Najla Ahmad
June 22, 2018

Capstone:
Reinforcement Learning in a Dynamic Multi-Agent Environment

1. Definition

1.1 Project Overview

In a multi-agent system, agents work together to complete tasks. An example of agents working together is a Sensor Web, such as one the one developed for NASA[1], where agents work in coalitions to determine if a weather phenomenon is occurring or not. Another example of agents working together is given by Shoham and Leyton-Brown (2009)[2]. They researched a sensor network, in which each unit has sensing capabilities and limited processing power. Individually, each sensor builds a local view of what is occurring around them. Working together, these agents can build a global view of the environment.

For my PhD thesis, I worked on intent recognition in multi-agent systems[3]. Idle agents that were otherwise not doing anything would observe other agents in the system, try to recognize what they were trying to achieve, and then attempt to assist them with their tasks. These agents chose between set plans, or a list of instructions of how to behave. During many of my Udacity lectures, I would think about how the agents I created would be much more intelligent if they used machine learning concepts.

A classic way to see how well agents are working together is to have a team of agents attempt to herd other "animal" agents. This can be seen in the Microsoft Malmo Collaborative AI Challenge[4] and in several years of the annual Multi-Agent Programming Contest[5]. The "animal" agents follow set behavior patterns while the other agents attempt to herd them into corrals. A score is calculated based on the number of animals herded and the time elapsed. We will refer to the animal agents as cow agents, which is the same terminology used by the Multi-Agent Programming Contest.

1.2 Problem Statement

The animal herding domain is a well-known problem in multi-agent systems. It is a good test of the agents' abilities because, as the Multi Agent Programming Contest states: "The contest consisted of applying (or developing from scratch) a multi-agent system to solve a cooperative task in a highly dynamic environment." Animal herding simulations test agents' abilities to simultaneously work cooperatively with their teammates and competitively with the opposing team.

In the cow herding contest, two teams of agents face off in a grid-like world. Cow agents move around in a swarm-like way while each of the two teams try to corral the most cows on their side.

---

[1] Tsatsoulis, C., N. Ahmad, E. Komp, and C. Redford (2008). "Using a Contract Net to Dynamically Configure Sensor Webs," IEEE Aerospace Conference, Big Sky, MT, 1-6.

[2] Shoham Y. and K. Leyton-Brown (2009). "Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations." Cambridge University Press, Cambridge, UK.

[3] Intent recognition in Multi-agent Systems: Collective Box Pushing and Cow Herding. N Ahmad (2013)

[4] "The Malmo Collaborative AI Challenge" https://www.microsoft.com/en-us/research/academic-program/collaborative-ai-challenge/

[5] "Multi Agent Programming Contest" https://multiagentcontest.org/2010/

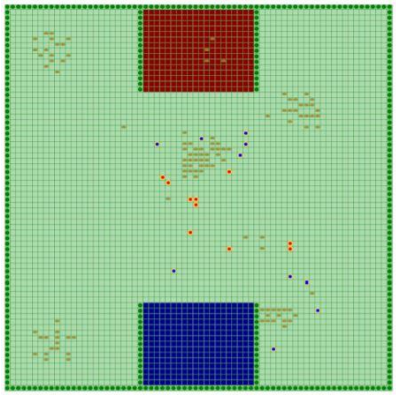

Figure 1 Example image from 2008 cow herding scenario document (Dix et al., 2008)[6]

The experiment in this capstone evaluates machine learning in a dynamic multi-agent system. While this project did not allow time for the competitive aspect of the cow herding scenario, it was easy to compare agent reasoning capabilities by looking at the scores and completion the agents are able to achieve.

I implemented three types of agent teams. These are: agents trained using Monte Carlo learning, agents following a random policy to represent poor reasoning, and agents that follow a strategy based on aspects of the winning teams from the multi-agent contest[7].

By adding machine learning agents to the cow herding scenario, I was able to examine reinforcement learning in this dynamic, cooperative, and competitive environment.

1.3 Metrics

In the first year that the cow herding scenario was used, the score was calculated by the number of cows that were brought into the corral. Once a cow entered the corral, it was not able to leave. After the first year, the score was calculated as the average number of cows in the corral instead. Cows could repeatedly escape from the corral and be herded again.

In this new version of the scenario, cows could repeatedly escape from the corral and be herded again until the simulation reached the maximum time limit. To calculate the score for this version, at every timestep, the cows in a team's corral are counted and added to the sum for all the previous timesteps. The final sum is then divided by the number of steps, which yields the final score for that team.

For both my PhD work, I used the first version of the score calculation and for my capstone, I used the second version of the score calculation to compare agent groups. However, I also chose to explore training the reinforcement learning agents using both of these scenarios to see if they were able to learn the task. I also had a maximum number of timesteps so that the simulations would not run too long if an agent group was not able to herd the cows in a reasonable amount of time.

