Navigation project

1. Models tested

We tried and trained different DQN parametrizations:

- simple DQN algo
- double DQN algo
- simple DQN algo with priority sampling

For each we tried to train with different set of parameters:

- simple DQN algo: epsilon decay for values [0.994, 0.995, 0.996, 0.997]
- double DQN algo: epsilon decay for values [0.994, 0.995, 0.996, 0.997]
- simple DQN algo with priority sampling: [prio_a x prio_b] for values [[0., 1.] x [0., 1.]] (0.1 steps)

Notation

A Simple or Double DQN model will be identified by the following string: "[dqn*type]*[neural_network_fully_connected_hidden*layers]*[epsilon_decay]"

A priority sampling model will add "[prio*a]*[prio_b]"

For instance "simple_[32; 32]_0.994" designates a simple DQN algo, trained with a neural network composed of 2 hidden layers fully connected of 32 nodes each, using a epsilon decay every episode of 0.994

In [1]:

```
%load_ext autoreload
%autoreload 2
```

In [2]:

```
import results_analysis
```

2. Training results

In [3]:

```
#Get results from 'train_results.csv'
#results_df: full dataframe of results
#solved_df: nb of episodes to reach 14 reward over last 100 episodes
#tops_df: fastest models to solve environment
results_df, solved_df, tops_df = results_analysis.get_training_results_df(results_file=
'train_results.csv', tops_nb=5)
```

2.1 Fastest trained models

In [4]:

```
#print tops_df
display(tops_df)
```

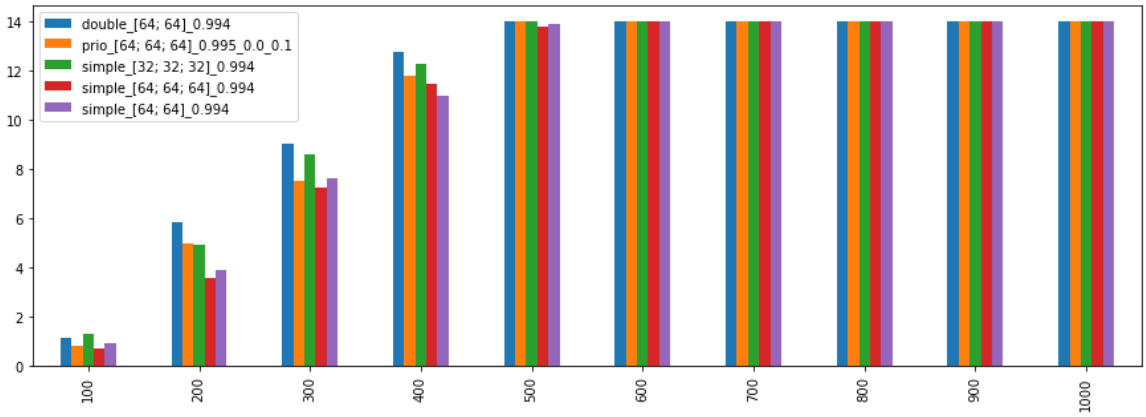
model	episode
double_[64; 64]_0.994	450
prio_[64; 64; 64]_0.995_0.0_0.1	483
simple_[32; 32; 32]_0.994	496
simple_[64; 64; 64]_0.994	505
simple_[64; 64]_0.994	506

In [5]:

```
#Plot fastest models
results_analysis.plot_results(results_df, tops_df.index, results_type='train')
```

Out[5]:

	double_[64; 64]_0.994	prio_[64; 64; 64]_0.995_0.0_0.1	simple_[32; 32; 32]_0.994	simple_[64; 64; 64]_0.994	simple_[64; 64]_0.994
100	1.16	0.81	1.31	0.73	0.91
200	5.86	4.99	4.93	3.58	3.89
300	9.07	7.51	8.61	7.28	7.65
400	12.76	11.81	12.27	11.47	10.98
500	14.00	14.00	14.00	13.79	13.91
600	14.00	14.00	14.00	14.00	14.00
700	14.00	14.00	14.00	14.00	14.00
800	14.00	14.00	14.00	14.00	14.00
900	14.00	14.00	14.00	14.00	14.00
1000	14.00	14.00	14.00	14.00	14.00



