CS698R: Deep Reinforcement Learning Project

Final Project Report

Team Name: Quad-A
Project Title: Foraging in Replenishing Patches
Project #: 2
Team Member Names: Roll No
Abhay: 180014
Ankur Gupta: 180109
Anshul Rai: 180117
Archit Bansal: 180134

1 Introduction

Foraging is the task of collecting natural resources such as food or wood. While foraging for food, animals have to make decisions often in the lack of complete information. They need to decide whether to keep exploiting a particular patch whose reward decreases with each exploitation or explore the environment for new patches with potentially higher rewards. The decision of leaving a patch is often associated with the energy and time cost of finding new patches during which the animal or human has no food. In such an environment making a wrong decision can often be fatal, and hence human brains have evolved to make decisions that are very close to the optimal decisions. Even though we know that the human brain most often takes the optimal decision, we do not know much about how the learning takes place and hence cannot create models to mimic human behaviour. While studying any such problem, the general aim is to understand how the human brain learns and how it can be mimicked.

Here we will be trying to find an optimal agent to collect berries from replenishing bushes in the environment. There is a time cost to travel between bushes and harvesting. The agent is allowed to go back to the bush they have harvested in the past. Each bush may or may not have berries, and the number of berries collected from a bush decreases with each harvest. This environment is hence very close to the real-life problem of foraging in which there are a limited number of bushes that the human keeps revisiting and bushes replenish when not being eaten, and there is a decay of berries when the bush is being eaten. A real-life example of such an environment is the movement of migratory birds and the movement of cattle by nomadic people. As discussed during the presentation, this also has many other applications, such as data collection in UAVs and deciding user behaviour on social media.

2 Related Work

The foraging task is an important problem in the field of neuroscience. It models two theoretical dilemma - stay-switch(patch leaving) and accept-reject(diet-selection). Various approaches to solve this have been proposed over the years, such as the Mean Value Theorem, as discussed in Charnov [1976], is optimal for non-replenishing patches. Papers like Collins and Frank [2012] discuss the drawbacks of pure RL approaches and include the role of working memory in decision making.

Schönberg et al. [2007] studied about the reinforcement learning signals in humans via a reward based four-armed bandit task. To model the decision of humans, they used a choice recency model as in Lau and Glimcher [2005]. We adapted this technique to train RL agent on our environment and analysed its results.

3 Problem Statement

This task aims to harvest the maximum reward from a field of replenishing bushes within a given time frame. Mathematically, we can define the problem as :

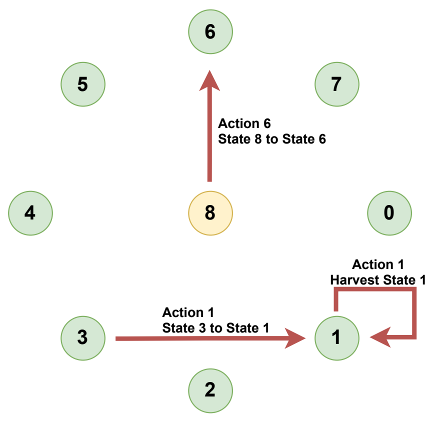
$$\text{Choose a sequence of actions : } (a_0, a_1 ... a_n) \quad \text{With the objective : } maximize(\sum_{i=0}^{n} r_i)$$

$$\text{Under the constraint : } \sum_{i=0}^{n} t_i <= T \quad \text{and} \quad \sum_{i=0}^{n+1} t_i > T \quad \forall\, a_{n+1} \in A$$

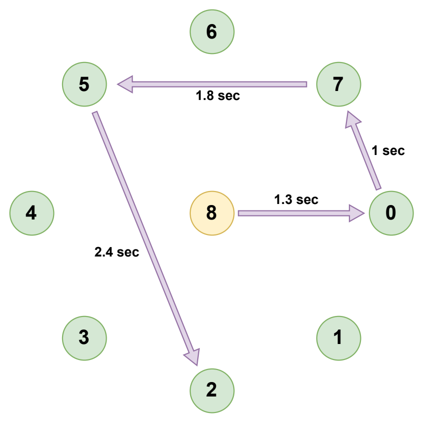
where a_i denotes the i^{th} action, r_i denotes the reward received from performing action a_i, t_i denotes the time taken to perform action a_i, T denotes the maximum time available to gather reward from the environment and A denotes the set of all possible actions. Solving this task would correspond to finding a policy π, such that the sequence of actions taken by the policy is optimal and the harvested reward is the max of the total reward possible to collect from the environment under the given constraints.

4 Environment Details and Implementation

Our environment is a combination of 8 bushes and a starting position, as shown in fig 2. The 8 bushes are arranged around the corners of a regular octagon and are numerically stated as 0-7, with the starting position being referred to as state 8. Out of the 8 bushes, 4 of them are rewarding (provides some positive reward upon harvesting them), and the others are non-rewarding (zero rewards upon harvesting them). The agent has to start from state 8 and move to one of the bushes, and then the agent can either harvest that bush or move to a different bush. The agent cannot return to the starting position ever again. For implementation we have defined 8 actions (action: 0-7) corresponding to 8 bushes (state:0-7). For agent currently in state i and performing action j, we define that if $i == j$, then the agent will harvest bush i else the agent transitions from bush i to bush j.



(a) States and Actions



(b) Time Cost of Actions

Figure 1: Enivronment Setup

Rewards: Consider the rewards stored in the rewarding bush i, j, k, l as R_i, R_j, R_k, R_l. Suppose the agent harvests bush i, then the reward collected by the agent is $0.9 * R_i$ and R_i is updated as $R_i = 0.9 * R_i$. In addition to the stored rewards, each rewarding bush has a pre-defined replenishing rate (call them r_i, r_j, r_k, r_l). Whenever a rewarding bush is harvested, stored rewards in all other rewarding bushes get replenished as:

$$R_a = min(R_a + r_a, 200) \text{ where } a = j, k, l \text{ and bush } i \text{ was harvested}$$

The overall cap of the stored reward for each bush is 200. At the start of the game, the stored reward on each rewarding bush is set to 70 ($R_i = 70, R_j = 70, R_k = 70, R_l = 70$).
Time Cost: Each action in the environment has a certain time cost associated with it. If a given action is to harvest a bush, a time cost of 1 second is incurred, otherwise the time cost is linearly scaled to the distance to be travelled (speed of the agent is constant). Time taken to travel between adjacent bushes is 1 second, and based on it, other distances are scaled and timed. Fig 2b shows the time taken by the agent for some of these cases. The total time limit for the agent to interact with the environment is 300 seconds.
Blocks: In order to study the need for working memory in our problem, there are 3 types of environment (block-1, block-2, block-3), which differ in the placement of rewarding bushes and their replenishing rate:

Block-1: i=1, j=2, k=4, l=6 with $r_1 = 4, r_2 = 4, r_3 = 4, r_4 = 4$

Block-2: i=2, j=3, k=5, l=7 with $r_1 = 8, r_2 = 2, r_3 = 2, r_4 = 8$

Block-3: i=0, j=3, k=4, l=6 with $r_1 = 2, r_2 = 4, r_3 = 8, r_4 = 16$

The demo of our rendered environment is available here

5 Proposed Solutions

5.1 Baseline

For the baseline of our task, we implemented Tabular RL methods having Q values of the form $Q_t(S_t, A_t)$ which is the generic formulation of a control problem in RL. This gave us a final policy which was far from optimal and as without the concept of decreasing rewards and replenishing patches in its understanding of state definition, it was stuck harvesting a single rewarding bush for the entirety of an episode. You can watch the baseline policy video here

NOTE: One thing worth pointing out is that we keep zero randomness in the way we finally play the environment from the learned q values. This is because the aim of a RL agent is to find an optimal policy which can never be attained if there is even an iota of randomness in the final way in which we play the environment.

5.2 Basic Algorithms

The problem with our baseline algorithms was that it did same action on a state using the final policy irrespective of the reward(feedback) that it gets from the patch about the current status of the bush's reward.

Keeping this is mind we have modified our Q value function such that it takes the previous reward as an input. Mathematically, the Q value for a particular action from a particular state when the previous step reward is R_{t-1} is defined at $Q_t(S_t, R_{t-1}, A_t)$ With this modification we ran the Tabular RL methods, namely SARSA, Q learning, Double Q Learning, SARSA Lambda.

5.3 Working Memory

Working Memory Collins and Frank [2012] is a concept in neuroscience which states that small amount of information can be stored in mind and used at the time of execution of tasks. Working memory is an important parameter used by humans for making decisions and to imitate human learning we introduce it.

We use the idea of working memory(WM) to estimate the replenishing rate of different patches. For each bush the replenishing rate can be estimated by calculating the ratio of increase in the reward of a bush since it's last harvest time and the time elapsed since it's last harvest. To improve our estimates of replenishing rate with time we have taken a running average of the estimated replenishing rate at each time step. Mathematically,

$$estimatedRepRate = (reward - lastReward)/timeElapsed$$

$$repRate = 0.6 * repRate + 0.4 * estimatedRepRate$$

We then create a WM decision module which uses the last reward experienced at each state and the time elapsed since we last visited each state, along with it's replenishing rate estimates to find the bush with maximum reward at each time step and hence move to it. Mathematically,

$$EstimatedReward = lastReward + replenishingRate * timeElapsed$$

Working memory in humans can generally store between 5 and 9 items. If we exceed the capacity of the memory we might experience loss of old data. The data stored in working memory also becomes imprecise with time. Keeping these things in mind We have implemented 4 renditions of working memory

- Perfect Memory: This is the ideal condition in which we assume perfect and infinite working memory
- Limited Memory: In this case we have limited the number of items that can be stored to 6 and 9 in the two editions we have run.

- Decaying Memory: Human memory degrades with time, to capture this we have introduced a decay to the memory with time. In this while trying to estimate the value of replenishing rate for estimating the rewards in different bushes the agent is not able to recollect the stored value precisely. Instead the agent gets a value which is sampled from $\mathcal{N}(\mu, \sigma^2)$ where μ is the estimated value stored in array and the σ^2 is dependent on the time elapsed since last time this state was harvested.
- Limited and Decaying Memory: This is a combination of the Limited Memory and the Decaying memory scenario where we incorporate both limitations, i.e. have a limited working memory and forget it as time proceeds.

5.4 Deep Networks (DQN)

We also created a DQN . The model takes as input a 10X1 vector where the first 9 indices are used to store the state as a one-hot encoded vector and the last index stores the previous reward the agent had received after normalisation. This is passed through two fully connected layers of 128 and 64 nodes and finally given output of 8X1 which is the estimate of the q value for those actions. We used MSE loss with 128 batch size and Adam optimisation to train this model.

5.5 Choice Recency Module with RL Agent

As shown by Schönberg et al. [2007], along with previous rewards, previous choices can be used to model the decisions(actions) of an RL agent. This is done via keeping a vector c for each state-reward combination. Whenever any action is performed at a particular state, the corresponding choice value is set to one. The choice values corresponding to other action for the given state-reward is decayed by a factor of d, known as *choice recency index*. We implemented this approach in the action select part of Sarsa algorithm. The action is chosen via the softmax function where probabilities are represented as shown below :

$$\frac{exp((\text{Q[s][r]+ b[s]*choice[s][r]})/\tau)}{sum(exp((\text{Q[s][r]+ b[s]*choice[s][r]})/\tau))}$$

In the above equation, τ is the softmax temperature which is decayed exponentially through the episodes and b is the coefficient. As stated by the authors, the coefficient b can be negative as it helps to capture a tendency to alternate or positive for perseveration . We have made b, a state dependent vector as each state should have different impact of previous chosen action. Since in the above equation, the coefficient b controls the overall probabilities used in softmax, it becomes essential for the RL agent to learn the correct values of b. We exploited the replenishing rates learnt by the working memory module in order to predict correct values of b. For rewarding bushes, the b coefficient is chosen as the state replenishing rate divided by the minimum replenishing rate. This ensures that the choice values of higher replenishing bushes have greater impact than those of lower rewarding bushes. The final policy on which the agent is evaluated used argmax function instead of softmax function used while training.

6 Experiments

We have performed extensive experimentation on all of the proposed solutions in order to decide upon the best hyperparameter settings as well as get insights into the nature of the problem and the solutions used.

The first set of experiments we tried out was to determine how to incorporate information about the previous reward received, into the state variable. We tried binning the received rewards into bins of different sizes into order to find a balance between being able to capture all points of transitions to new optimal action while at the same time not increasing the state space indefinitely as we are using Tabular RL methods. We found that using the integer part of the reward received by the agent was a good binning strategy and use the same for all our future approaches.

