# Final Project Proposal: MARL with Asymmetric zero-sum game

Eliav Shalelashvili  
Project supervised by Dr Yehudit Aperstein  
Afeka

## Table of Contents

[Final Project Proposal: MARL with Asymmetric zero-sum game 1](#_Toc49120085)

[1. Table of Contents 1](#_Toc49120086)

[2. Figure List 1](#_Toc49120087)

[3. INTRODUCTION 2](#_Toc49120088)

[4. LITERATURE REVIEW 2](#_Toc49120089)

[4.1 Reinforcement Learning 2](#_Toc49120090)

[*4.2* *Stochastic Games* 3](#_Toc49120091)

[*4.3* *Nash Equilibrium in SGs* 4](#_Toc49120092)

[*4.4* *Learning in SGs* 5](#_Toc49120093)

[*4.5* *The GVG-AI competition* 6](#_Toc49120094)

[5. OUR MODEL - Zombie invasion problem 7](#_Toc49120095)

[*5.1* *Assumptions* 7](#_Toc49120096)

[*5.2* *Stochastic game* 8](#_Toc49120097)

[*5.3* *Agents* 8](#_Toc49120098)

[*5.4* *Game rules* 9](#_Toc49120099)

[6. Next steps 10](#_Toc49120100)

[7. Building the simulation 10](#_Toc49120101)

[7.1 Architecture 10](#_Toc49120102)

[7.2 Performance Test 11](#_Toc49120103)

[8. REFERENCES 18](#_Toc49120104)

## Figure List

[Figure 1 – Game tree when there are two actions by player 4](#_Toc49120115)

[Figure 3 – simulation architecture 11](#_Toc49120116)

[Figure 4 – Double Deep Q-Network architecture and flow 12](#_Toc49120117)

[Figure 5 - Epsilon greedy, starts with 1, ends with 0.05 and decay of 0.00001 13](#_Toc49120118)

[Figure 6 – environment set-up for zombie master performance check with optional actions 13](#_Toc49120119)

[Figure 7 – Zombie master actions distribution along different ranges of episodes 14](#_Toc49120120)

[Figure 8 – Total zombies survived vs. the episodes (blue) with its moving average (orange) 15](#_Toc49120121)

[Figure 9 - environment set-up for light master performance check with optional actions 15](#_Toc49120122)

[Figure 10 - Light master actions distribution along different ranges of episodes 16](#_Toc49120123)

[Figure 11 - Total zombies survived vs. the episodes (blue) with its moving average (orange) 17](#_Toc49120124)

## Table List

[Table 1 – learning parameters while evaluating the zombie master 14](#_Toc49859281)

[Table 2 – learning parameters while evaluating the light master 16](#_Toc49859282)

## INTRODUCTION

One of the primary goals of the field of artificial intelligence (AI) is to produce fully autonomous agents that interact with their environments to learn optimal behaviors, improving over time through trial and error. Crafting AI systems that are responsive and can effectively learn has been a long-standing challenge, ranging from robots, which can sense and react to the world around them, to purely software-based agents, which can interact with natural language and multimedia.  
Imagine a board of zombies approaching from some locations in the left side of the board towards the right side. Above all that, there is a light that can be positioned anywhere on the board.  
The two agents will be called 'Zombie Master' and 'Light Master' the Zombie Master is responsible of positioning the zombies in the left side and determine their initial angle and speed that will stay constant for each zombie. On the other hand, the Light Master decides where to project his light in every turn. Each zombie that leaves the left side of the board and goes under the light of the Light Master is damaged and his strength meter is lowered by some value.  
In general, the goal of the Zombie Master/Light Master is to maximize/minimize the strength of the zombies that are reaching the right side of the board.   
We will use some variants of traditional RL methods and examine their profit to our work.

## LITERATURE REVIEW

The area of learning agents that master a particular game or on the other side, agents that seek the highest score over a set of games, grew to huge scales in the past few years. Since we are not facing with a studied problem nor a known game, we will divide our review into five sections:

1. Reinforcement Learning (RL) – review of some traditional and relevant RL algorithms and concepts
2. Stochastic Games
3. Nash Equilibrium in Stochastic Games
4. Learning in Stochastic Games
5. The GVG-AI competition

All along with elaboration of the potential contribution of each topic to our research due to the successes of similar problems and previous research of the domain.

### Reinforcement Learning

Reinforcement learning is an area of Machine Learning. It is about taking suitable action to maximize reward in a particular situation. Essentially an agent (or several) is built such that it can perceive and interpret the environment in which is placed, furthermore, it can take actions and interact with it.  
Basic reinforcement learning problems are modeled as a Markov Decision process (MDP) which is a 4-tuple , where:

* is a finite set of states.
* is a finite set of actions.
* is the probability that action  in state at time will lead to state , due to action .
* is the immediate reward (or expected immediate reward) received after transitioning from state to state , due to action .

The goal is to learn a policy that maximizes the cumulative sum of discounted rewards

where are the rewards and is a discount factor, tuning parameter through which we can influence the amount of weight we give to future awards in relation to the immediate reward.

We can split the subject of RL into two main partitions: ***Model-Free*** and ***Model-Based***. In Model-Free RL,  
the agent does not have access to a model of the environment (The agent couldn’t estimate the consequences of his actions). In Model-Based RL, the agent has access to a model of the environment.  
Our focus is on the Model-Free type of learning mainly due to the advantage that it doesn’t require a model of the environment.