Statistical analysis, including one and two tailed t-tests, were done to determine whether the team scores are significantly different with an alpha level of 0.05.

---

[6] Dix, J., M. Köster, and F. Schlesinger. (2013). "Multi-Agent Programming Contest." Multi Agent Programming Contest. Retrieved July 30, 2013, from http://www.multiagentcontest.org.

[7] Heßler, A., Hirsch, B., and T. Küster. (2010). "Herding cows with JIAC V." Annals of Mathematics and Artificial Intelligence, 59(3-4): 335-349.

2. Experimental Analysis

2.1 Data

For this project, I built a version of the cow herding simulation in Python. The input to the agent was the environment state, which is detailed in a following section. The reward for each state is based on the number of cows in the goal and is also detailed in a following section. The score was calculated the same way as it is calculated in the Multi-Agent Programming Contest as noted above. The agent used this state and reward to perform reinforcement learning. The project was a scaled down from the original cow herding simulation due to time and processing constraints.

The output that I collected consisted of the scores and completion time of three groups of agents: randomly acting agents, agents following a set plan, and reinforcement learning agents that were pre-trained using a Monte Carlo algorithm.

2.2 Experiment

In order to study reinforcement learning in a dynamic multi-agent environment, I created a cow herding environment in Python. It is based off the environment detailed in the Multi Agent Programming contest and my PhD work. It is a scaled down version in terms of grid size and number of agents per team. This is because of time constraints for the capstone project and processing power since machine learning agents will require more processing power than the agents I worked with previously. However, this is an expansion and continuation of my previous work in terms of the intelligence of the agents. In my previous work, agents followed preset matching and herding algorithms. In this experiment, agents are able to learn from their environment. To accomplish this, I implemented a reinforcement learning team which learned a policy using ε-Greedy Monte Carlo learning.

The cow herding simulation was run with the random agents, then the plan recognition agents, and finally the reinforcement learning agents. Once trained, the reinforcement learning agents did not perform learning anymore. Instead they followed an ε-Greedy version of their learned policy.

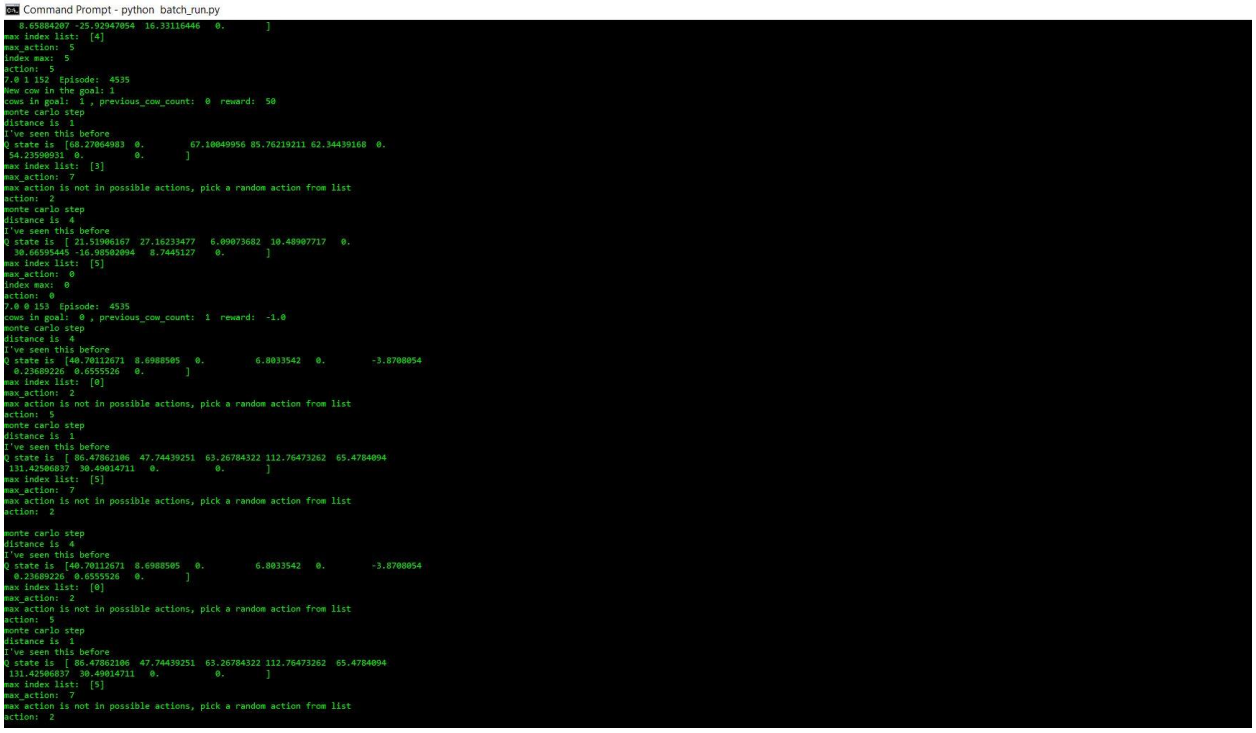

Figure 2 Monte Carlo Agents During the Learning Process

The scores and completion times that the agents were able to achieve were calculated and saved over multiple episodes for analysis.