The next set of experiments we carried out was to determine the gamma hyperparameter. This hyperparameter is responsible for determining how much influence an action has on later rewards. As in our environment certain actions such as harvesting non-rewarding patches have no effect on the rewards received by the agent later on in the episode, we hypothesized that we may require to use a lower gamma value to correctly model the environment. We perform a grid search to find the optimal hyperpatameter value and find that the optimal results are obtained for a gamma value of 0.3-0.4. For each algo. we separately run experiments on both gamma 0.3 and 0.4 in order to get the best results.

Apart from that, certain hyperparameters such as how the learning rate decays over the complete training cycle as well as how the epsilon decays when using the epsilon greedy action selection strategy are unique to each algorithm and are tuned separately for each of them.

The next set of experiments we carried out was regarding the comparison of different action selection strategies in the Tabular RL models. In order to solve the control problem, the agent is required to balance between exploration and exploitation, and there exist a number of different strategies for doing the same. In our experiments, we contrast the nature of learning curves obtained when using Epsilon-Greedy, Softmax and UCB action selection strategies. For epsilon-greedy strategy, we decay the epsilon value linearly from 1 to some finetuned final value, such that towards the end of the training cycle, the agent does little exploration and mostly exploitation. In the case of Softmax and UCB, we tune the hyperparameters tau and c for the two algorithms respectively at each of the 3 blocks of environment.

For all the results we make during our experimentations in order to compare the algorithms, we run each algorithm for five different seeds and average over their results in order to get a more consistent analysis. Also, we compare the algorithms by plotting the rewards received from a greedy policy over the estimated state-action value function at the end of each reward. This graph performs a function similar to the % optimal policy graph we use when we know the exact

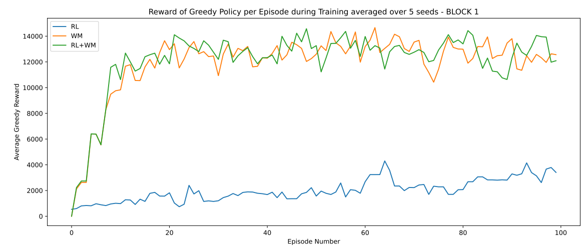
Note that for all further experiments, we use the Epsilon-Greedy action selection strategy alongside the different control algorithms we try.

For the WM part we conduct 2 experiments. In first we change the number of values that we can store in limited working memory, we try the values 6 and 9. Secondly we need to experimentally fix the standard deviation of the recall normal function in forgetful WM. It has been set so as to keep the deviation within 1 decimal place for realistic gameplay.

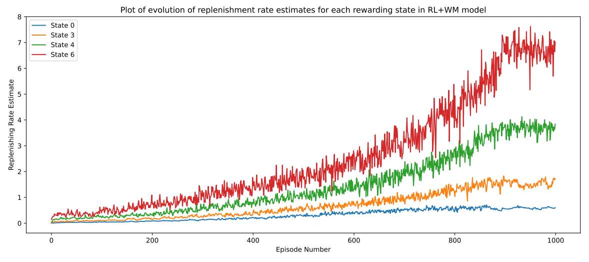
In case of choice recency module, we experimented with the the softmax temperature τ, the choice vector and the coefficient b. The softmax temperature was adjusted so as to balance out the effect of large Q values. The choice vector was initially initialised to 1 as discussed by authors, however it was later initialized to 10, so as to have a meaning impact in front of Q. Additionally, various settings for coefficient b was experimented out. Initially it was chosen as a single constant, but changing to a state dependent quantity gave meaningful results as discussed in previous section.

7 Results and Analysis

Tabular Methods: The result for tabular method are as shown in table 1. Q learning is performing much better than double Q and SARSA for block 1 and 3 but SARSA is performing best on block 2.



(a) Comparision between performance on WM, RL and RL+WM agents over 1000 episodes (Block-1)



(b) Evolution of estimate of replenishment rates with training episodes (Block-3)

Figure 2: Working Memory with RL Agent

Action Selection Strategy: Amoung the different available action selection strategies, UCB seems to be performing the best closely followed by softmax. Epsilon-Greedy strategy perform poorly compared to these two. Apart from the final scores, UCB also shows higher scores (for greedy policy of current Q-function estimates) in the first few episodes itself when compared to epsilon-greedy. This clearly shows that UCB learns a lot faster than epsilon-greedy strategy. (For more info. regarding the exact plots refer to the appendix)

Control Algo.	Action Selection Strategy	Block-1	Block-2	Block-3
SARSA	Epsilon-Greedy	18093.4	18473.5	23484
	Softmax	21797	19670.75	24209.5
	UCB	21366	19769	24363
Q-Learning	Epsilon-Greedy	19586.8	18093.4	24073
	Softmax	19186.6	21797	23607
	UCB	20827	21366	24642
Double Q-Learning	Epsilon-Greedy	18983.8	19181.6	22949.6
RL+WM	Epsilon-Greedy	20309.8	18929.8	23857.0
DQN	Epsilon-Greedy	22321	17132	23867
RL + Recency Module	Epsilon-Greedy	19139.33	20240.67	23856.5

Table 1: Results for Different Proposed Approaches

Blocks: Comparing across block we can see that the third block has higher reward when compared to block 2 and 1. This is because block 3 has higher replenishing rate for its constituent bushes.
Working Memory : From Fig.2 (a), we can see the effect of incorporating the WM module along with the RL agent. In the initial few episodes, the WM module is able to get some idea about the environment and hence provides a sudden boost to the agent's performance as can be seen by the difference between the pure RL and the RL+WM as well as pure WM agents. From plot b in the same figure, we can see that the WM module is able to learn the difference between the replenishing rates of the 4 rewarding bushes in the third block of the environment.
Choice Recency Module: Choice recency module improved the rewards obtained over the normal sarsa algorithm. This is consistent across all three blocks. Also the difference is much more visible in block 2 and 3, since they have states with different replenishing rates and the factor of b helps agent in understanding the corresponding bias. See Appendix for the Choice Recency Module graphs for three blocks.

8 Future Directions and Conclusions

We implemented various algorithms to find the best policy to play these foraging environments. Many of these algorithms give results which are very close to each other. To try to understand and mimic human learning we included the WM module which quickly learns the rewarding and fast replenishing bushes. This module helped us drastically reduce our training time but suffered drawback as real human have a very limited capacity of working memory. However if we want to mimic real human behaviour even more closely we should also add a randomness error in the q value estimates as they will also not be stored perfectly for a real human.

9 Member Contributions

All team members believe that each member has contributed significantly to the project till now.

Member	**Contribution**
Abhay	• Baseline algorithm • Implemented Choice based RL model • Literature review, Presentation & Report
Ankur Gupta	• Environment rendering • Implemented Deep RL models • Literature review, Presentation & Report
Anshul Rai	• Environment class code • Implemented WM model • Literature review, Presentation & Report
Archit Bansal	• Environment testing • Tabular RL methods and experimentation • Literature review, Presentation & Report

Table 2: Contribution of each member in the project

References

Eric L Charnov. Optimal foraging, the marginal value theorem. *Theoretical population biology*, 9(2):129–136, 1976.

Anne GE Collins and Michael J Frank. How much of reinforcement learning is working memory, not reinforcement learning? a behavioral, computational, and neurogenetic analysis. *European Journal of Neuroscience*, 35(7): 1024–1035, 2012.

Brian Lau and Paul W Glimcher. Dynamic response-by-response models of matching behavior in rhesus monkeys. *Journal of the experimental analysis of behavior*, 84(3):555–579, 2005.

Tom Schönberg, Nathaniel D Daw, Daphna Joel, and John P O'Doherty. Reinforcement learning signals in the human striatum distinguish learners from nonlearners during reward-based decision making. *Journal of Neuroscience*, 27(47):12860–12867, 2007.

Appendix

A Experimental Details

A.1 General Parameters :

Sarsa

- α : 0.5 to 0.01 linear decay in 3*noEpisodes//4
- ϵ : 1 to 0.01 linear decay in 19*noEpisodes//20 (decayEpsilon)
- gamma : 0.4
- τ : 1000 to 10 exponential decay (SoftMax)
- c : 15 (UCB)

Q Learning and Double Q Learning

- α : 0.05 to 0.01 linear decay in noEpisodes
- ϵ : 1 to 0.01 linear decay in 9*noEpisodes//10 (decayEpsilon)
- gamma : 0.4
- τ : 1000 to 10 exponential decay (SoftMax)
- c : 1(Block 1 and Block 3) and 3(Block 2) (UCB)

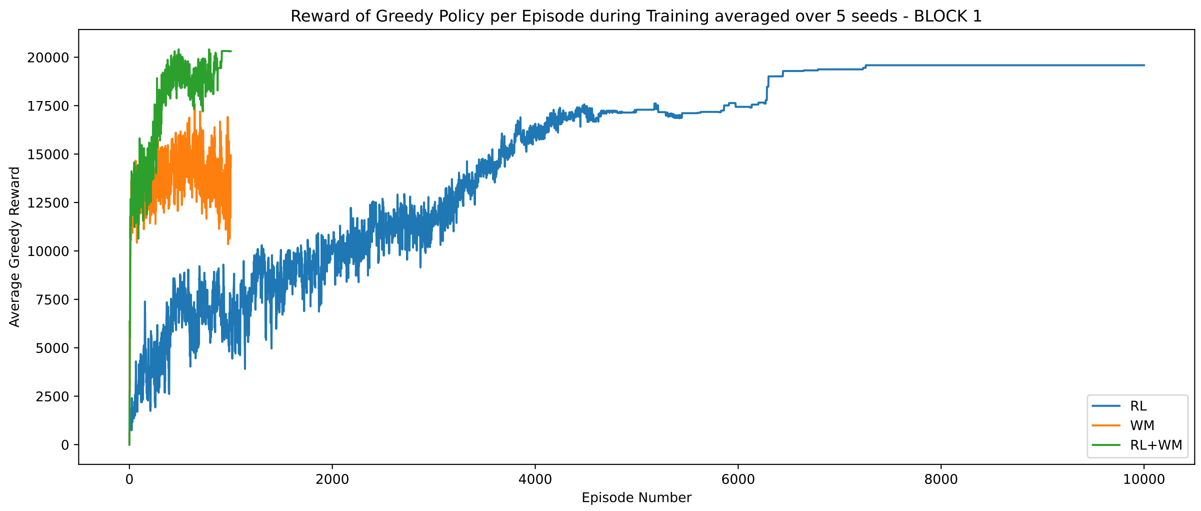
DeepQN

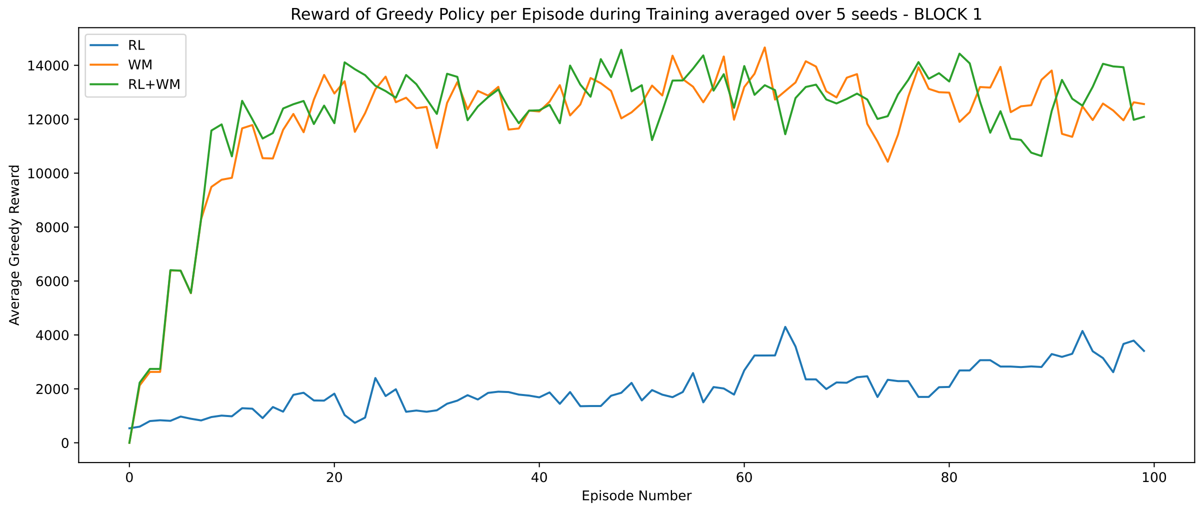
- lr : 1e-4
- Batch_size : 128
- Memory_size : 10000
- Optimizer : Adam
- Output Update : 4