The Model-Free learning can be considered as two parts of ***off-policy***learning and ***on-policy***learning. an agent might be acting using one or two control policies. In *on-policy* control the agent is evaluating and simultaneously improving the exact policy that it follows. Conversely, in *off-policy* control, the agent is following one policy, but may be evaluating another – it is following a behavior policy while evaluating a target policy. In our work we will implement some *off-policy* algorithms alongside an algorithm from the tree search area called MCTS for comparison and evaluation.

### *Stochastic Games*

In this paper, two-player zero-sum Stochastic Games (SGs) are considered. These games proceed like MDPs, with the exception that in each state, both players select their own actions simultaneously, which jointly determine the transition probabilities and their rewards. The zero-sum property restricts that the two players’ payoffs sum to zero.

A *Stochastic Game* (SG) is a tuple , Where:

* N is the number of the players/agents
* T: is the transition function
* is the action set for the player
* is the discount factor
* : is the reward function for player

The objective of the n agents is to find a deterministic joint policy (aka. joint strategy aka. strategy profile) (where ) so as to maximize the expected sum of their discounted payoffs. The Q-function, , is the expected sum of discounted payoffs given that the agents play joint action in state and follow policy thereafter. The optimal -function ­ is the -function for (each) optimal policy . So, captures the game structure. The agents generally do not know in advance. Sometimes, they know neither the payoff structure nor the transition probabilities.

For example, consider a zero-sum game with two players, one player (Player 1) wants to maximize his/her total reward, the other (Player 2) would like to minimize that amount. Similar to the case of MDPs, the reward can be discounted or undiscounted, and the game can be episodic or non-episodic.

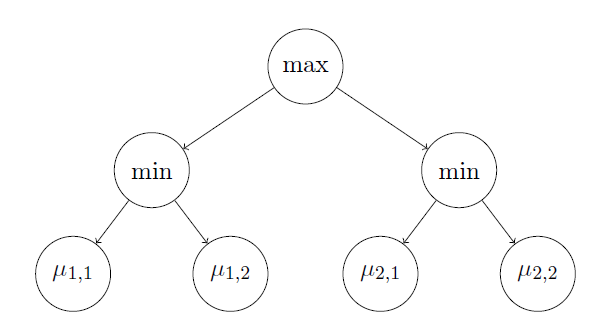


Figure 1 – Game tree when there are two actions by player

We consider a two-player two-round zero-sum game, in which player A has available actions. For each of these actions, indexed by , player B can then choose among possible actions, indexed by . , when player A chooses action and then player B chooses action j, the probability that player A wins is . We investigate the situation (see Figure 1 for an example) from the perspective of Player A, who wants to identify a maximin action

Assuming that Player B is strategic and picks, whatever A’s action , the action minimizing , this is the best choice for A.

### *Nash Equilibrium in SGs*

"In Game Theory, A Nash Equilibrium is a stable state of a system that involves several interacting participants in which no participant can gain by a change of strategy as long as all the other participants remain unchanged"   
Princeton University

A Nash equilibrium is a joint strategy where each agent’s is a best response to the others. For a stochastic game, each agent’s strategy is defined over the entire time horizon of the game.

Given a with players, a Nash Equilibrium is a tuple of strategies such that for all and ,

Where, is the set of strategies available to agent , And,

.

Is the discounted sum of rewards, with discount factor .

A Nash equilibrium is strict if the inequality above is strict. An optimal Nash equilibrium is a Nash equilibrium that gives the agents the maximal expected sum of discounted payoffs.

In the literature, SGs are typically learned under two different settings, and we will call them online and offline settings, respectively. In the offline setting, the learner controls both players in a centralized manner, and the goal is to find the equilibrium of the game [9]. This is also known as finding the worst-case optimality for each player (a.k.a. maximin or minimax policy). In this case, we care about the sample complexity, i.e., how many samples are required to estimate the worst-case optimality such that the error is below some threshold. In the online setting, the learner controls only one of the players, and plays against an arbitrary opponent [10]. In this case, we care about the learner’s regret, i.e., the difference between some benchmark measure and the learner’s total reward earned in the learning process. This benchmark can be defined as the total reward when both players play optimal policies [3], or when Player 1 plays the best stationary response to Player 2. Some of the above online-setting algorithms can find the equilibrium simply through self-playing.

### *Learning in SGs*

Learning in stochastic games can be formalized as a multi-agent reinforcement learning (MARL) problem. we can say that the goal of RL is to learn equilibrium strategies through interaction with the environment.

Our work focuses on competitive settings with partially-observable MARL that has received limited attention throughout the years [2]. There were works include model-free gradient-ascent based method [3][4], simulator-supported methods to improve policies using a series of linear programs [5], Recent scalable methods use Expectation Maximization to learn finite state controller (FSC) policies [6].

The most interesting approach I've found related to our problem of competitive relation between the agents and partial observability framework is described in DEC-HDRQNS [7], that means a Decentralized Hysteretic Deep Recurrent Q-Networks model. Their approach is model-free and decentralized, learning Q-values for each agent. In contrast to policy tables or FSCs, Q-values are amenable to the multi-task distillation process as they inherently measure quality of all actions, rather than just the optimal action.

