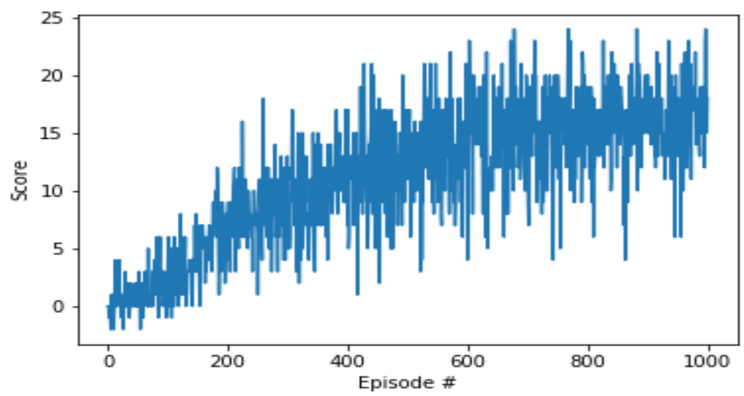
Udacity Deep Reinforcement Learning Nanodegree
Navigation Project Report

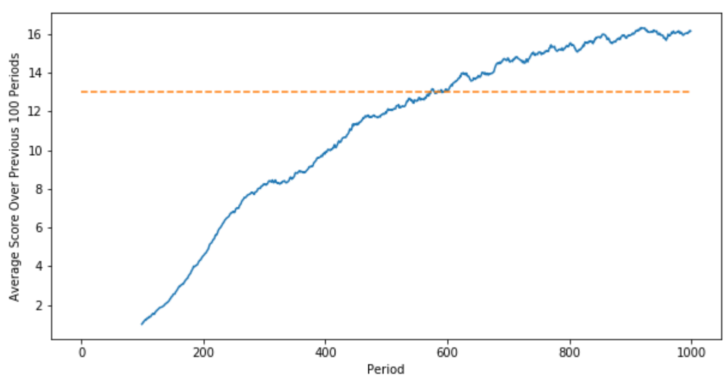
1. Learning algorithm
   I used deep Q-Learning algorithm to solve the environment. Below is a summary of the algorithm:
   - Initialize replay memory D with capacity N (replay buffer size)
   - Initialize action-value function $\hat{q}$ with random weights w
   - Initialize target action-value weights $w^- \leftarrow w$
   - For the episode $e \leftarrow 1 \; to \; M$:
     - State input: S
     - For time step $t \leftarrow 1 \; to \; T$:
       - Sample:
         - choose action A from state S using policy $\pi \leftarrow \epsilon - Greedy\big(\hat{q}(S, A, w)\big)$.
         - Take action A, observer reward R, and next state S'.
         - Store experience tuple (S, A, R, S') in replay memory D. $S \leftarrow S'$
       - For every C steps, learn:
         - Obtain random minibatch of tuples (s_j, a_j, r_j, s_{j+1}) from D
         - Set target $y_j = r_j + \gamma \max_a \hat{q}(s_{j+1}, a, w^-)$
         - Update: $\Delta w = \alpha \left(y_j - \hat{q}(s_j, a_j, w)\right) \nabla_w \hat{q}(s_j, a_j, w)$
         - $w^- \leftarrow \tau \times w + (1 - \tau) \times w^-$

2. Hyperparameters:
   - Exploration vs exploitation:
     - Initial ϵ: 1.0
     - ϵ decay parameter: 0.995
     - Smallest ϵ: 0.01
   - Experience replay:
     - Replay buffer size: 100000 (maximum number of experiences in memory)
   - Minibatch size of experience to learn: 64
   - Discount factor: 0.99 ($\gamma = 0.99$)
   - Frequency to learn: 4 steps (C=4)
   - Learning rate for updating local network: ($\alpha = 0.0001$)
   - Parameter governing soft update of target network weights: 0.001 ($\tau = 0.001$)
   - Neural network:
     - Two hidden layers
     - Each hidden layer has 64 nodes
     - Linear activation is used for both layers

3. Plot of scores
    - Period by period score:



    - Average scores over 100 periods. The environment is solved in 594 periods.



4. Future work
    - Use double DQN. The main difference would be on the target estimate during learn step: $y_j = r_j + \gamma \hat{q}\left(s_{j+1}, argmax_a\left(\hat{q}(s_{j+1}, a, w)\right), w^-\right)$

    - Use dueling DQN where the final value functions are estimated as a combination of state values V(s) and advantage values A(s, a). This could potentially improve performance because many value functions may not change much with the actions.