# Project Progress

Table of contents

[Project Progress 1](#_Toc47911187)

[Building the simulation 1](#_Toc47911188)

[Architecture 1](#_Toc47911189)

[Performance test 2](#_Toc47911190)

Figures list

[Figure 1 – simulation architecture 2](#_Toc47911138)

[Figure 2 – Double Deep Q-Network architecture and flow 2](#_Toc47911139)

[Figure 3 - Epsilon greedy, starts with 1, ends with 0.05 and decay of 0.00001 3](#_Toc47911140)

[Figure 4 – environment set-up for zombie master performance check with optional actions 4](#_Toc47911141)

[Figure 5 – Zombie master actions distribution along different ranges of episodes 5](#_Toc47911142)

[Figure 6 – Total zombies survived vs. the episodes (blue) with its moving average (orange) 5](#_Toc47911143)

[Figure 7 - environment set-up for light master performance check with optional actions 6](#_Toc47911144)

[Figure 8 - Light master actions distribution along different ranges of episodes 7](#_Toc47911145)

[Figure 9 - Total zombies survived vs. the episodes (blue) with its moving average (orange) 7](#_Toc47911146)

## Building the simulation

### Architecture

Building a simulation for Reinforcement learning purposes is mostly a manner of creating an environment and throw there some entities that follow certain rules.

In our case the entities thrown into the environment are zombies and light.  
The managers of the above entities are the zombie and light masters. The zombies master has the ability to place a zombie at some starting position as it wishes, and the light master has the ability to place the light somewhere on top the board.

Once the zombie master ordered to place a zombie, the only rule for the zombie is move straight to the other side of the board.

Hence, the two parts of the simulation are the: environment and entities.

#### Environment

As said above, the environment is storage place of all entities that are going to join the simulation. Since we are facing a 2D board game, the environment implemented as a grid with cells. Each cell can accommodate up to one zombie.

On top of the grid, there is the environment class which is able to query and contact the zombies inside the grid. In addition, all the outer communication from the environment is managed by the environment manger. Its purpose is to pass the environment to the learning agents command and process/reshape the environment state before sending forward.

Over view of the above:

