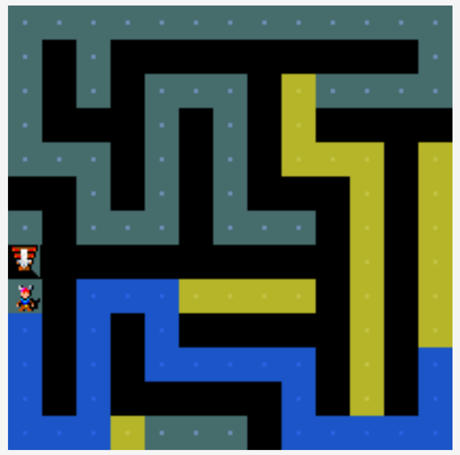
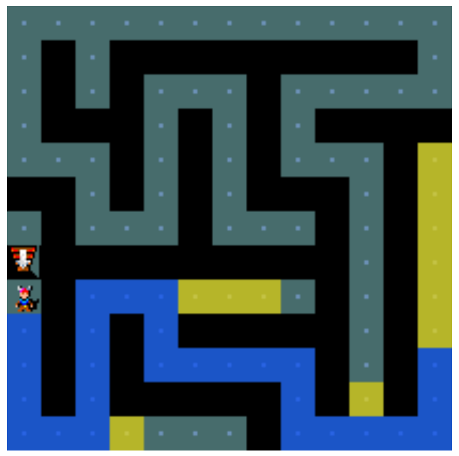
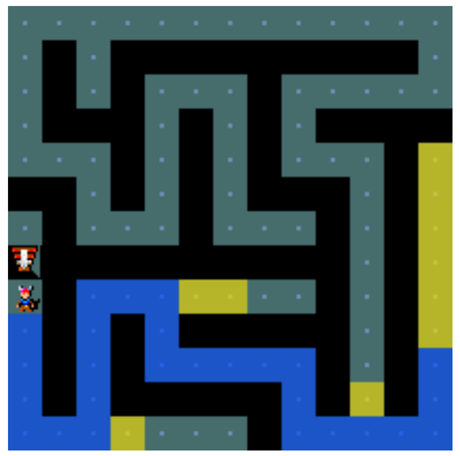
It took some time to train them, but it was a pretty "not involving" task, so we did it.

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

```
WARNING: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:NNHeuristic:This is a pseudo-heuristic. It is not a real heuristic.
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.
100%|██████████| 26/26 [00:00<00:00, 188.79it/s]
100%|██████████| 26/26 [00:00<00:00, 193.05it/s]
```

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* NNHeuristic	1.393	37	27		Greedy NNHeuristic	2.432	39	28	

To have a brief comparison between the heuristics and our "pseudo heuristic" we'll show the table with the results

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* NNHeuristic	1.393	37	27		Greedy NNHeuristic	2.432	39	28	
A* Manhattan	0.166	49	27		A* Euclidean	0.114	51	27	
Greedy Manhattan	0.11	39	28		Greedy Euclidean	0.112	38	28	

As we can clearly see from the table above, our pseudo heuristic outperforms all the other heuristics, 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

