Reinforcement Learning Agent in Wumpus World

A. Nicholas Powell (120217167), B. Omkar Popalghat (1217191287), C. Siddhant Sangal (1217670883)

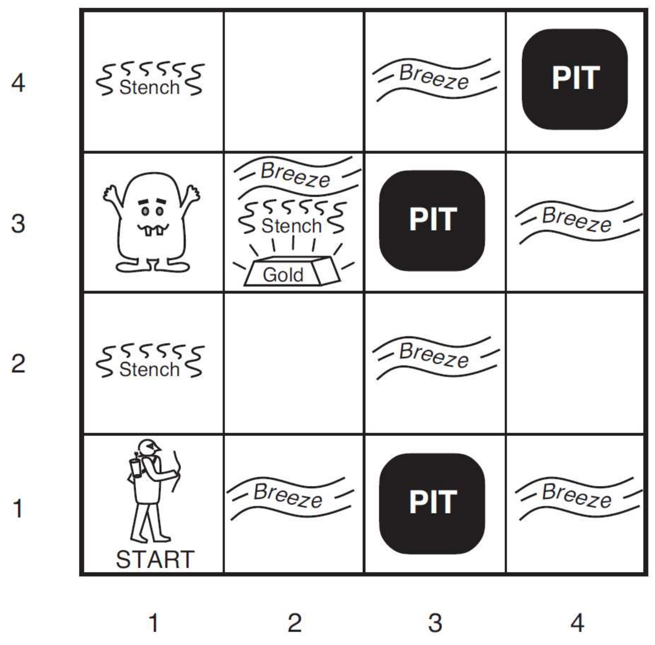
***Abstract*—The project as described in the report has been developed under the team project-based submission for the class CSE 571 Artificial Intelligence. It focuses primarily on incorporating noise action model into the Wumpus world environment that was taught in class and projects were also performed on the same. This project aims to implement the agent that is primarily a hybrid logic agent as a reinforcement learning agent that can reason about and learn from performing various trials on the environment and learning various strategies and then finding the optimal one. The implementation of the various versions as explained above are various reinforcement learning techniques that have varied degree of accuracy and situational advantages. The agent as shown learns over time about the various characteristics or variables in the environment that contribute to it's optimal end reward when it exits the environment setting. The behaviour is governed by tuning the rewards obtained from entering various states. Furthermore, the project aims to compare the performance of the agent under different settings and draw conclusions regarding the same.**

***Index Terms*— Agent, Wumpus, Reinforcement Learning, environment, rewards, state, deterministic, stochastic, noise model, utility, convergence.**

I. Introduction

THIS project has been performed as a team with regular updates and code files being posted on the GitHub repository. As the name of the project suggests, various reinforcement learning techniques are implemented onto the same agent in the same agent and the performance of the agent has been mapped out for comparison. As taught in class and mentioned in the textbook, there are various reinforcement learning techniques that can be used in order to convert a basic logic or a hybrid logic agent into an intelligent one that learns from mistakes and successes. The environment that has been given to work on is that of the Wumpus world which is a classic grid search environment used for testing and implementing various algorithms. We later modify the environment which by default has a deterministic action model into a noisy model where the actions are now stochastic. The next step is to implement various Reinforcement Learning techniques so as to make the agent learn by explorations and exploitation of various strategies under noise and find the behaviour or action sequence that yields an optimal solution.

The important variables in the environment that are directly associated to the rewards are the pit, the ghost, the gold and the arrow. The agent performs exploration in the initial stages so that it can understand the dynamics of the game setting. This environment also contains adversaries that end the game with a negative terminal reward that is not favourable for finding optimal solution. A typical Wumpus world environment can be represented as following:



(image credit: Artificial Intelligence: A modern approach, Stuart Russell & Peter Norvig)

II. Technical Approach

The approach for the simulation begins with understanding the dynamics of the problem. The idea behind the construction of the environment being arranging the pit, Wumpus and gold in various combinations while not violating the rule of overlapping with each other. It can be observed that the pit has a breeze around it and the Wumpus has a stench around it. The agent uses its sensors to detect these and update its knowledgebase when it is a logic agent. But when it behaves as a reinforcement learning agent, it does not know the implications of the presence of a breeze or stench and will end up moving in an intuitively wrong direction but learn from the obtained reward simultaneously.

