

BACHELOR THESIS
ARTIFICIAL INTELLIGENCE

Radboud University



Prospect theory in exploration

Author:

Kamiel Kunst
S1037679

First supervisor:

Dr. S. Thill
Artificial Intelligence
Serge.thill@donders.ru.nl

Second reader:

Dr. Ing. L.P.J. Selen
Artificial Intelligence
Luc.selen@donders.ru.nl



27 January 2023

Abstract

Exploration problems are situations where an agent should explore an environment without any prior knowledge. The standard approach to this problem is by using reinforcement learning. This is a method where the agent learns a sequence of actions that lead to the best cumulative reward. Problems with this approach are the learning inefficiency and the inapplicability to real-world problems. A possible solution is to use cheaper and less complex approaches. One possibility is expected utility theory. In this theory the agent always chooses the option with the highest expected utility. This utility is calculated by taking the weighted sum of all possible outcomes. In 1979 Kahneman and Tversky offered critique on this theory as a model of human behaviour when risks are involved. They presented a better alternative called prospect theory, which encapsulates the risk averse and risk seeking behaviour of humans. These three theories are compared in the setting of an exploration problem. This leads to the research question: How does prospect theory compare to expected utility theory and reinforcement learning in exploring an environment? To answer this question a general framework was created. This framework contained many necessary functions to implement the before mentioned theories and algorithms. The final implementations were tested in several environments with increasing complexity. Ultimately, it seems that reinforcement learning is superior but the results are inconclusive due to several limitations. Unfortunately this means that the original problem with reinforcement learning has not been solved and it is uncertain if either prospect theory or expected utility can be used as an alternative.

Contents

1. Introduction.....	4
2. Background	6
2.1 <i>Prospect theory</i>	6
2.2 <i>Expected utility</i>	9
2.3 <i>Reinforcement learning</i>	9
3. Methods.....	11
3.1 <i>General framework</i>	11
3.2 <i>Application of theories</i>	15
4. Results	17
4.1 <i>Environment 1</i>	17
4.2 <i>Environment 2</i>	18
4.3 <i>Environment 3</i>	19
5. Discussion.....	20
5.1 <i>Analysis of results</i>	20
5.2 <i>Limitations</i>	20
5.3 <i>Further research</i>	21
5.4 <i>Conclusion</i>	21
6. Bibliography	22

1. Introduction

Uncertainty is all around us, it is something we must deal with every single day. It ranges from what kind of weather can we expect tomorrow? Or what is the probability that this coin flip comes up heads? According to Li et al. (2013), uncertainty can be divided in two main categories: Aleatory uncertainty and epistemic uncertainty (Li et al., 2013). Uncertainty is epistemic if there is a possibility to reduce the uncertainty. An example of epistemic uncertainty is to estimate the distance between two points, the uncertainty in this estimation can be reduced if you know the distance between two points that are close to the original points. Aleatoric uncertainty is when there is no possibility to reduce the uncertainty (Kiureghian & Ditlevsen, 2009). An example of this can be the probability that a fair coin comes up heads. It does not matter how many experiments you do beforehand it is impossible to know whether the next flip comes up heads or tails.

Uncertainty is also a key element of robotics. Robotics is the science of observing and operating in a real-world setting by using computer-controlled mechanical devices. Uncertainty in robotics can arise from numerous factors. Examples of these factors are:

1. Environment. Unstructured and dynamic environments are the cause of a lot of unpredictability. Especially in real-world scenarios, the environment is a huge factor to unpredictability and uncertainty.
2. Sensors. They are inherently limited in what they can perceive. This is due to two factors. First, their range and resolution are always limited by physical laws. Second, noise is unavoidable and can cause disturbances in different measurements.
3. Robots. This cause of this factor is the motors that drive robots. They can produce inaccuracies due to noise and extensive usage.
4. Models. Models are abstractions of the real world. Therefore, they only partially model the actions and their consequences of the robots and the environment. However, this factor is mostly ignored when developing new systems.
5. Computation. This factor is especially prevalent in the real-world. Since many robots act in real-time, the number of computations is limited.

All these factors contribute to the complexity of robotics. Traditionally, a substantial portion of these uncertainties has been ignored. Often, research focuses on one or two of these factors since the main goal is to create robust software that can handle various challenges (Thrun & Burgard, 2002).

In this thesis the main focus is on the first point: the environment. An agent is placed in an environment without any prior knowledge of this setting. The goal of this agent is to explore this environment as efficiently as possible. This efficiency can be measured using multiple metrics, for example, distance driven, battery usage, and time spent. In this situation we assume that the agent has a good vision of the environment. Thus, a coordinate system is used to represent its own location and the location of their target. However, since the vision is not

perfect, there is noise added to the target's location relative to the distance of the agent. This setting is also known as the exploration problem in robotics. A solution to this problem can be applied in many different fields and real-world applications, such as, a search and rescue mission, space exploring, or something as trivial as cleaning the house or mowing the lawn (Stormont & Allan, 2009).

There are many different approaches to try and solve this problem, but every technique has its own advantages and disadvantages. The most popular approach to this problem is Reinforcement Learning (RL). In RL an agent is incentivized to learn the best sequence of actions. It is the most popular because robotic exploration is addressing some typical RL problems such as exploration-exploitation, curse of dimensionality, and slow learning convergence. However, RL also has some serious drawbacks (Garaffa et al., 2021). One of the main disadvantages is the inefficiency of RL. It requires a lot of time and resources to be applied in the real-world. A solution to this can be to use a different and simpler method.