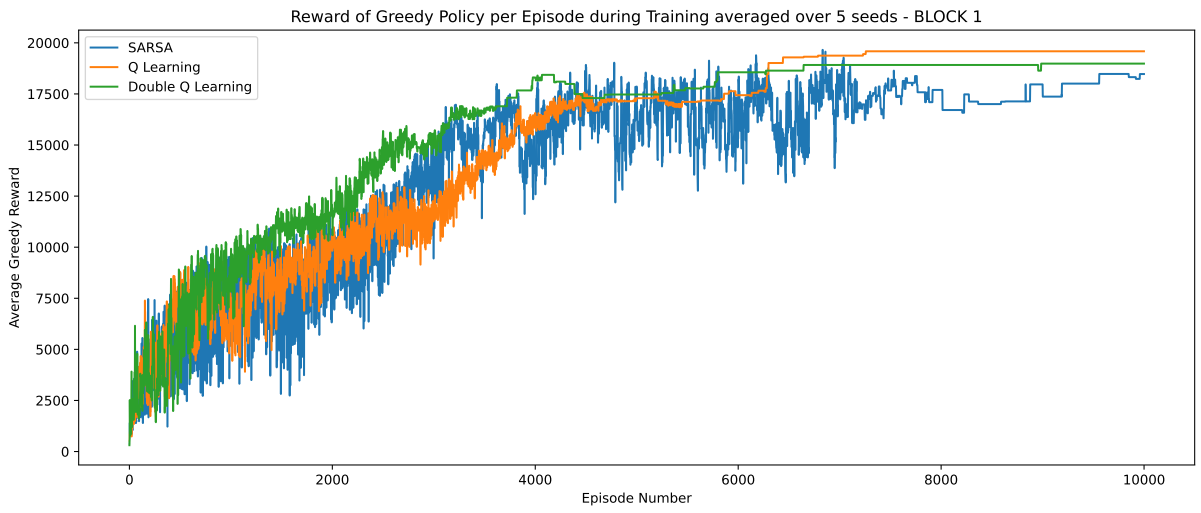
B Extra Plots

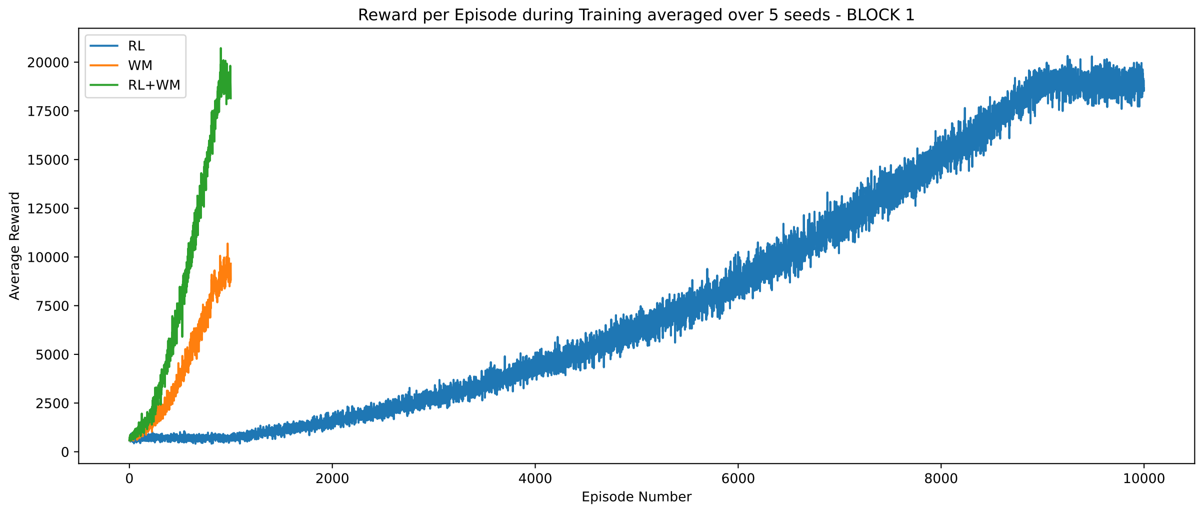
B.1 Tabular Method AND WM

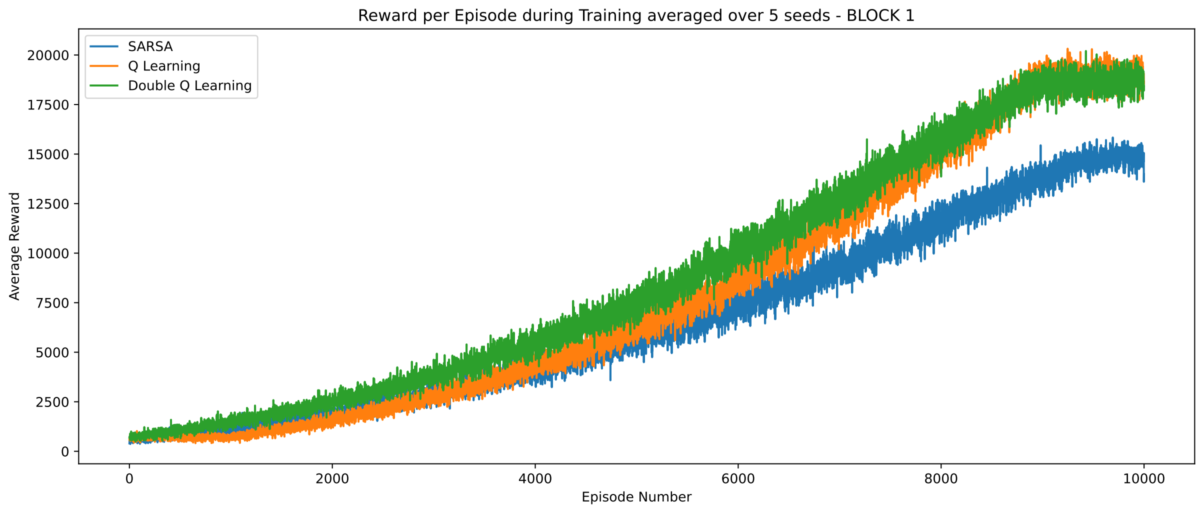
B.1.1 Block1

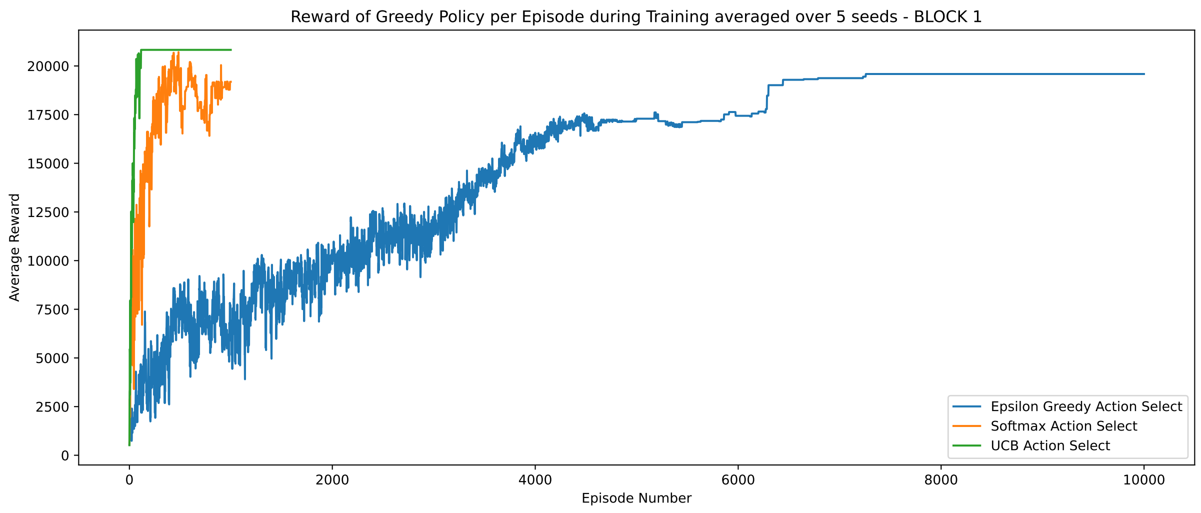


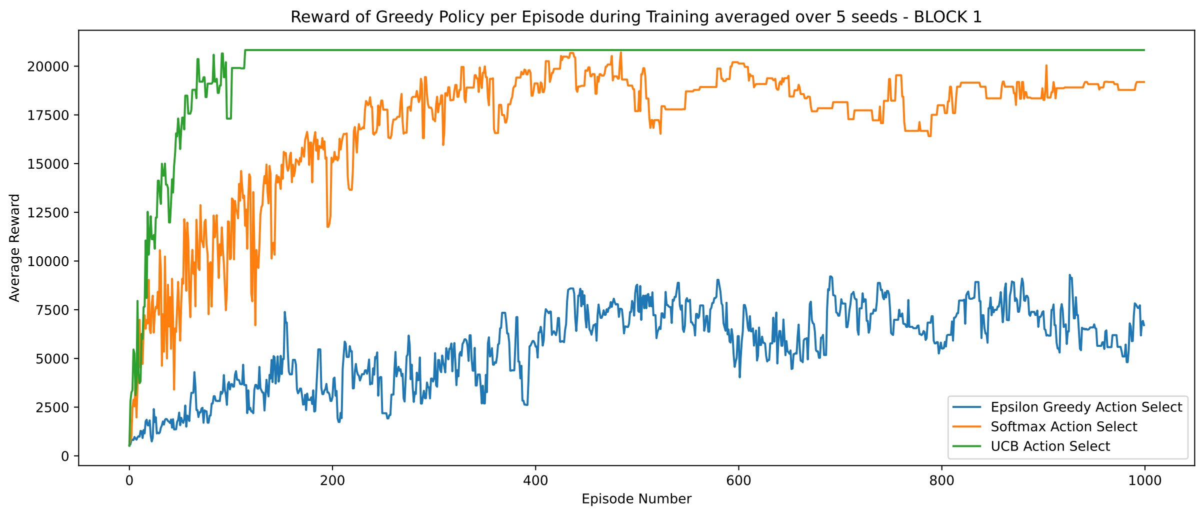



Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 1
SARSA
Q Learning
Double Q Learning
Average Greedy Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

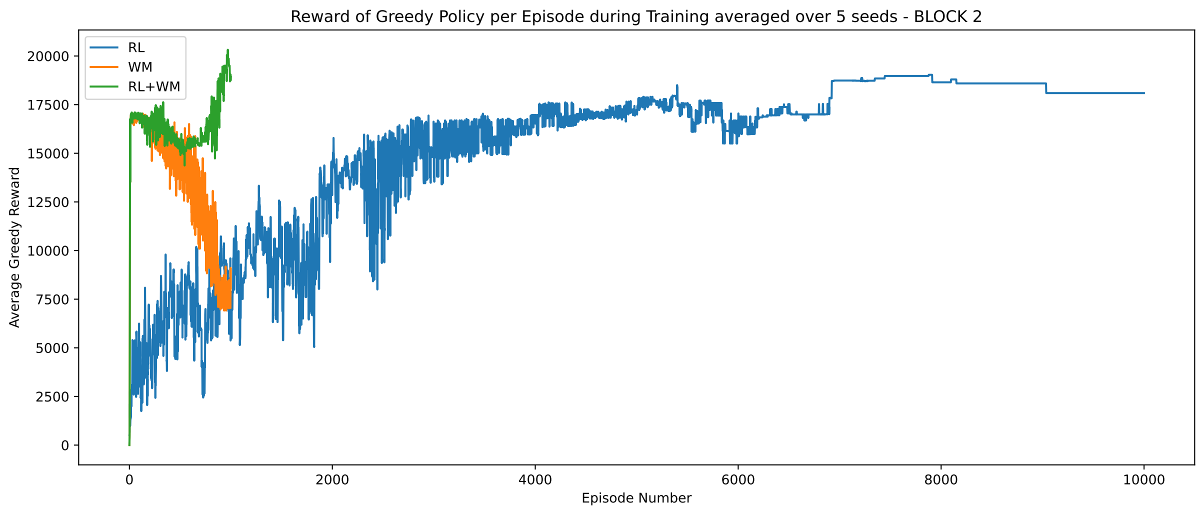

Reward per Episode during Training averaged over 5 seeds - BLOCK 1
RL
WM
RL+WM
Average Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