.

A. Wumpus World Noise Action Model

The agent that tries to maximize the utility based on logic-based behavior primarily functions on the knowledgebase and sensor perceptions. The 'world' consists of grid locations (henceforth referred to as 'state') connected by tunnels along with environment variables [1]. The sensor on the logic agent can sense only the current state and its characteristics with a maximum of [breeze, stench and gold/shine] existing simultaneously. The noise action model behaves such that the agent has a Transition probability T(s,a,s') of going from state 's' to ' s' ' by performing an action 'a'. The probability of performing this correct action is 0.8. With a probability of 0.2, it randomly choses one of the other actions from [Forward, Left, Right, Shoot]. This makes the model non-deterministic. A tabular representation for the same is as following:

P(selected action)	0.8
P(incorrect action)	0.2

In terms of the code, there were two implementations of the noise action model of which the later was correct and the former was not. The first attempt to incorporate noise was done where action is decided i.e. the path is not yet formed. After the implementation, it was realized that action sequence returned from the planning was such that it would break the logic agent as it was derived from a noisy search of the states. Thus, a second approach was implemented where the stochasticity of 0.8 transition probability was implemented on the returned list of states. This implementation is intuitively correct and performs as required. The second approach was also confirmed by discussion and review by the instructor.

B. Reinforcement learning techniques

Various reinforcement learning techniques have been taught and learnt over the course of the semester which can be broadly classified into model-based and model-free reinforcement learning techniques. There are various methods such as Monte-Carlo method (Direct Evaluation), Q learning method, Temporal Difference learning, Approximate Q learning, Adaptive Dynamic Programming, SARSA(state-action-reward-state-action) and more [7]. This project focuses on the comparison between Monte-Carlo method and Q learning techniques. An explanation for the two and a technical intuition for the implementation of the same are as following:

(i) Monte-Carlo method (Direct Evaluation): The main goal in either of the techniques is to calculate the value of each state depending on the reward gained from reaching that state over various iterations. Monte-Carlo method is in a way a greedy algorithm that takes into consideration the summation of the immediate rewards i.e. living and terminal in order to evaluate the state and learn from the overtime convergence to enter the state or not and if so, with what action [3]. The formula for evaluation is as following:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

The implementation in code is using the above formula where the assigned living reward and the discount factor gamma are summed till the terminal state. This update takes place over each trial till the value converges. The convergence occurs when the agent learns the best state action pair sequence to maximize the overall utility.

(ii) Q Learning technique: The Q learning technique is essentially a modification of the Bellman equations which the same as above, are used to calculate the value of a state [4]. The implementation of this had been done previously in a deterministic setting for the course. The equation utilized is as following:

$$Q(s,a) \leftarrow Q(s,a) + \alpha(R(s) + \gamma \max_{a} Q(s',a') - Q(s,a))$$

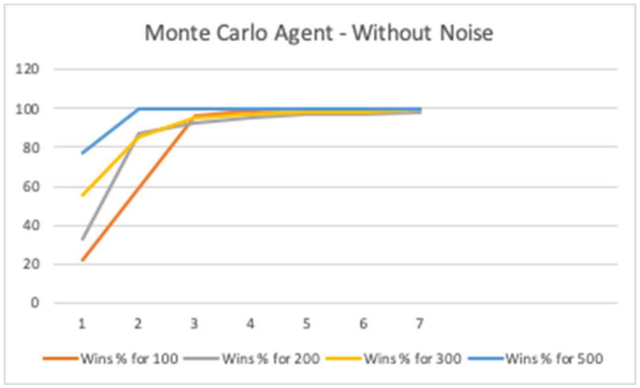
We start with an initial percept. Beginning with the initial state which is always (1,1). We initialize the Q value for all states as zero. As we choose an action for each state, we update the corresponding Q value for each iteration till we reach a convergence, or the value update behaves asymptotically. In order to make the implementation quicker, an upper limit of 1000 iteration has been set which provides us with satisfactory values to work with. Once the Q values has been set for each state, the agent will now behave in a greedy manner selecting the highest q value-action pair at each state to determine the optimal path. Using Q values has been found to be better than direct evaluation in terms of performance and overall accuracy in the given setting which will be explained in the subsequent module.