The proposed approach takes into consideration the concept of Hysteresis (lag) [8].  
Overly-optimistic MARL approaches completely ignore low returns, which are assumed to be caused by teammates’ exploratory actions. This causes severe overestimation of Q-values in stochastic domains.  
Hysteretic Q-learning, instead, uses the insight that low returns may also be caused by domain stochasticity, which should not be ignored. This approach uses two learning rates: nominal learning rate, α, is used when the TD-error is non-negative; a smaller learning rate, β, is used otherwise (where 0 < β < α < 1).

In our work we will take the proposed methods under consideration and will cover some RL algorithms in stochastic games, like:

* Minimax-Q learning [1] (based on linear programing duality)
* Double Deep-Q network
* Monte Carlo Tree Search (MCTS)
* Combination of RL with MCTS
  + MaastCTS2
  + OLMCTS
  + SARSA-UCT
* Decentralized Hysteretic Deep Recurrent Q-Networks model

### *The GVG-AI competition*

The General Video Game AI competition (GVGAI) was created in order to test these general agents on a multitude of real-time games (both stochastic and deterministic) under the same conditions and constraints. It has received significant international attention in the seven years it has been running and has allowed for many interesting algorithms to be tested on the large number of problems.

The GVG-AI Competition explores the problem of creating controllers for general video game playing, in such platform, researches have an opportunity to test their agents via participating in the competitions.

The past few years have led to some great RL algorithms like 'MaastCTS' and 'OLMCTS' [11], both based the MCTS [12] algorithm. All of these have proven themselves in the competitions, therefor, might be useful and beneficial to our research.

The platform of GVG-AI letting the competitors test their algorithms on some environments built specially for them (by DeepMind) to challenge and push them to their limits. By going over all the proposed environments, I found some games with a lot of resemblance to us, like in our research, there were two players in a zero-sum stochastic game alongside the fact that the action space of the agents is much like ours, for the illustration in this paper I want to elaborate on two games that might be of our interest:

1. Ghostbusters – a version of the known Atari game with improvements to satisfy the competition demands.
   1. Players:
      1. One player is the ghost
      2. The other is the hunter
   2. Actions:
      1. The ghost can pass through walls and wraps around the level
      2. The hunter shoots missiles and moves faster than the ghost
   3. Goal:
      1. The aim of the ghost is to either avoid dying or catch the hunter
      2. The goal of the hunter is to avoid the ghost that can hurt him and shoot the ghost
   4. Algorithms with best results:
      1. MCTS
      2. [MaastCTS2](https://github.com/DennisSoemers/MaastCTS2/tree/master/Two-Player/src/MaastCTS2)

The game Ghostbusters briefly described above, reminds our game in many manners, the similarity of the actions the agents take, the ghost that is capable of moving around the grid while targeting to catch the hunter, much like our Light Master (described later). And on the other hand, the hunter that shoot missiles in many directions to try and catch the ghost, just like our Zombie Master (described later).

1. Upgrade-X
   1. Environment:
      1. Two players (agents)
      2. Both of the players are located in a two-dimensional square which they can't leave
   2. Actions:
      1. Both of the players have some laser cannons, which they can move around
      2. If they run into a laser, they lose health points
   3. Goal:
      1. The winner is the player that survives or the one with most points at the end of the game
   4. Algorithms with best results
      1. OLMCTS algorithm
      2. SARSA-UCT algorithm
      3. [MaastCTS2](https://github.com/DennisSoemers/MaastCTS2/tree/master/Two-Player/src/MaastCTS2)

Like in the previous game, the Upgrade-X games have a bunch of similarities with our game like, the two agents moving around with laser cannons are responsible of the canon direction and their goal is to maximize their health/strength points until the end of the game, a process that reminds a lot our Zombie Agent (described later) that is responsible to his zombies direction and velocities alongside the goal of maximizing their strength throughout the game.

## OUR MODEL - Zombie invasion problem

Our learning model is based on a two-dimensional game grid on which the zombies live. In this chapter we'll expand beyond the idea and intuition, we will formally review the rules of the agents, and define the principles of stochastic game in our particular case of two-player zero sum game.

### *Assumptions*

we'll start with some basic assumptions relating the environment and the agent's movement.

Throughout the game, the time and space will be considered discrete while the system operates in discrete time over a horizon T. The system area is represented by N-by-M grid with integer coordinates

Around the board will revolve two types of agents, light and zombies. A zombie and light marking might take coordinates on the integer grid (cells), while at each time moment a zombie can move one cell in right direction.

Furthermore, the light-mark is represented by a square area A-by-A.

Figure 2: grid set-up for example with light-mark of 3x3 and 8 zombies

### *Stochastic game*

As stated above, we deal with two player stochastic game, we will now define the problem we are facing similar to what is stated in the section ‎4.2.

Our stochastic game is defined by the tuple where:

* + 1. is the number of players
    2. is the transition function
    3. is the action set for the player
    4. is the discount factor (for now )
    5. is the reward function for player i

We deal with stochastic two players zero-sum game, i.e.

with limited information on one side (asymmetric information).

### *Agents*

#### Zombie Master

The first player is the Zombie Master, its objective is to maximize the average lifetime of zombies. One way of defining the above is the sum lifetime of all zombies and average over all game rounds.  
In each round, the zombie master must decide on a coordinate Y, where the next zombie should start. Action is an integer number from 0 to N (N is the size of the board). Therefore, the actions it can take are the set: .