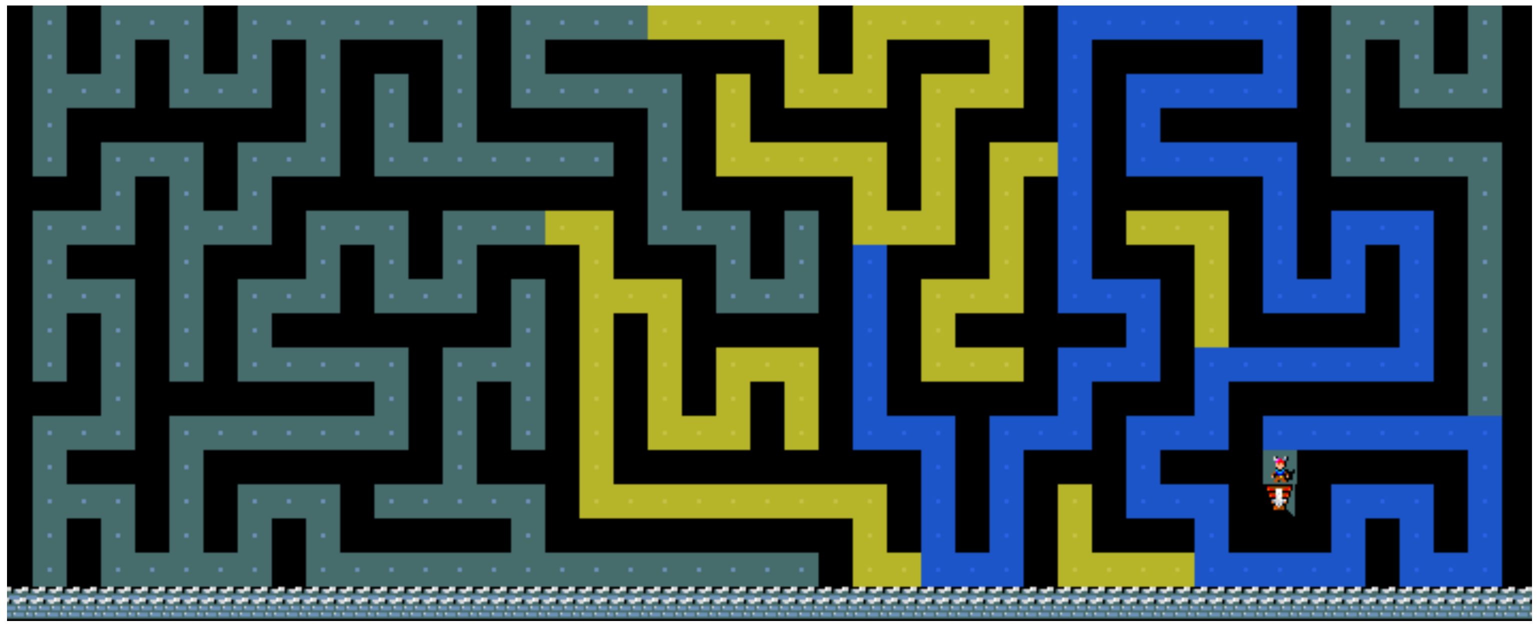
For the bigger mazes we'll use the a new seed, so we can compare the results with the previous ones.

```
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.optimize
rs.legacy.SGD`.
WARNING:absl:Skipping variable loading for optimizer 'SGD', because it has 11 variables whereas the saved optimizer has 1 variables.
```

