# Assignment 4

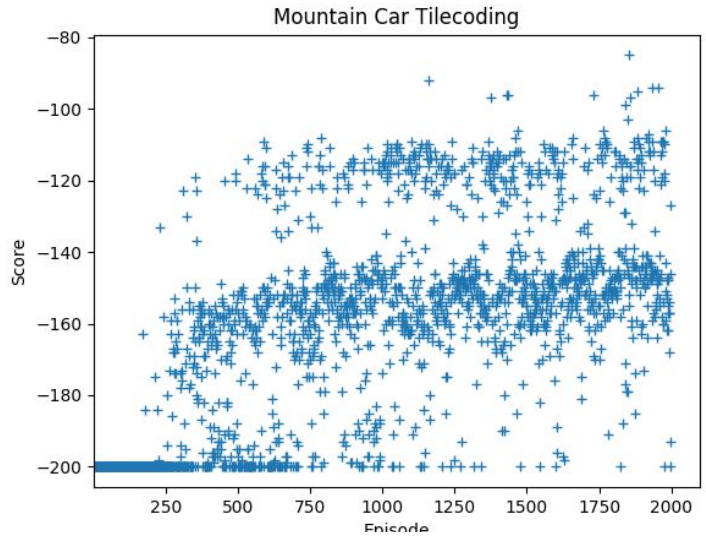
Siddharth Sagar 160030034

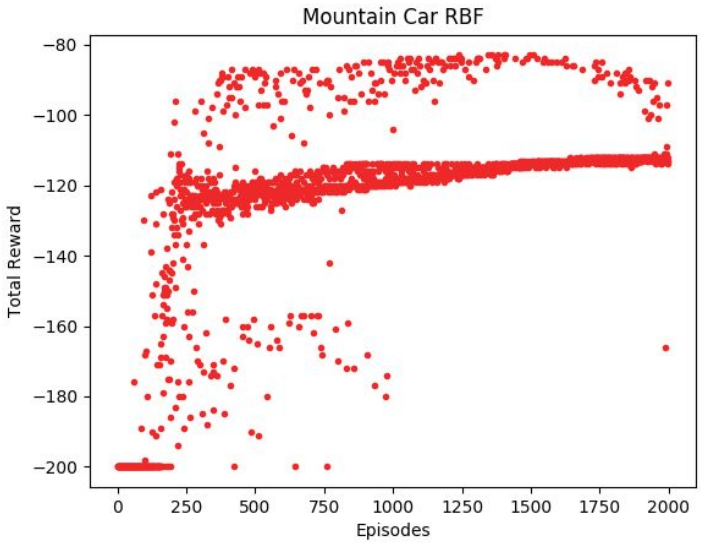
## Mountain Car

I tried to solve mountain car problem using the Open AI gym environment using:

- Multiple Tile Coding
- Radial Basis Functions
- REINFORCE algorithm

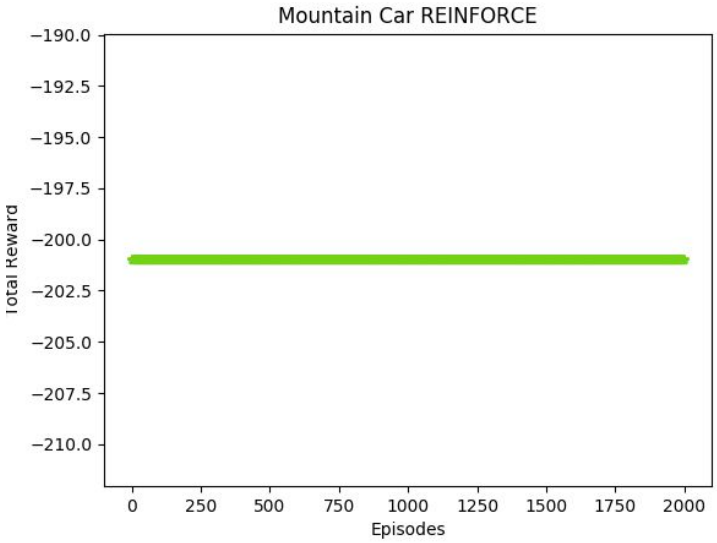
Of the above 3 algorithms, multiple tile coding and RBF converged while the REINFORCE algorithm refused to learn even after trying various combinations of the parameters.