One option is to use expected utility theory. This theory is simpler and requires less computational power. In expected utility theory the agent acts perfectly rational. It is not influenced by factors such as risk, which does affect the behaviour of human decision making. It always chooses the option with the highest expected utility. This utility is calculated by taking the weighted sum of all possible outcomes. Thus, each step the agent chooses the option that has the highest return (Fishburn, 1968).

Another option is to use prospect theory. Prospect theory was developed in 1979 by Kahneman and Tversky as an improved model of human behaviour when dealing with risks. They created this theory as a critique on expected utility theory as a model of human behaviour (Kahneman & Tversky, 1979). Even though they created this theory several decades ago, it is still shown to be relevant. Not only that, currently it is also one of the better models of human decision making with risks (Barberis, 2013). However, to my knowledge this approach has not been tested in an exploration setting.

The goal of this thesis is to compare these three different theories and algorithms when applied to the exploration problem. Therefore, the research question is: How does prospect theory compare to expected utility theory and reinforcement learning in exploring an environment?

In the following sections prospect theory and the other approaches will be explained in more detail. Later the implementation of the problem will be shown. This is followed by how the three before mentioned theories will be implemented and applied. Next, I will show the results of my research and what conclusions are drawn from this. Finally, the limitations of my work will be mentioned as well as possibilities for further research.

2. Background

2.1 Prospect theory

As mentioned before, prospect theory was created by Kahneman and Tversky in 1979. The goal of their paper was to offer critique on expected utility theory as a model of decision making with risks. Their paper aims to create a better model to represent human decision making when risks are involved. They found several inconsistencies for human decision making compared to expected utility. Furthermore, they highlighted these inconsistencies by showing hypothetical situations in which the axioms of expected utility are violated. These situations consisted of two options where the outcomes represent Israeli currency. They were presented to students and faculty members of various universities (Kahneman & Tversky, 1979). Even though they created this theory several decades ago, it is still shown to be relevant (Barberis, 2013).

Initially, they presented several problems in which they showed the certainty effect. This effect states that people tend to underweight outcomes that are paired with a probability in comparison to outcomes that are certain. One example problem they showed is shown below in figure 1.

PROBLEM 1: Choose between

A: 2,500 with probability .33,	B: 2,400 with certainty.
2,400 with probability .66,	
0 with probability .01;	
$N = 72$ [18]	[82]*

PROBLEM 2: Choose between

C: 2,500 with probability .33,	D: 2,400 with probability .34,
0 with probability .67;	0 with probability .66.
$N = 72$ [83]*	[17]

Figure 1; problem 1 & 2 (Kahneman & Tversky, 1979)

The responses show that 82% of participants choose option B over option A. It also shows that 83% of people choose option C over option D. However, problem 1 and 2 are remarkably similar. The only difference is that in problem 2 on both sides a 66% chance for 2400 is removed. Thus the difference in the results can be attributed to the certainty effect as mentioned before. This is not the only instance of the certainty effect they show in their paper.

Next, they expand on these problems by creating similar situations, to show the difference between gains and losses. These variants would have the opposite outcomes (gains became losses). When people were presented with these problems in a setting where they would suffer

losses, their behaviour drastically changed. Initially, they would try to avoid risks to maximize their gains. This behaviour is called risk averse. However, when they showed situations where losses are involved the participants were choosing the options paired with a probability. The reason for this shift in behaviour is that people want to minimize their losses. This behaviour is called risk seeking. So, they conclude that when gains are involved people tend to avoid risk (risk averse) and with losses people tend to choose the option with more risk (risk seeking). Again an example of a problem with gains is shown below in figure 2, however now the results also provide insight on how people would act when presented with options where losses are unavoidable.

PROBLEM 7:

A: (6,000, .45), B: (3,000, .90).

N = 66 [14] [86]*

PROBLEM 8:

C: (6,000, .001), D: (3,000, .002).

N = 66 [73]* [27]

Figure 2; problem 7 & 8 (Kahneman & Tversky, 1979)

As can be seen from the data, originally 86% of participants prefer option B over option A. However, in their research they showed that when these gains are converted to losses the majority (92%) would choose option A over option B. This also holds for problem 8, 73% choose option C with gains and 70% choose option D when losses are involved.

Furthermore, they showed that the behaviour of people can be influenced by how problems are presented, even when both options have the same result. An example of this can be found below in figure 3.

PROBLEM 11: In addition to whatever you own, you have been given 1,000.
You are now asked to choose between

A: (1,000, .50), and B: (500).

N = 70 [16] [84]*

PROBLEM 12: In addition to whatever you own, you have been given 2,000.
You are now asked to choose between

C: (-1,000, .50), and D: (-500).

N = 68 [69*] [31]

Figure 3; problem 11 & 12 (Kahneman & Tversky, 1979)

In this case, both option A and C result in a 50% chance to end up with 1000 or 2000. In both option B and D the result is a guaranteed 1500. However, the results from both problems are significantly different. This is due to how the problems are presented. As mentioned before, when gains are involved (problem 11) people show risk averse behaviour and when losses are involved (problem 12) they are risk seeking. Additionally when all outcomes are calculated using expected utility, all options are the same.