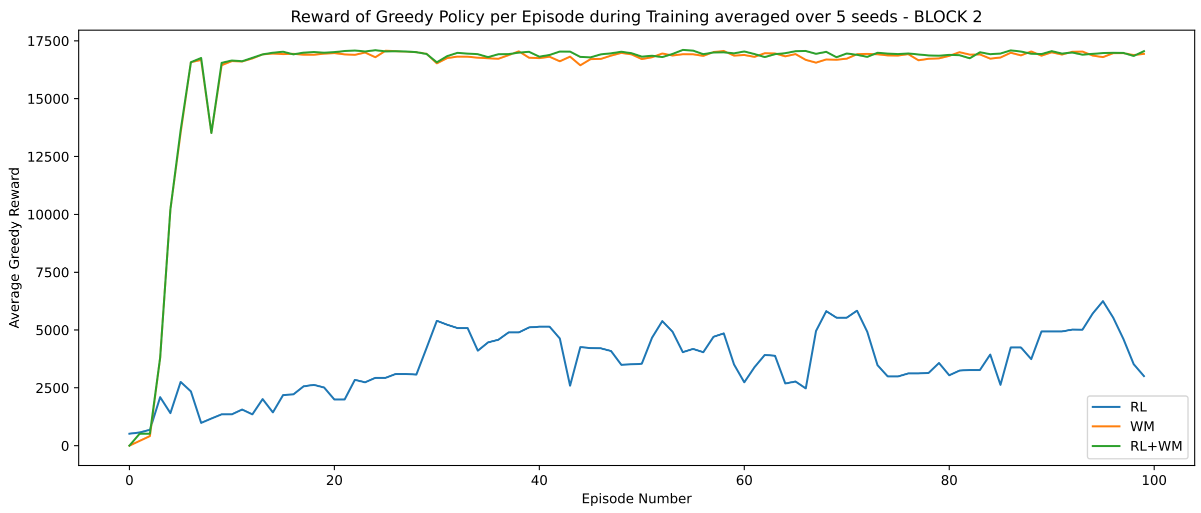

Reward per Episode during Training averaged over 5 seeds - BLOCK 1
SARSA
Q Learning
Double Q Learning
Average Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

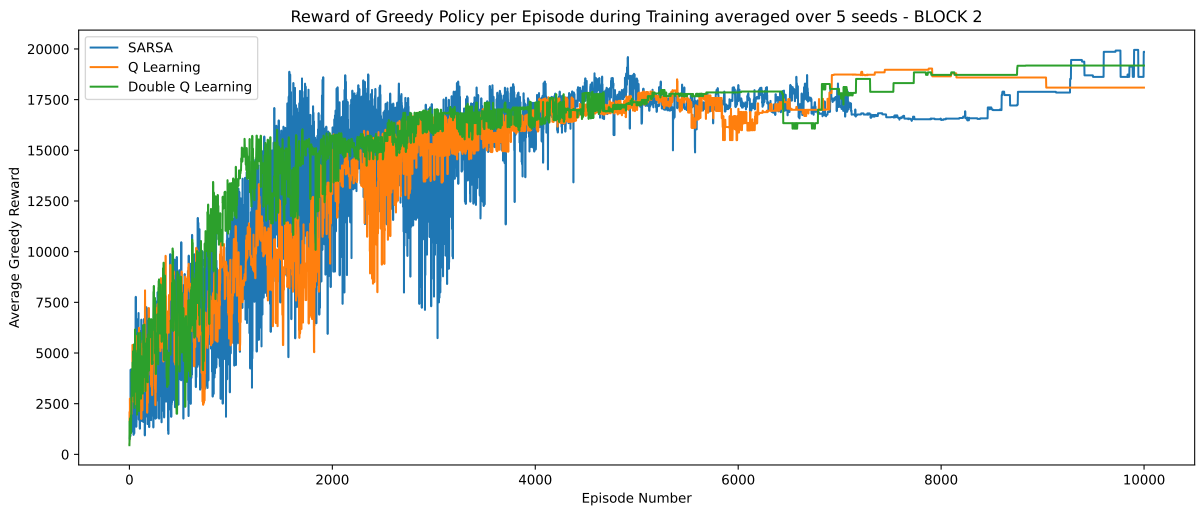

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 1
Average Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select

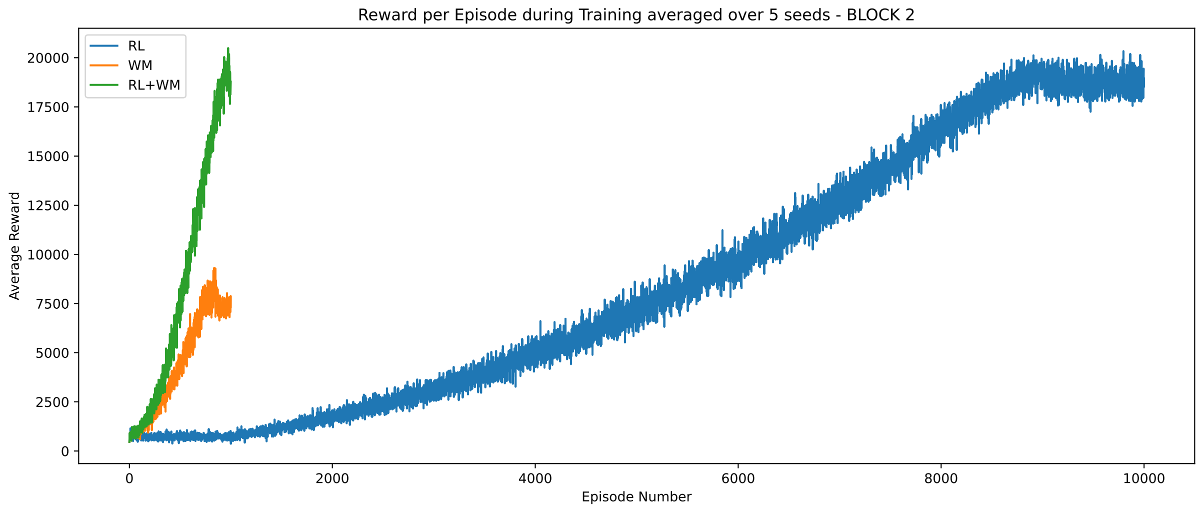

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 1
Average Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select

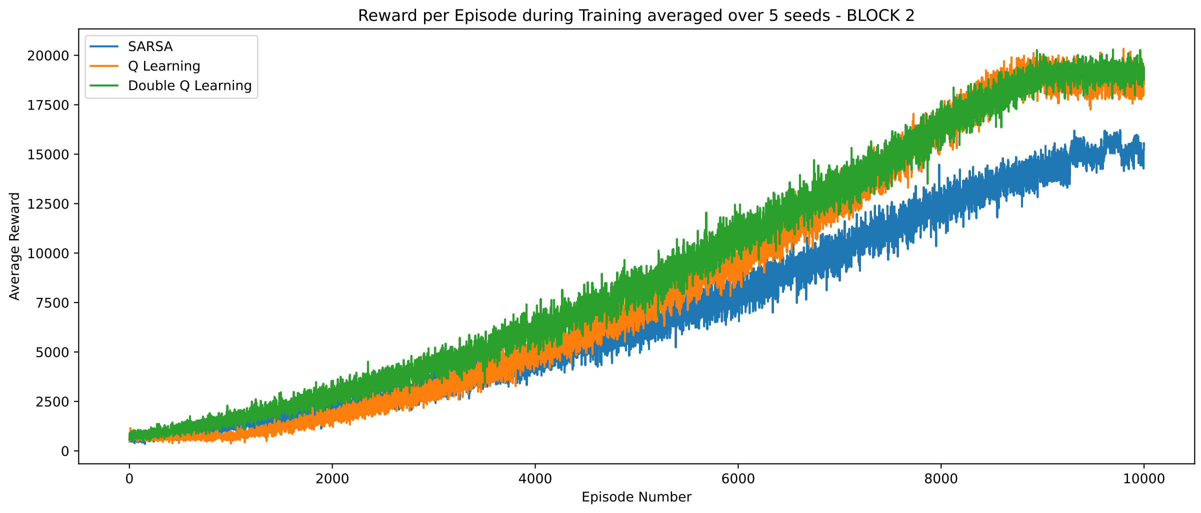
B.1.2 Block2

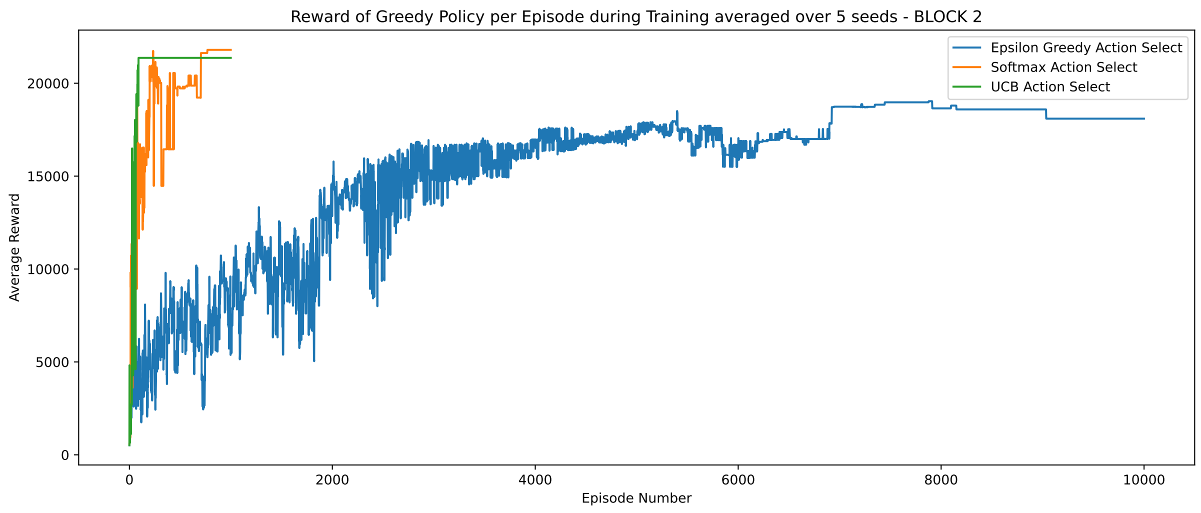

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
RL
WM
RL+WM
Average Greedy Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

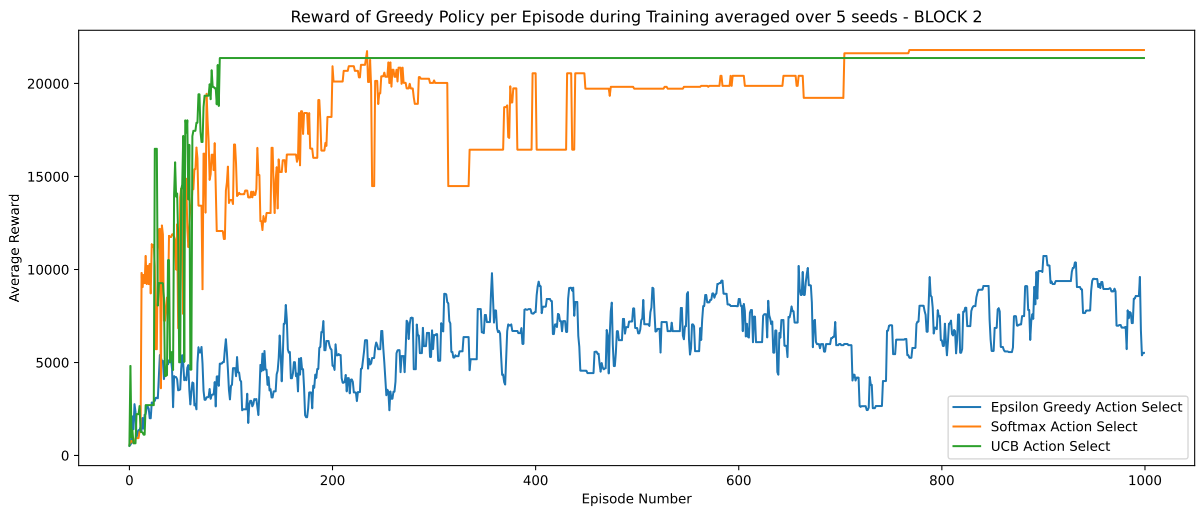

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
RL
WM
RL+WM
Average Greedy Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
0
20
40
60
80
100


Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
SARSA
Q Learning
Double Q Learning
Average Greedy Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