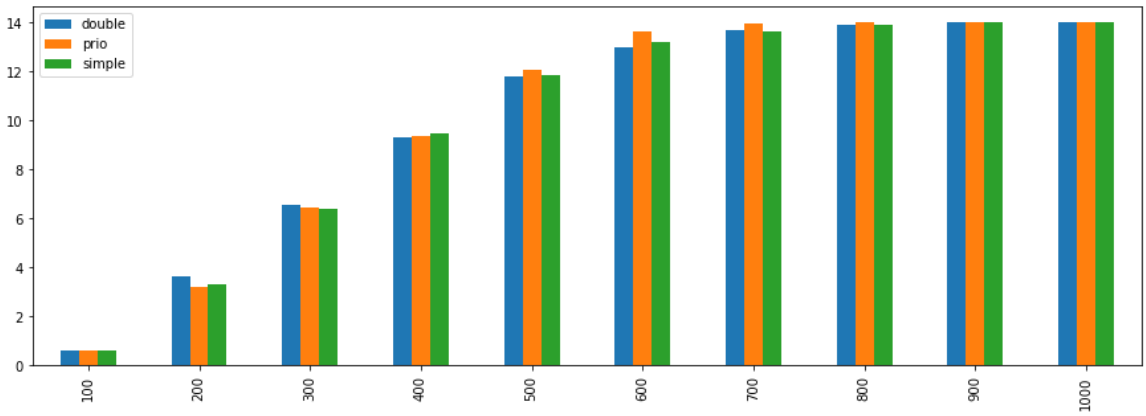
2.2. Per model type training results (simple, double, prio)

In [6]:

```
results_analysis.plot_results_per_model_type(results_df, results_type='train')
```

Out[6]:

	double	prio	simple
100	0.627500	0.616300	0.586250
200	3.620625	3.230300	3.319375
300	6.575625	6.450900	6.395000
400	9.315625	9.394400	9.507500
500	11.781250	12.098700	11.875000
600	12.980000	13.656832	13.206250
700	13.711250	13.987900	13.650000
800	13.905000	14.000000	13.903125
900	14.000000	14.000000	14.000000
1000	14.000000	14.000000	14.000000



3. Test results

In [7]:

```
#Get results from 'test_results.csv'
#results_df: full dataframe of results
#solved_df: nb of episodes to reach more than 13 reward in testing_phase
#tops_df: highest score models during testing phase (in average over 10 episodes)
results_df, solved_df, tops_df = results_analysis.get_test_tesults_df(results_file='tes
t_results.csv', tops_nb=5)
```

3.1 Best test models

In [8]:

```
#print tops_df
display(tops_df)
```

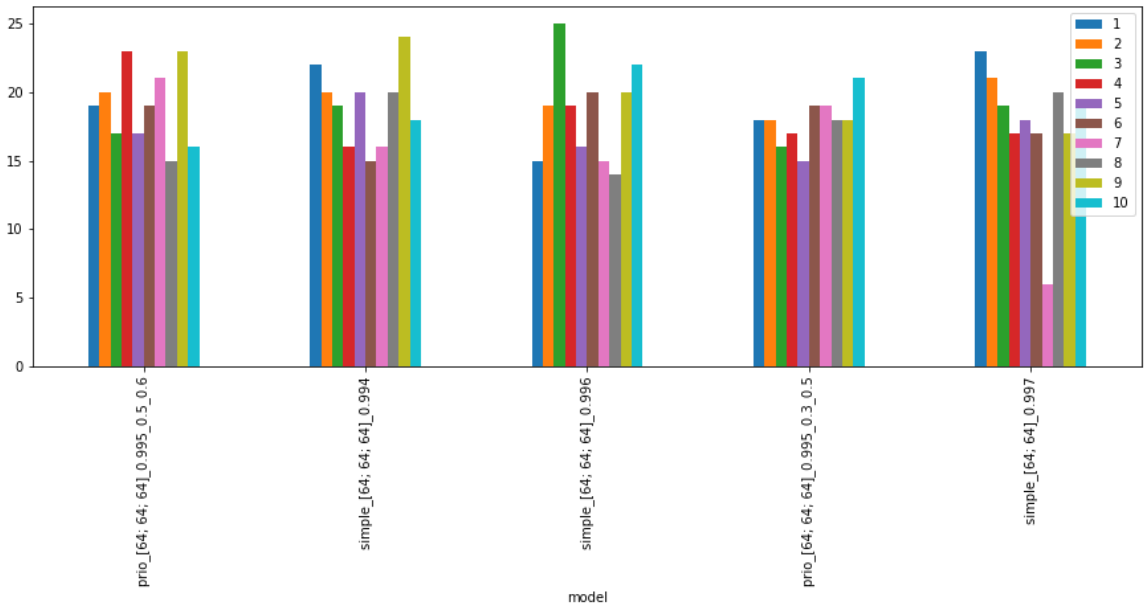
model	score
prio_[64; 64; 64]_0.995_0.5_0.6	19.0
simple_[64; 64; 64]_0.994	19.0
simple_[64; 64; 64]_0.996	18.5
prio_[64; 64; 64]_0.995_0.3_0.5	17.9
simple_[64; 64]_0.997	17.7

In [9]:

```
#Plot fastest models
results_analysis.plot_results_transpose(results_df, tops_df.index, results_type='test')
```