Ultimately, they present two new functions, a value- and a weight function. The value function represents the changes with respect to a reference point in function of the results. They describe this function using two arguments, the reference point, and the magnitude of change (positive or negative). Furthermore, they conclude that the perceived difference between 100 and 200 is larger than the perceived difference between 1100 and 1200. Thus, they hypothesize that the function is concave for gains and convex for losses. Also, they determined that this new function is steeper for losses compared to gains. A hypothetical function that satisfies all these constraints can be seen below in figure 4. The weight function they created presents decision weights with respect to different prospects. Decision weights are inferred from choices between prospects but are not probabilities. For this function numerous properties need to be considered. For example: subadditivity, subcertainty, subproportionality, and overweighting. For more detail on these properties, or prospect theory in general, see their original paper (Kahneman & Tversky, 1979). In figure 4, there is also a possible decision weight function shown that satisfies the mentioned properties.

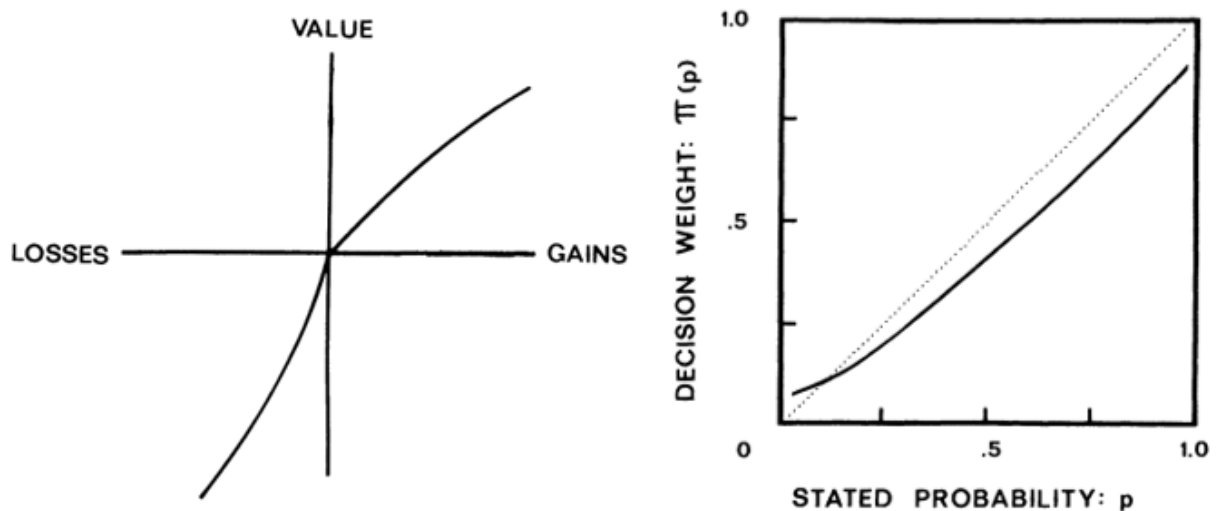


Figure 4; Possible value function (left) and possible decision weight function (right) (Kahneman & Tversky, 1979)

2.2 Expected utility

As mentioned before, prospect theory was created as a critique on expected utility. Just like prospect theory, expected utility is a theory for making decision when faced with uncertainty. However, in contrast to prospect theory we assume a decision is based on perfect rationality. The expected utility is calculated by taking the weighted sum of all possible options. The corresponding formula is shown below. In this formula, $E(u)$ is the expected utility, $u(x_i)$ is the utility of outcome x_i , and finally p_i is the probability of x_i (Fishburn, 1968). When we apply this formula to the example shown before in figure 3, we can conclude that all options have the same expected utility. Therefore, there should be no difference in the results.

$$E(u) = \sum u(x_i) * p_i$$

2.3 Reinforcement learning

Reinforcement learning (RL) is one of the three components of machine learning, alongside supervised- and unsupervised learning. In supervised learning there is labelled data to train an agent. In contrast to unsupervised learning where the data is not labelled. RL does not need labelled data but does get feedback. This feedback does not state whether a decision is correct or not as in supervised learning but provides a result to the agent. In RL an agent is incentivized to learn what actions lead to the best results. Actions are paired with positive or negative rewards, depending on the desired outcome. This encourages the agent to learn a sequence of actions that leads to the optimal cumulative reward. RL can be applied in numerous fields, such as games, categorization, or exploration (Li, 2019). It has been shown that RL has achieved significant success in numerous domains. However, it still suffers from serious drawbacks that stop RL from solving real-world problems. One of the main disadvantages is the learning inefficiency. It can require millions of interactions with the environment to solve simple problems. This is not feasible in the real-world due to time and hardware constraints of the agent and the environment (Hao et al., 2023).