2.3 Algorithms and Techniques

The reinforcement learning technique I implemented was ε-Greedy Monte Carlo learning. I selected this due to the fact that there were many distinct episodes that could be used for learning. I refer to the agents which used this learning technique as Monte Carlo Agents. This algorithm fits the problem nicely since there is an end to each episode and many episodes can be run. In this case, an episode ended once the maximum timesteps had elapsed.

Before training began, an empty Q table was created and loaded during the first episode. At each subsequent episode, Monte Carlo agents loaded the Q table from the previous episode. At each step, the Monte Carlo agents took the current state and selected an action in an ε-Greedy way. At the end of the timestep, the Monte Carlo agent was given the reward from the environment and the Q table was updated. After all of the episodes, the final Q table was saved to be used later.

Reinforcement learning was done in two phases. First was the training phase where the agents built the Q table over many episodes. Agent scores were not saved at this time. In the next phase, agents read the saved Q table on initialization. Once trained, the agents used an ε-Greedy method to follow the pre-trained Q table. These pre-trained agents were used to collect data and compare the Monte Carlo agents to the other types of agents. The Q table was not updated in this data collection phase.

2.4 Benchmark

The scores of the reinforcement learning agents were compared to two other agent groups. The first group being a team of randomly acting agents, referred to as Random Agents. This was the example of agents with poor reasoning capabilities. At each step, these agents move one space in a randomly selected direction. This continues until all cows are herded or until the maximum number of timesteps is reached.

The second group, referred to as Plan Agents, were agents modeled after the JIAC V team by Heβler et al. (2010) whose work is referenced in a previous section. These agents won the cow herding competition in 2009. These agents follow a set plan. They move randomly around until they locate a cow agent. Then they attempt to herd this cow agent into the team corral by first moving so that the cow is between the agent and the goal, then moving towards the cow. This part of the plan repeats until the cow enters the corral. The agent then looks for another cow and repeats the process. This continues until all cows are herded or until the maximum number of timesteps is reached. These agents are used as the benchmark and are an example of high functioning agents.

To compare the learning agents with the other groups, each agent group (random, plan, and Monte Carlo) were run in the environment to record their scores and completion times. There were two agents per team across all agent types. The scores were then compared to see if one team performed statistically better than the others.

3. Methodology

3.1 Data Preprocessing

Many iterations of the environment were run before settling on the final parameters. I tried various options for different parameters including grid sizes, corral sizes, number of agents per team, number of cows, ε, state representations, episodes for training, and episodes for testing. It took most of my time working with these values until I was able to settle on a model which I felt like gave enough balance between time and function. Some of these parameter choices are detailed in a later section.

3.2 Implementation

The cow herding simulation will be built in Python using Mesa[8]. According to the description, "Mesa is an Apache2 licensed agent-based modeling (or ABM) framework in Python. It allows users to quickly create agent-based models using built-in core components (such as spatial grids and agent schedulers) or customized implementations; visualize them using a browser-based interface; and analyze their results using Python's data analysis tools. Its goal is to be the Python 3-based alternative to NetLogo, Repast, or MASON." This is the Python alternative to the Java based Repast which was used for my PhD.

This capstone project is a continuation of my PhD thesis work. A screenshot the environment from my PhD thesis can be seen in the figure below.

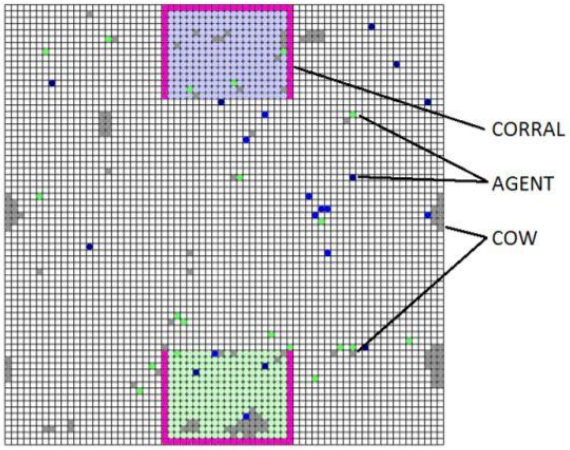

Figure 3 Cow Herding Simulation (Ahmad, 2013)

The agent environment above consists of a grid world. Some grid areas are designated to be corrals. Two agents cannot occupy the same grid space at the same time.