We can see here that **Tile Coding algorithm** learns how to solve the problem slowly but ends up converging to the solution. Although it fluctuates between -160 and -120 mostly, We can see that it learn a good solution. It even reaches a solution of >-100 after a 100 episodes and the frequency of such episodes increases as we continue the algo for a few more Episodes the frequency of episodes with scores >-100 increases. Hence we can conclude that the algorithm works.

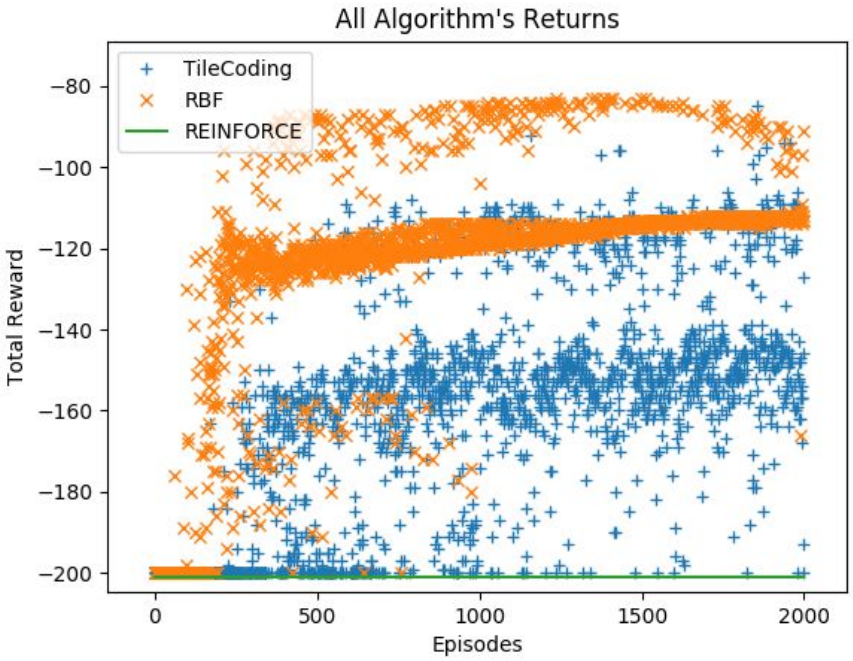
**Radial basis function** clearly works better than Tile Coding, converging at around 200 episodes to a solution which gives -120 reward and the frequency of getting >-100 is much more as we can see in the graph.



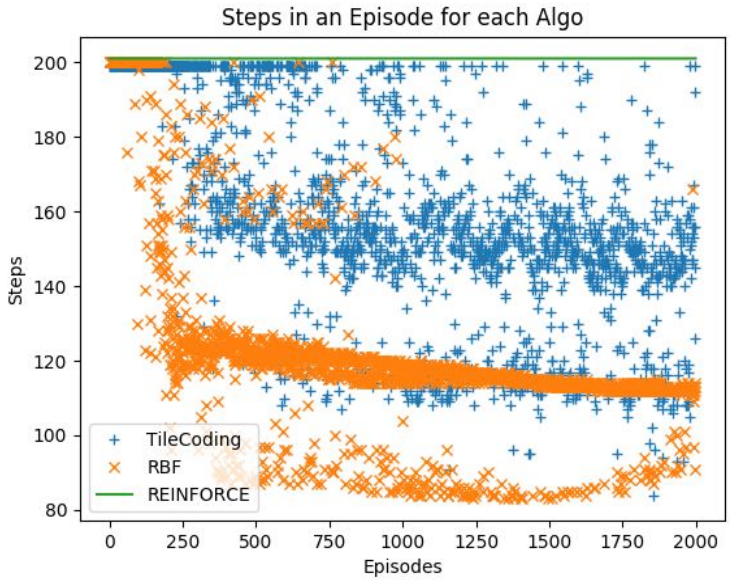

Reinforce Algorithm learns nothing over the course of 2000 episodes.