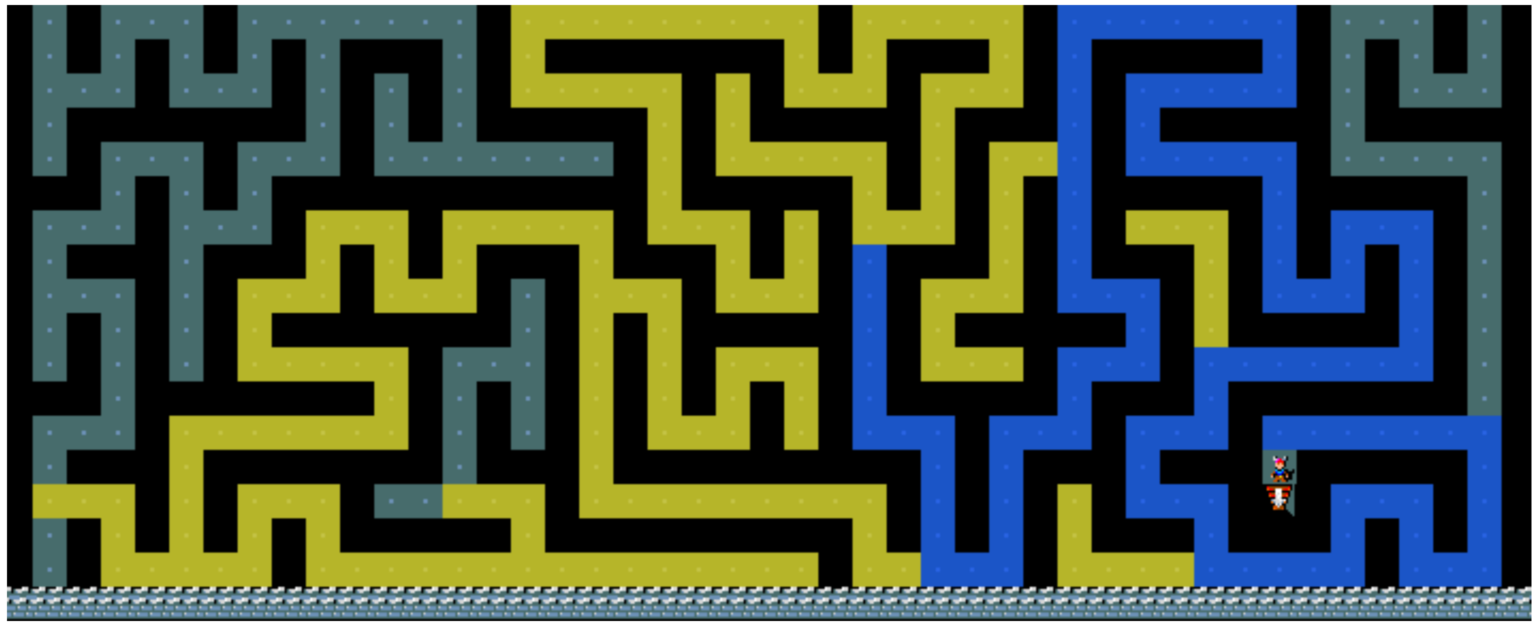
```
WARNING:NNHeuristic:This is a pseudo-heuristic. It is not a real heuristic.
```

```
100%|██████████| 106/106 [00:01<00:00, 92.22it/s]
100%|██████████| 106/106 [00:01<00:00, 89.64it/s]
100%|██████████| 106/106 [00:01<00:00, 86.55it/s]