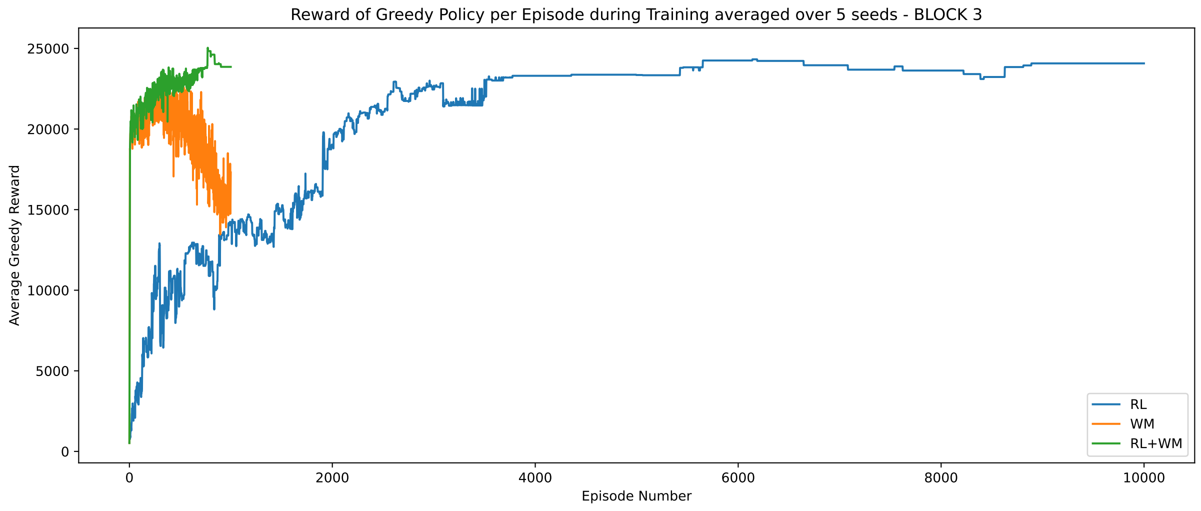

Reward per Episode during Training averaged over 5 seeds - BLOCK 2
RL
WM
RL+WM
Average Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

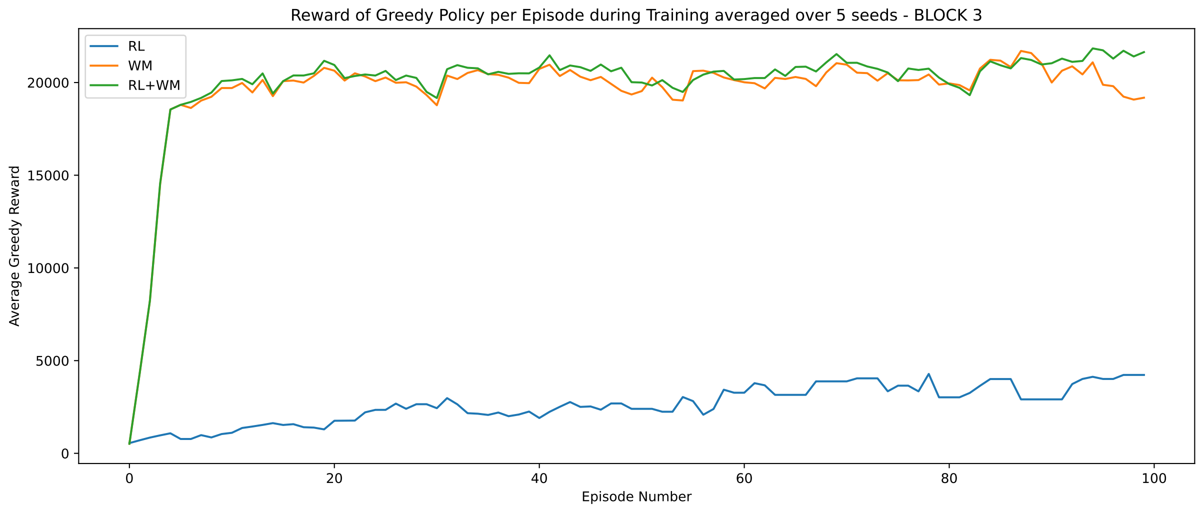

Reward per Episode during Training averaged over 5 seeds - BLOCK 2
SARSA
Q Learning
Double Q Learning
Average Reward
Episode Number
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

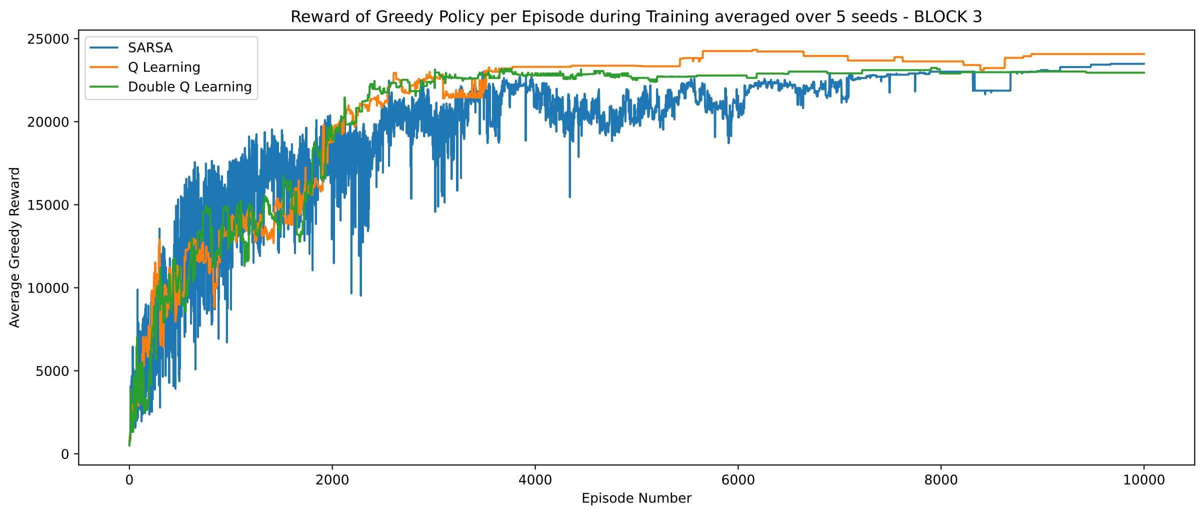

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
Average Reward
Episode Number
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select
0
5000
10000
15000
20000
0
2000
4000
6000
8000
10000

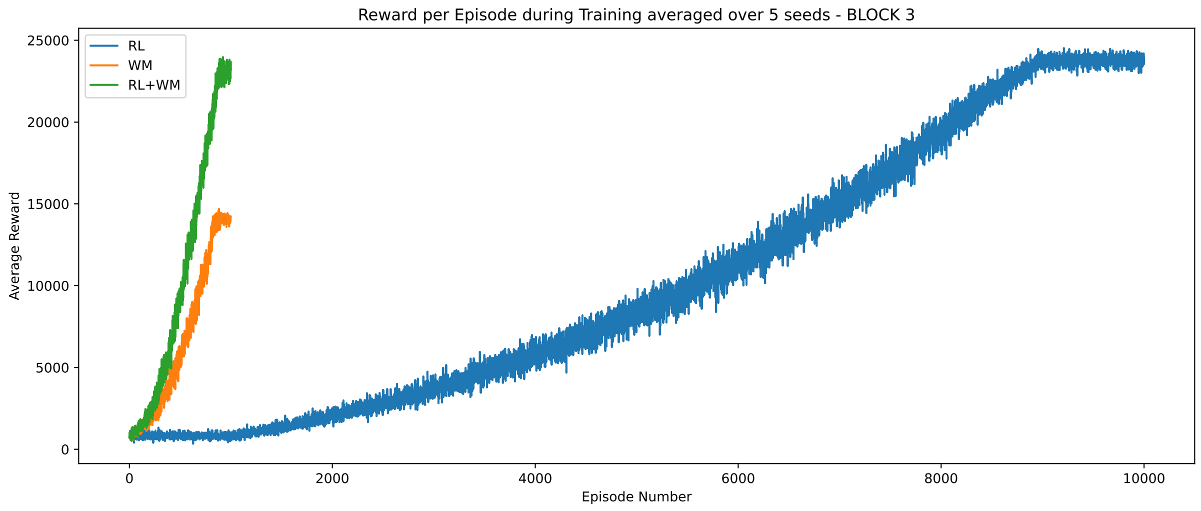

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
Average Reward
Episode Number
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select
0
5000
10000
15000
20000
0
200
400
600
800
1000

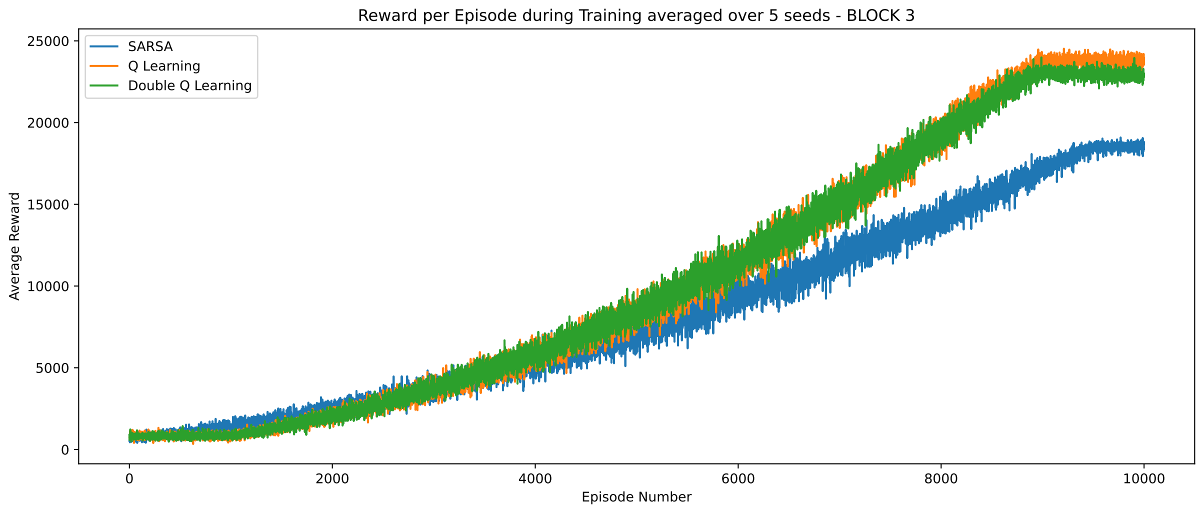
B.1.3 Block3

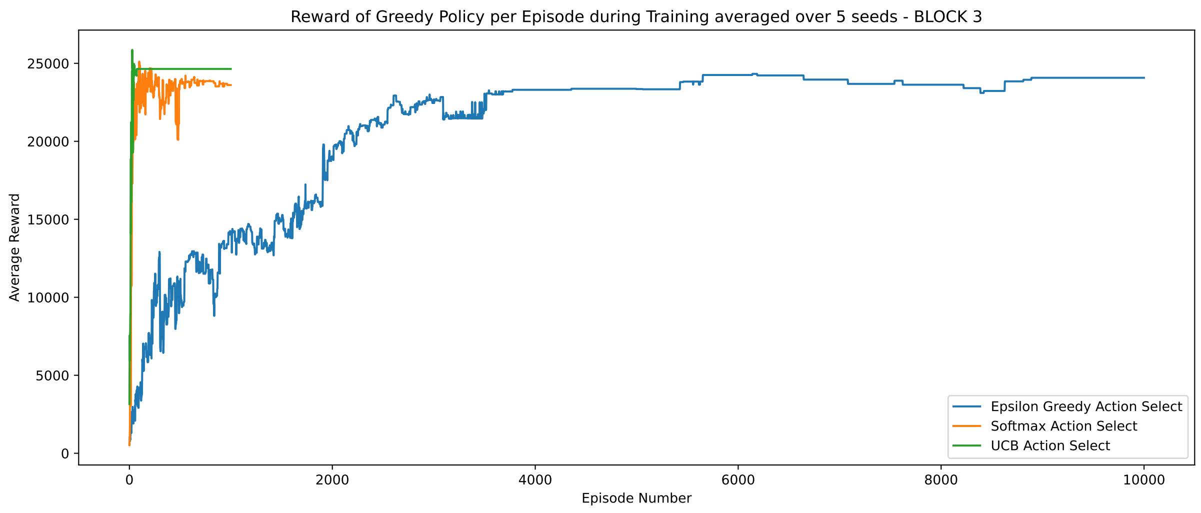

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Average Greedy Reward
0
5000
10000
15000
20000
25000
Episode Number
0
2000
4000
6000
8000
10000
RL
WM
RL+WM