We can see that the best average is for RBF > TileCoding > REINFORCE

| Algorithm | Average Reward |
|---|---|
| Tile Coding | -159.496 |
| Radial Basis Function | -122.9005 |
| REINFORCE | -200 |



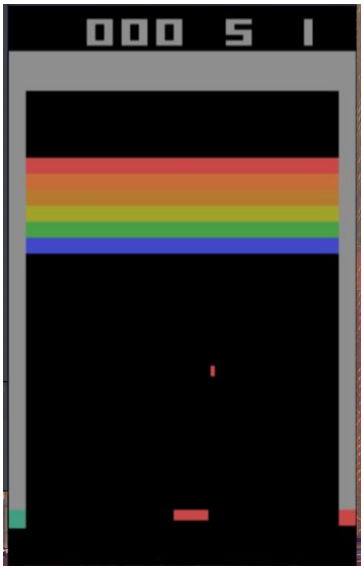
Here the reward is the -ve of the steps taken in the episode. Hence, the steps per episodes also show the same trend but inverted for obvious reasons.




## Breakout Atari

The objective in this OpenAI gym environment is to maximize your score in the Atari 2600 game Breakout. In this environment, the observation is an RGB image of the screen, which is an array of shape (210, 160, 3) Each action is repeatedly performed for a duration of kk frames, where kk is uniformly sampled from {2,3,4}.

The game looks like this.



Since the output was an image we need to process to derive a state. For us the relevant information would be ball position and the player position. The ball can move in two dimensions X and Y, while the player can move only in the X direction.

The image was processed in the following way.

1. Downsizing (By 2x)
2. Convert to Grayscale.
3. Crop the image to remove the score and the borders
4. Read this processed image pixel by pixel to identify positions of the player and the ball
    a. The ball and the player can be identified by checking the value of the pixel being 114
    b. There is also a red(114) block line in the image. This must be considered and code was written in a way that the ball would be found even if this was the same color.

