CMP9764M Comaring a Q-learning driven agent to an IRL based agent



line 1: 2nd Stephen Rerri-Bekibele  
line 2: *School of Computer Science. University of Lincoln   
(of Affiliation)*line 3: Lincoln, United Kingdom  
line 4: 16663359@students.lincoln.ac.uk

Table of Contents

[Introduction 2](#_Toc95791866)

[Concept 2](#_Toc95791867)

[Evaluation 3](#_Toc95791868)

[Appendix 5](#_Toc95791869)

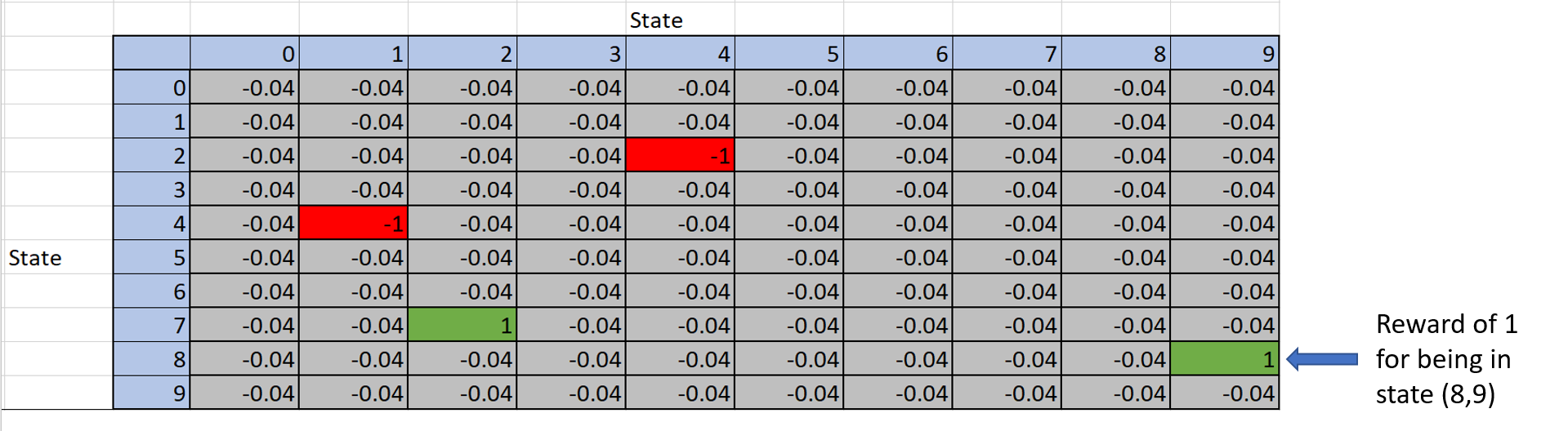
*Abstract*— Programming an AI to navigate an environment is a very common occurrence in the gaming world. There are many different path finding algorithms [2] and ways this has been done. From A\*[3] to Dijkstra’s algorithm [4]. The aim of this research paper is to take on such a task and use an MDP [1] approach to program an AI to survive as-long-as possible in arenas of varying sizes, whilst also collecting bonuses in a non-deterministic World simulation. This is repeated for a partially observable world as well as a fully observable world while the AI is also dealing with Stochastic actions [6]. The outcome of this research paper proves that an AI is more capable of effective planning and surviving longer when using MDP solutions in a fully observable scenario than it is in a partially observable scenario.

Keywords—MDP, survive, deterministic, planning, solutions, observable, Stochastic actions (key words)

# Introduction

Preliminary results showed that the Q-learning is much faster than an IRL agent because of the need for human confirmation at each step as not only could the human also make mistakes at time since there is no policy being used aside from the human’s choice of what they think is the right action, the human can also carelessly classify bad actions as good and good as bad. Moreover the IRL agent needs to suggest a good alternative to an action if the human says an action is bad otherwise the IRL agent ends up stuck in its current position until a good move is suggested, this means the agent takes a lot more time to move if it keeps suggesting wrong actions, there needs to be a way to let the agent know an action is bad due. This can be done by looking at the last actions suggested by the agent and if they are bad actions then the agent suggests a new action at random that is not one of the previous actions. This reduces how long it takes the agent to move but also doesn’t take the freedom of choice from the agent because if it given exactly 3 chances to pick a right action then in a true deterministic world the agent would have picked a different potentially right action as there are only 4 actions.

The mean number of states the IRL agent took to get to

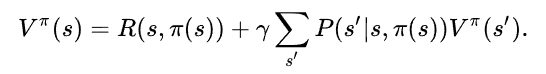


The example above is a replica of the reward matrix used in this research and illustrates that there are positive and negative rewards i.e., +1 and -1, for being in certain states of the grid such as the positive rewards in states (8, 9) and (7, 2). In non-terminal states, the reward is set as -0.04. By assuming that the utility of a run is the sum of the reward states, the -0.04 is an incentive for the AI to take fewer steps to get to the terminal state [11].

For the base grid of size 10 by 10 used in this project, the MDP requires as input, two matrices of data, one is a Probability matrix A.K.A transition model P (s’| s, a), of shape (A, S, S) where A are actions and S are states. This specifies an array of possible actions (A) for each state. Each S x S, then specifies the transition probabilities of reaching the second state by applying that action in the first state [6]. This meant the probability matrix would have been a (4,100,100) matrix because there were 4 possible actions the AI could take at any state.