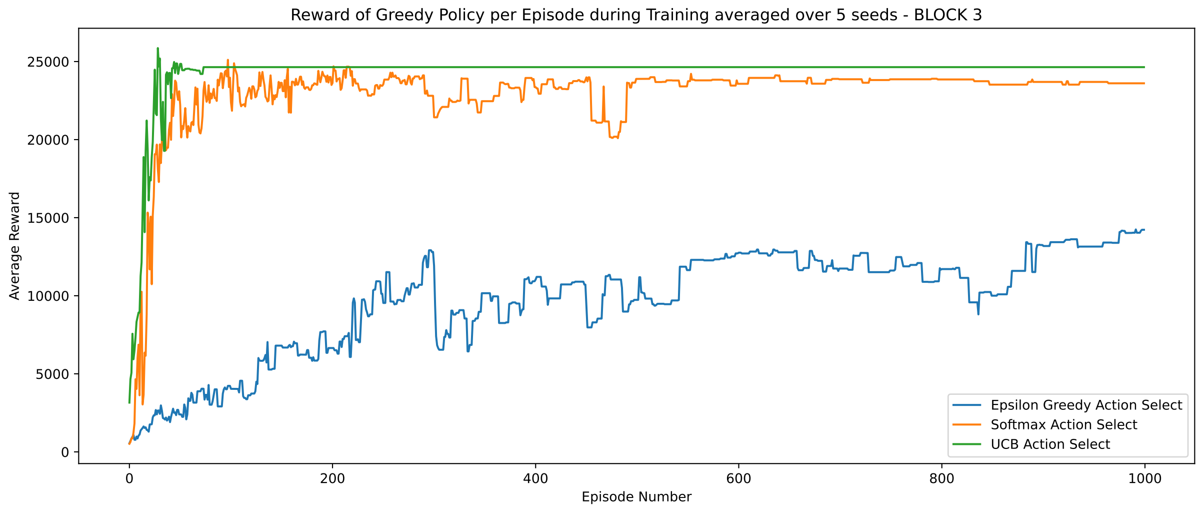

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Average Greedy Reward
0
5000
10000
15000
20000
Episode Number
0
20
40
60
80
100
RL
WM
RL+WM


Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
SARSA
Q Learning
Double Q Learning
Average Greedy Reward
Episode Number
0
5000
10000
15000
20000
25000
0
2000
4000
6000
8000
10000

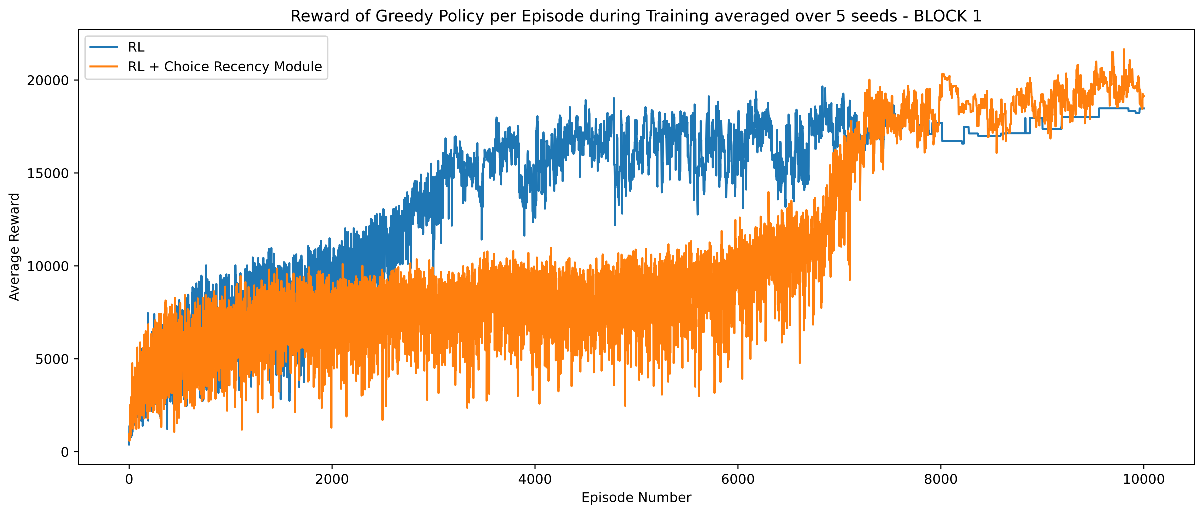

Reward per Episode during Training averaged over 5 seeds - BLOCK 3
RL
WM
RL+WM
Average Reward
Episode Number
0
5000
10000
15000
20000
25000
0
2000
4000
6000
8000
10000

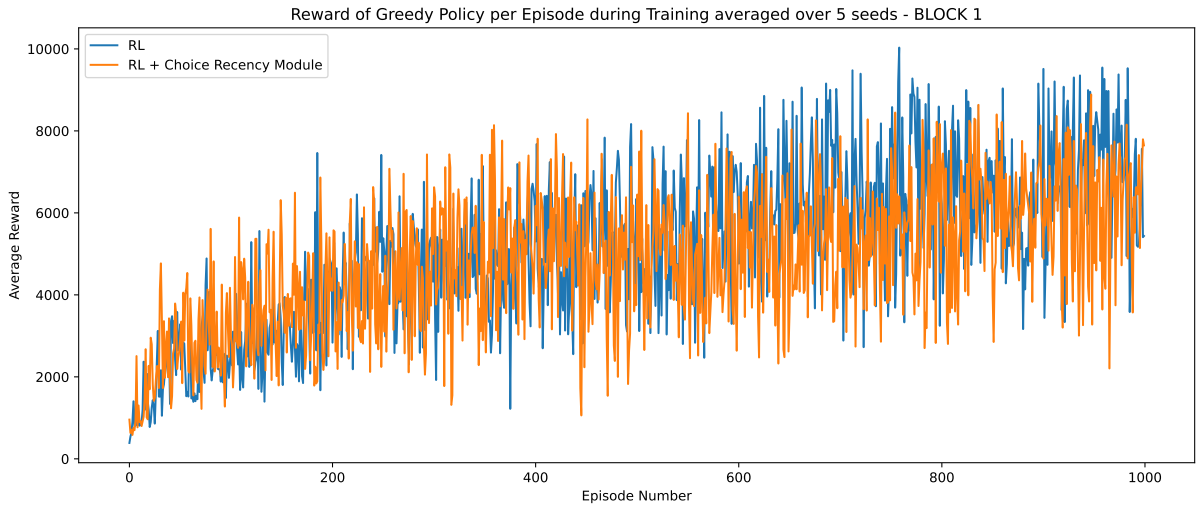

Reward per Episode during Training averaged over 5 seeds - BLOCK 3
SARSA
Q Learning
Double Q Learning
Average Reward
Episode Number
0
5000
10000
15000
20000
25000
0
2000
4000
6000
8000
10000

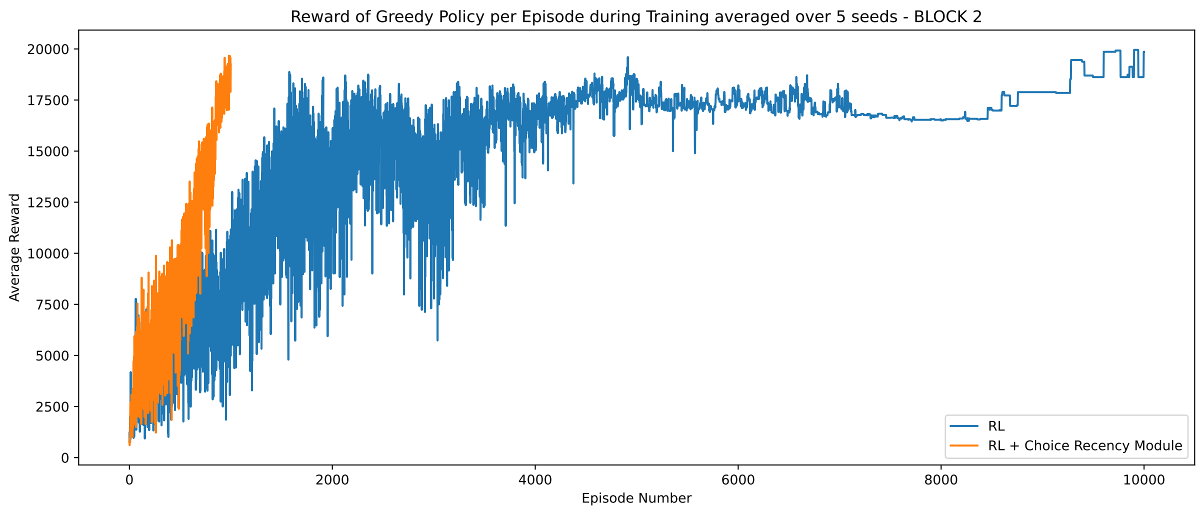

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Average Reward
Episode Number
0
5000
10000
15000
20000
25000
0
2000
4000
6000
8000
10000
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select

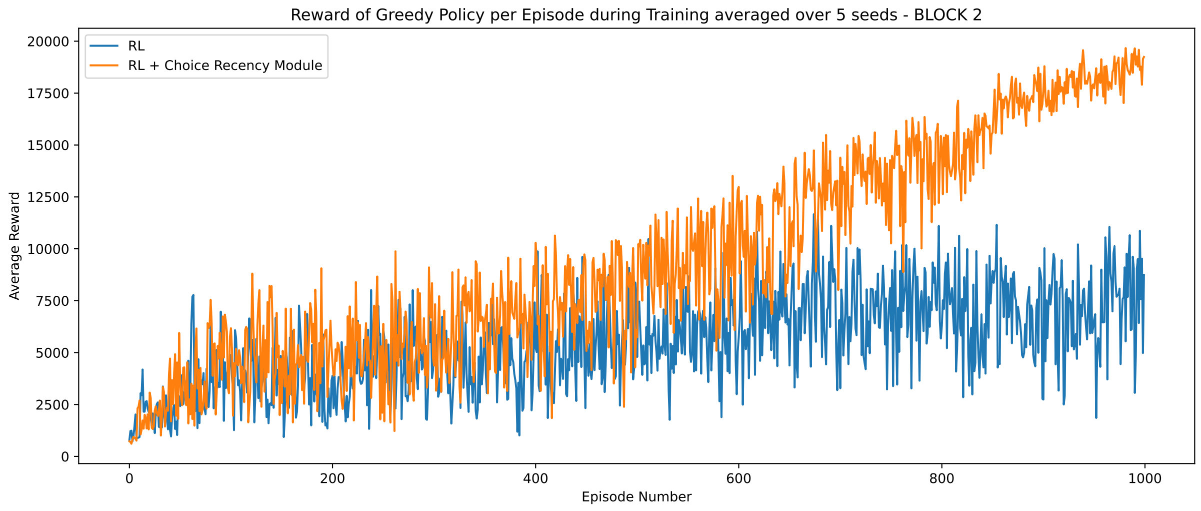

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Average Reward
Episode Number
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select