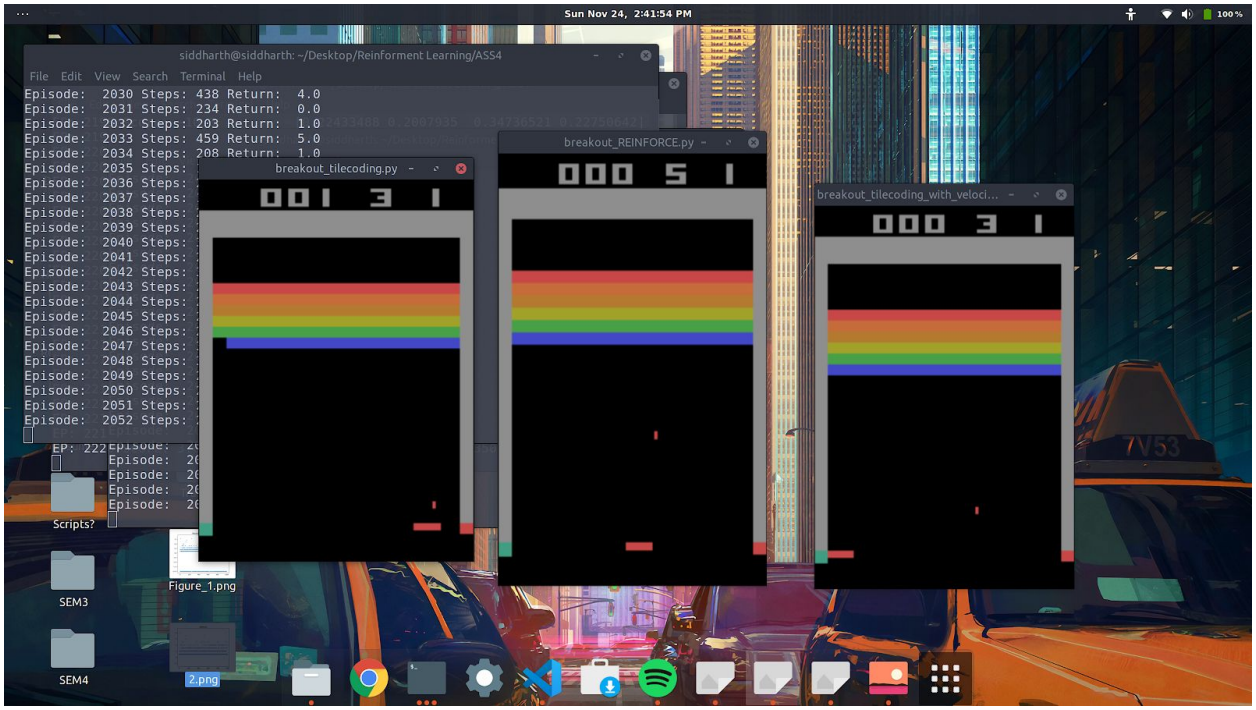
Using 4 of these images which we got consecutively we get the velocity of the ball in the following way.

```
def get_velocity(last4_states):
   frame_by_frame_vel_x = []
   frame_by_frame_vel_y = []
   for i in range(1,len(last4_states)):
       xvel = last4_states[i][0][0] - last4_states[i-1][0][0]
       yvel = last4_states[i][0][1] - last4_states[i-1][0][1]
       if xvel > 5 or xvel < -5 : xvel = 0 This is if the ball resets
       if yvel > 5 or yvel < -5 : yvel = 0
       frame_by_frame_vel_x.append(float(xvel))
       frame_by_frame_vel_y.append(float(yvel))
 return[np.average(frame_by_frame_vel_x),np.average(frame_by_frame_vel_y)]
```

Basically we average the x and y pixel displacements over 4 frames.

For All algorithms, The algorithm was run for 2500 episodes which took about 6hrs with the following parameters wherever relevant