The zombie master bases its decisions on the available information which is the matrix of the zombies' locations N-by-M. Each cell of the matrix is 0 (no zombie) or 1 (occupied by zombie). Denote the collection of variables (i.e. observations, actions) available to player 1 at time t by

, where

Denote a subset of all observations until time t and actions until time t-1 by . Therefore contains a set of t N-by-M matrices and a history of choices from the set .

#### Light Master

Another player in the game is the Light Master. Its objective is to minimize the average lifetime of zombies, that is defined similarly to the definition of the Zombie Master: the sum lifetime of all zombies and average over all game rounds.  
In each round of the game, the Light Master has to choose where to put the center of the light (x, y coordinates):

Thus, the space of action in its possession is:

Again, similarly to the Zombie Master, the Light Master must choose an action based on some available information it has like the N-by-M matrix of zombies and zombies history as defined above (see *Zombie Master*).  
In addition, it has information on the strength of the zombies at any given moment and the history of light locations. To sum up, all the information available to him is:

* A tensor (two 2-d matrix) of:
  + A zombie location matrix, such that
  + A zombie strength matrix, with non-empty cells at zombie location, such that

* The mark (light) at time t, , i.e. the player’s action

Therefore, the available information at time t is

Denote a subset of all observations until time t and actions until time t-1 by

### *Game rules*

The game has discrete clock, in each clock tick:

* The zombie master decides where a new zombie will appear. The new zombie's hit points is equal to one.  
  All previous existing zombies are moved right 1 cell, i.e. if for   
  In general,
* Zombies that go over the right boundary disappear
* Each zombie that inside marked region (light) got additional hit of the amount c. Meaning the hit points are increased by c (default value c=1)
* Each remaining zombie heal itself by multiplying hit point by (1-epsilon) factor. Thus:
* Once all hit points are calculated and falling zombies are removed from the board, there is a “kill process” that might remove some zombies from the board
* For each zombie a utility function U is calculated based on the hit point. U produce values from [0,1] such that zombie with no hit point get 0 and zombies with large value of hit points get utility close to 1
* If the outcome is positive, zombie is removed from the board
* The reward (for zombie master) for the round is computed and its equal to the number of zombies that are still in play
* The round ends by plants selecting a new place for the light

## Next steps

* Build the entire environment of the game
  + will contain two agents' possible interactions
  + will be compatible with the openAI gym framework to enable potential of wider research.
* Examine the proposed algorithms
  + DQNs – basic and successful
  + MCTS – from the tree search area – will use for reference
  + Combining RL with MCTS – GVG-AI competition (alphaGo\*, MaastCTS2, OLMCTS, SARSA-UCT etc.)
  + DEC-HDRQNS – proven successful
* Test the results over a different utility and reward functions
* Increase degree of simulation precision
  + Consider round markings, finer resolution, continuous coordinates etc.

## Building the simulation

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.

### Architecture

The two parts of the simulation are: 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:

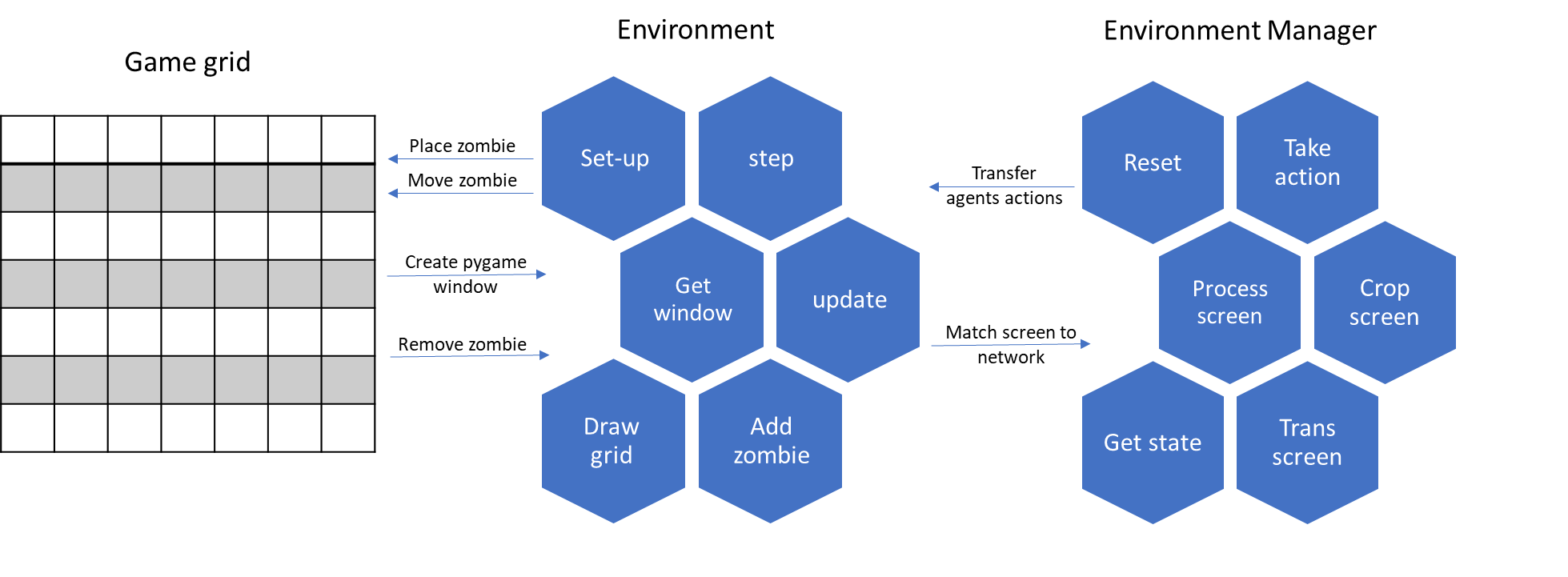


Figure – simulation architecture

#### 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.

Following are the steps of the tests:

* First, we will test the zombie master performance with a random (foolish) light master.
* Therefore, we will test the light master performance with a random zombie master.
* Then, we will test the performance of both the agents trying to learn while playing against each other.