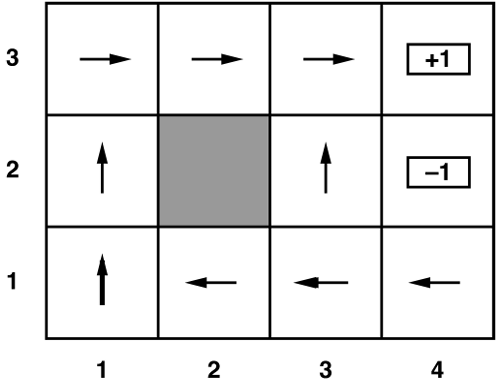
The other, a reward matrix of shape (S, A) A.K.A reward function R(s), which specifies a set of S vectors equivalent to the S set of states, one for each state and each is a vector with one element for each of the actions. Each element, (also called the expected utility), is then the reward for executing the relevant action in the state. (This models the cost of actions) [6]. For a grid of size 10 by 10 the reward matrix was a (4,100) matrix, each column belonging to an action and each row, a state of the grid and within that state was the reward for being in said state.

The Probability and reward matrices can be generated recursively using the bellman equation [10] to compute the expected utilities for each state of the maze grid.



1. The Bellman Equation used to compute the probability value of each cell for a change of state by a given action

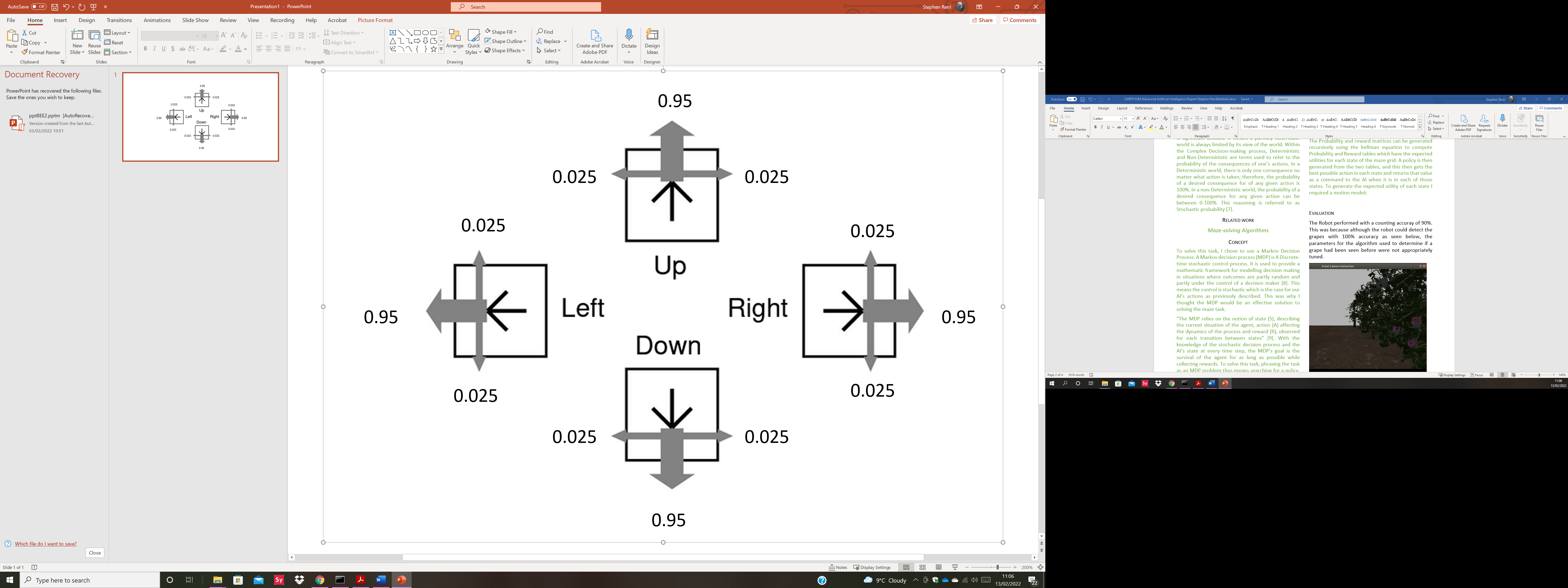
A policy much like:



1. An example of a simplified policy that will be generated by the MDPtoolbox library from the Transition model and Rewards function.

is then generated from the two tables, and this then gets the best possible action in each state and returns that value as a command to the AI when it is in each of those states.

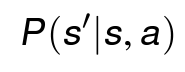
To generate the expected utility of each state I required a motion model:



1. A motion model to illustrate the stochastic nature of the AI’s actions

95% of the time the AI moved as intended and 0.5% of the time it moved perpendicular to the direction indented. Half the time to the left and half the time to the right.

To describe each action a transition model can be used and since the actions are stochastic, each can be described as:



Where a is the action that takes the AI from s to s’. These transitions are assumed to be (first order) Markovian and therefore only take into consideration, the current and next states.

In a special case such as whenever the AI hit a wall, a “wall bouncing algorithm” had to be developed. This algorithm simply returned the probability of the action in the current state plus the probability of taking the action as the value of the current state.