- Alpha = 0.0001
- Gamma = 1
- Lambda = 0.9
- Epi = Emu = epsilon = 0.4

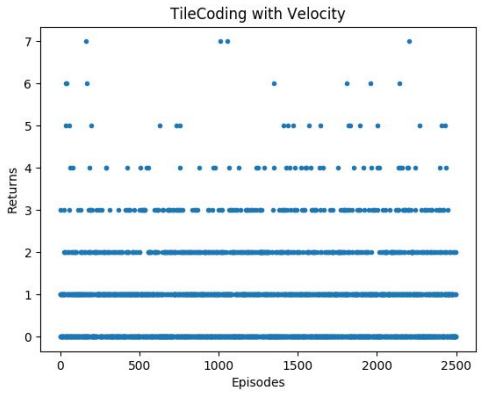


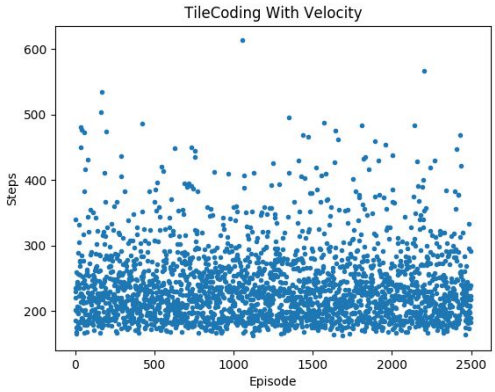

*All 3 code running together*

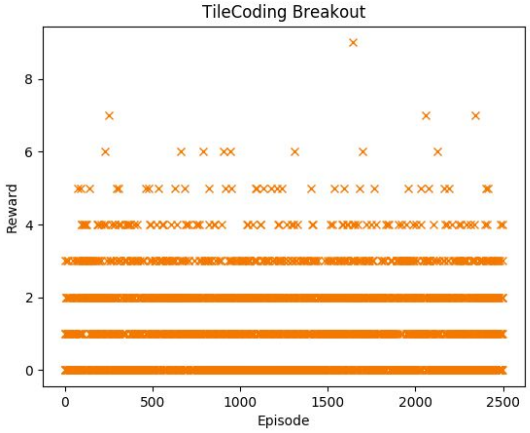
The algorithm was run for 2 variations, one with velocity and one without velocity

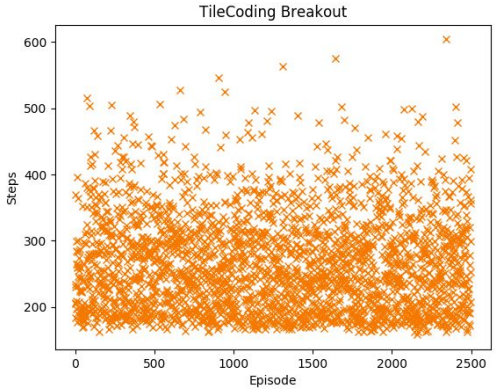
| Algorithm | Average Return | Average Step |
|---|---|---|
| TileCoding with velocity | 0.9524 | 236.3116 |
| TileCoding | 1.316 | 265.386 |
| REINFORCE | 1.4596 | 256.6224 |

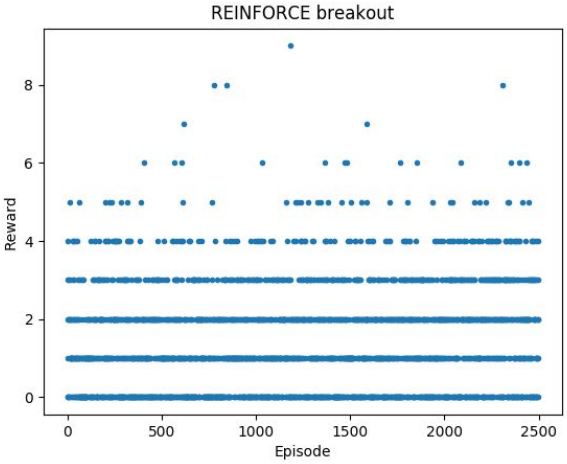
Looking at the averages we can clearly see that including velocity in the **TileCoding** Algorithm has not helped much. This could be because we increase 2 more dimensions to now make the state 5 dimension. Hence the code requires a much longer run time to actually learn something. **REINFORCE** algo works best and can be seen to learn. The following are the graphs generated. Not much can be understood for them.

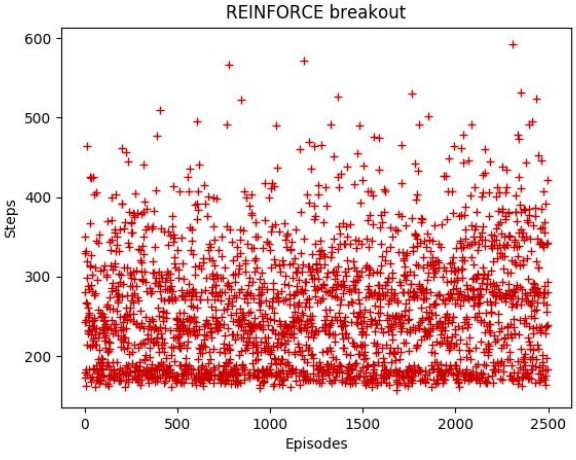

TileCoding with Velocity
Returns
Episodes
0
1
2
3
4
5
6
7
0
500
1000
1500
2000
2500