Now, Before we start the tests like mentioned , lets introduce the algorithm we are going to use for learning.

#### The algorithm

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

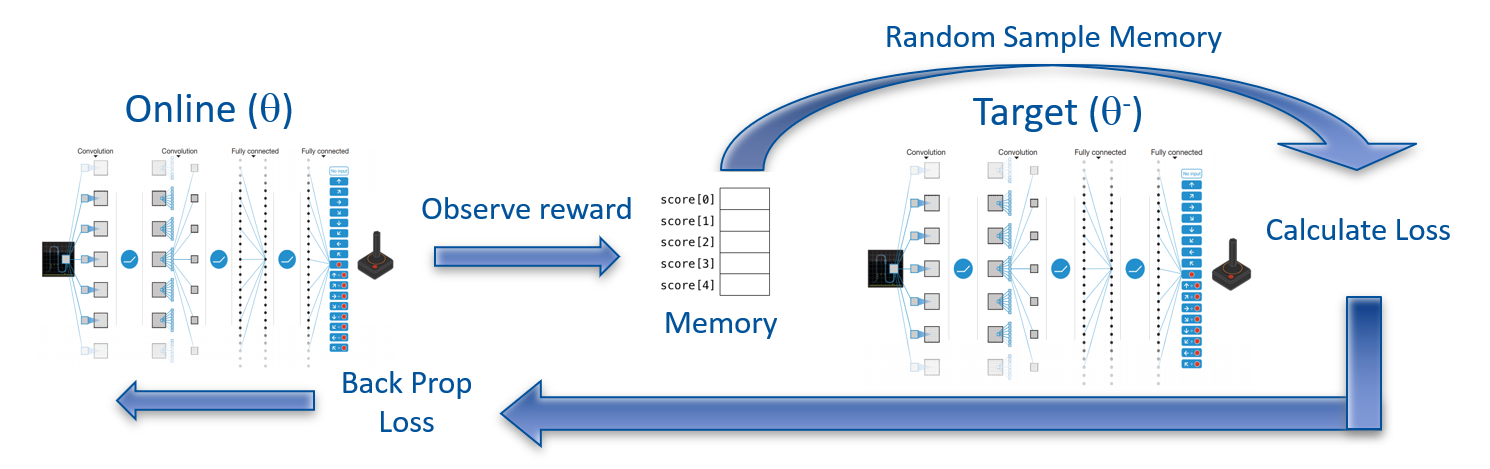


Figure – Double Deep Q-Network architecture and flow

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 4 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:

While the 'start' and 'end' parameters stand for the starting value and ending value of the epsilon function. The 'step' parameter represents the current step of an episode and is multiplied by the 'decay' parameter that is equal to , for achieving the start-end values of the epsilon function.

Which with 200,000 steps looks like: TODO – change to epsilon 0 in the last episodes

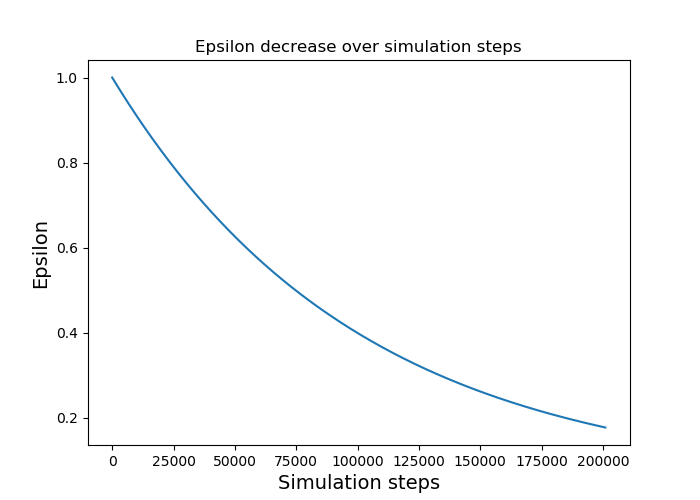
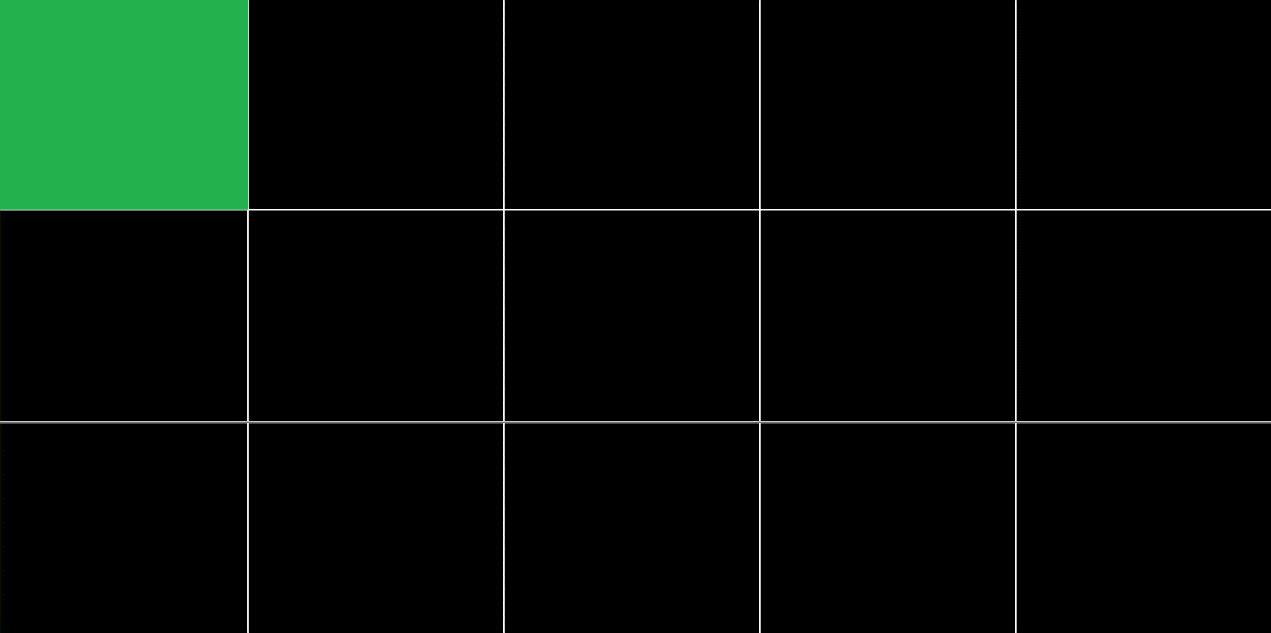