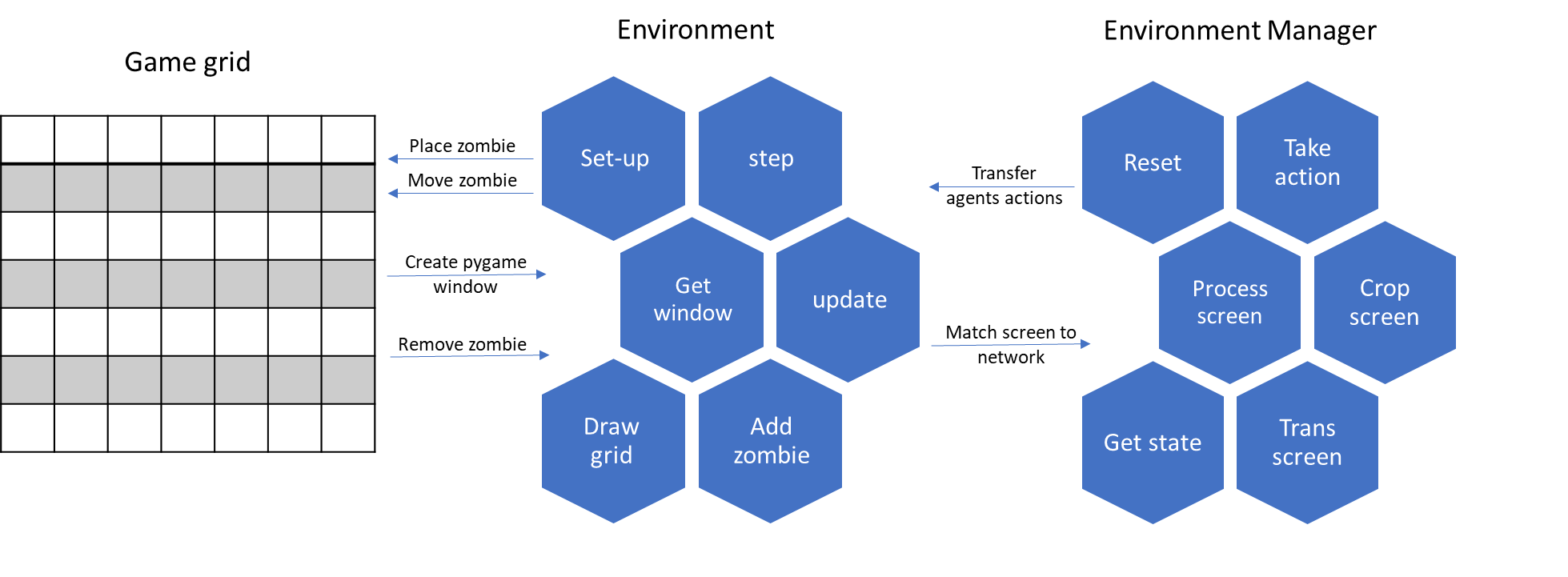


Figure – simulation architecture

#### Entities

There are four types of entities living in our environment that are:

* Light master
* Light
* Zombie master
* Zombie

The light and zombie masters are intelligent agents with the ability to create zombies and move the light accordingly. On the other hand, the zombie and light per se are only data entities that are using to visualize the state in some conditions.

### Performance test

Usually in RL projects, we will use some known and tested environment, since that’s not the case, we have to test the performance of the environment with some simple scenarios in order to prove sanity and stability.

#### The algorithm

For testing the performance we'll use a model known as DDQN, which stands for Double Deep Q-Network.

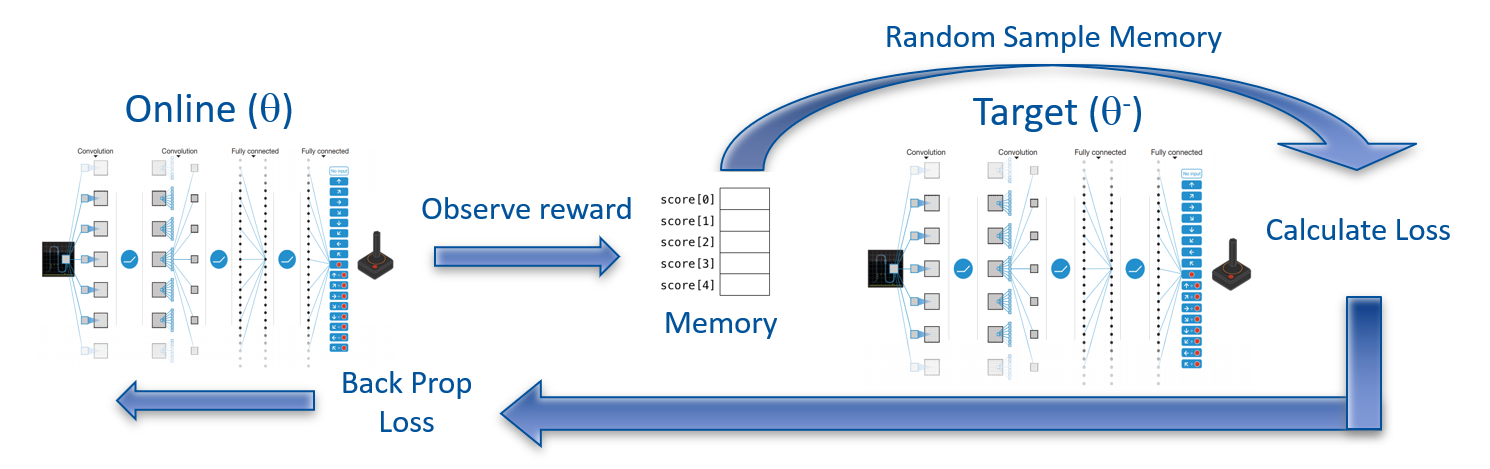


Figure – Double Deep Q-Network architecture and flow

The learning algorithm we used in this project called: 'Double Deep Q Learning'.  
In **Double Deep Q Learning**, the agent uses two neural networks to learn and predict what action to take at every step. One network, referred to as the **online network**, is used to predict what to do when the agent encounters a new state. It takes in the state as input and outputs Q values for the possible actions that could be taken.  
The other network, referred to as the **target network**, is used to evaluate what is the best action to take for the next state (the action with the highest Q value).

For the evaluation process we use something called **replay memory**, which holds the last history up to sometime in the past. And eventually, for **loss calculation** we sample a **random batch** (with some size smaller than the memory size) from the replay memory and updating by **back propagation** the online network. After some number of rounds called **replace target frequency**, we **update the target** net weights according to the online net. We can look at Figure 2 that sums up the whole idea.