Reinforcement learning takes much more time and processing power than my previous agents. In order to finish the project within the allotted time, several adjustments were made. The environment used in this capstone is smaller than the original environment guidelines. Team size was minimized with teams of two agents per team. Instead of having two teams running simultaneously, there was only one team and one corral at a time. Comparison was done on scores achieved by each team and time to completion. The team scores and times were statistically compared with each other.

The grid world, like the grid I used for my PhD, is a wrap around grid (moving off of the edge moves an agent so that it enters the other side) where each space can only be occupied by one agent at a time.

3.2.1 Cow Agents

The cow agents behave the same way as the cow agents in the Multi-Agent Programming contest. The algorithm is detailed in the figures below. This algorithm leads to a weak swarming behavior.

[8] "Project Mesa" https://github.com/projectmesa/mesa

Algorithm 1 Cow movement algorithm.

Require: a cow represented by its position vector $c \in \mathbb{N} \times \mathbb{N}$

1: let N be the set of the 9 cells adjacent to c, including c;
2: remove from N all those cells that are not reachable;
3: calculate the weights of all cells $n \in N$;
4: determine the set $M \subseteq N$, where the weight for each $m \in M$ is maximal;
5: randomly pick a cell $m \in M$;
6: move the cow to m;

Figure 4 Cow Algorithm 1 located in 2010 MAPC scenario document (Dix et al., 2010).

Algorithm 2 Calculate the weight of a given cell.

Require: a cell represented by its position vector $n \in \mathbb{N} \times \mathbb{N}$, and a cow-visibility range $r \in \mathbb{N}$

1: determine the set C of all cells that are a in the rectangle $[n_x - r, n_y - r + n_x + r, n_y + r]$ and that are on the map;
2: set ret to 0;
3: **for all** $c \in C$ **do**
4: calculate d the distance between c and n;
5: get the weight w of c in respect to the cell content;
6: add w/d to ret;
7: **end for**
8: **return** ret

Figure 5 Cow Algorithm 2 located in 2010 MAPC scenario document (Dix et al., 2010).

3.2.2 Random Agents

The team of randomly acting agents were created to see what scores a "bad" or "random" team would achieve. At each timestep, these agents scanned the adjacent cells and checked which ones were currently unoccupied. Then the agents randomly selected one of these grid spaces to move to.

3.2.3 Plan Agents

The plan agents followed a set of pre-given instructions, known in multi-agent systems research as a plan[9]. The plan followed by the plan agents is based on one of the winning teams of the Multi-Agent Programming Contest and the plan agents in my PhD research. At the beginning of the plan, the plan agents look within their given vision range and locate a cow to follow. Then they attempt to herd them into the corral by moving so that the cow is between them and the corral. Once the cow is herded, the agent then looks for another cow to follow.

Plan agents access movement methods in a file created for this experiment called movement_control.py. This file contains methods that allows the agents to move towards a location, compute distances, and more.

[9] Goldman, RP., F. Kabanza, and P. Bellefeuille (2010). "Plan Libraries for Plan Recognition: Do We Really Know What They Model?" In Plan, Activity, and Intent Recognition (PAIR) 2010, AAAI Workshop Proceedings, AAAI, Atlanta, Georgia.

3.2.4 Monte Carlo Agents

Monte Carlo agents were used to train a Q table over many episodes. This Q table was then saved and loaded by agents that no longer updated the Q table. Instead, they selected actions using the pre-trained Q table as a policy in an ε-Greedy manner. The value of ε was set through experimentation. The final value of ε used in the training part of this experiment was 1.0/((episode_number/800) + 1) where episode_number refers to the number of the current episode. The final value of ε used in the testing portion of this experiment was .00001.

Helper methods were defined in a separate file called rl_methods.py and could be accessed by all reinforcement learning agents. Methods in this file included functions to encode the state, select an ε-Greedy action based on a Q table, and others. Details of how the state was encoded are in a future section.

The maximum size of the action space for an agent is eight since an agent can move one space in any direction if that space is not already occupied. All actions can not be performed in every state. In each state, the Monte Carlo agents make a list of possible actions, which are the actions that lead to the non-occupied adjacent locations and select from them in an ε-Greedy manner.

The state was stored as tuples with the different agent types stored in a numbered system (for example 1 is Wall Agent, 2 is Random Agent, etc.) and empty grid spaces stored as 0.

3.2.5 Wall Agents

In order to define the corral boundaries, non-mobile wall agents were created. These can be seen on the visualization as squares. All mobile agents are represented by circles. These agents are necessary to block of the spaces that surround the corral so that other agents did not enter these locations. At each timestep, the wall agent's step method did nothing.

