RL homework 3

Name: Eric Hambro

SN: 17096751

Start date: *7th March 2018*

Due date: *21st March 2018, 11:55 pm*

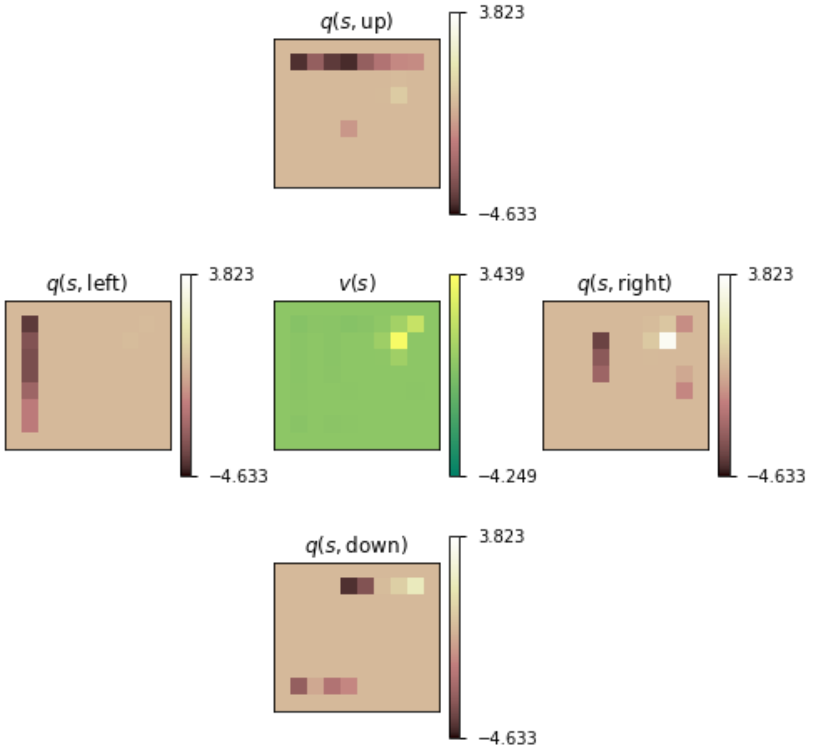
In [2]:
```python
from IPython.display import Image
```

Data Efficiency Plots

In [10]:
```python
print("ONLINE Q - DATA EFFICIENCY")
Image('./online_1.png')
```

```
ONLINE Q - DATA EFFICIENCY
```

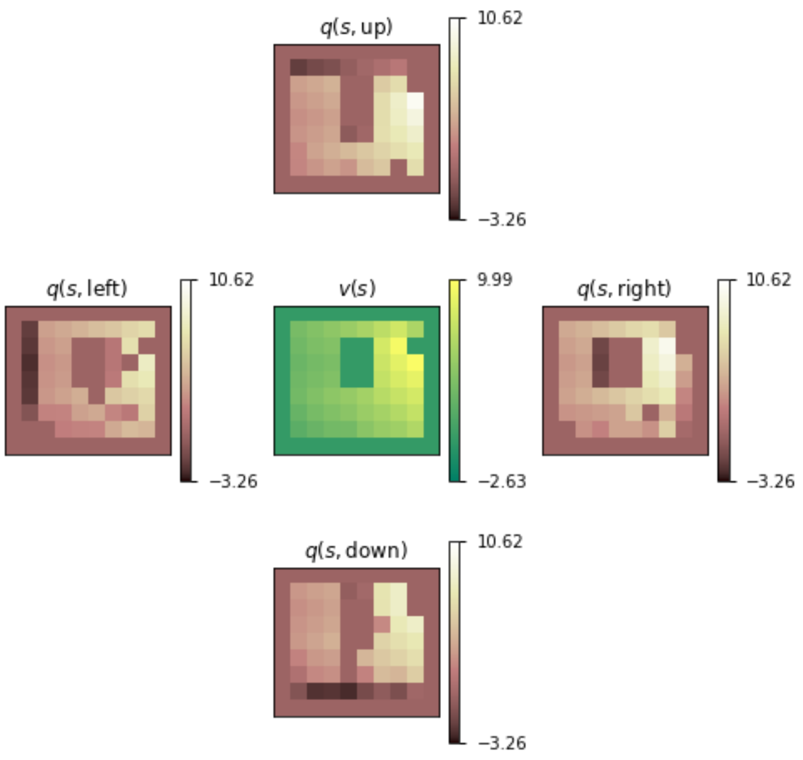
Out[10]:



In [11]:
```python
print("EXPERIENCE REPLAY - DATA EFFICIENCY")
Image('./replay_1.png')
```

```
EXPERIENCE REPLAY - DATA EFFICIENCY
```

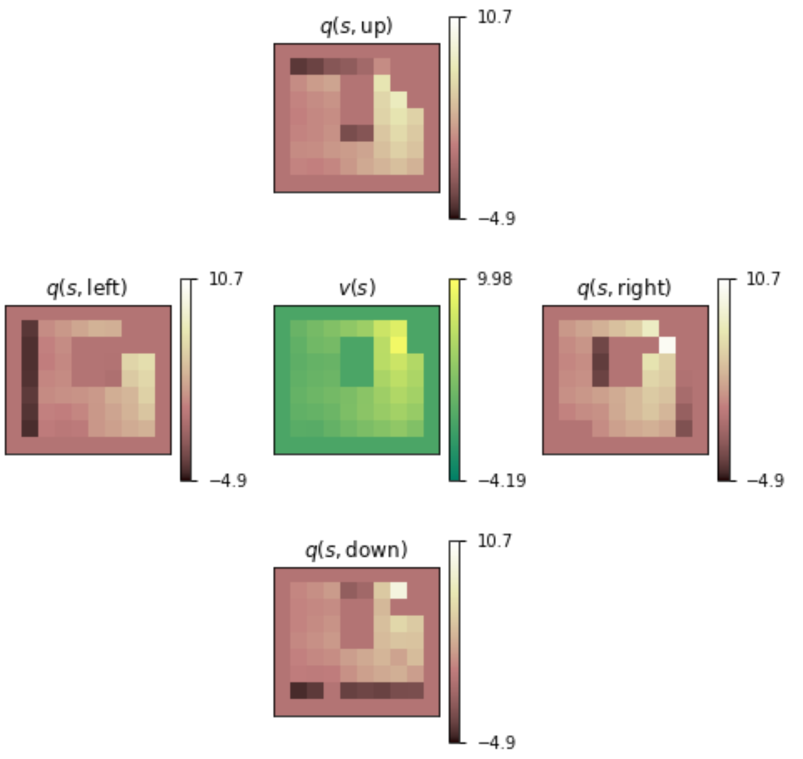
Out[11]:



In [12]:
```python
print("DYNA Q - DATA EFFICIENCY")
Image('./dyna_1.png')
```

```
DYNA Q - DATA EFFICIENCY
```

Out[12]:

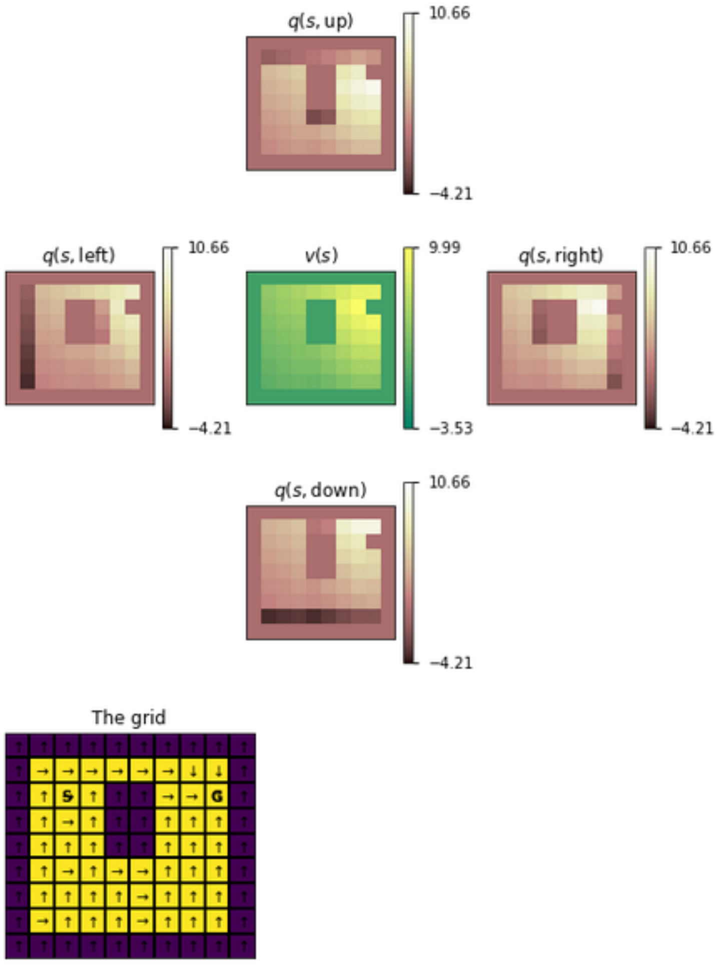


Computational Complexity Plots

```
In [14]: print("ONLINE Q - COMPUTATIONAL COMPLEXITY")
         Image('./online_2.png')
```

```
ONLINE Q - COMPUTATIONAL COMPLEXITY
```

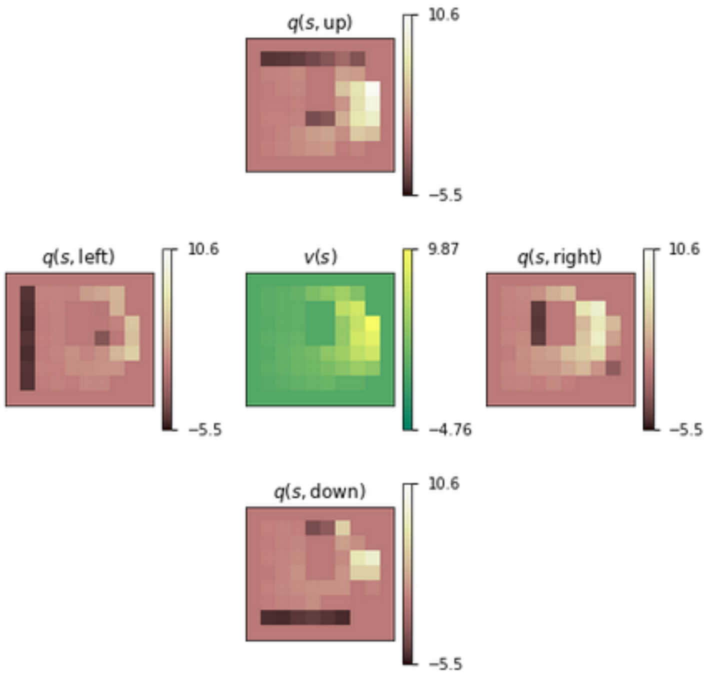
Out[14]:

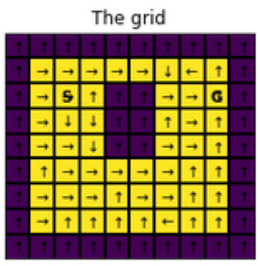


In [15]:
```
print("REPLAY - COMPUTATIONAL COMPLEXITY")
Image('./replay_2.png')
```

```
REPLAY - COMPUTATIONAL COMPLEXITY
```

Out[15]:

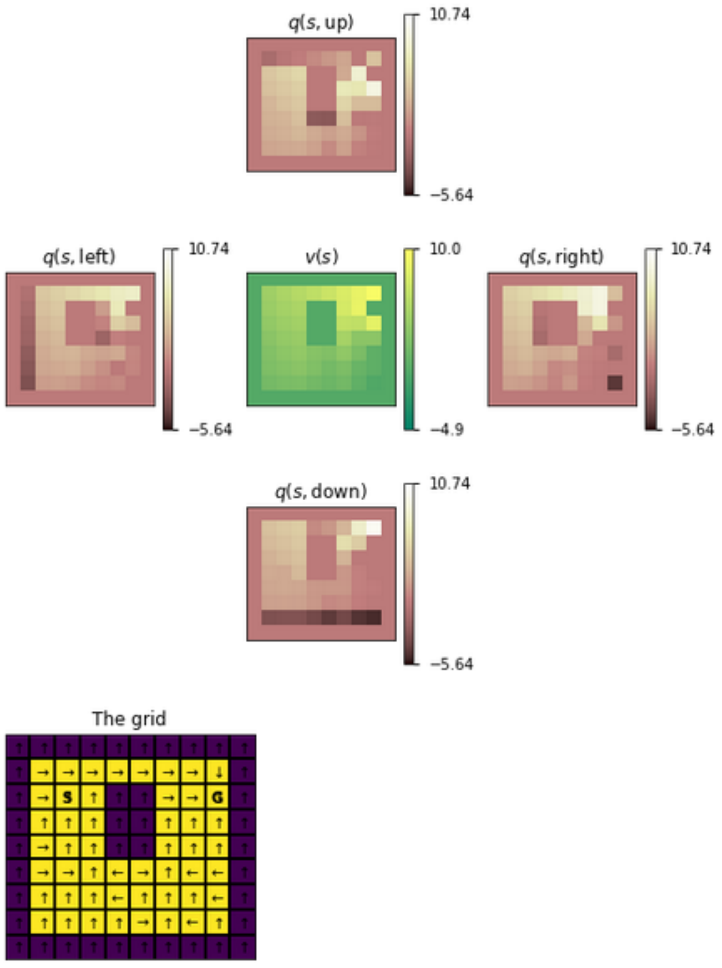




In [16]:
```
print("DYNA Q - COMPUTATIONAL COMPLEXITY")
Image('./dyna_2.png')
```

```
DYNA Q - COMPUTATIONAL COMPLEXITY
```

Out[16]:



Q1. Data Efficiency & Computational Complexity

[5 pts] Why is the ExperienceReplay agent so much more data efficient than online Q-learning?

Experience replay is more data efficient because events that have been experienced once can be reused to iteratively improve the Q value estimates (especially in this stationary setting). The replay buffer, allows the model to store previous experiences, and so the same piece of data can be reused (by sampling) to learn more efficiently, particularly in this stationary environment. The online Q learning has no such buffer, and can only learn by interacting with the environment - each interaction is only used once!

[5 pts] If we run the experiments for the same number of updates, rather than the same number of steps in the environment, which among online Q-learning and Experience Replay performs better? Why?

In this case, online Q performs better. This is because every update involves a new sample from the environment, compared to the replay, in which 1 in 31 updates is new from the environment. This allows online Q to effectively explore the space much more for the same amount of computation. This can be seen in both the plots, as the online q learning has much more "filled out" action values, compared to experience replay.

[5 pts] Which among online Q-learning and Dyna-Q is more data efficient? why?

For the same reasons above Dyna-Q is appears to be more data efficient. Although both models perform updates on the same "true experiences", Dyna-Q also uses its experiences to update an estimated model of the environment, and then to query the model repeatedly to update its q values.

In our current set up, in the limit of many samples, this model will tend to the correct model of the environment, and dyna-q will be able to effectively sample previous experiences, and perform updates based on these. Given our deterministic tabular model and stationary setting, in this limit, dyna-q will be much like experience replay. Of course, this depends on the model converging to the true model reasonably quickly. However, assuming the model is reasonably accurate (or not pathologically wrong in the beginning), dyna-q should be more data efficient than online q.

[5 pts] If we run the experiments for the same number of updates, rather than the same number of steps in the environment, which among online Q-learning and Dyna-Q performs better? Why?