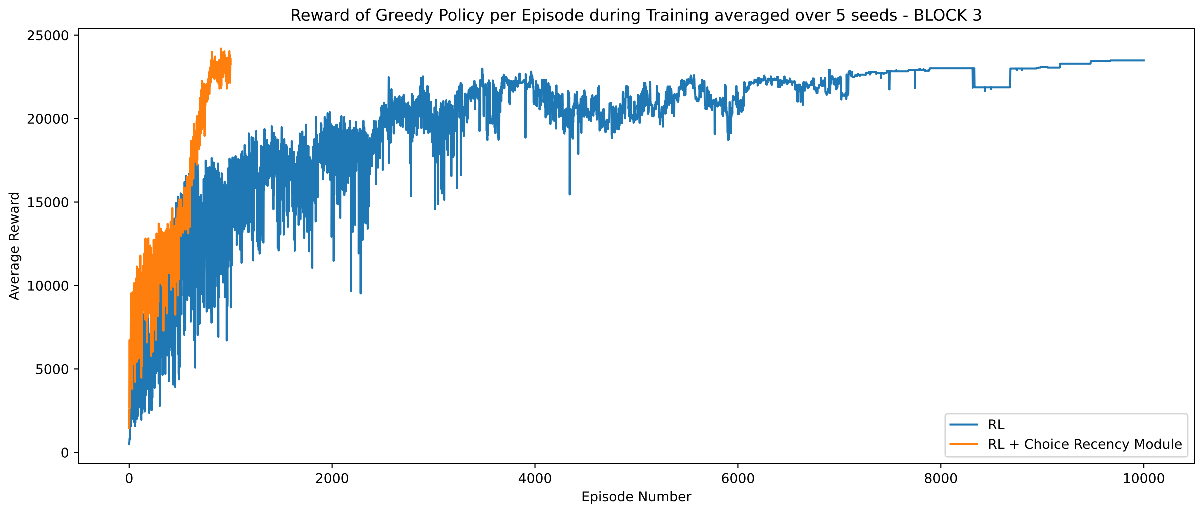
B.2 Choice Recency Model

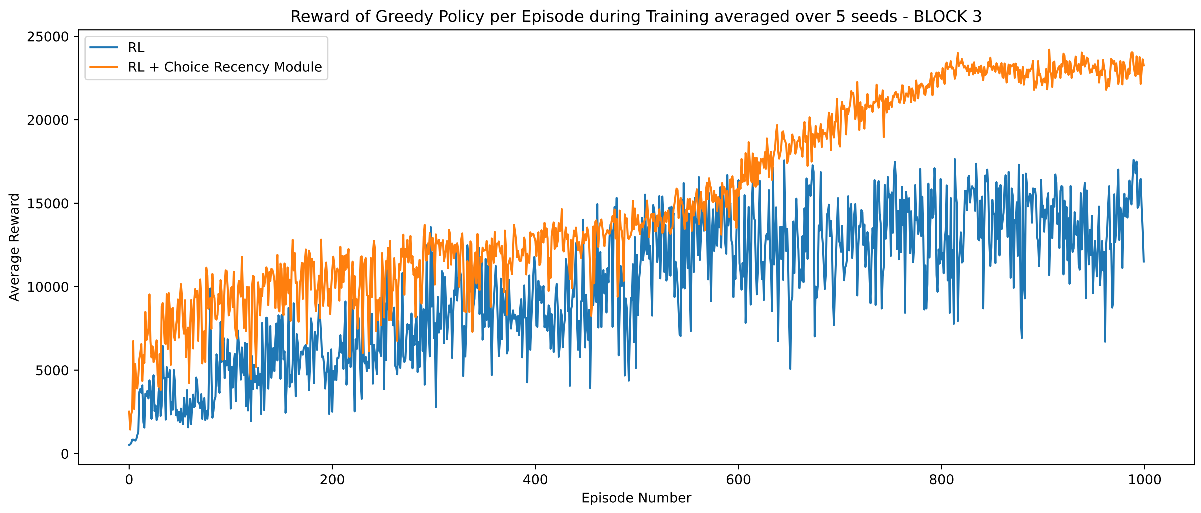


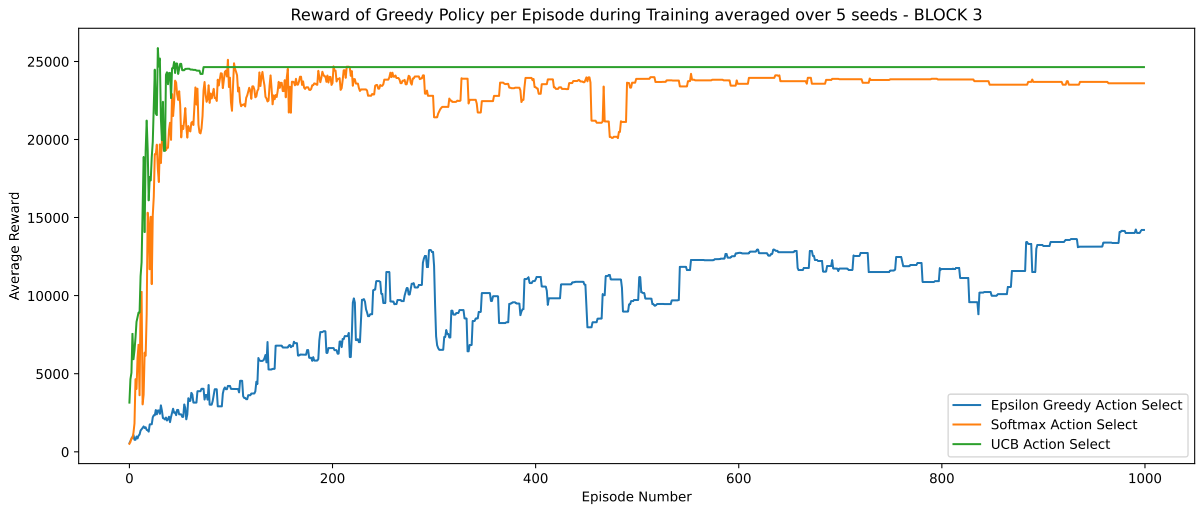



Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
Average Reward
Episode Number
RL
RL + Choice Recency Module
0
2500
5000
7500
10000
12500
15000
17500
20000
0
2000
4000
6000
8000
10000

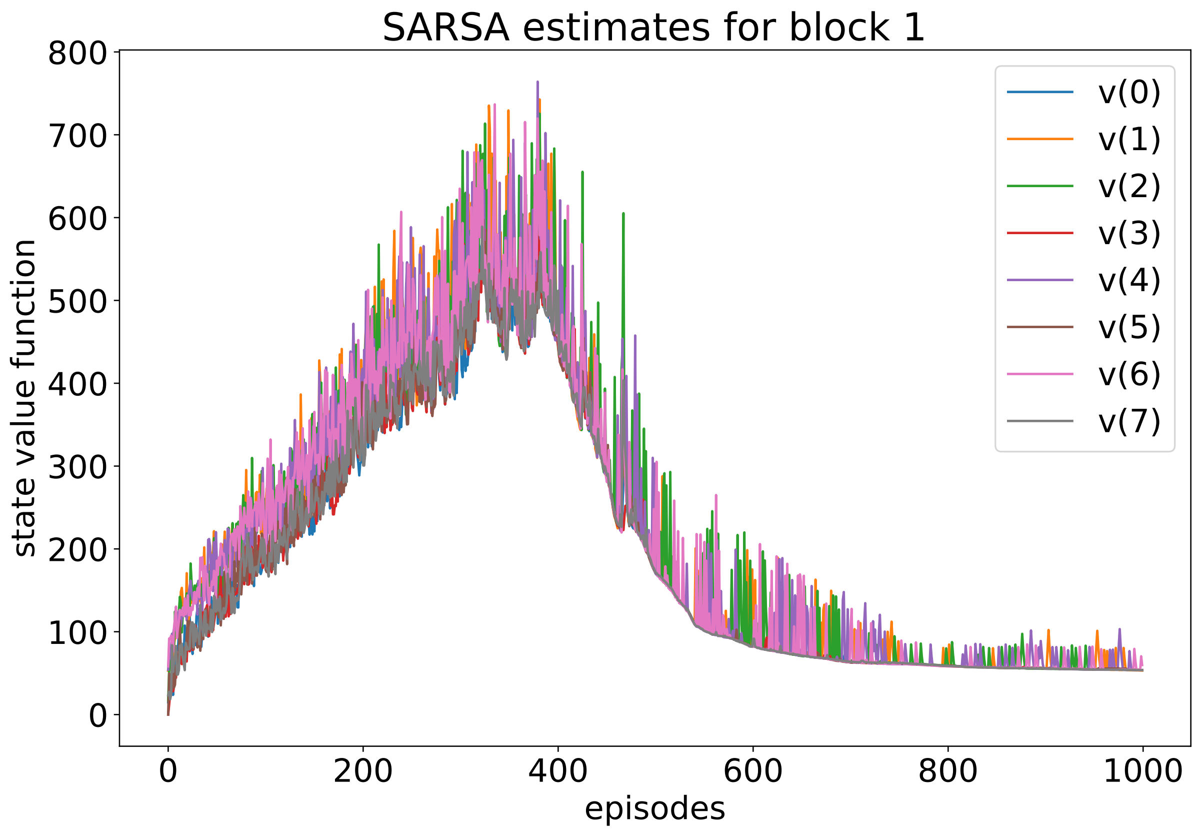

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
Average Reward
Episode Number
RL
RL + Choice Recency Module
0
2500
5000
7500
10000
12500
15000
17500
20000
0
200
400
600
800
1000

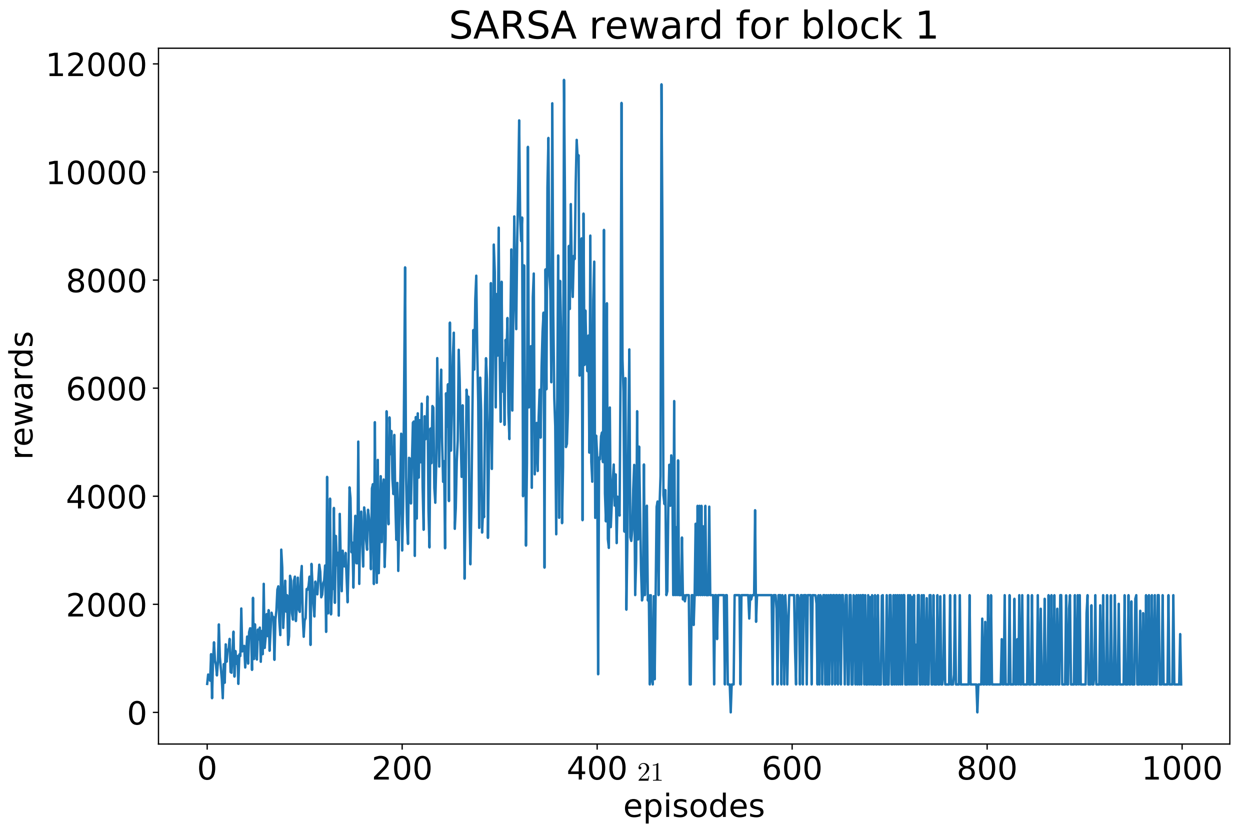

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Average Reward
Episode Number
RL
RL + Choice Recency Module
0
5000
10000
15000
20000
25000
0
2000
4000
6000
8000
10000


Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
RL
RL + Choice Recency Module
Average Reward
Episode Number
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000

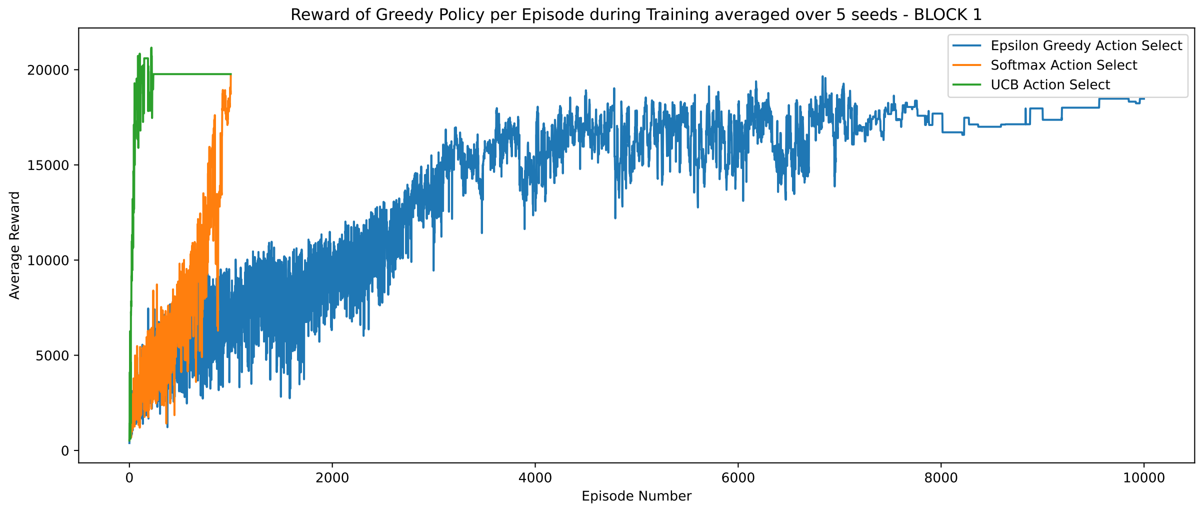

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select
Average Reward
Episode Number
0
5000
10000
15000
20000
25000
0
200
400
600
800
1000