As this research was divided into two solutions, one for a fully observable environment and another for a partially observable environment, the policy was only used whenever the AI had enough information to create a policy, i.e., the AI had to have seen at least one bonus, a meanie, and a pit. This meant the policy would be inactive a lot of the time in the partially observable scenario and instead the AI would have to use a greedy approach to find bonuses and avoid pits and meanies.

This method was chosen because it allowed the AI to dynamically switch between two states, one where it moved with a policy that told it the best action for any state the AI was in and another that allowed the AI to successfully navigate to bonuses whilst avoiding meanies and pits coming in and out of visibility limit.

# Evaluation

Surprisingly, the AI didn’t always perform better when the partially observable parameter was set to FALSE. This is evident because the MDP was not perfect and could not always provide a better policy to follow before too many obstacles were present as new meanies were added after a few time steps of the simulated world.

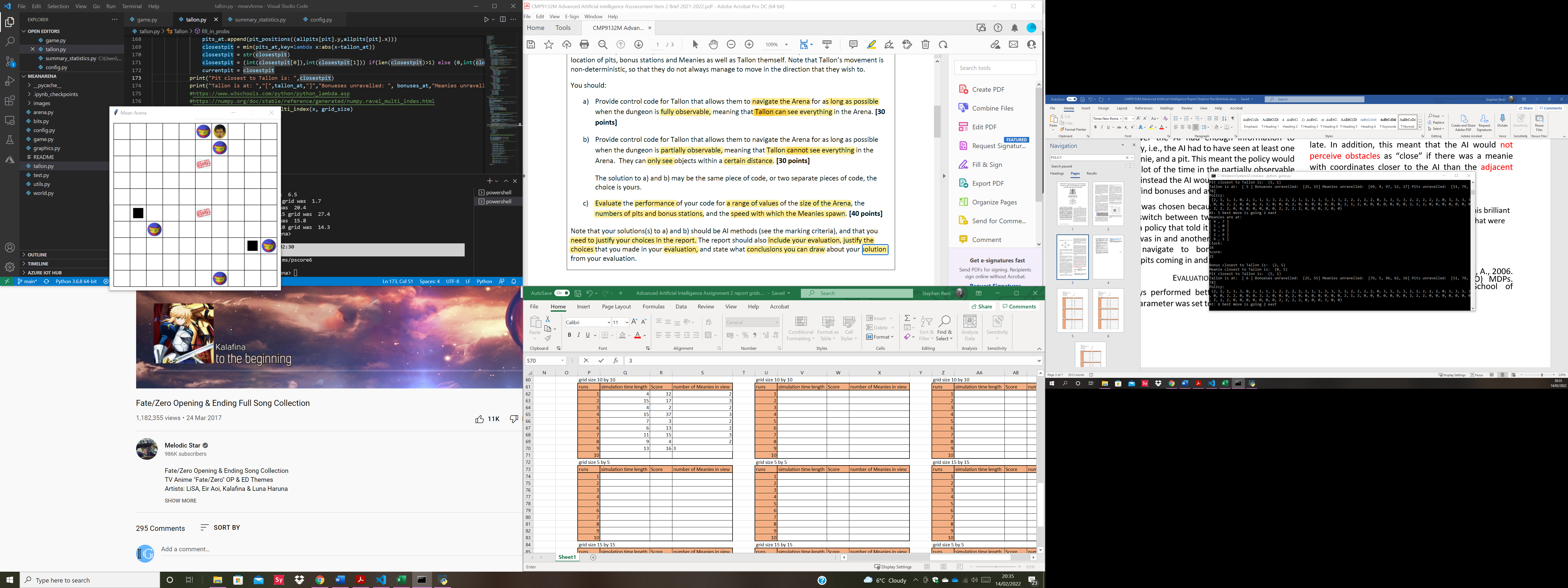
Three scenarios were created with varying amounts of bonuses, pits, arena sizes and speed of spawning meanies. For the sake of word count only the 10 by 10 grid will be analyzed in detail and the result tables in the appendix will show the results of the other tested parameters.

When partially visible was TRUE, the first scenario had a mean score of 14.6 and survival time of 9.6(s) and the second scenario which has double the parameters of the first scenario had a mean score of 26.5 and survival time of 21.4(s). Lastly, the 3rd scenario which yet again doubled the parameters had a mean score of 15.8 and survival time of 14.3(s).

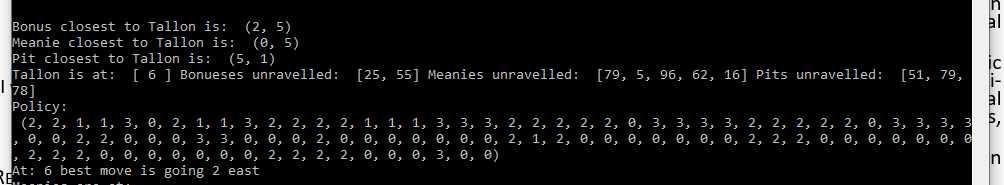
In contrast, when partially visible was FALSE, the first scenario had a mean score of 15.6 and survival time of 11.2(s). The second scenario had a mean score of 16.9 with survival score 7.6 and lastly, the third scenario had a mean survival time of 19.7 and survival time of 10.3.

At the time of this test, this demonstrated that in most cases, a larger grid size led to longer survival times and a higher bonus. In smaller grid sizes the AI was getting caught more quickly, resulting in a lower score and survival time. Tables of results can be found below in the Appendix.