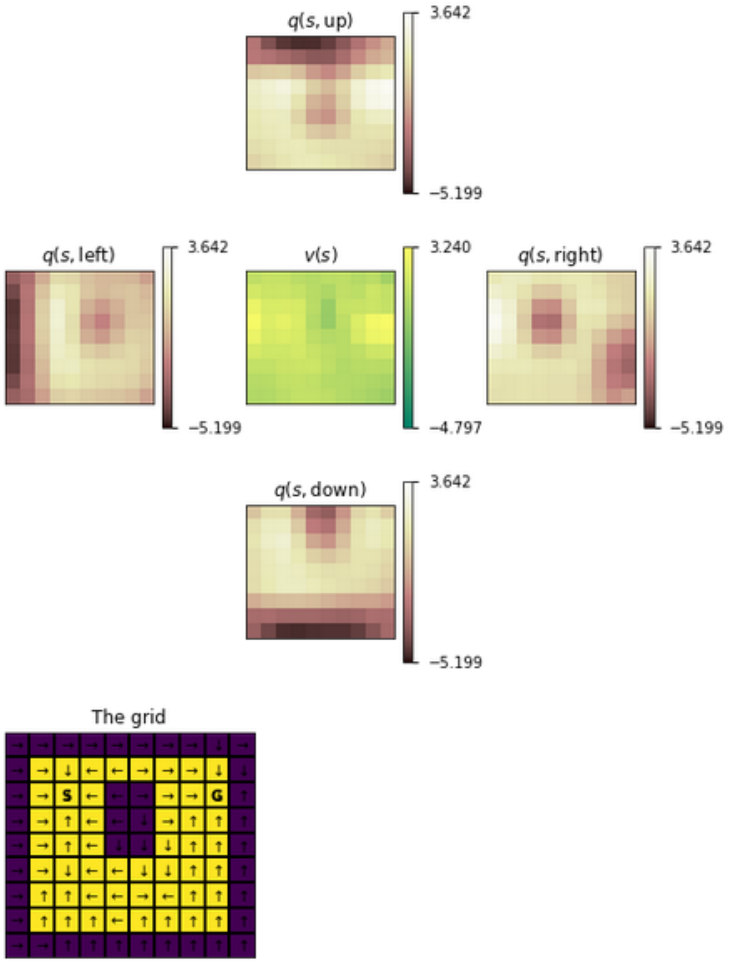
Once again, online-q should be more data efficient. In the limit of large exploration, with an accurate model and varied experiences in the buffer, these will be very similar, but initially, when the model is still being learnt and the experience buffer is empty dyna will perform worse. Every online Q update is based on a "true" experience, whereas only 1 in 31 of these is the case for Dyna-Q. The other 30 are approximations to the true experience that may be lead to less efficient updates, especially initially. For this reason online q performs better for the same number of updates.

Linear Function Approximation Plots

```
In [17]: print("ONLINE Q - FEATURE APPROXIMATION")
         Image('./online_feature.png')
```

```
ONLINE Q - FEATURE APPROXIMATION
```

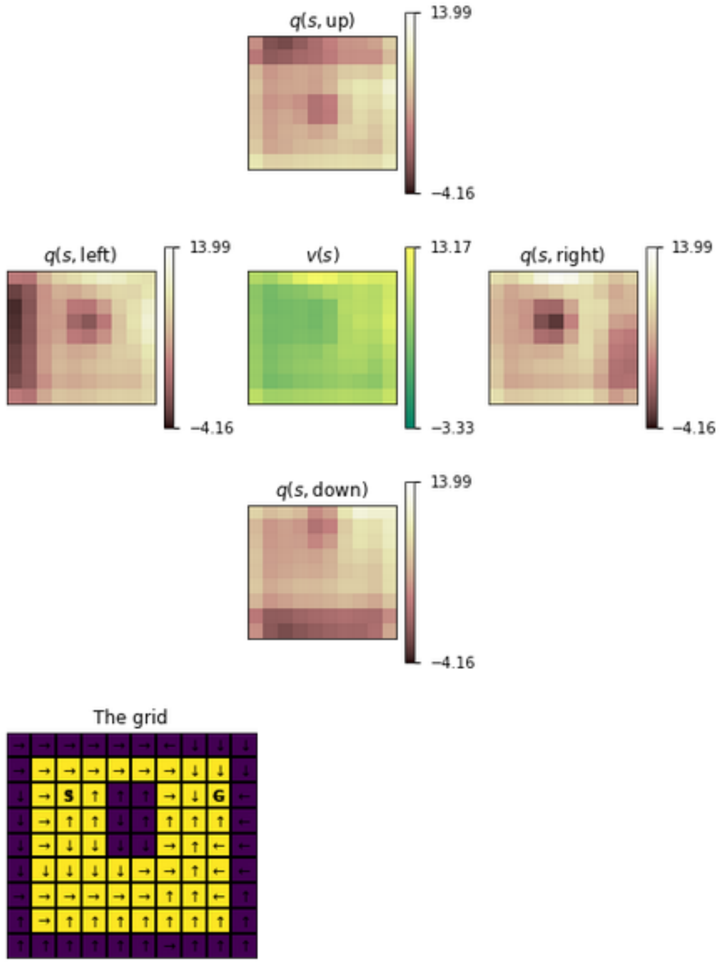
```
Out[17]:
```



In [18]:
```
print("REPLAY Q - FEATURE APPROXIMATION")
Image('./replay_feature.png')
```

```
REPLAY Q - FEATURE APPROXIMATION
```