One of the main achievements of RL in the past decade is AlphaGo zero. This is the evolved form of its predecessor AlphaGo. AlphaGo was the first program to achieve superhuman performance in the game Go. It achieved this behaviour by learning from two deep neural networks. The first network was trained by looking at human expert moves. The second network was trained using RL. It played matches against itself and from this it gathered data to determine the best move in every position. This was done by determining the most likely move by using Monte-Carlo Tree Search (MCTS). In a nutshell, MCTS is a heuristic search algorithm that combines tree search algorithms and RL. In is most often used in the context of playing board games or other turn-based games (Wang, 2021). AlphaGo Zero was based solely on RL using a MCTS approach. Using this it used self-play to learn what the best actions at a given

state were. This new and improved version of the original AlphaGo beat its predecessor 100 to 0 and showed that it was the best Go program at that time (Silver et al., 2017).

One of the main problems in RL is exploitation vs exploration. Exploitation refers to repeating actions that are already known to lead to a good result. Exploration is the action of choosing actions that lead to unknown results, in the hope of finding better alternatives (Gupta et al., 2006). A common real-life example is to visit your favourite restaurant or choose a new place to eat. In the first case you know that the food is good, but by choosing a new option you may find even better food. This problem is also relevant in exploration. In exploration you can choose a route that is guaranteed to lead to the target, or you can decide to take a new route. This new route may be cheaper or shorter than the route you knew before, but it can also lead to worse results.

One solution to this problem is by using a ϵ -greedy approach. This approach offers a mix of both exploitation and exploration. It can be decided what fraction of actions you want to explore vs exploit by choosing an acceptable value for ϵ . For example, if you set ϵ to 0.6, you will explore a new action 60% of the time. By choosing a small value you can guide the agent to exploit the actions that you know lead to a good result. By increasing this value, you can force the agent to take more exploring actions. This can possibly lead to better results. However it does require more time and computational power (Lindwurm, 2019).

3. Methods

3.1 General framework

To compare the theories and algorithms, a suitable environment is needed. The software used to create this environment is Unity. Originally, Unity was created as a game engine to support cross-platform gaming. Currently it has been adopted in a much wider range of industries, such as film, engineering, and architecture. This is because it enables the user to create 2- or 3-dimensional settings and provides a lot of possibilities for customization. These possibilities arise from the ability to create, combine, and customize objects and scripts (Dealessandri, 2020).

There are several reasons to support the choice of Unity as the environment. First, it provides the ability to create a 3-dimensional terrain. This terrain is in a nearly continuous space, this means there are a lot of options to explore for the agent. Furthermore, this space provides a better representation of real life when compared to graphs or a grid. Second, Unity is widely used for a variety of applications, this is because of the many possibilities it provides to the user. Not only does it provide many tools, but it also makes it easy to integrate and create your own tools needed for a project. This includes libraries to increase the number of options even further. Third, since it is widely used, there is a lot of documentation to help solve complex problems.

Now that we have an environment to implement the problem, a terrain to explore can be created. First, a simple terrain with all the necessary aspects is created. Later the complexity of the terrain can be increased. The terrain is made up of a continuous space and is one of the basic objects in unity. This terrain is covered in many individual tiles that all need to be covered to complete the exploration. A good analogy of this situation is to see the agent as a small paint brush, this brush needs to paint the entire canvas (terrain) to finish exploring. To create this canvas, a new script is created. This script places tiles in a square at a small interval all throughout the terrain.

A snippet of this script can be found below. In this snippet, the `envSize` is inherited from the agent script. This determines the size of the square in which the tiles are placed. To place the tiles a for loop is created with the size of the complete environment. In this loop the built-in Unity function *Instantiate* is used. This function clones an object and places it elsewhere and it requires three arguments. First, is the object that needs to be placed. In this case it uses a prefab that has been set to a tile object. Next is the location. This also requires three arguments, x-, y-, and z-coordinates. The specific coordinates are calculated by the formula shown in the snippet. Every tile is placed slightly below the original terrain object since it is focused on covering the horizontal part of the tile. The interval between the tiles is determined by the variables `x_space`

and *y_space* and is set beforehand. Last is the rotation of the object. This is kept the same as the original tile and does not require changing in this setting.

```
envSize = Agent.envSize;
for (int i = 0; i < envSize * envSize; i++)
{
    Instantiate(
        prefab, // placed object (tile)
        new Vector3(x_space * (i % envSize) + 0.5f, -0.05f, y_space * (i / envSize) + 0.5f), // location
        Quaternion.identity // rotation (original)
    );
}
```

Additionally, an agent is needed to explore. This agent should be viable for all theories and algorithms. It consists of a basic spherical object and a script since this allows for easy movement in all directions and the script enables intelligent behaviour. This behaviour includes selecting optimal targets and finding the shortest or cheapest route to this target. The target selection can differ per theory or algorithm but movement is needed for all three. Usually, agents in an exploration setting have a limited amount of energy. Therefore, it is important to use this energy efficiently (Mei et al., 2006). To further increase the realisticness, gravity is also considered. When on uneven terrain the agent is pulled downward. This makes moving downward cheaper in comparison to moving on a flat surface. However, moving uphill requires even more energy than both these situations.