Figure 4 - Epsilon greedy, starts with 1, ends with 0.05 and decay of 0.00001

#### 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 looks like:



0

1

2

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

As we can see in Figure 6 – environment set-up for zombie master performance check, 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:

Table – learning parameters while evaluating the zombie master

|  |  |
| --- | --- |
| Light master action | 0 |
| Target update | 10 |
| Num episodes | 100 |
| Steps per episode | 100 |
| Batch size | 264 |
| Gamma (discount factor) | 0.999 |
| Epsilon-greedy start | 1 |
| Epsilon-greedy end | 0.05 |
| Epsilon-greedy decay | 0.000222 |
| Replay memory size | 1000 |
| Learning rate | 0.001 |

With a deep NN of three layers, all fully connected (called 'Linear' in pytorch formulation): Linear (15,128), Linear (128,128), Linear (128,3). 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):

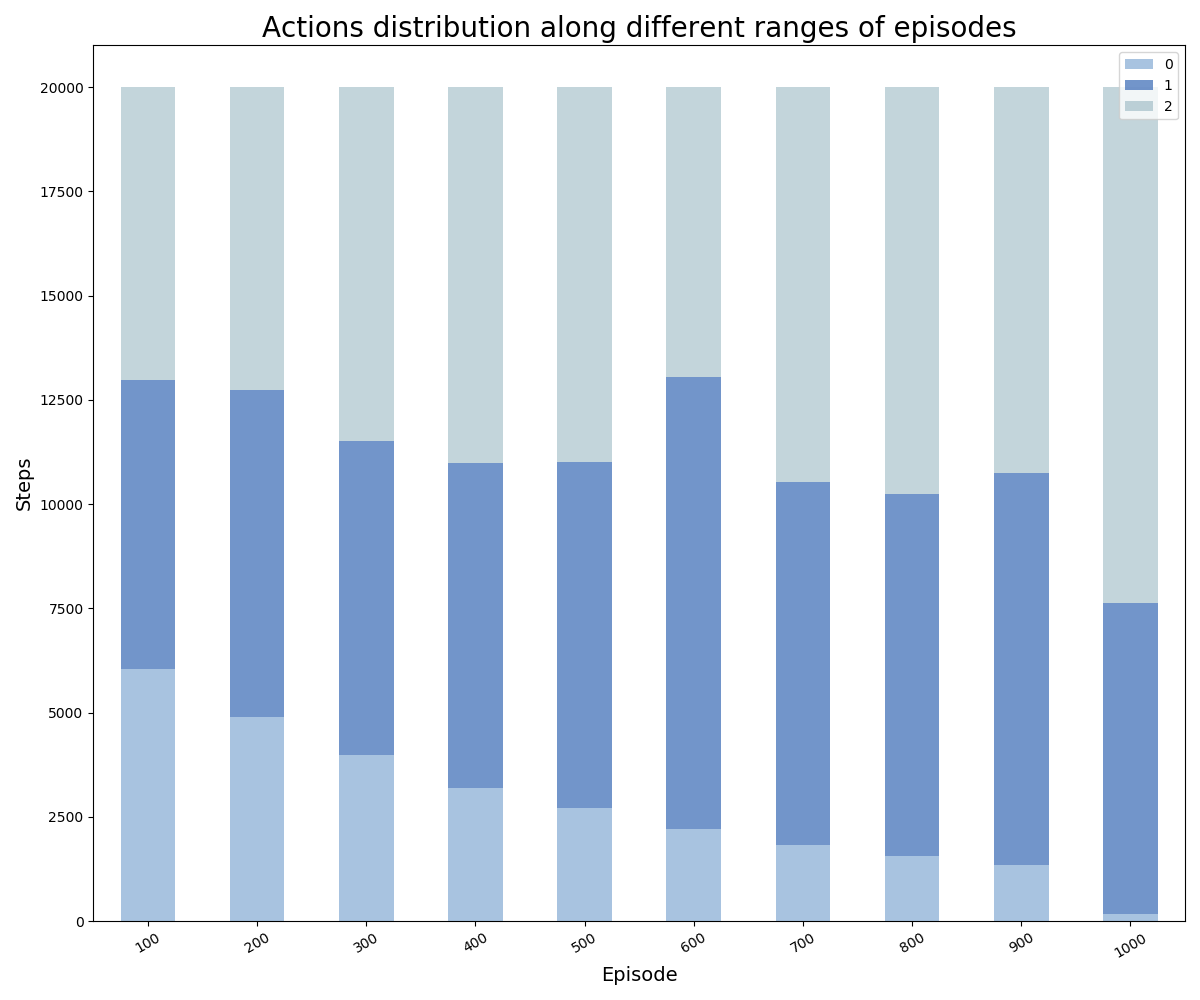


Figure 6 – Zombie master actions distribution along different ranges of episodes

In Figure 7 we can notice the fading of the lower blue which means that the zombie master decides to choose action one or two outright as the episodes go on.

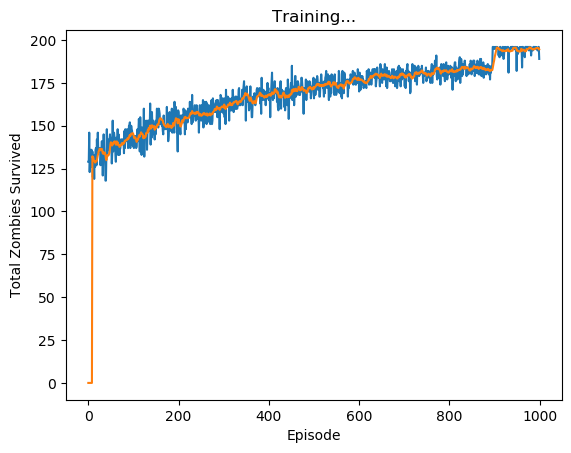
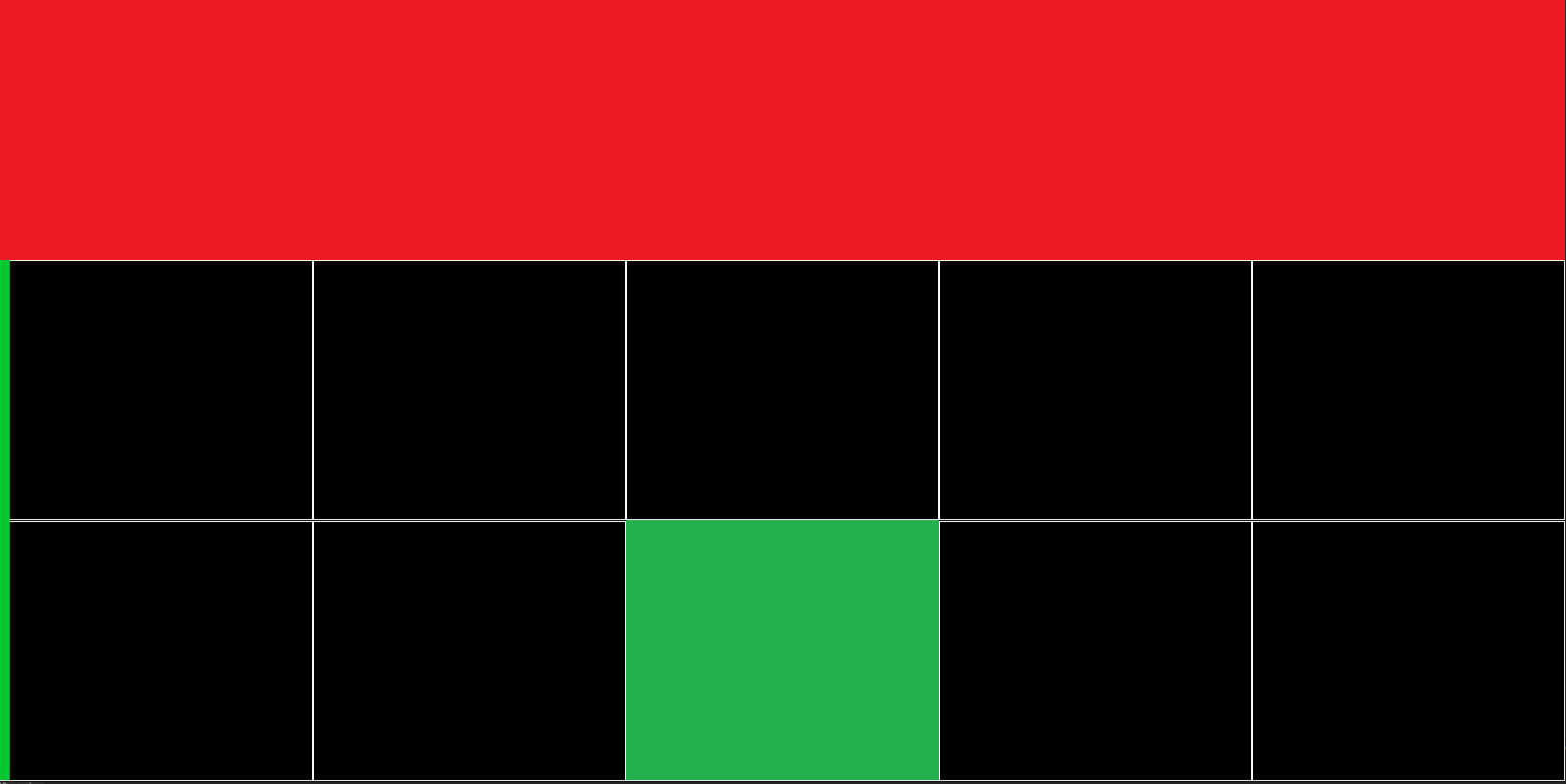