III. Results, Analysis and discussion

The coding part for Monte-Carlo simulation involved two stages where the model was initially trained in an environment that is noise-free and then on a noisy environment in order to compare the statistics. The agent was exposed to two environments to compare its performance in different environments and the statistics obtained reflect the intuitive behavior. The explanation for each section is as follows:

A. Deterministic Monte-Carlo technique

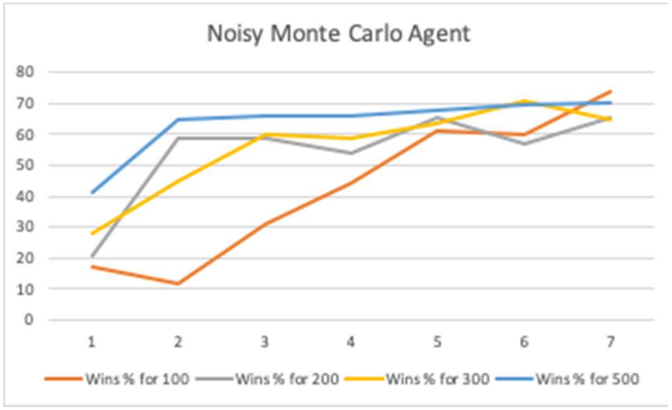
This part of the project is the metric for comparison with the performance under stochasticity. As it can be observed from the graph below, the agent performs very well from the onset. The number of wins since the first trial is relatively high and it converges to a high win rate over multiple iterations of dictionary update and learning. The agent has been trained up to 500 iteration training set at a time which requires relatively high amount of time by the last trial which as it was observed over the course of the project is an expected behavior. The following graph depicts the convergence of win rate over a high learning period and reflects a good learning-based performance.



The amount of time required for the training of the agent was quite high and it is expected as it is one of the drawbacks of the direct evaluation method. The direct evaluation method when executed over high number of instances turns out to be a very useful algorithm as the agent performs in an almost ideal manner. It can be observed that the agent does not have a win rate of 1for hundreds of initial instances which is due to the fact that the agent maintains a balance between exploration and using the newest optimal policy. It eventually converges to a win rate of 1 and maintains the same as it has converged to the most optimal set of actions. This behavior is unique to deterministic models and it will be demonstrated in the following sections that even a small uncertainty makes the agent perform in a less optimal fashion.

B. *Mont-Carlo with noise action model*

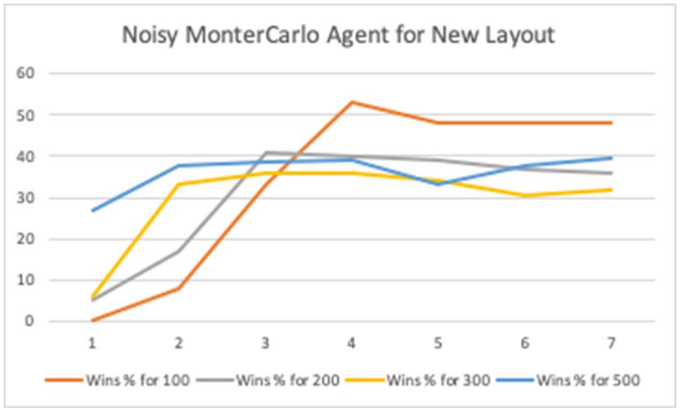
In this part of the project, the noise model has been incorporated and the effect on the performance is clearly visible. Furthermore, the inclusion of noise or randomness is also present in the algorithm. It is such that 20% of the times, the action is completely ignored which is toggled by an addition variable in the code. It uses a normal distribution to vary its selection among any similar choices present. The agent starts with 100 iterations and performs up to 500 iterations with the last one taking remarkably high time as expected. It is clearly visible from the attached graph below that the number of losses is quite high in the beginning but as the agent updates the dictionary at each instance, the number of wins increase and finally converges to a respectable performance. The noise in this version is such that the agent ends up in the intended state with a transition probability of only 0.8. The effect of this is reflected from the reduced performance as compared to the noiseless model.