The implementation of the cost function can be found below. It is a function that returns a float value that contains the distance of moving between two given points. First, the horizontal difference is covered. Next, when the agent is moving uphill the difference in elevation is multiplied and added to the horizontal difference. When the agent is going downward, the elevation is also multiplied but subtracted from the horizontal distance. If there is no difference in elevation the *y_diff* value is 0, so does not influence the cost.

```
// return cost of moving between two points better
float Cost3( Vector3 pos, Vector3 pos2){
    float d; // horizontal distance
    float x_diff = pos.x - pos2.x; // x difference
    float z_diff = pos.z - pos2.z; // z difference
    d = Mathf.Sqrt(x_diff*x_diff + z_diff*z_diff);

    float y_diff = pos.y- pos2.y;
    return d + 3*y_diff;
}
```

However, there is also a possibility to recharge the battery. Numerous charging points have been scattered throughout the environment. Initially, their locations are set randomly but this can be specified beforehand if need be. Beforehand, the agent is unaware of these locations. Once the agent enters the vicinity of these chargers, it stores their location internally. However, as mentioned in the introduction noise is involved. This noise is relative to the distance between the agent and its target. These chargers are represented by a blue cube in the environment. They provide the agent with the possibility to fully recharge the battery but only once per station. In figure 5 below, a small sample environment is shown including three charging stations and the agent itself.

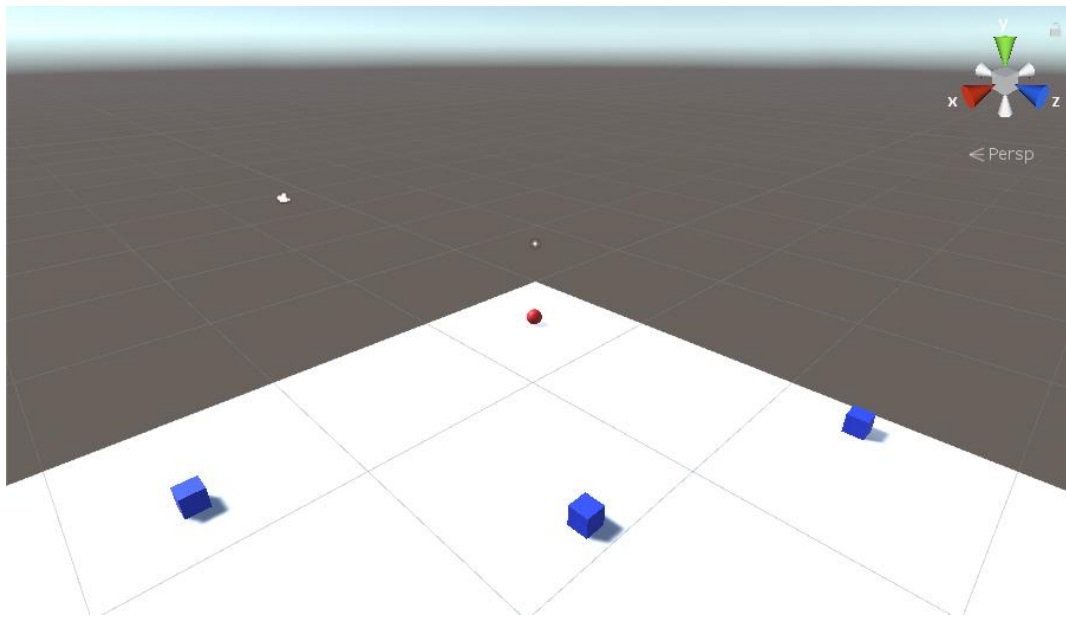


Figure 5; simple example terrain

Finally, there are some functions needed for all three implementations. The noise function is one of them. It gets the position of the target as an argument. First it determines the distance between the location of the agent and the target. The noise added is relative to the distance between the position of the target and the agent. This is done to simulate a noisy sensor that can be used in the real world. Next, the x and z position of the target are determined. However, these positions are altered by adding a randomly sampled number from a normal distribution. Finally the target position is returned. There is a snippet of this implementation down below.

```
private Vector3 Noise(Vector3 target){
    float dist = Vector3.Distance(transform.position, target);
    float stdDev = dist / 2;
    target.x += randomNormal(0, stdDev);
    target.z += randomNormal(0, stdDev);
    return target;
}
```

In the previous snippet it is visible that the numbers are sampled from the *randomNormal* function. Unfortunately, Unity does not provide a random normal distribution function as shown above. An alternative way to create randomly sampled numbers is by using the box-muller approach. This method requires randomly generated numbers from a uniform distribution (which Unity does provide) to generate samples that follow a normal Gaussian distribution. For more detail on this method see their original paper (Box & Muller, 1958). An implementation of this method can be seen in the snippet below.

```
// return a random number sampled from normal distribution
private float randomNormal(float mean = 0, float stdDev = 1){
    // standard values -> mean = 0 & stdDev = 1
    // Box-muller transform
    float u1 = 1f - Random.value;
    float u2 = 1f - Random.value;
    float randStdNormal = Mathf.Sqrt(-2f * Mathf.Log(u1)) * Mathf.Sin(2f * Mathf.PI * u2);
    return mean + stdDev * randStdNormal;
}
```

To create more complex environments elevation and/ or obstacles are added. Obstacles are represented by a grey wall in the environment. They can blockade certain routes and force the agent a different way. These obstacles can help to make the environment more realistic. An increase in elevation in the environment can lead to bigger difference between the cost and distance between two points. This is because moving up or down influences the cost. Furthermore, by increasing the complexity more things need to be considered when deciding where to go next. Examples of this are that moving down is cheaper and often has a higher probability of succeeding compared to moving upwards due to the discrepancy between the cost.