#### Epsilon Greedy strategy

**Epsilon greedy policy** is a way of selecting random actions with uniform distribution from a set of available actions. Using this policy either we can select random action with epsilon probability and we can select an action with 1-epsilon probability that gives maximum reward in given state.

During the learning process we will use the epsilon greedy strategy with non-linear decrease in epsilon of:

Which with 200,000 steps looks like:

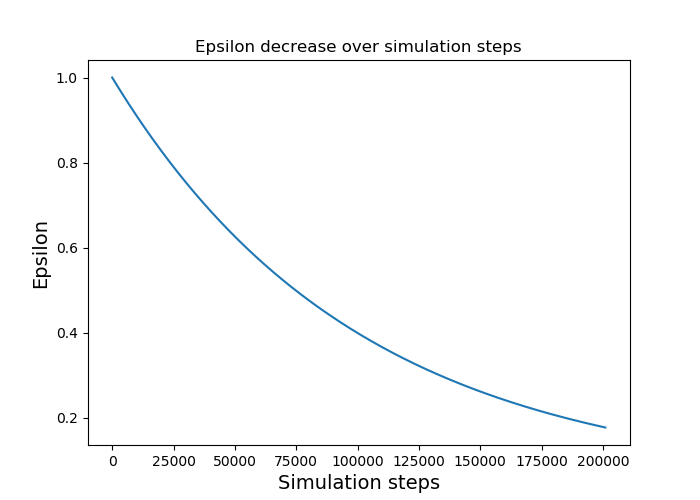
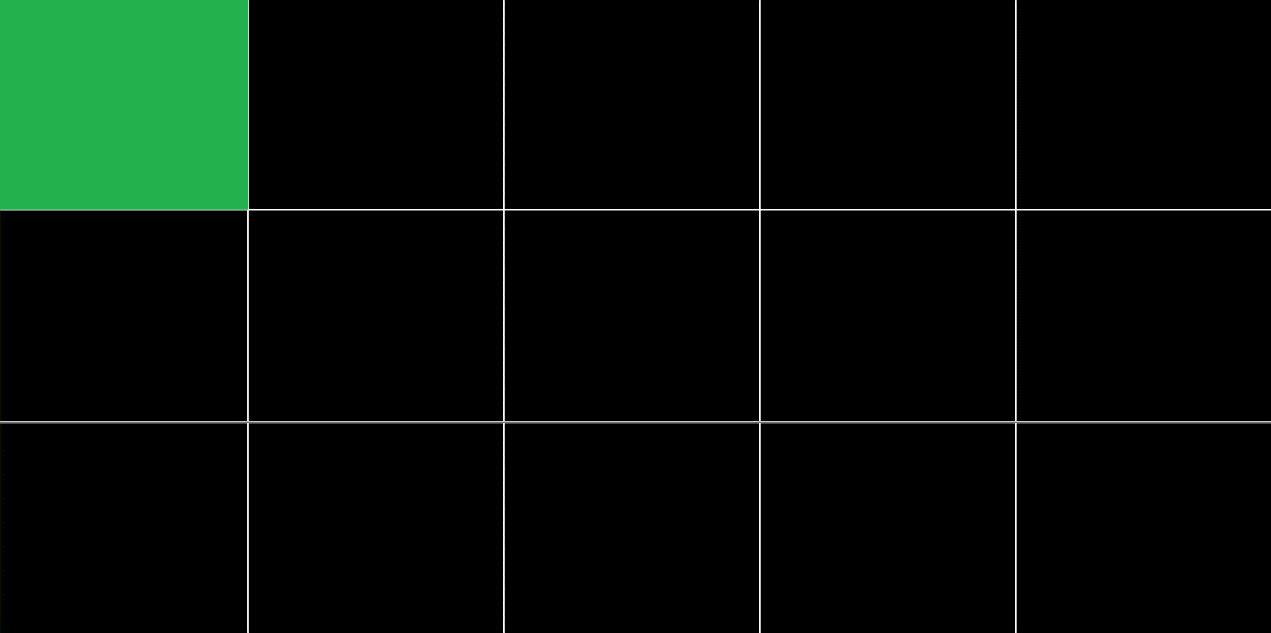


Figure - Epsilon greedy, starts with 1, ends with 0.05 and decay of 0.00001

#### Zombie master test on a 3x5 board

As of the first test of learning, consider the zombie master that learns alone while the light master is forced to take some predetermined action.