Out[9]:

model	1	2	3	4	5	6	7	8	9	10
prio_[64; 64; 64]_0.995_0.5_0.6	19.0	20.0	17.0	23.0	17.0	19.0	21.0	15.0	23.0	16.0
simple_[64; 64; 64]_0.994	22.0	20.0	19.0	16.0	20.0	15.0	16.0	20.0	24.0	18.0
simple_[64; 64; 64]_0.996	15.0	19.0	25.0	19.0	16.0	20.0	15.0	14.0	20.0	22.0
prio_[64; 64; 64]_0.995_0.3_0.5	18.0	18.0	16.0	17.0	15.0	19.0	19.0	18.0	18.0	21.0
simple_[64; 64]_0.997	23.0	21.0	19.0	17.0	18.0	17.0	6.0	20.0	17.0	19.0



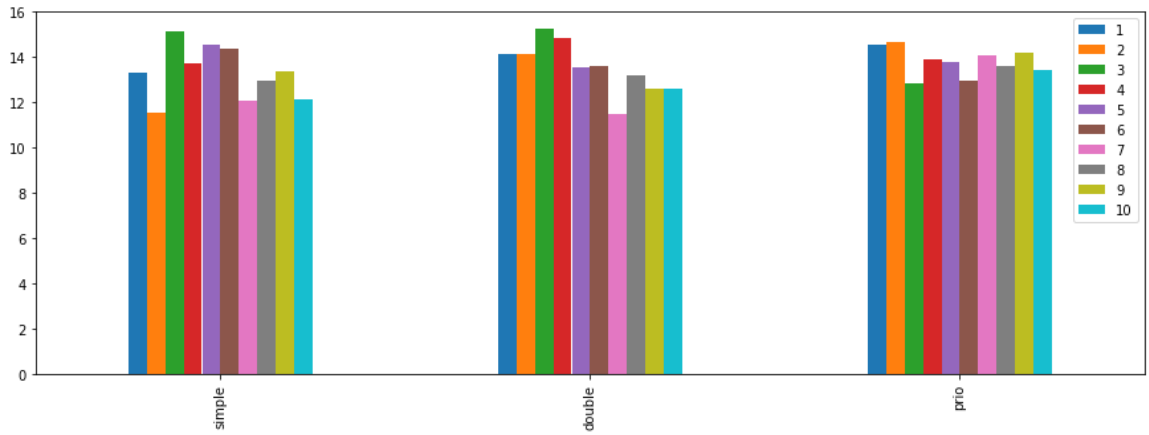
3.2. Per model type test results (simple, double, prio)

In [10]:

```
results_analysis.plot_results_per_model_type_transpose(results_df, results_type='test')
```

Out[10]:

	1	2	3	4	5	6	7	8	9	10
simple	13.3125	11.5625	15.125	13.6875	14.5625	14.375	12.0625	12.9375	13.375	12.125
double	14.1250	14.1250	15.250	14.8125	13.5625	13.625	11.5000	13.1875	12.625	12.625
prio	14.5400	14.6300	12.850	13.8700	13.8000	12.920	14.0400	13.5700	14.210	13.440



4. Conclusion

All models perform comparably but it seems like the priority sampling models perform slightly better than the rest, both in terms of training speed and test results. I then decided to select the "prio_[64; 64; 64]_0.995_0.5_0.6" model as it was the best on the test set and it trained raisonably fast in 610 episodes

In [11]:

```
import os
import results_analysis

#You need to be at the root directory of the repo to run the model in the next cell
os.chdir('..')
print(os.getcwd())
selected_model = 'prio_[64; 64; 64]_0.995_0.5_0.6'
```

```
C:\Users\J\Programming\Visual Studio Code\UdacityRL
```

4.1 Selected model training

In [12]:

```
#The selected model was trained in 610 episodes (reached a score of 14 over the last 10
0 episodes)
results_df, solved_df, tops_df = results_analysis.get_training_results_df(results_file=
'./Results/train_results.csv', tops_nb=5)
display(solved_df.loc[selected_model])
```

```
episode    610
Name: prio_[64; 64; 64]_0.995_0.5_0.6, dtype: int64
```

4.2 Selected model test run

In [13]:

```
#If you get the "handle is closed" error, you need to restart your kernel and execute f
rom the Conclusion first cell;
#I don't know how to fix that
%run -i navigation.py test --test_params=best_params.json --test_model="auto" --test_re
sults_path="''"
```

```
INFO:unityagents:
'Academy' started successfully!
Unity Academy name: Academy
        Number of Brains: 1
        Number of External Brains : 1
        Lesson number : 0
        Reset Parameters :

Unity brain name: BananaBrain
        Number of Visual Observations (per agent): 0
        Vector Observation space type: continuous
        Vector Observation space size (per agent): 37
        Number of stacked Vector Observation: 1
        Vector Action space type: discrete
        Vector Action space size (per agent): 4
        Vector Action descriptions: , , ,

Create NN with layers: [(37, 64), (64, 64), (64, 64), (64, 4)]
Create NN with layers: [(37, 64), (64, 64), (64, 64), (64, 4)]
Load model weights ./ModelWeights/prio_[64, 64, 64]_0.995_0.5_0.6.pth
Score: 16.0
```

In [ ]:

```

```