There are numerous occasions when the simulation ends. The first possibility is that the agent has explored the entirety of the environment. This means the agent has succeeded in exploring the presented environment. In most cases this is the desired outcome of the program. Another possibility is when the battery of the agent runs out (the agent has failed to explore everything). Usually when this happens there is still a part that has not been visited and thus not been explored. Lastly, there is a possibility of the agent getting stuck in a more complex environment. In this scenario the battery has not run out, but movement is no longer possible or does not benefit the agent in any way. However, this last situation is rather uncommon.

3.2 Application of theories

Now that the framework is set up, the key ideas behind the different theories and algorithms can be implemented in this setting. All implementations are based on a target selection method. This method works by selecting a single object out of numerous possibilities as the goal of movement. Target selection directly influences exploration time and energy consumption (Mei et al., 2006). In the context of this thesis that means selecting a specific tile or charging point as the target.

The implementation of prospect theory and expected utility theory are similar. Both implementations are based on a greedy algorithm, at every possibility they choose the target with highest immediate reward. This reward is based on two factors; the gain and the probability of reaching the target. However, initially all tiles in the environment have the same value of 1. This changes when the agent has visited a tile, the reward on this tile now becomes 0. Since all tiles have the same reward, the probability is key in deciding the target. The targets are sorted on their distance to the agent. The reasoning behind this is that the smaller the distance the bigger the probability of reaching the target. Furthermore, Mei et al. (2016) found that sorting targets on distance instead of utility can lead to a significant decrease in energy consumption. The main difference between prospect- and expected utility theory is how to act given certain probabilities. The implementation based on prospect theory is very risk averse, this means that it tries to avoid getting stuck in every situation. The reason for this is to maximize their gains by avoiding risky situations.

The behaviour of the agent can be roughly divided in three separate situations. In the first situation there is a significant amount of exploring to do and the battery is relatively full. An instance of this is at the start, assuming that the agent starts with a charged battery. In this case the agent keeps exploring to decrease the number of tiles that it must visit while keeping track of the battery and distances to the chargers. The second situation is when the battery level starts to reach the limit of reaching the charger. In this case the agent needs to determine the possibility of completing the environment compared to reaching the charging point. These possibilities are an estimation based on the size of the environment and the area that is already explored. One option is that the agent determines charging a necessity, then the agent makes its way to the charger and continues exploring. Another possibility is to finish exploring, this should be preferred over the previous option, if possible. The third situation arises when the agent has utilized all chargers but is still exploring. When this happens the agent should try and maximize the terrain explored with the remaining battery.

The main difference between the implementations of prospect- and expected utility theory happens in the second situation. The estimates in both implementations are the same but their decision can significantly differ given this estimation. It can happen that both agents come to the conclusion that finishing exploring is impossible, then both will decide to charge. Both can also choose to finish exploring if they believe it is possible. However, as mentioned before their estimation is based on the size of the environment and the area that is already explored. Since this area is unknown they cannot be sure of the possibility of completing exploration. This means there is risk involved in their decision making. The implementation of prospect theory is based on risk averse behaviour. This implies that the agent tries to avoid getting stuck or running out of battery more than the implementation based on expected utility theory. This behaviour influences the decisions of the agents, often prospect theory chooses charging when it is uncertain about the results. On the other hand, expected utility theory makes its decision based on perfect rationality.

Another difference between the implementations of these two theories is the decision when charging is necessary. As mentioned before, prospect theory has risk averse behaviour, which discourages it from not finishing exploration. As a result of this behaviour prospect theory will generally go charge earlier compared to expected utility to reduce to possibility of not finishing exploring since it tries to maximize the gains in every situation. This is because there is noise involved in the sensors and this influences the guessed distance to the charging point.

Finally reinforcement learning needs to be applied in exploration as well. As mentioned in chapter 2, there are multiple options for RL. The implementation of RL in this thesis is based on the 'Unity Machine Learning Agents' library from Unity. Their implementations are based on PyTorch and enable enthusiasts to easily train agents for several use-cases. Furthermore, this library can be used to train agents on many methods such as RL, imitation learning, or many others. The specific implementation in this thesis is based on the epsilon greedy approach as mentioned in chapter 2. This library is also used in the scientific community to create intelligent agents that perform various tasks. Even when the environment itself is rather simple, it has been shown that the agents trained by these methods can demonstrate complex and intriguing solutions (Urmanov et al., 2019). For more information to this specific library and training methods I refer to their GitHub page (*Unity ML-Agents Toolkit*, 2017/2023).

4. Results

These results are based on three different environments. The first was a completely flat environment to see how they compare when there is little to no complexity in the exploration part. This can also be seen as a baseline. The second was a bit more complex. This complexity was achieved by adding changes in elevation throughout the entire environment. The last environment was even more complex, the changes in elevation were increased and different obstacles were spread out across the environment.

The size of all three environments was consistent and set to a value of 20. This means that the agent had to explore an environment that was a square of 20 by 20, so 400 individual tiles. The speed of the agent was also kept consistent. The base speed was 7 units per second; however, this could fluctuate due to elevation, turning, or stopping. In every setting the battery started at 100% and there were 3 charging points placed randomly in the environment. These locations differed between the three environments but were consistent for every implementation in the same environment.