3.2.6 Model

The model, as defined by the Mesa module, controls the flow of time and agents. All agents are created in the model as defined for that episode. The model places the wall agents at the correct locations and randomly places all other agents. At each timestep, the model calls each agent's step method in a random order. At the end of the timestep after all agents have taken their step, the model updates the reward function of all learning agents. The current episode score is stored in the model. The model also is aware of the current episode number which is then used to calculate ε for reinforcement learning.

3.4.7 Final Experiment Configuration

In the final configuration for the experiment, a team of 2 agents attempted to herd four cow agents. They were spawned randomly in an environment consisting of a 10x10 wrap around grid. A corral was located on the grid with an interior size of 2x4.

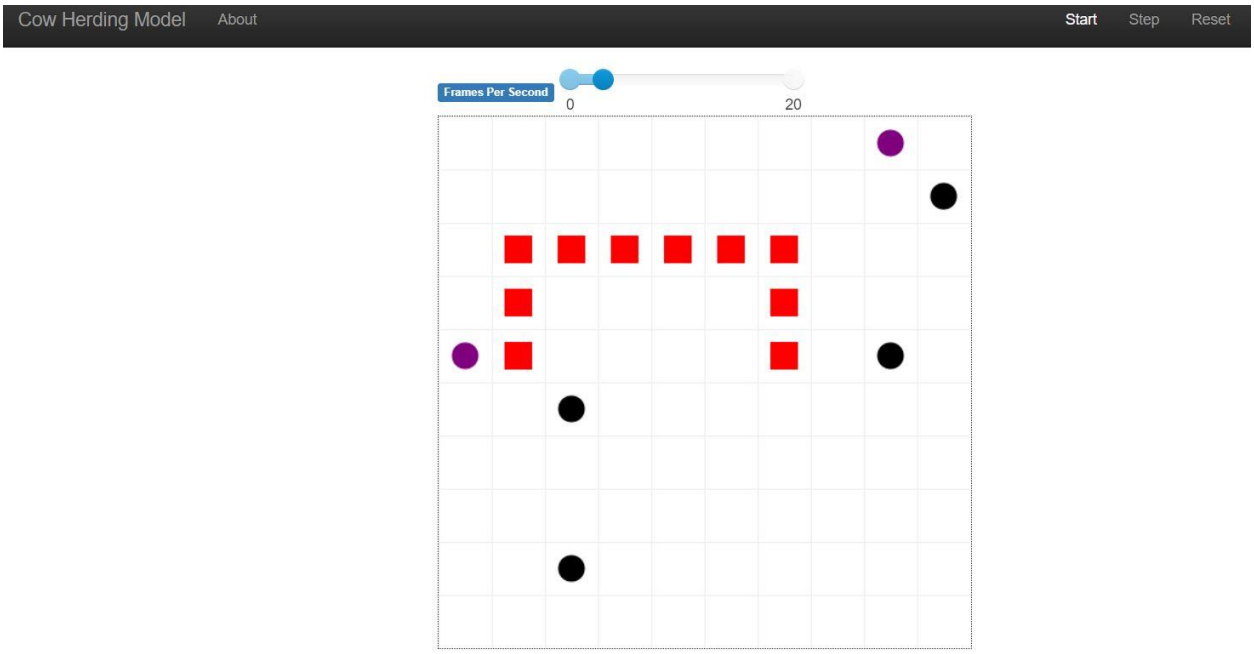

Figure 6 Environment running. Black circles: Cow Agents. Purple circles: Plan Agents. Red Squares: Wall Agents.

The environment runs in three modes. The first mode is for visualization and is used to observe agent behavior. This can be seen in the figure above and is not used for reinforcement learning or data collection. The other two modes, data collection and Q table learning, were not run with a visual component since they run too fast for human eyes to observe. Instead, they gave updates about the current episode and score in the command prompt.

Each episode lasted until all the cows were herded or a maximum time of 100 timesteps was reached. At the end of each episode, the final score and time were recorded. Each agent team was run for 3000 episodes. Statistical analysis was performed on the resulting data.

3.3 Refinement

I went through many iterations before finding a satisfactory model on which to perform analysis. As with many experiments, there was a tradeoff between time and function. The following sections detail improvements made during the implementation process.

3.3.1 Vision Range

As in the Multi-Agent Programming Contest, agents do not have a view of the whole environment. Instead, they have a range around their current location that they can see, called the vision range. I tried various combinations of having a vision range between 1 and 4. A vision range of radius 1 means the center space where the agent is currently located and 1 space in every direction, resulting in a total of 9 grid spaces. A vision range of radius 4 means 4 spaces in every direction, resulting in a state which includes 9 x 9 or 81 grid spaces. A vision range of 2, which resulted in a state space of 25 grid spaces, is what worked best for this project given the time constraints. The bigger the vision range, the bigger the state space. All plan agents and reinforcement learning agents had this same vision range.