At the beginning, we implemented the DDQN algorithm for the zombie agent with grid of 3x5 that look like:



0

1

2

Figure – environment set-up for zombie master performance check with optional actions

As we can see in Figure 4, the light master is forced to take the top left cell as action in every step as stated above.

Consider the following parameters for the learning process:

|  |  |
| --- | --- |
| Light master action | 0 |
| Target update | 10 |
| Num episodes | 1000 |
| Steps per episode | 200 |
| Batch size | 256 |
| Gamma (discount factor) | 0.999 |
| Epsilon-greedy start | 1 |
| Epsilon-greedy end | 0.05 |
| Epsilon-greedy decay | 0.00001 |
| Replay memory size | 1000 |
| Learning rate | 0.001 |

With a deep NN of three layers, all fully connected (called 'Linear' in pytorch formulation): Linear (15,32), Linear (32,16), Linear (16,3). As we know, the zombie master has three possible actions to play – the meaning of the '3' in the last layer.

We achieved a convergence in the amount of times the zombie master chose to send a zombie from the top row (the worst decision it could make):

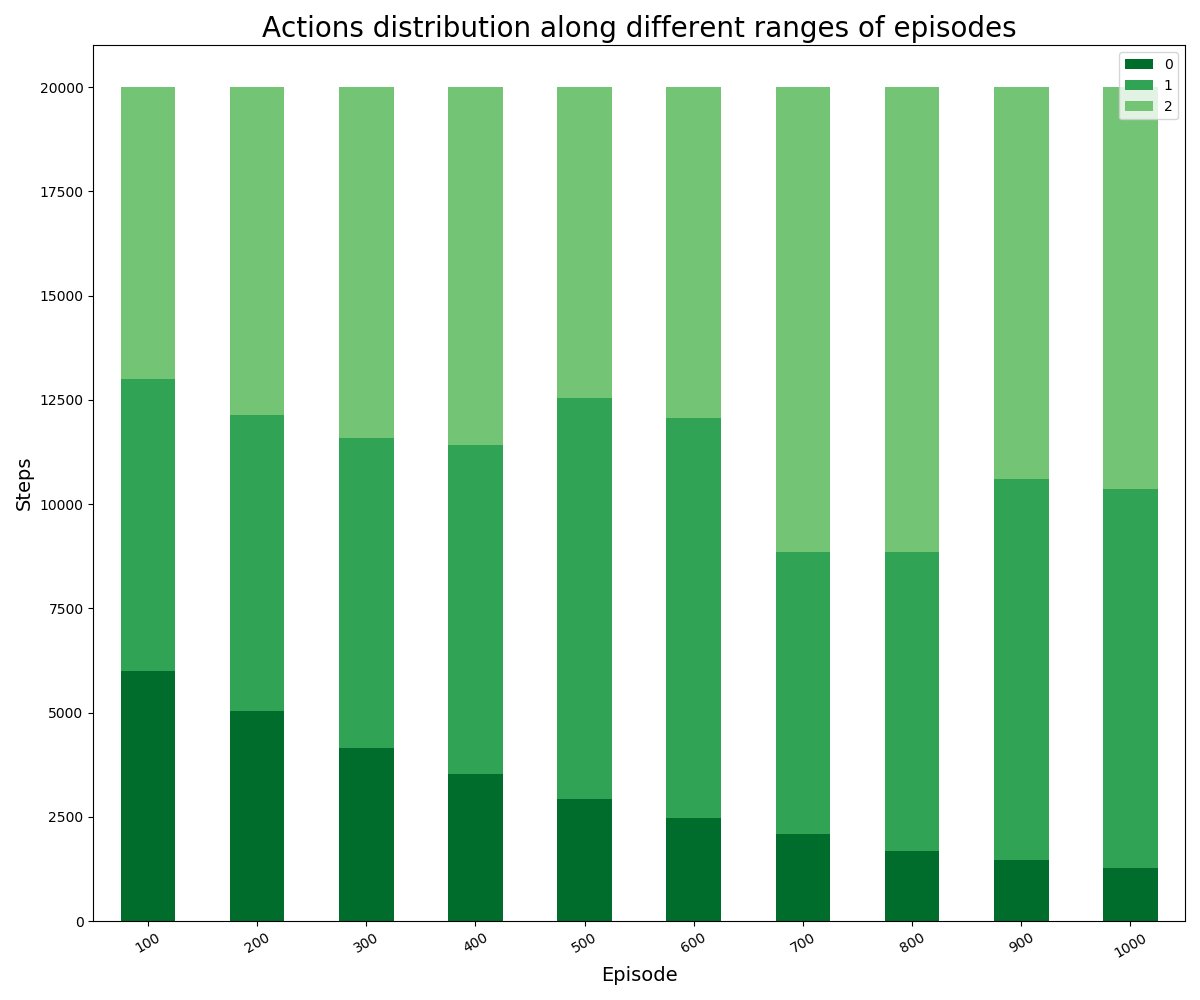


Figure 5 – Zombie master actions distribution along different ranges of episodes

In Figure 5 we can notice the fading of the darker green which means that the zombie master decides to choose action one or two outright as the episodes go on.

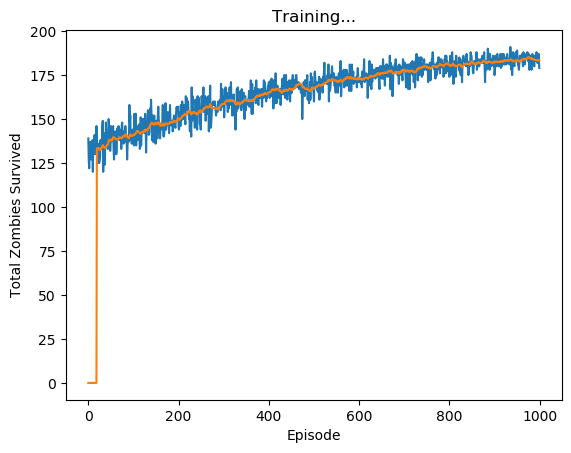
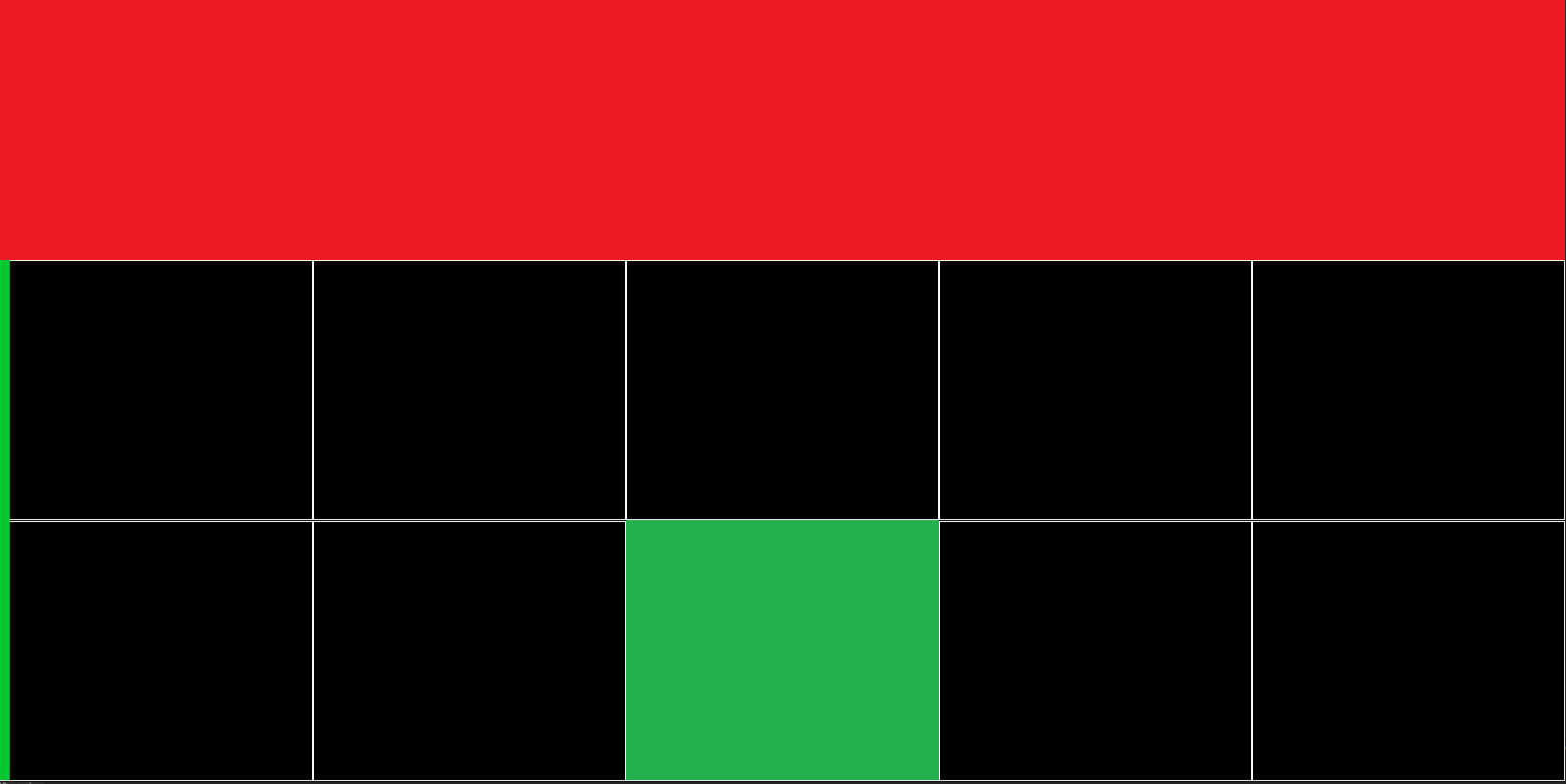