The resulting tables are made up of three different parts. The first row shows the average of 10 runs per algorithm per environment. The next row represents the runs that were completed out of the 10 total. In these runs the agent has fully completed exploring. The last row shows the failed runs. These are runs where the program finished before the exploration was complete. This could be either due to running out of battery or getting stuck. There are multiple metrics used to compare the theories. They are time spent, distance driven, remaining battery percentage, percentage of total terrain explored, and the number of successes out of 10 runs.

4.1 Environment 1

Table 1, environment 1, prospect theory:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	56,32	389,41	0,64	98,6%	3/10
FINISHED	56,59	390,90	2,16	100%	3
FAILED	56,22	388,84	0	98,2%	7

Table 2, environment 1, expected utility theory:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	49,80	341,18	6,39	86,9 %	5/10
FINISHED	55,76	384,69	12,78	100 %	5
FAILED	43,83	297,54	0	73,9 %	5

Table 3, environment 1, reinforcement learning:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	55,82	383,33	7,45	99,6 %	9/10
FINISHED	55,41	383,24	8,27	100 %	9
FAILED	59,51	384,14	0	96 %	1

4.2 Environment 2

Table 4, environment 2, prospect theory:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	54,71	379,61	0,26	91,7 %	1/10
FINISHED	59,33	410,44	2,6	100 %	1
FAILED	54,19	376,18	0	90,8 %	9

Table 5, environment 2, expected utility theory:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	52,05	360,52	3,50	86,8 %	3/10
FINISHED	58,25	402,28	11,70	100 %	3
FAILED	49,39	299,79	0	81,2 %	7

Table 6, environment 2, reinforcement learning:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	56,34	397,92	3,41	96,7 %	8/10
FINISHED	57,40	401,34	4,26	100 %	8
FAILED	52,07	384,24	0	83,5 %	2

4.3 Environment 3

Table 7, environment 3, prospect theory

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	57,66	400,22	0	94,0%	0/10
FINISHED	-	-	-	-	0
FAILED	57,66	400,22	0	94,0 %	10

Table 8, environment 3, expected utility theory:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	50,99	353,05	0,48	85,3 %	1/10
FINISHED	59,61	410,28	4,78	100 %	1
FAILED	50,04	346,69	0	83,6 %	9

Table 9, environment 3, reinforcement learning:

	TIME SPENT	DISTANCE DRIVEN	BATTERY LEFT	TERRAIN EXPLORED	COMPLETED
TOTAL	55,68	398,89	3,4	94,1 %	6/10
FINISHED	57,53	402,12	5,7	100 %	6
FAILED	52,90	394,05	0	85,3 %	4

5. Discussion

5.1 Analysis of results

We start by taking a closer look at the results in the first environment. To interpret these results analysis of variance is used. From this analysis it becomes clear there is no significant difference in time spent. Furthermore, there is no significant difference between distance driven in the three categories. There is a significant difference between the amount of battery left between prospect theory and both expected utility and RL. But no significant difference in the percentage of terrain explored. However, we can see that prospect theory, expected utility theory, and RL have completed 3, 5, and 9 runs respectively.

In the second environment the results were analysed in the same way as before. Again there was no significant difference in time spent. There was not a significant result in the comparison between distance driven. In the second environment there was no significant result between battery left as seen in the previous environment. Finally, there was also no significant result between the percentage of terrain explored. Prospect theory completed 1 run, expected utility theory completed 3 and RL completed 8 out of 10 possible runs.

The last environment also did not provide any significant results for time spent. This was also the same for distance driven. However, there is a significant difference between the amount of battery left in RL compared to the other two methods. But no significant difference between battery left when prospect theory and expected utility are compared. Finally there also is a significant difference between RL and expected utility theory when percentage of terrain explored is compared.

The average times of the failed runs are lower compared to the finished runs. The reason for this is that in some cases the agent did not reach the charging point in time so their battery ran out. This caused the runs to end early and decrease the average time.

5.2 Limitations

Overall, it seems that RL produces the best results. However this is not supported by much statistical analysis. The reason there is no significant results can be due to the underwhelming number or runs in every category. This number could be drastically increased by running simulations in the Unity environment. This was not possible due to the way the general framework was set up. This is a significant limitation to the produced results and should be considered for further research.

Furthermore, it should also be noted that the RL implementation required training time. This was because it finds a possible sequence of actions by using the epsilon greedy approach. It tries to tackle the exploitation vs exploration problem by using this method. Even though the results produced were good we cannot be certain this is the best possible method. Given more training time, RL could possibly have found even better actions which lead to better results. On the other hand, both prospect theory and expected utility do not need any training since the implementations are based on a greedy algorithm. This should definitely be considered when interpreting these results.

Additionally, the implementation for prospect theory could be improved. One way to achieve this is by considering both the value and decision weight function by Kahneman and Tversky. It should also be considered that they originally did not intent for prospect theory to be used this way. Therefore, it should not be considered the best model of human behaviour in this specific situation. Also, the rewards for both exploration and not finishing the run were chosen arbitrarily. This influenced the behaviour of all three implementations. Different values could lead to significantly different outcomes.