The analysis of the behavior is in tandem with the intuition that the agent tries to move away from the negative reward environment variables which are the pit and the Wumpus but due to the stochastic nature lands in the wrong state and ends the game. It can also be observed from the gradual improvement in performance that the agent learns to keep it's distance from such locations and if it does detect one in its neighborhood, to take action that will be in the opposite direction so as to not land up in the negative reward states even by accident. The number of losses is relatively high as compared to the noiseless setting which is also due to the fact that the environment, we are working with is quite small and the pits and Wumpus are quite close to each other.

C. *Monte-Carlo noise action model under new environment.*

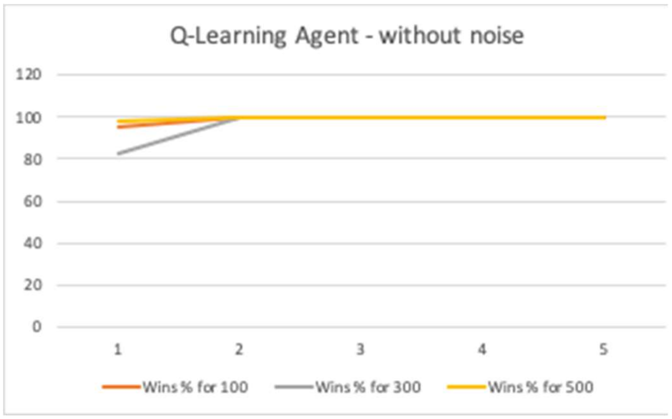
The agent now is trained on a new version of the environment with the same size and environment variables but the pit, Wumpus and gold are now present in different locations. This version of implementation shows the versatility of the agent and proves that the logic implemented works in any type of environments long as it does not violate the game rules. As expected, the performance of the agent is low in the beginning but later converges to a respectable win rate and learning performance. The graph below is intuitively quite like the performance in the previous environment.



When the agent is pretrained on one environment and then run on a new one, the performance off the bat is significantly improved. This has been further discussed in the following sections. For an environment where the size is higher and the negative reward environment variables are further apart, the agent should be able to perform even better as it has more freedom to chose which action to take. It is possible in the current environment that the agent might not be able to take the action to move in the opposite direction to the pit or Wumpus as there would be another pit or a Wumpus in the opposite direction or in the orthogonal one making the decision making process harder. Thus, for an environment with lots of open spaces, the agent will perform much better, although the exploration aspect would take considerably high amount of time and so will the convergence in win rate. The above is due to the fact that the number of states and permutations of the same increases exponentially.

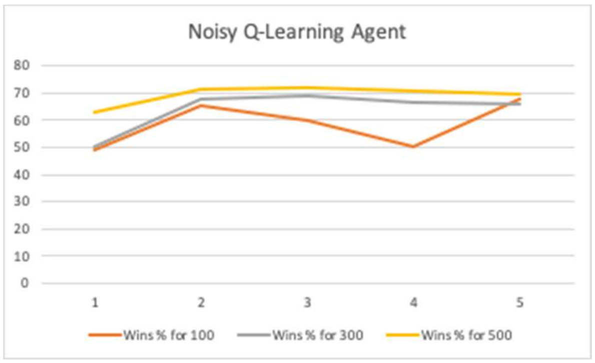
D. *Deterministic Q-Learning technique*

The implementation of this part of the program utilizes a completely different learning and value calculating strategy as explained in the previous section. The graph displays a performance that is expected and has been previously observed. It shows a respectable and quick in the win rate of the agent in the first environment. This implementation was the more complex of the two. The implementation of Q-Learning is not as straight forward as direct evaluation as it requires an iterative update of Q values for a huge state action pair set.