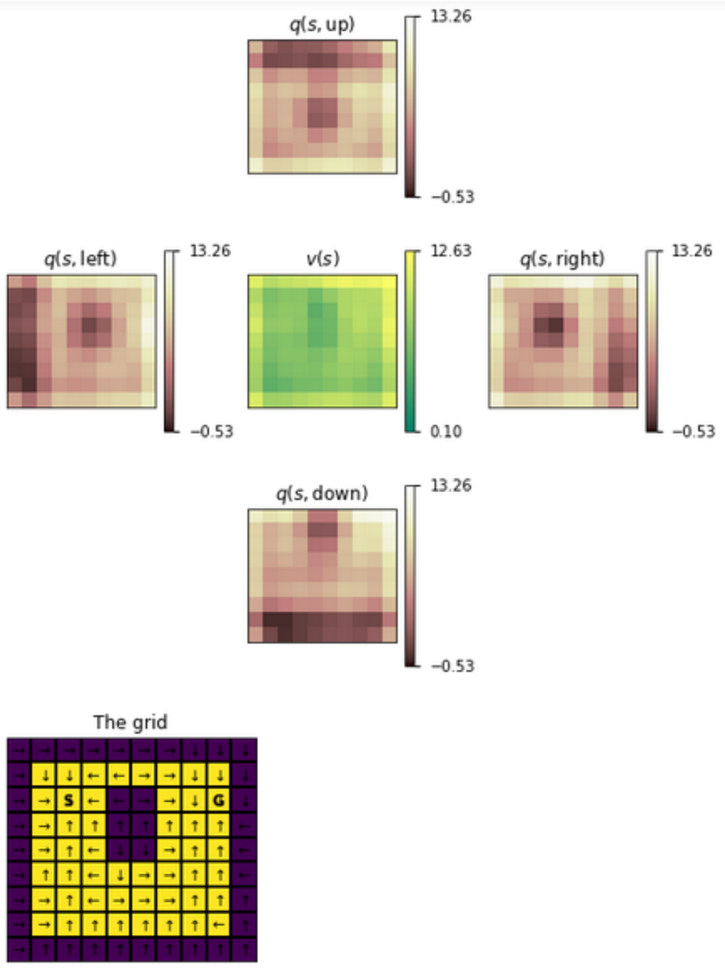
Out[18]:



In [19]:
```python
print("Dyna Q - FEATURE APPROXIMATION")
Image('./dyna_feature.png')
```

```
Dyna Q - FEATURE APPROXIMATION
```

Out[19]:



Q2. Linear Function Approximation Questions

[5 pts] The value estimates with function approximation are considerably more blurry than in the tabular setting despite more training steps and interactions with the environment, why is this the case?

This is because in the function approximation case, we derive out information about which state we are in from "features", rather than an enumerated state. This allows for some degree of aliasing in certain condiions, where some states may have very similar features, and so be difficult to distinguish - even if they have very different properties. This means that the function has to learn from these similar features, for instance, the point where a wall might be, or not be - even though the state vector represent path or wall may be very similar! To do this case, our algorithms require more training and interaction with environment. Furthermore the learning rate for our algorithms is lower for our linear feature updates than for our tabular case, so the agent updates slower its function approximations slower (though this may be a good thing), and furthermore in the case of the our dyna and experience replay agents, there are fewer offline updates.

[5 pts] Inspect the policies derived by training agents with linear function approximation on `FeatureGrid` (as shown by `plot_greedy_policy`). How does this compare to the optimal policy? Are there any inconsistencies you can spot? What is the reason of these?