In conclusion, my method of generating a reward matrix via the bellman equation fell short because it recursively added each reward but only one at a time for the arrays of bonuses, pits, and meanies. I therefore had to create a priority queue whereby only the instances of each obstacle or reward closest to the AI at every time step would be added to the reward matrix and dealt with at each time step. This meant that when dealing with multiple meanies (Figure x) the AI could get caught with a pincer move because only the closest meanie chasing it would have a negative reward associated with it and thus it would not see another meanie coming from another direction until it was too late. In addition, this meant that the AI would not perceive obstacles as “close” if there was a meanie with coordinates closer to the AI than the adjacent obstacle



1. An image of the grid showing how Tallon can get trapped by multiple meanies. In this example the policy gave the best action for that state and got Tallon out to survive for longer.



1. This shows the policy in Figure 4 and how Tallon interprets the closest obstacles.

This shows that although the meanies at (1,6) and (0,5) were equally close to Tallon if we look at the graph, Tallon still sees the closest Meanie as the one at (0,5) since Tallon is at (0,6) and this is dangerous because it means Tallon could have moved into the adjacent meanie at (1,6).

Suggestions for improvement include finding a way to add all current meanies, pits and bonuses into the reward function so the MDP library can create a better policy and making the code more programmer friendly.

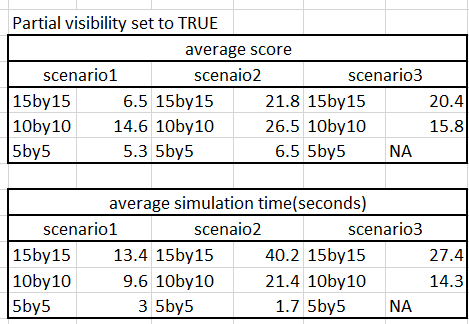
##### Acknowledgment

I would like to acknowledge Simon Parsons for his brilliant teaching and for bringing to light, techniques that were applied to this project

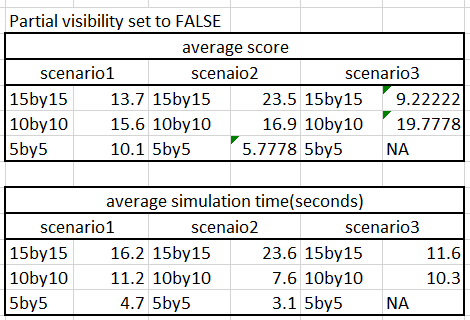
##### References

1. Toussaint, M., Harmeling, S. and Storkey, A., 2006. Probabilistic inference for solving (PO) MDPs. Technical Report EDI-INF-RR-0934, School of Informatics, University of Edinburgh.
2. Korf, R.E., 1988. Optimal path-finding algorithms. In Search in artificial intelligence (pp. 223-267). Springer, New York, NY.
3. Lester, P., 2005. A\* pathfinding for beginners. online]. GameDev WebSite. http://www. gamedev. net/reference/articles/article2003. asp (Acesso em 08/02/2009).
4. Goyal, A., Mogha, P., Luthra, R. and Sangwan, N., 2014. Path finding: A\* or dijkstra's?. International Journal in IT & Engineering, 2(1), pp.1-15.
5. Qudrat-Ullah, H., Spector, J.M. and Davidsen, P. eds., 2007. Complex decision making: Theory and practice. Springer.
6. Simon Parsons, 2022. Lecture 7: Complex decision making. https://uol.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=b69f615e-0f8f-45d0-9dcf-adef00962046
7. Florescu, I., 2014. Probability and stochastic processes. John Wiley & Sons.
8. Wikipedia, 2022. Markov decision process. https://en.wikipedia.org/wiki/Markov\_decision\_process
9. Garcia, F. and Rachelson, E., 2013. Markov decision processes. Markov Decision Processes in Artificial Intelligence, pp.1-38.
10. Peng, S., 1992. A generalized dynamic programming principle and Hamilton-Jacobi-Bellman equation. Stochastics: An International Journal of Probability and Stochastic Processes, 38(2), pp.119-134.
11. Simon Parsons, 2022. Lecture 9: Markov Decision Process. <https://uol.cloud.panopto.eu/Panopto/Pages/Viewer.aspx?id=6f0e8ec4-045d-401d-b4c1-adf600963e25>
12. Chadès, I., Chapron, G., Cros, M.J., Garcia, F. and Sabbadin, R., 2014. MDPtoolbox: a multi‐platform toolbox to solve stochastic dynamic programming problems. Ecography, 37(9), pp.916-920.