Figure – Total zombies survived vs. the episodes (blue) with its moving average (orange)

Plus, we can tell from Figure 6, the zombie master reward is almost at the maximum it can get – reaches 185 zombies from possible of 195 (there are 200 steps with grid width of 5), something that can be explained be the epsilon greedy ending parameter at 0.05 which means that at the end of the learning, the zombie master is still taking random steps with 5% chance.

#### Light master test on a 3x5 board



7

8

6

5

9

0

1

2

3

4

10

11

13

14

12

Figure - environment set-up for light master performance check with optional actions

Figure 7 illustrates the simulation while testing the performance of the light master.   
As we can see, the zombie master takes only the action 0 (red cells, predetermined for simplicity) what caused the zombies to exit from the upper cell solely. Which after few steps made the entire top row full of zombies. In addition, the green cell represents the light action in the current step.

Furthermore, in general, we can tell in the first episodes there should survive roughly ~130 zombies since the actions are taken random and there is 33% chance for the light master to light the top row.

Once again, consider the following parameters:

|  |  |
| --- | --- |
| Zombie master action | 0 |
| Target update | 10 |
| Num episodes | 1000 |
| Steps per episode | 200 |
| Batch size | 256 |
| Gamma (discount factor) | 0.999 |
| Epsilon-greedy start | 1 |
| Epsilon-greedy end | 0.05 |
| Epsilon-greedy decay | 0.00001 |
| Replay memory size | 1000 |
| Learning rate | 0.001 |

With a deep NN of three layers, all fully connected (called 'Linear' in pytorch formulation): Linear (15,32), Linear (32,16), Linear (16,15).  
In this case we have fifteen outputs. Hence the output of the last layer equals to 15.

This time we achieve increase in the amount the light master chooses to light the first row. The phenomenon indicates the light agent's recognition of the fact that the zombies are coming out of the upper cell.