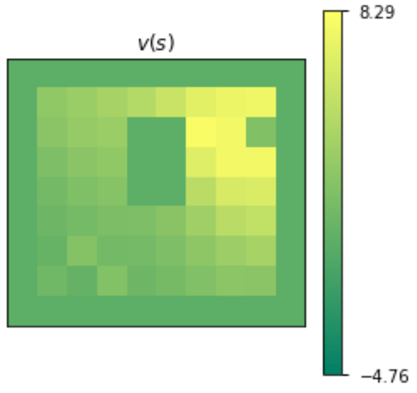
The optimal policy is going "up and over" the narrow pass at the top of the map. The experience replay manages to get very close to this policy, although the others struggle to achieve this. The experience replay gets closest because, being the most data efficient (in this stationary environment) it is able to learn the best approximation function for the q values, which this model should do in the long run. There are however some discrpancies, in particular online q and dyna's optimal policy apparently pushes them away from the centre. Here it would suggest the linear model has learnt some features that strongly indicate a wall in the centre of the model and pushes the agent away from the centre. Unfortunately this applies to the section in the "up and over" section. It is possible there is one feature who's presence is being taken as an indicator for this, hence the erroneous classification. With more training it the models should learn the optimal path.

Non Stationary Plots

In [20]:
```
print("Online Q - NON STATIONARY")
Image('./online_alt.png')
```

```
Online - NON STATIONARY
```

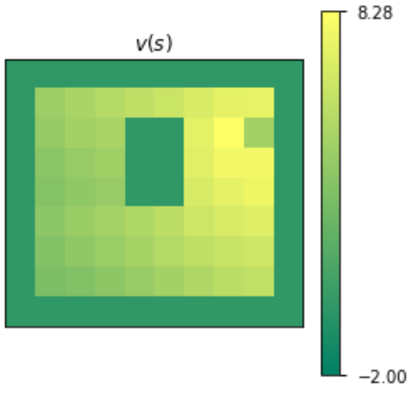
Out[20]:



```
In [23]: print("Replay - NON STATIONARY")
         Image('./replay_alt.png')
```

```
Replay - NON STATIONARY
```

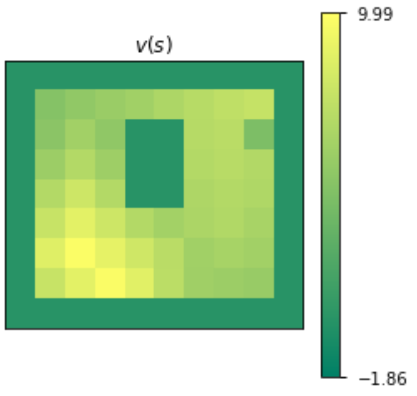
Out[23]:



```
In [22]: print("Dyna - NON STATIONARY")
         Image('./dyna_alt.png')
```

```
Dyna - NON STATIONARY
```

Out[22]:



Q3. Non Stationary Questions

[10 pts] Compare the value estimates of online Q-learning and Experience Replay, after training also on the new goal location, explain what you see.

Looking at the plots it is clear the Online-Q learning manages, in some capacity to slightly adjust to the new target whereas the Experience Replay really fails. All the high value states for Experience replay are still next to the former goal, and have not moved, whereas the online Q has managed to at least assign a slightly higher value to the states directly next to the new goal. In both cases though they have not managed to adapt their whole overall value estimates that well.

Here experience replay performs worse because in the replay cycles, the algorithm is "learning" from out of date experiences, that still have the goal in the top right. Online q doesnt do this, but still struggles to adjust its overall q values to the new ones, as it has only runs for a smaller fraction of steps on the new environment.

[10 pts] Compare the value estimates of online Q-learning and Dyna-Q, after training also on the new goal location, explain what you see.

The value estimates are much stronger for the Dyna-Q algorithm, compared to the online q. Despite only learning for 1/30th of the steps on this new map, dyna-q has increased the value of many squares near the new goal, and decrease the ones near the old goal. Online-q has not managed to do this, only increasing the value of squares immediately neighbouring the new goal.

This is because the Dyna-Q is able to update its model to the changed goal, and therefore benefit from the model-based q-improvements from the replay buffer. In this implementation of the tabular case, the model sets rewards, states and discounts according most recent reward and state and discount, so the model updates very quickly. This means the 30x replay buffer can then update all the new state values very efficiently, based on the model - which the online version of course can't.

```
In [ ]:
```