3.3.2 State

Over the implementation process, I spent time refining how the state was stored. At first, I tried making the entire grid into a tuple of tuples and stored that as the state. However, the state space was so large that after almost 1000 episodes, an agent would only revisit a state it had seen before once or twice. I realized that I needed to limit the state to the vision range of the agents, which is also something

that is done in the Multi Agent Programming Contest. The larger the state size, the more episodes and time needed to train the agent, so there was an important tradeoff between data stored and time. Still, this strategy did not give the agent enough information about the state for learning. I then changed the state given to the agent to a tuple representing the grid within vision range and the distance to the goal rounded to the nearest integer. In other words, the state looked like this: ((grid in range tuple), distance). This worked better, but I wanted to refine the state representation even further. I then replaced the distance with the current location. The state then looked like ((grid in range tuple), (x,y)) and this was the final state used for the experiment.

3.3.3 Cow Behavior

At the beginning of implementing the experiment, the cow agents were able to escape the goal repeatedly until the maximum number of timesteps was reached. However, there were two issues to this approach. First, training and collecting data took a long time because every episode reached the maximum number of timesteps. Second, since the scenario was much more complex, the reinforcement learning agents needed much longer for training. I finally changed the cow behavior so that it aligned with the cow behavior in my PhD. This meant that once a cow entered the corral, it could not escape. If all cows were herded before the maximum number of timesteps had elapsed, the episode ended at that point.

3.3.4 Rewards

I tried many iterations of the rewards given to the Monte Carlo agents to see which variations led to the best behavior in the most efficient manner. The first version of the reward system I tried were giving the reinforcement learning agents the current score as stored in the model. Next, I tried the number of cows currently in the goal. However, these reward systems were not training the agents effectively. It may have worked well if I had much more time and processing power, but I needed a reward function that worked well within my limitations. The final reward function that I settled on was a positive reward if there were more cows in the goal than the previous timestep and -1 otherwise. I tried +100 and +50 as rewards for the positive reward and +50 produced better training results.

3.3.5 Training episodes

In an ideal world, I would have trained the Monte Carlo agents as long as they needed to converge on a near optimal policy. However, this was obviously not possible given the time constraints. This was especially true since the agents needed to be retrained every time there was a change to other variables, such as the state representation. The final number of episodes used to train the Monte Carlo agents was 25,000 episodes with 500 timesteps per episode. Since there were two agents per team, this was 25,000 x 500 x 2, or 25 million Q table updates.

3.3.6 Maximum Time per Episode

Given enough time, all cows would eventually end up in the goal state just by random chance. Since I wanted to test an efficient herding of cows, I set a maximum time per episode in the testing phase. I experimented with various times until settling on a value I felt gave agents enough time to work while minimizing the number of times cows wandered into the goals on their own. This value was 100 timesteps.

4. Results

4.1 Model Evaluation and Justification

For Random Agents, Plan Agents, and Monte Carlo Agents, teams of two agents were run over 3000 episodes to collect data about scores achieved and task completion time. This data was analyzed using t-tests with an α level of .05. A summary of findings can be found in the tables below.

Agent Group 1	Agent Group 2	Statistically Significant Difference	Higher Mean Score
Random	Plan	Y	Random
Monte Carlo	Random	Y	Monte Carlo
Monte Carlo	Plan	Y	Monte Carlo

Table 1 Summary of Score Comparisons Across All Experiments

Agent Group 1	Agent Group 2	Statistically Significant Difference	Lower Mean Time
Random	Plan	Y	Random
Monte Carlo	Random	Y	Monte Carlo
Monte Carlo	Plan	Y	Monte Carlo

Table 2 Summary of Time Comparisons Across All Experiments

This experiment found that the Monte Carlo agent team performed better than both the pre-set plan team and the randomly acting team. By better, I mean able to achieve a higher score and lower completion time. A surprising finding of this study is that the randomly acting team performed better than the pre-set plan team. My hypothesis about this is that the pre-set plan agents need more time to complete their task than the time allotted for the experiment. In fact, following their plan lead to an initial period with fewer cows in the goal, which let the random agents take the lead. This also shows the reinforcement agents in a good light because they were able to complete the task quickly. A preliminary exploration of the pre-set plan agents shows this to be true. I temporarily set the maximum timesteps per episode to 500 and the preset plan agents performed better. However, for experimental analysis I kept the maximum timesteps to 100 per episode in order to test efficiency.