```

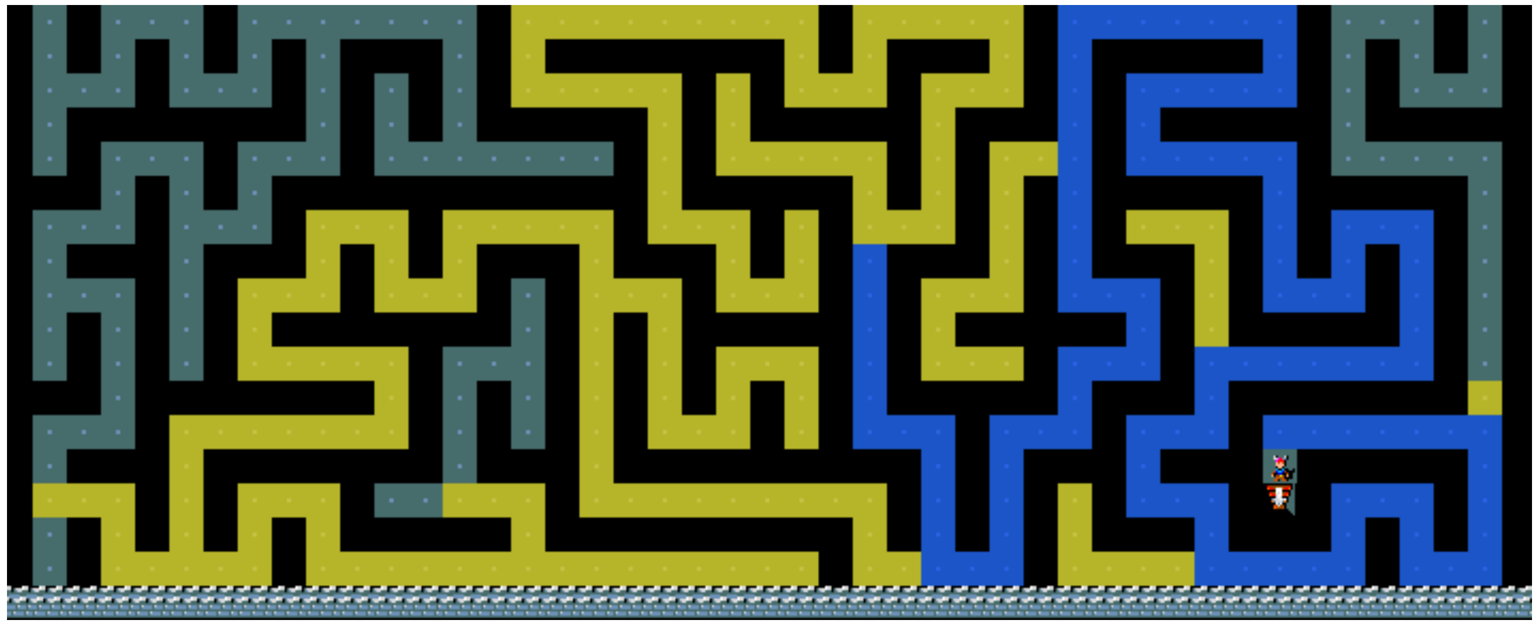
| Algorithm | Time | Visited | Path | Image | Algorithm | Time | Visited | Path | Image | |:------------------|------:|--------:|--------:|---:|------------------|------:|--------:|--------:|---:| | A*