TileCoding With Velocity
Steps
Episode
200
300
400
500
600
0
500
1000
1500
2000
2500

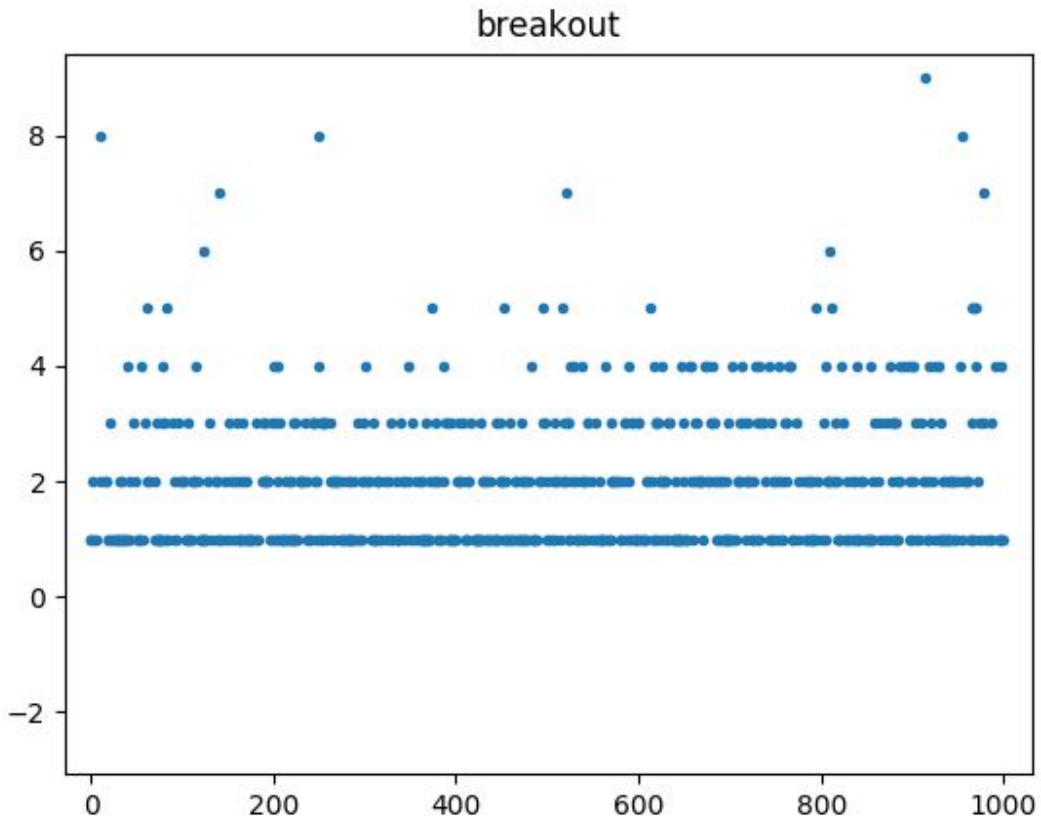

TileCoding Breakout
Reward
Episode
0
2
4
6
8
0
500
1000
1500
2000
2500


TileCoding Breakout
Steps
Episode
200
300
400
500
600
0
500
1000
1500
2000
2500


REINFORCE breakout
Reward
Episode
0
2
4
6
8
0
500
1000
1500
2000
2500


REINFORCE breakout
Steps
Episodes
200
300
400
500
600
0
500
1000
1500
2000
2500

But if we change the 0 Reward case by giving it a penalty of -10 instead we get the following graph.




We can see here that the frequency of 4 reward is increasing as the number of episodes increases. This shows us that the algorithm is indeed learning and can be made better by running it for number of episodes (maybe millions) and tweaking some parameters