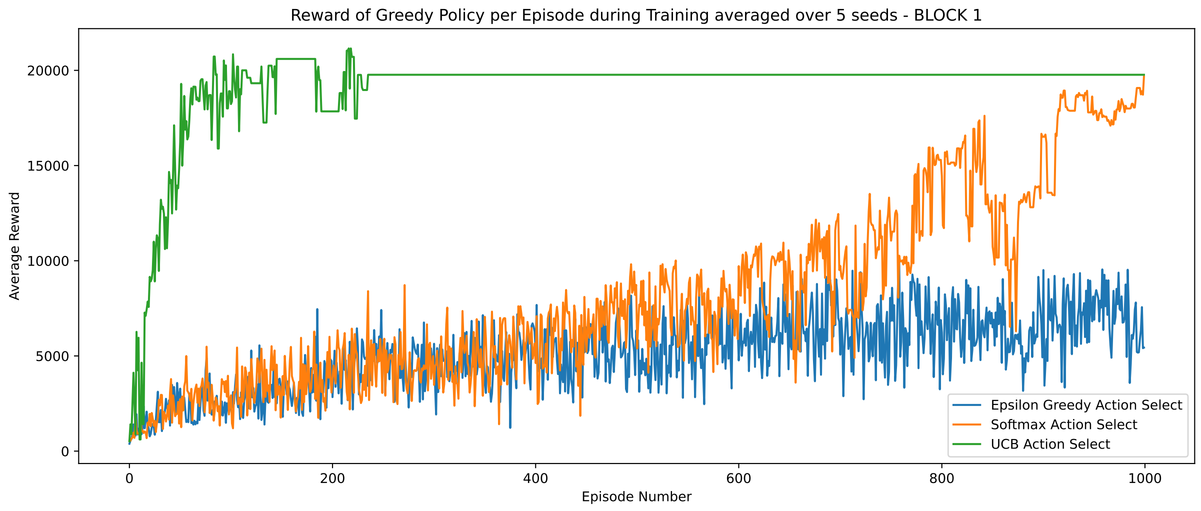
B.3 Initial Baseline

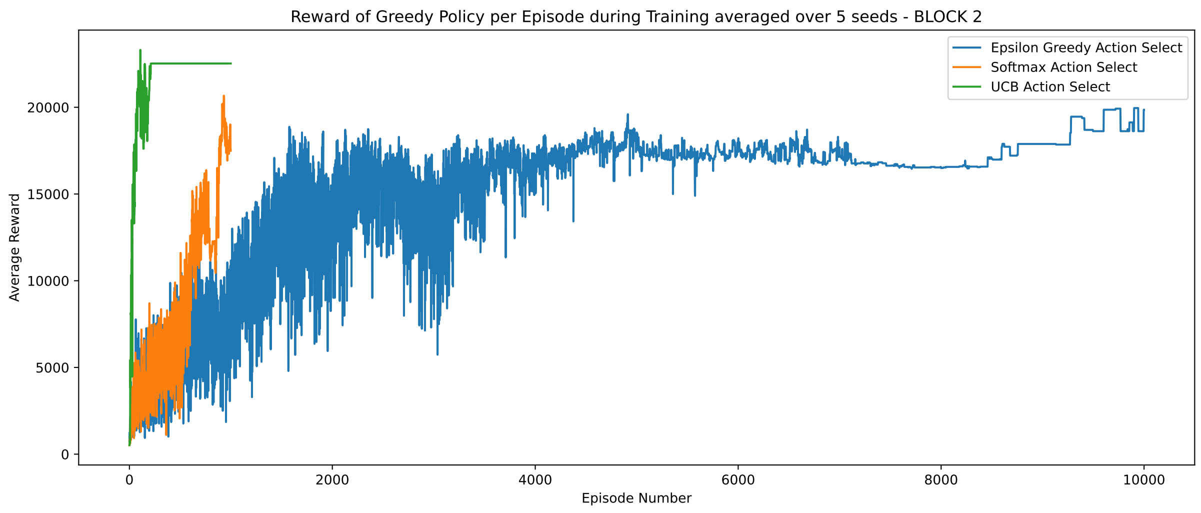


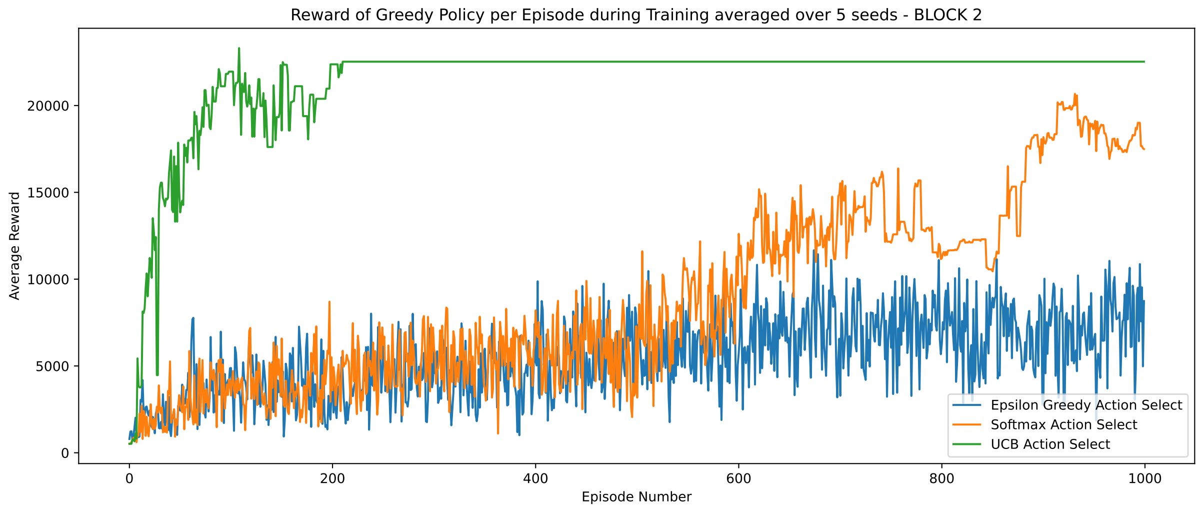


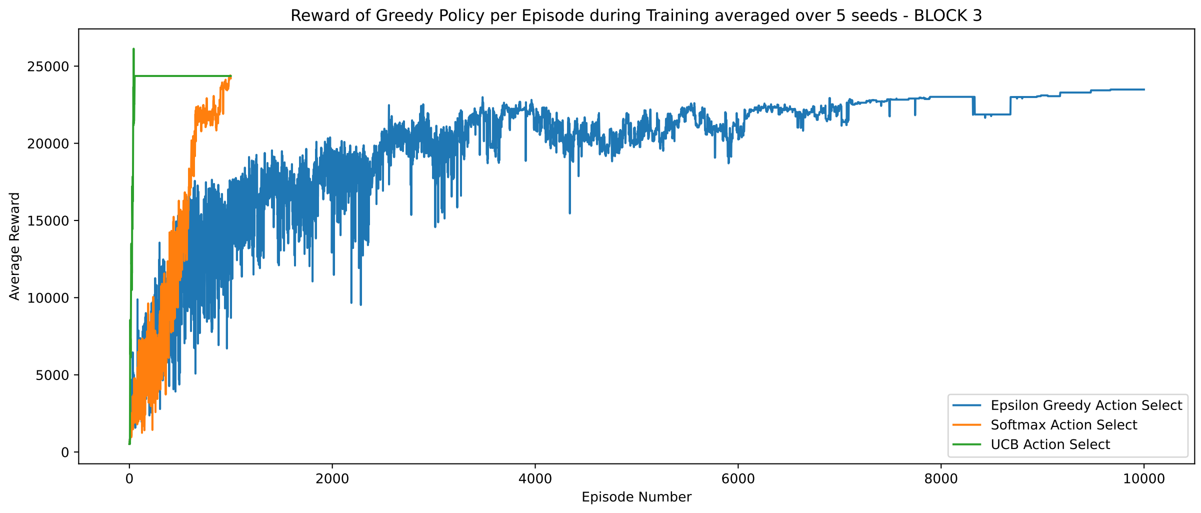
B.4 SARSA TABULAR Methods

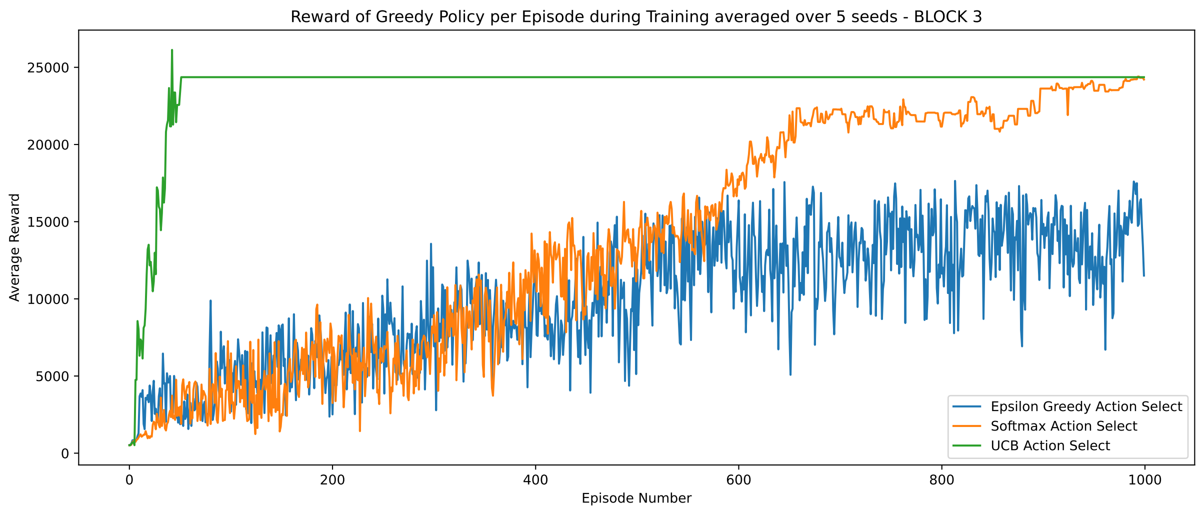




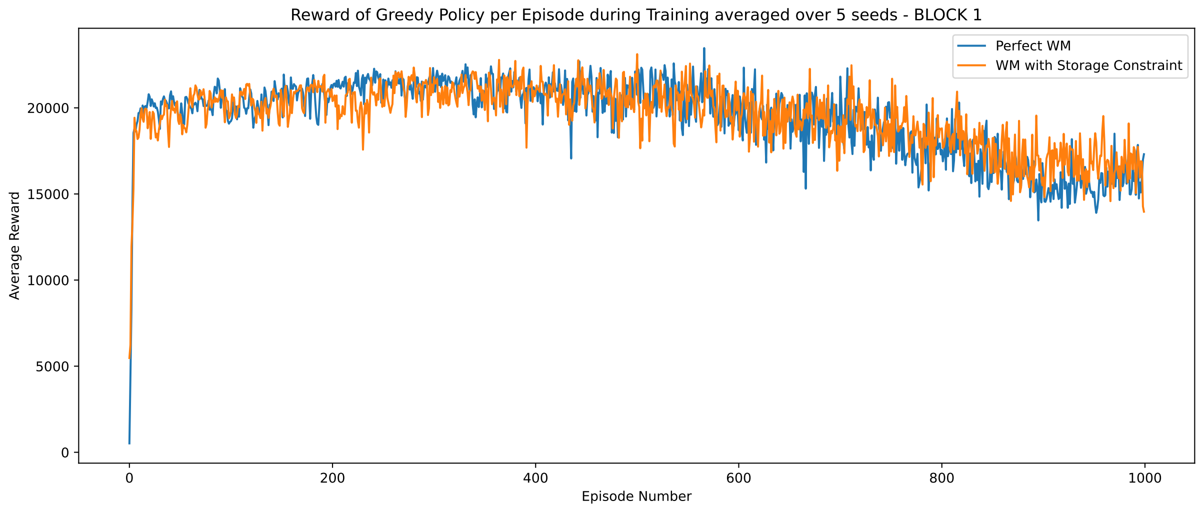

Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select
Average Reward
Episode Number


Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 2
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select
Average Reward
Episode Number


Reward of Greedy Policy per Episode during Training averaged over 5 seeds - BLOCK 3
Epsilon Greedy Action Select
Softmax Action Select
UCB Action Select
Average Reward
Episode Number



B.5 Perfect VS Limited WM



C More Information, etc.