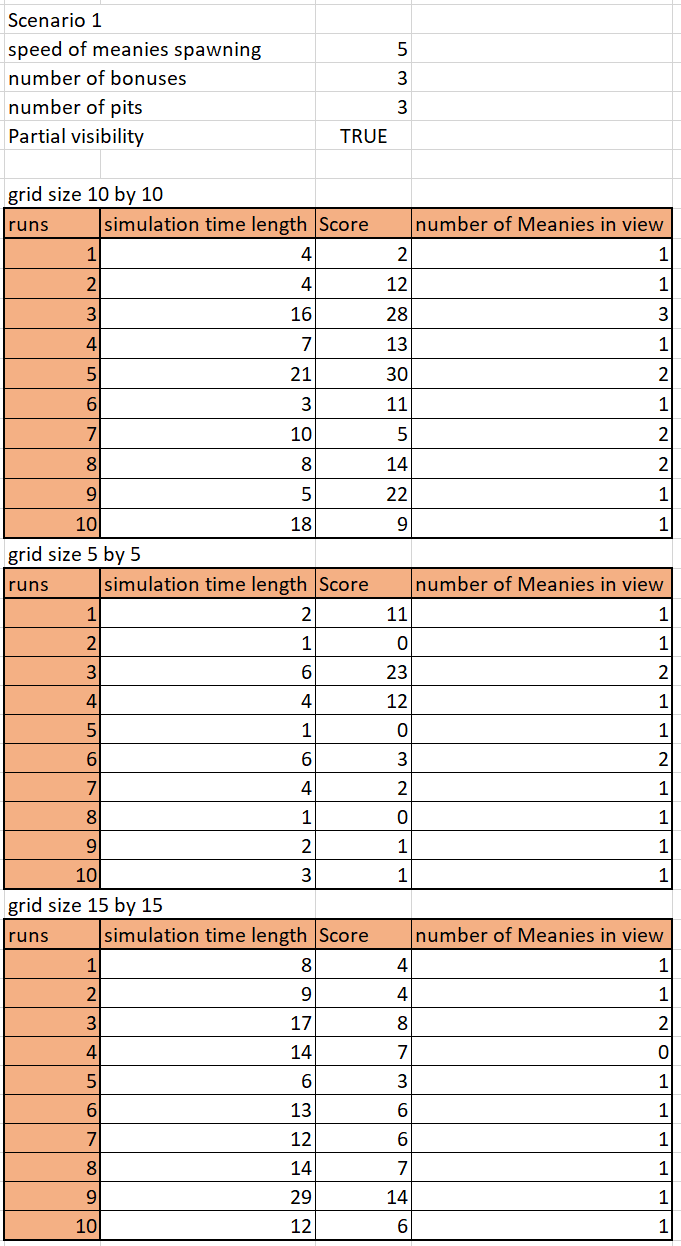
# Appendix

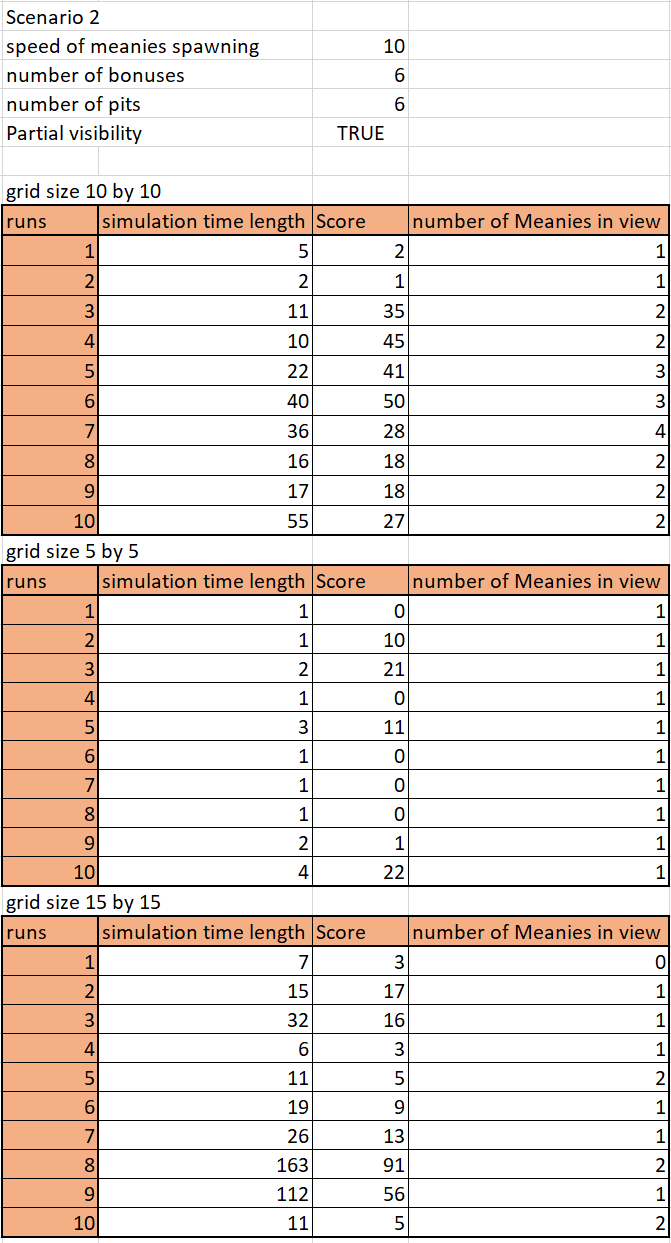
**

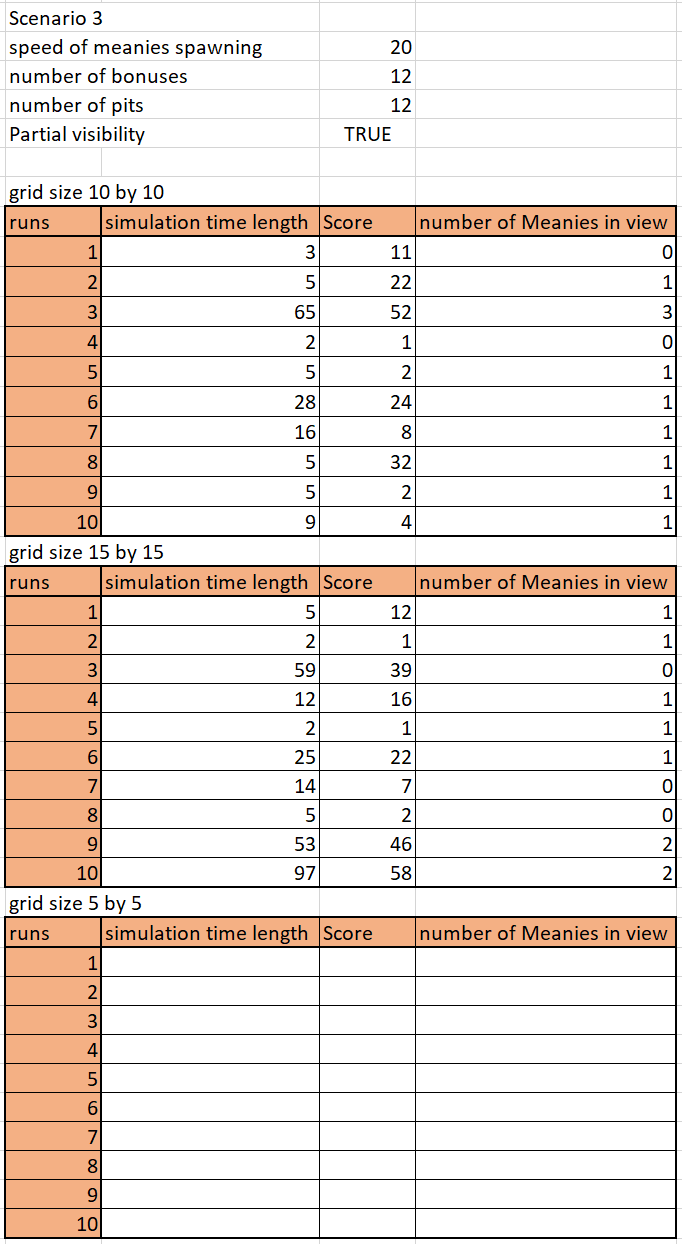
1. *This is an image of my stat table which shows Tallon’s performance when partial visibility was set to TRUE for 3 scenarios that are outlines in more detail in the images below*