The following tables show the details of the statistical analysis that was done.

t-Test: Two-Sample Assuming Equal Variances

	RandomScore	*PlanScore*	*PlanScore*	*MCScore*	*RandomScore*	*MCScore*
Mean	2.006461697	1.918109467	1.918109	2.148966	2.006462	2.148966
Variance	0.995744226	1.288367164	1.288367	1.006818	0.995744	1.006818
Observations	3000	3000	3000	3000	3000	3000
Pooled Variance	1.142055695		1.147593		1.001281	
Hypothesized Mean Difference	0		0		0	
df	5998		5998		5998	
t Stat	3.201986541		-8.34632		-5.51565	
P(T<=t) one-tail	0.00068598		4.33E-17		1.81E-08	
t Critical one-tail	1.645107713		1.645108		1.645108	
P(T<=t) two-tail	0.00137196		8.66E-17		3.62E-08	
t Critical two-tail	1.960359573		1.96036		1.96036	

Table 3 Details of Score Statistical Analysis

t-Test: Two-Sample Assuming Equal Variances

	RandomTime	*PlanTime*	*RandomTime*	*MCTime*	*PlanTime*	*MCTime*
Mean	74.35933333	75.93266667	74.35933	71.2643	75.93267	71.2643
Variance	477.2486291	495.6966985	477.2486	507.5656	495.6967	507.5656
Observations	3000	3000	3000	3000	3000	3000
Pooled Variance	486.4726638		492.4071		501.6312	
Hypothesized Mean Difference	0		0		0	
df	5998		5998		5998	
t Stat	-2.762721789		-5.40187		-8.072633	
P(T<=t) one-tail	0.002874814		3.42E-08		4.12E-16	
t Critical one-tail	1.645107713		1.645108		1.645108	
P(T<=t) two-tail	0.005749629		6.85E-08		8.25E-16	
t Critical two-tail	1.960359573		1.96036		1.96036	

Table 4 Details of Time Statistical Analysis

4.2 Robustness

In order to achieve good results in the allotted time, I had to make the encoding of the state space more specific than I otherwise would have liked. In an earlier iteration, the state space was saved as a tuple of the grid in the vision range of the agent and the distance to the goal. While learning was taking place, it was not a rate that was conducive to finishing this project on time. Instead, I changed it to a tuple of the grid in the vision range of the agent and the current location. This allowed learning to be faster, but it did make the agent less flexible. In order to make the model more robust, I plan to return to my first state representation and run these experiments when I have more time. This will be much more generalizable as it will be able to use the same Q table no matter where on the grid the goal is located.

Aside from that fact, I am satisfied with the final parameters of the experiment. I went through a great deal of trial, error, and experimentation to reach the final values.

5. Conclusion

5.1 Free-Form Visualization

As can be seen in the charts below, the team of Monte Carlo agents achieved better results than the other two teams in terms of both score and completion time. These differences were shown to be statistically significant in the previous section.