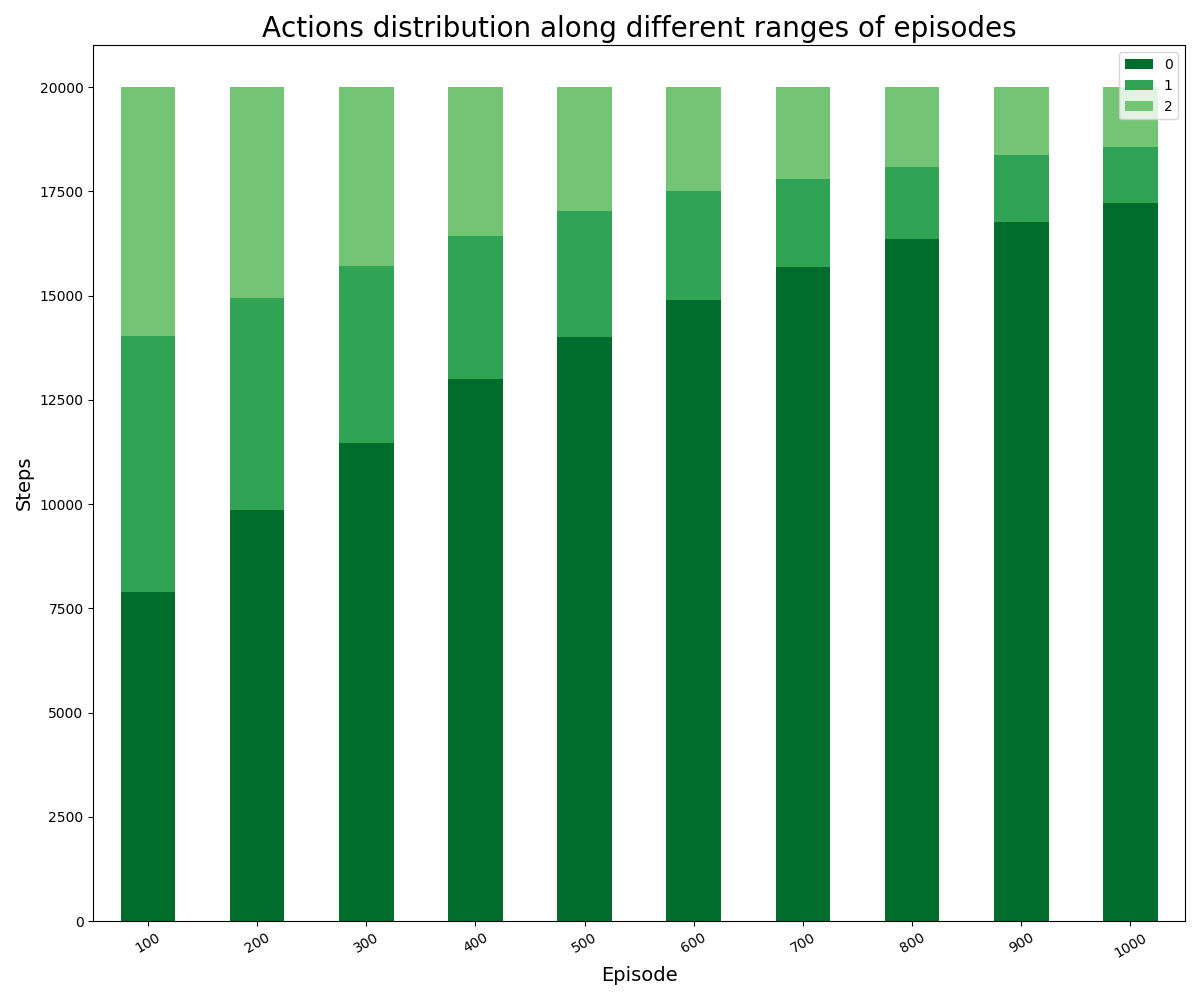


Figure 8 - Light master actions distribution along different ranges of episodes

In Figure 8 we can see the significant increase in the number of times the light agent selected the top row illumination throughout the simulation progress.  
After 800 episodes, the light master chooses to light the correct row more than 80% of the time. Note that the value of the greedy epsilon here is decreasing to 0.05 at the 1000th episode, this tells us the agents are still taking random actions sometimes, which can lead to poor and not helpful choices.

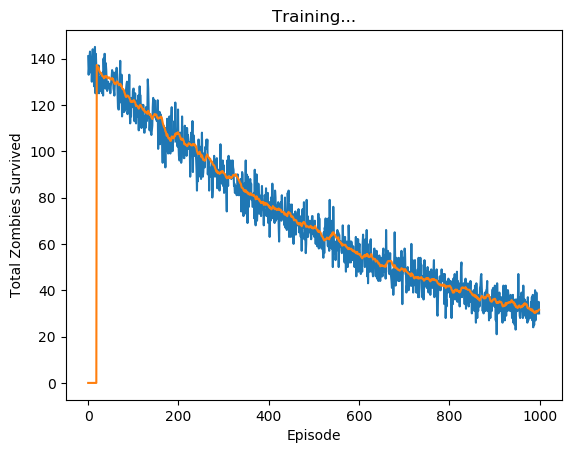


Figure - Total zombies survived vs. the episodes (blue) with its moving average (orange)

In Figure 9 we can clearly see the learning process of the light agent, from the first episodes with 130 zombies survived (out of possible 195, it's approximately two thirds), it managed to eliminate over 80% of the zombies by the 1000th episode.