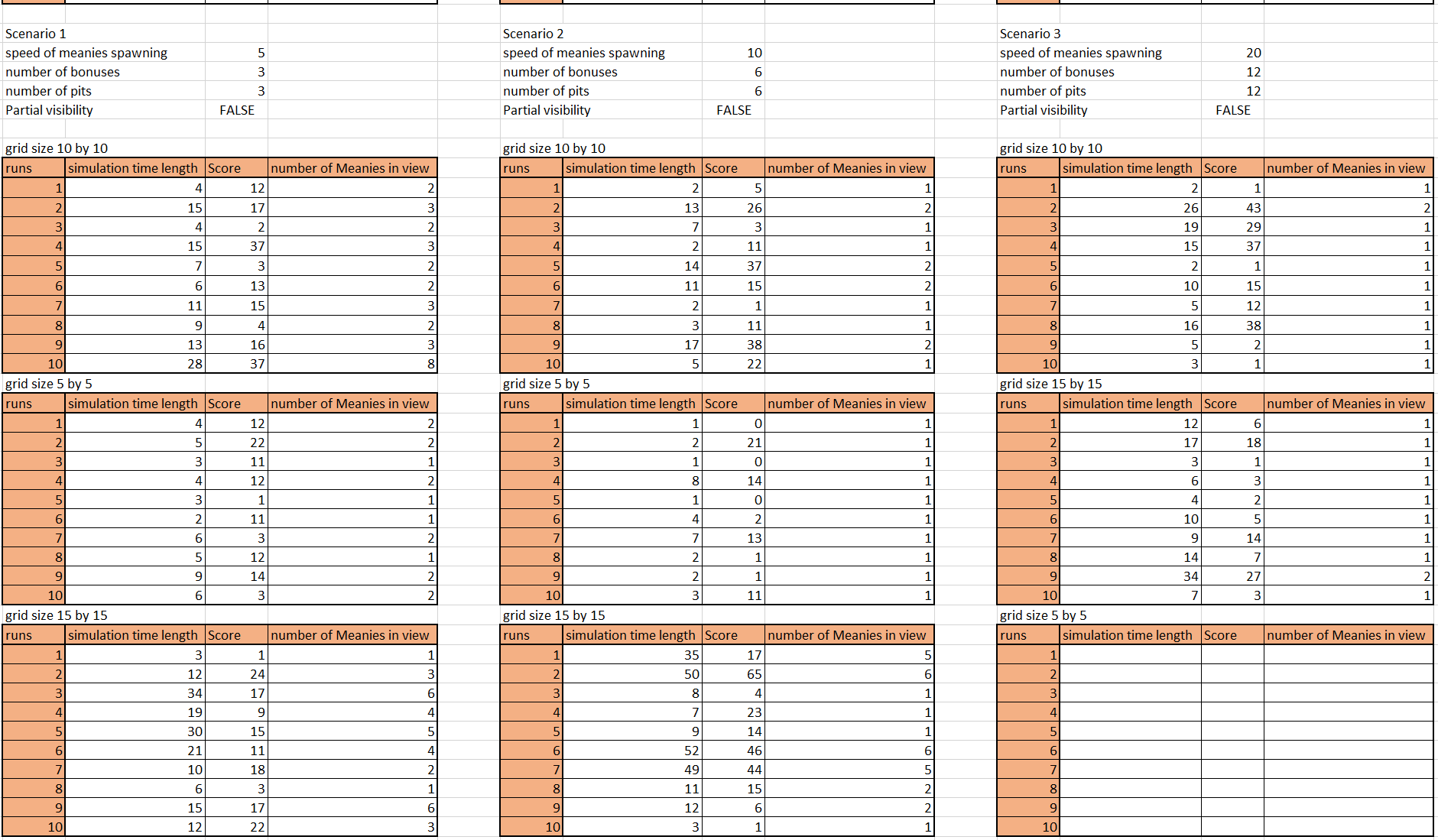
**

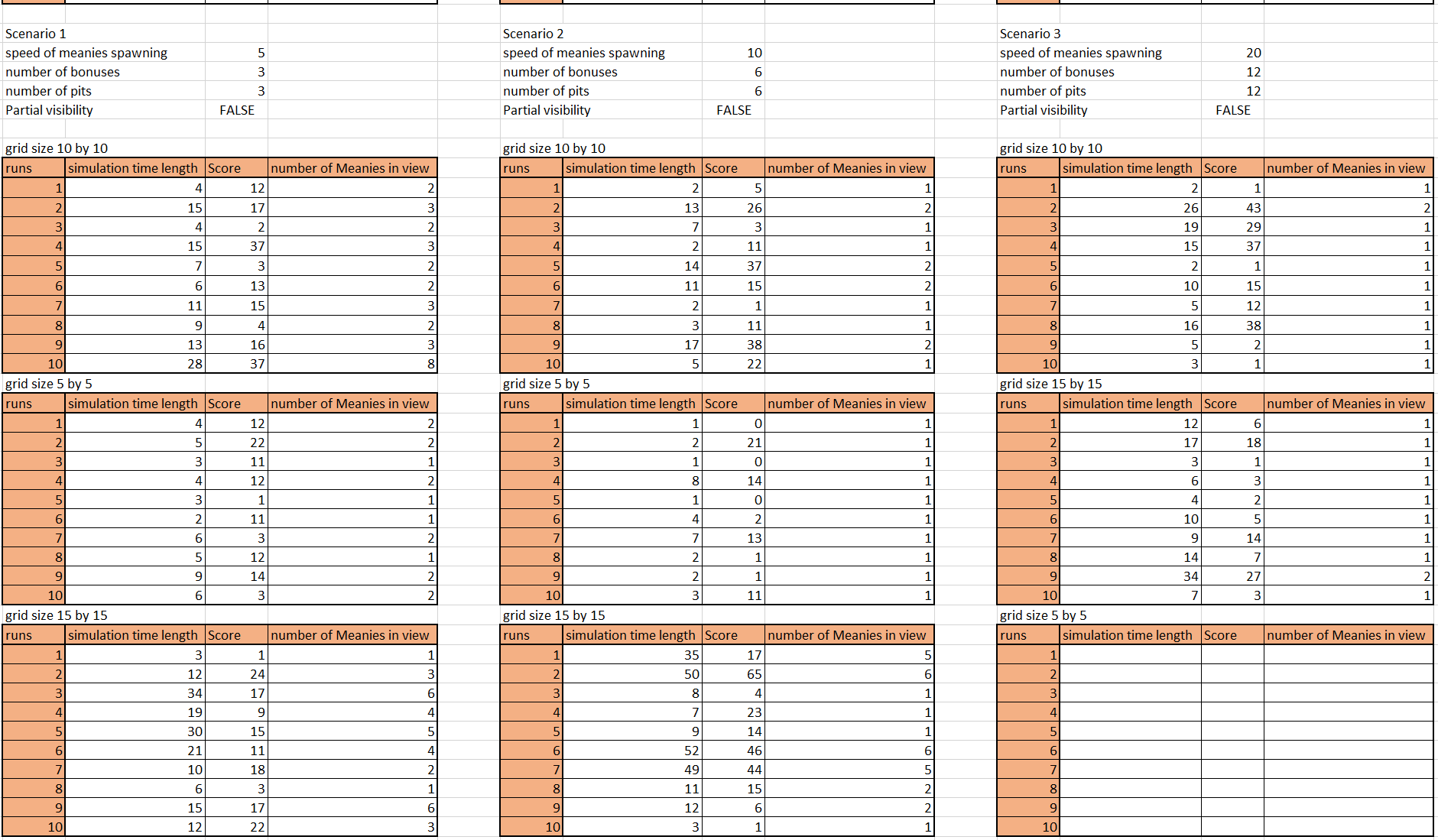
1. *This shows Tallon’s performance when partial visibility was set to FALSE for the 3 scenarios outlines in more detail in the images below*