E. *Q-Learning with noise action model*

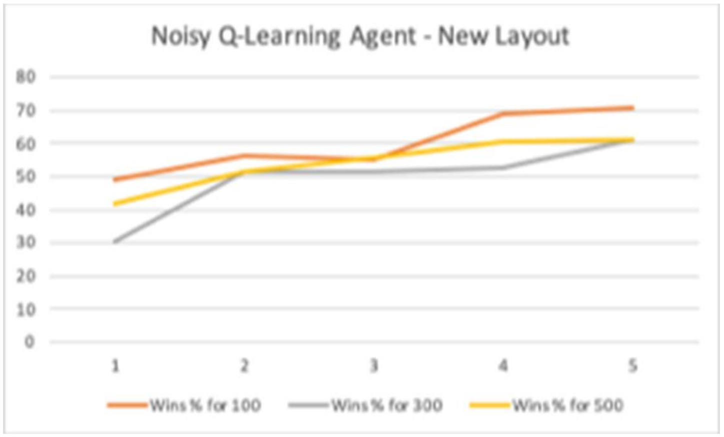
The difference in the performance of the two models is quite visible as the agent has a win rate of 0.62 from the beginning and converges to 0.72. The agent learns very quickly and has a sudden jump in win rate due to very fast learning and a decision-making process that is more complicated than before and performs with higher accuracy in theory. It can be inferred from this data that using the Q-learning model would be better if the convergence has to reached faster. A comparative analysis of the two models displays a near equal convergence point but a measurably faster convergence in Q-learning.



A comparison under identical space constraints shows that Q-Learning is the preferred model. One of the major contributors of this superiority is the absence of algorithm randomness as discussed in the previous section. The algorithm does not completely ignore the action choice anymore and adheres to the set to state action pairs provided by the dictionary.

F. *Q-Learning noise action model under new environment*

The graph of the model when implemented under the new environment is slightly different than that of the previous environment although the convergence is just as fast. One of the prime reasons for this being the fast convergence nature and independence from a pretraining knowledge. The model implemented here does not have prior training and shows different behavior under different environment settings.



IV. Conclusion and discussion

The hybrid logic agent that we initially worked with performs poorly under stochasticity as it has a fixed model of working and does not perform the learning aspect which is the key concept under reinforcement learning. The Q-Learning algorithm performs exceedingly better than Monte-Carlo (direct evaluation) algorithm. The convergence in win rate happens much faster for Q-Learning. Another important learning from this implementation is the importance of a pretrained model in both instances. Using a model which has been already trained using Monte-Carlo algorithm even under noise performs visibly better as the unfolding begins. This advantage cannot be seen in the Q-Learning implementation as the success of the algorithm is largely a function of the position of the environment variables like the positive reward bearing Gold. Thus, for implementing over various versions of an environment, Monte-Carlo reinforcement learning technique is the preferred one and for new environment maps, Q-Learning is the preferred one.

V. References

[1] Stuart Russell & Peter Norvig, "Artificial Intelligence: A modern approach" 3rd edition.

[2] Andrew Barto and Richard S. Sutton," Reinforcement Learning: Introduction"

[3] "Exploration:Probailistic Robotics" Sebastian Thrun et Al, Pg. 569-604

[4] "A Comparison of Exploration/Exploitation Techniques for a Q-Learning Agent in the Wumpus World" A. Friesen.

[5] "Foundations of Machine Learning II Project: Bandits and Wumpus", Guillaume Charpiat, Corentin Tallec & Ga´etan Marceau Caron, 2017

[6] *Daniel Bryce. 2011. "Wumpus World in introductory artificial intelligence". J. Comput. Sci. Coll. 27, 2 (December 2011), 58–65.*

[7] "Reinforcement Learning algorithms — an intuitive overview", *Robert Moni,SmartLabAI, 2019*

VI. Team effectiveness

Each member of the team was active since the beginning of the project and performed their assigned readings. Although the readings were supplementary and the contribution towards the project was minimal, each member completed their task on time. In order to not create any biases, each topic was worked on by all three members of the team with various ideas being contributed, being proven wrong and worked on together to find the best approach. The report too has been written as a team effort. The report has been reviewed, altered and finalized over multiple iterations. Every member of the team holds equal right over any appraisal or critique given towards the submission for this team project. The expertise of Nicholas Powell did prove to be a vital factor in giving the project the right direction. All the members performed to the best of their abilities.