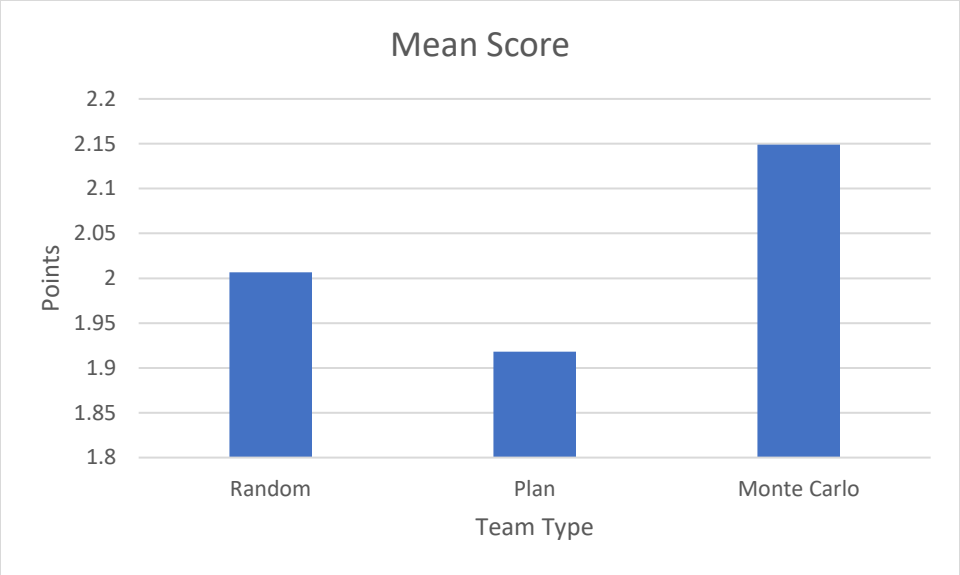

Table 5 Visualization of Mean Scores by Team

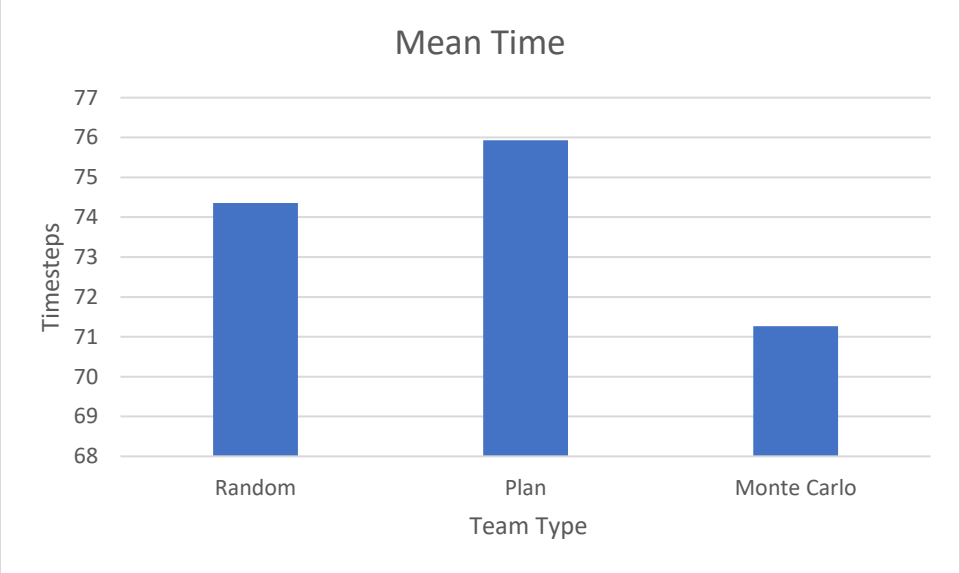

Table 6 Visualization of Mean Completion Time by Team

5.2 Reflection

In this project, I created a multi-agent environment in Python where teams of two agents herd cow agents with swarm like behavior into a corral. Teams were scored based on the amount of time it took to herd cows and the number of cows herded. The first team consisted of randomly acting agents, the second team consisted of agents following a pre-set plan, and the third team consisted of agents that selected actions in an ε-Greedy manner from a Q table that was trained using the Monte Carlo algorithm. Each team completed 3000 episodes consisting of a maximum of 100 timesteps. The score and completion time data were collected after every episode. After all episodes were completed, the data was analyzed using statistical methods.

5.2.1 Time and Changes

When I first started this project, I did not anticipate the number of changes and adjustments I would have to do to get the reinforcement learning agents to learn efficiently. It was easy to code the initial Monte Carlo process, but beyond that there had to be many changes. For example, I had initially planned to only use the raw score as the reward. However, it turned out that this was not enough for the agents to learn the task properly. I then tried a penalty whenever a cow escaped from the goal so that the agents could learn that the cow agents should not escape. This also did not produce the behavior in an efficient way. After many iterations, I settled on the final reward function of +50 if a new cow enters a goal and -1 otherwise.

Other variables that I changed which affected learning were the number of timesteps per episode and the number of episodes. Both were maximized as much as possible without causing a computation time that was too long. These values were determined using trial and error.

I also didn't anticipate the sheer amount of time it would take to run the agents. From the nanodegree, I knew machine learning takes time. However, I did not anticipate the orders of magnitude it would increase over my PhD thesis experiments.

5.2.2 Success

There were two points where I felt that this experiment was the most successful. At first, I was feeling dejected because I would train the Monte Carlo agents in the training mode and then watch them in the visualization mode and they did not seem to be working efficiently. I kept changing the reward function and training parameters until suddenly it began to work. I would watch the agents working to herd the cows and it was very exciting.

The next part where I felt elated was when I was performing the statistical analysis and saw with mathematical proof that the reinforcement learning agents were able to perform better than not only the random agents, but also the plan recognition agents.

5.2.3 Expectations

This experiment shows to me that reinforcement learning can be successfully used in a dynamic multi-agent environment. With more training time and processing power, I envision agents that vastly outperform agents with pre-set plans. Agents with pre-set plans can not adapt to new situations, while reinforcement learning agents are able to learn and adapt based on their environments.

5.3 Improvement

Machine learning takes time and resources. This capstone was able to analyze reinforcement learning in a dynamic environment, but there are many improvements that I plan to make and implement in the future.

First, I will increase the scope of the scenario used in this capstone. I will test various grid sizes, increasing the dimensions until I reach the full size used for my PhD. Next, I plan to test various numbers of agents per team, increasing the number per team until it reaches the number in the Multi-Agent Programming Contest. Finally, I will increase the number of cows until the number is the same as the Multi-Agent Programming Contest. These increases will greatly increase the state size and processing time needed to run the episodes.

The next logical step is to modify the environment so that there are two teams and two corrals at a time. This will greatly increase complexity and I will be able to study how the teams perform in a competitive setting. It will be interesting to see if the agents learn offensive, defensive, or a mixture of strategies.

Another change I would like to make is to store the state space in a more generalized way so that the same Q table can be used on more than one environmental setup.

Another expansion of this project is to incorporate new teams with new and different learning techniques and see how these teams fare in this environment. In my proposal I had thought I would also have time to implement TD learning, but I did not. I plan to implement that in the future.