5.3 Further research

Further research could increase the applicability of this thesis or similar research. This can be achieved in multiple ways. One option is to create situations where the utility of different object could differ. For example, this could be that higher points provide more vision, and thus have a higher utility. This could be achieved by using frontiers instead of tiles like the paper by Mei at al. (2006). Furthermore, it could also be interesting to give a negative reward for tiles that are visited more than once. This could possibly increase the efficiency of learning for RL and decrease repeated coverage of the same area to increase the efficiency of the agents. Another intriguing option is to compare total battery usage of different methods. This could be used to see how much battery is needed and also possibly increase efficiency. Finally, a more accurate implementation of prospect theory could be beneficial. Both the value and decision weight function should be integrated. This could lead to an even better model of prospect theory and be more applicable to different situations.

5.4 Conclusion

Finally, I conclude that it may seem that RL produces the best results. However, this could be due to chance since most results did not provide any significant differences. When the number of completed runs are compared, RL performs best, followed by expected utility and lastly prospect theory. However, there are some major limitations to the research presented in this thesis. Therefore, it becomes unclear what to conclude from this research and the original problem with RL has not been solved and it is uncertain if either prospect theory or expected utility can be used as an alternative.

6. Bibliography

- Barberis, N. C. (2013). Thirty Years of Prospect Theory in Economics: A Review and Assessment. *Journal of Economic Perspectives*, 27(1), 173–196.
<https://doi.org/10.1257/jep.27.1.173>
- Box, G. E. P., & Muller, M. E. (1958). A Note on the Generation of Random Normal Deviates. *The Annals of Mathematical Statistics*, 29(2), 610–611.
<https://doi.org/10.1214/aoms/1177706645>
- Dealessandri, M. (2020, January 16). *What is the best game engine: Is Unity right for you?* GamesIndustry.Biz. <https://www.gamesindustry.biz/what-is-the-best-game-engine-is-unity-the-right-game-engine-for-you>
- Fishburn, P. C. (1968). Utility Theory. *Management Science*, 14(5), 335–378.
<https://doi.org/10.1287/mnsc.14.5.335>
- Garaffa, L. C., Basso, M., Konzen, A. A., & de Freitas, E. P. (2021). Reinforcement Learning for Mobile Robotics Exploration: A Survey. *IEEE Transactions on Neural Networks and Learning Systems*, 1–15. <https://doi.org/10.1109/TNNLS.2021.3124466>
- Gupta, A. K., Smith, K. G., & Shalley, C. E. (2006). The Interplay Between Exploration and Exploitation. *Academy of Management Journal*, 49(4), 693–706.
<https://doi.org/10.5465/amj.2006.22083026>
- Hao, J., Yang, T., Tang, H., Bai, C., Liu, J., Meng, Z., Liu, P., & Wang, Z. (2023). *Exploration in Deep Reinforcement Learning: A Comprehensive Survey* (arXiv:2109.06668). arXiv. <http://arxiv.org/abs/2109.06668>
- Kahneman, D., & Tversky, A. (1979). Prospect Theory: An Analysis of Decision under Risk. *PROSPECT THEORY*, 29.
- Kiureghian, A. D., & Ditlevsen, O. (2009). Aleatory or epistemic? Does it matter? *Structural Safety*, 31(2), 105–112. <https://doi.org/10.1016/j.strusafe.2008.06.020>
- Li, Y. (2019). *Reinforcement Learning Applications* (arXiv:1908.06973). arXiv. <http://arxiv.org/abs/1908.06973>
- Li, Y., Chen, J., & Feng, L. (2013). Dealing with Uncertainty: A Survey of Theories and Practices. *IEEE Transactions on Knowledge and Data Engineering*, 25(11), 2463–2482.
<https://doi.org/10.1109/TKDE.2012.179>
- Lindwurm, E. (2019, November 3). *Intuition: Exploration vs Exploitation*. Medium. <https://towardsdatascience.com/intuition-exploration-vs-exploitation-c645a1d37c7a>

- Mei, Y., Lu, Y.-H., Lee, C. S. G., & Hu, Y. C. (2006). Energy-efficient mobile robot exploration. *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, 505–511.
<https://doi.org/10.1109/ROBOT.2006.1641761>
- Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., & Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359. <https://doi.org/10.1038/nature24270>
- Stormont, D., & Allan, V. (2009). Managing Risk in Disaster Scenarios with Autonomous Robots. *Journal of Systemics, Cybernetics and Informatics*, *7*.
- Thrun, S., & Burgard, W. (2002). Probabilistic robotics. *Communications of the ACM*, *45*(3), 52–57. <https://doi.org/10.1145/504729.504754>
- Unity ML-Agents Toolkit*. (2023). [C#]. Unity Technologies. <https://github.com/Unity-Technologies/ml-agents> (Original work published 2017)
- Urmanov, M., Alimanova, M., & Nurkey, A. (2019). Training Unity Machine Learning Agents using reinforcement learning method. *2019 15th International Conference on Electronics, Computer and Computation (ICECCO)*, 1–4.
<https://doi.org/10.1109/ICECCO48375.2019.9043194>
- Wang, B. (2021, January 11). *Monte Carlo Tree Search: An Introduction*. Medium.
<https://towardsdatascience.com/monte-carlo-tree-search-an-introduction-503d8c04e168>