NNHeuristic| 5.644 | 200 | 107 | | A*



Manhattan | 0.103 | 287 | 107 | | | A*

Euclidean | 0.106 | 288 | 107 |  |

What a pleasure to see our pseudo heuristic working so well

But Why ?

After some research and comparison of the manhattan distance and the euclidean distance, we found that the distance that our pseudo heuristic approximates is the...
We don't know actually, 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).

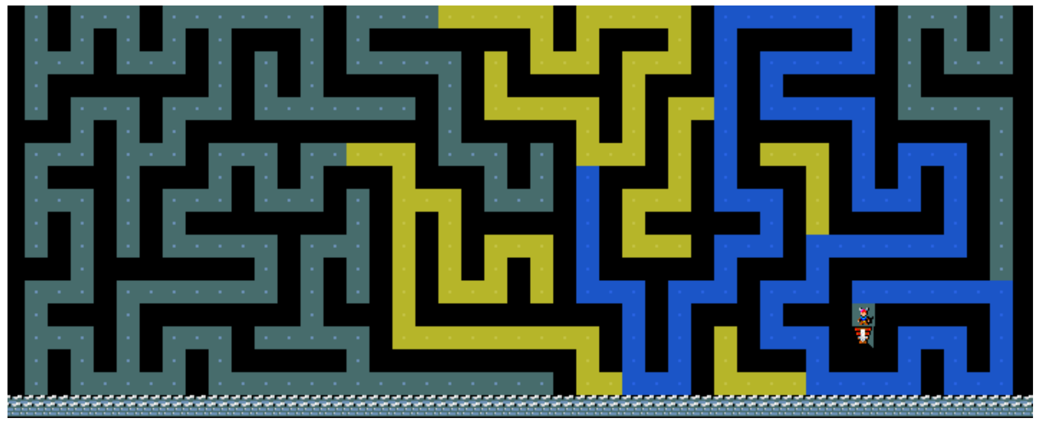
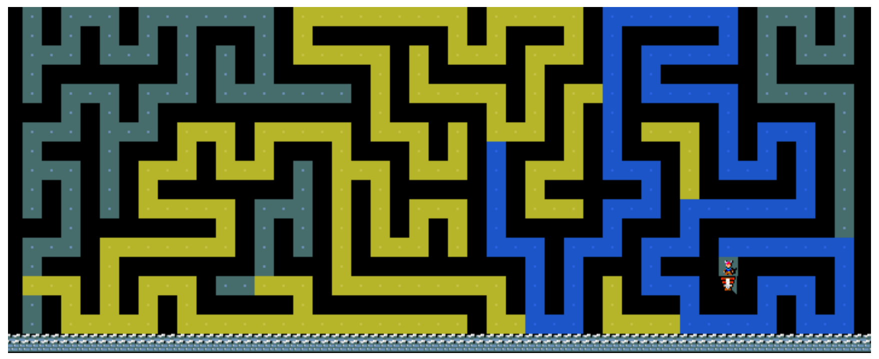
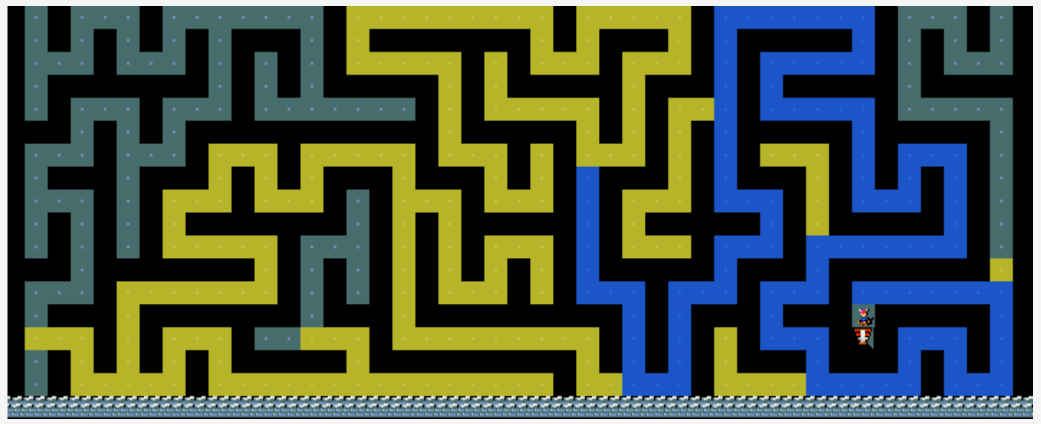
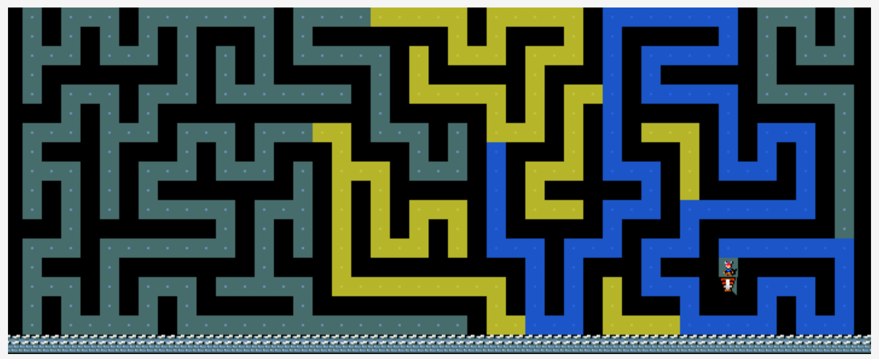
What is the SManhattan distance?

The SManhattan distance is the Manhattan distance with a little twist, it's the Manhattan distance mutlipied by a factor that is a costant..

We took our version of the Manhattan distance and we modified it to be the SManhattan distance... being defined as:

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

```
100%|██████████| 106/106 [00:01<00:00, 90.18it/s]
```

Algorithm	Time	Visited	Path	Image	Algorithm	Time	Visited	Path	Image
A* SManhattan	0.107	201	107		A* Manhattan	0.103	287	107	
A* Euclidean	0.106	288	107		A* NNHeuristic	5.644	200	107	

As we can see, the SManhattan distance is the best 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.

Objective

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.

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

Methodologies

Now.. How can we 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.optimize
rs.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 now have a csv file with all the results, we can now analyze them, but we are gonna do something better. Let's create a scoring system to compare the algorithms.