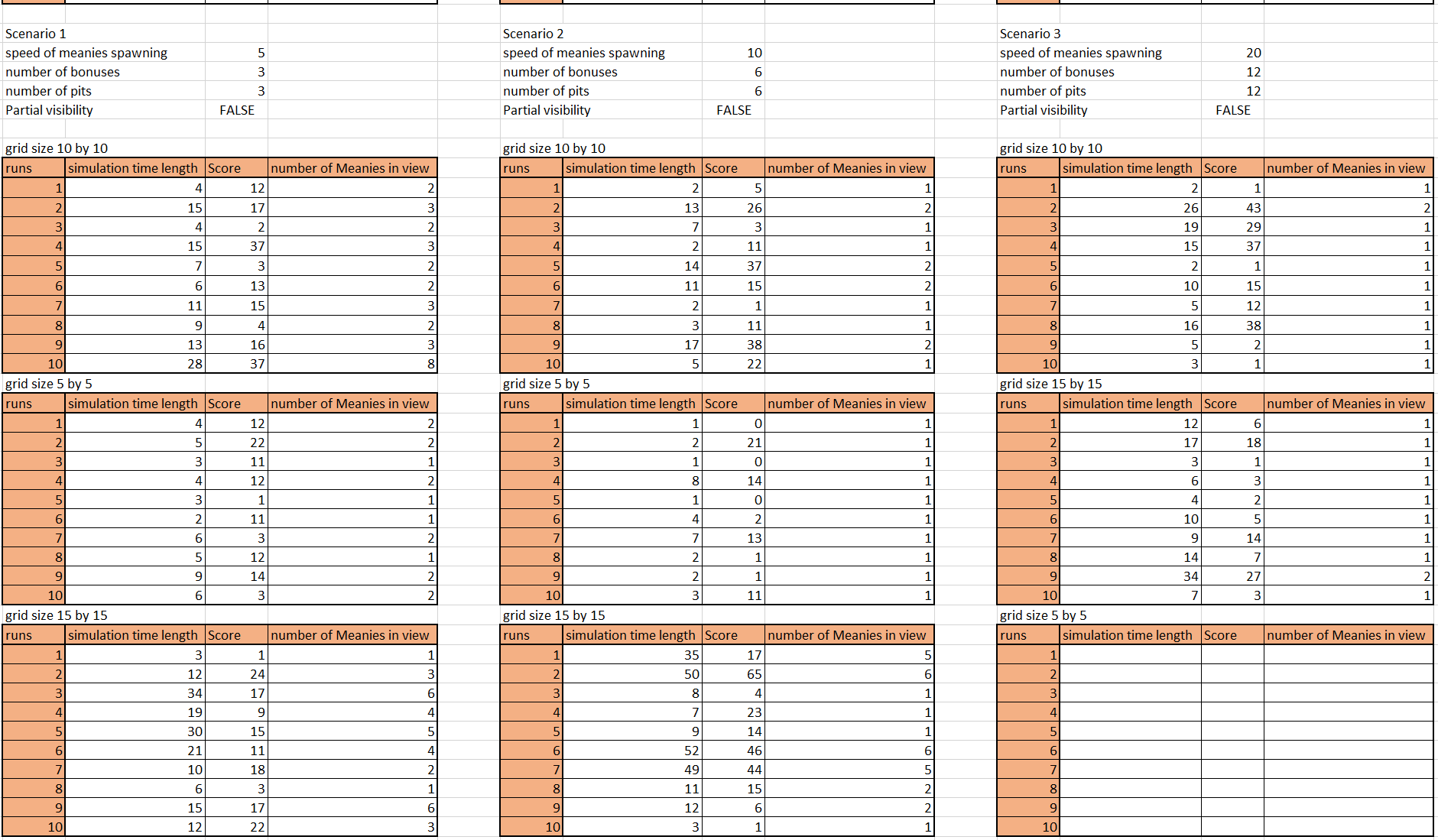
**

**

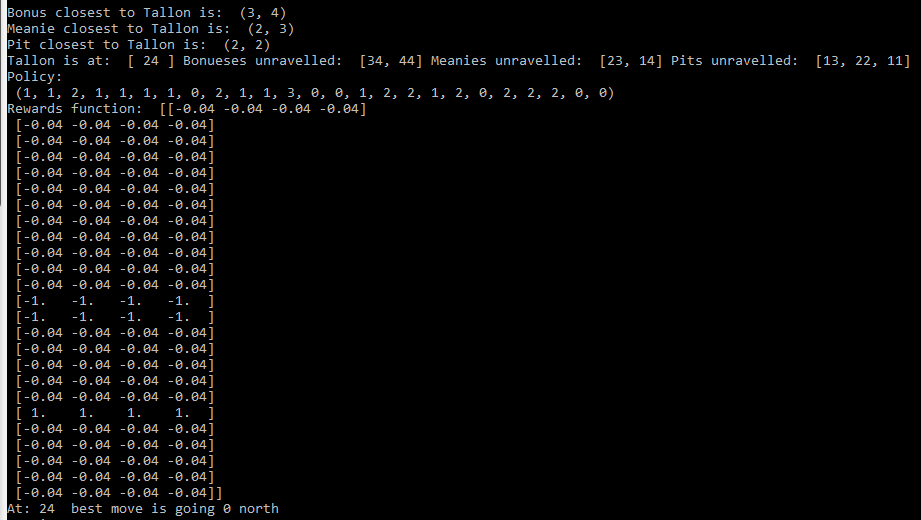
**

**

**

**

*The Third scenario does not test a 5 by 5 grid for either partial or full visibility because it is not possible to simulate this as the grid dimensions are too small to spawn in all the assets*

*Image of rewards function which shows that although there were two bonuses, two meanies and three pits all active, the rewards table only has one present at each time step, the closest one to Tallon is the only one present. This enables Tallon to get caught off guard by enemies or traps ahead of it when it is moving in the y axis.*