Scoring System (Assessment)

Basically the ratio between the path length and the visited cells length.

```
0    [(15, 41), (15, 42), (15, 43), (15, 44), (14, ...
1    [(11, 36), (11, 35), (11, 34), (10, 34), (9, 3...
2    [(7, 41), (7, 40), (7, 39), (7, 38), (8, 38), ...
3    [(15, 41), (15, 42), (15, 43), (15, 44), (14, ...
4    [(11, 36), (11, 35), (11, 34), (10, 34), (9, 3...
Name: Path, dtype: object
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000 entries, 0 to 2999
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   Algorithm       3000 non-null   object 
 1   Solved          3000 non-null   bool   
 2   Path            3000 non-null   object 
 3   Visited         3000 non-null   object 
 4   Time            3000 non-null   float64
 5   Maze_Name       3000 non-null   object 
 6   Seed            3000 non-null   int64  
 7   Path_Length     3000 non-null   int64  
 8   Visited_Length  3000 non-null   int64  
dtypes: bool(1), float64(1), int64(3), object(4)
memory usage: 190.6+ KB
None
```

Results

Drumroll, please! The moment of truth has arrived.

	Algorithm	Time_mean	Time_std	Visited_Length_mean	Visited_Length_std	Path_Length_mean	Path_Length_std	Path_Score_mean	Path_Score_std	Solution_Score_mean	Solution_Score_std
0	GREEDY-Manhattan-Env1	0.011785	0.007558	122.03	84.575763	77.94	50.930497	0.686225	0.157906	0.69098	0.204586
1	GREEDY-SManhattan-Env1	0.012042	0.007545	122.03	84.575763	77.94	50.930497	0.686225	0.157906	0.69098	0.204586
2	ASTAR-NNManhattan-Env1	3.822897	2.794039	129.22	96.026993	75.84	47.964153	0.678001	0.177119	0.69942	0.192659
3	ASTAR-SManhattan-Env1	0.012503	0.007660	131.24	96.968194	75.84	47.964153	0.664671	0.172712	0.69942	0.192659
4	ASTAR-Manhattan-Env1	0.013183	0.010007	160.89	109.755734	75.28	47.089230	0.541945	0.163292	0.70168	0.189126

	Algorithm	Time_mean	Time_std	Visited_Length_mean	Visited_Length_std	Path_Length_mean	Path_Length_std	Path_Score_mean	Path_Score_std	Solution_Score_mean	Solution_Score_std
0	ASTAR-NNManhattan-Env2	1.047521	0.708385	34.13	24.992385	23.47	15.930693	0.772143	0.179603	0.90992	0.064036
1	ASTAR-SManhattan-Env2	0.011277	0.007620	34.55	25.176298	23.47	15.930693	0.765570	0.180732	0.90992	0.064036
2	GREEDY-Manhattan-Env2	0.011131	0.007682	35.11	24.036544	24.21	16.623152	0.725002	0.155965	0.90693	0.066837
3	GREEDY-SManhattan-Env2	0.011163	0.007603	35.11	24.036544	24.21	16.623152	0.725002	0.155965	0.90693	0.066837
4	ASTAR-Manhattan-Env2	0.011269	0.007829	37.48	25.528942	23.47	15.930693	0.699493	0.174474	0.90992	0.064036

	Algorithm	Time_mean	Time_std	Visited_Length_mean	Visited_Length_std	Path_Length_mean	Path_Length_std	Path_Score_mean	Path_Score_std	Solution_Score_mean	Solution_Score_std
0	ASTAR-NNManhattan-Env3	0.399369	0.223056	12.00	7.739444	9.63	5.176979	0.878512	0.162095	0.96548	0.020708
1	ASTAR-SManhattan-Env3	0.011111	0.007642	11.95	7.576379	9.59	5.146530	0.875009	0.160810	0.96564	0.020586
2	ASTAR-Manhattan-Env3	0.011035	0.007396	12.36	7.852838	9.49	5.042236	0.845445	0.158404	0.96604	0.020169
3	ASTAR-EUCLIDEAN-Env3	0.011041	0.007799	13.01	8.129625	9.49	5.042236	0.811554	0.165666	0.96604	0.020169
4	ASTAR-Chebysev-Env3	0.011254	0.008491	13.60	8.336363	9.49	5.042236	0.776940	0.169756	0.96604	0.020169

Small Considerations

We can see, our Pseudo Heuristic is doing well... Always being in the top scores.
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, both are doing great.

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.

Challenges Faced

We had a lot of challenging moments on trying to use the minihack envirorment...
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.

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 new heuristic that we called SManhattan, which is the Manhattan distance multiplied by a constant.

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.