Figure – Total zombies survived vs. the episodes (blue) with its moving average (orange)

Plus, we can tell from Figure 8, the zombie master reaches the maximum it can get – reaches 196 zombies from possible of 196 (there are 201 steps with grid width of 5), we were able to achieve that thanks to the last 100 episodes with zero epsilon greedy parameter.

#### 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 9 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:

Table – learning parameters while evaluating the light master

|  |  |
| --- | --- |
| 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 (15,128), Linear (128,128), Linear (128,15).  
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.

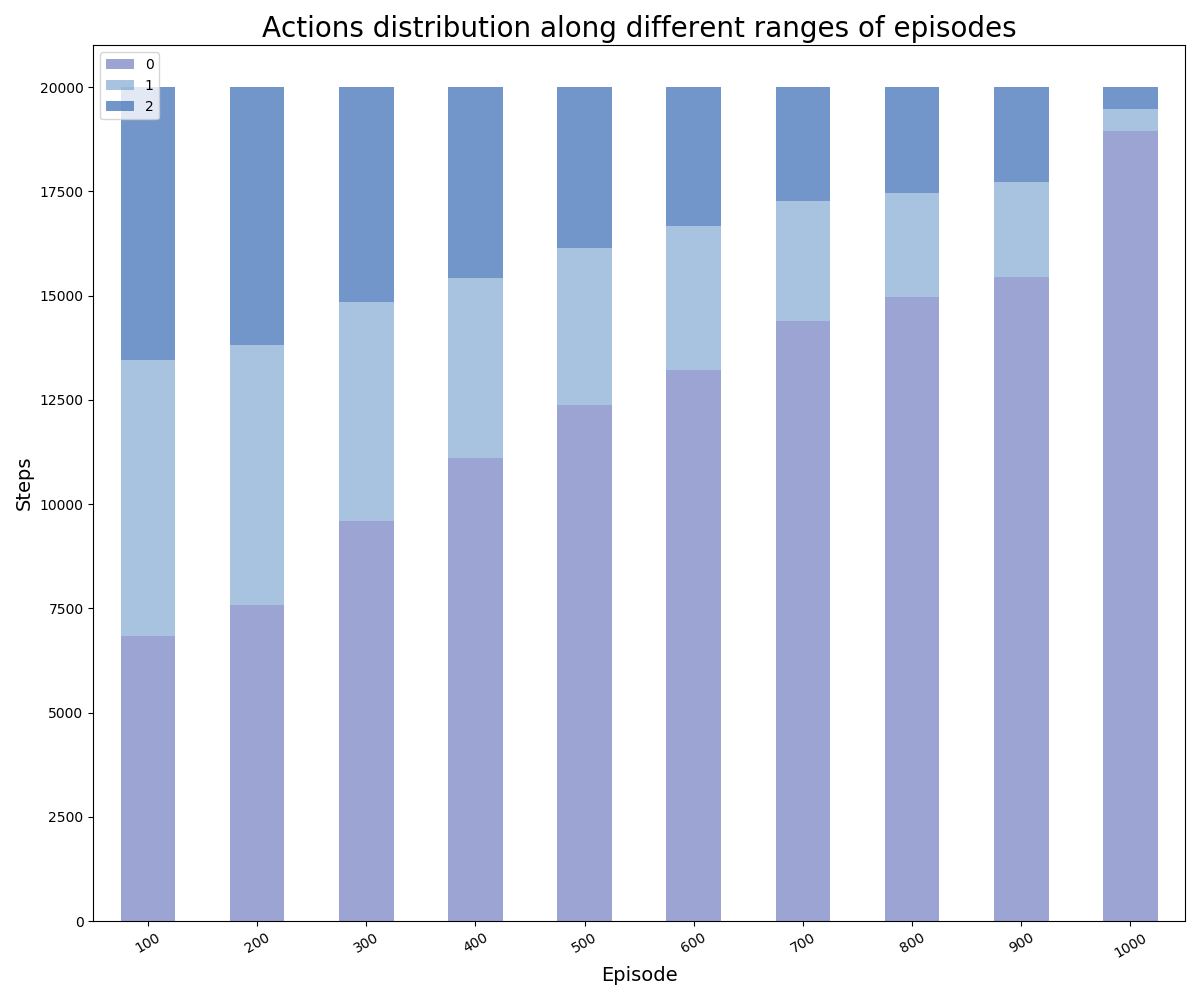


Figure 9 - Light master actions distribution along different ranges of episodes

In Figure 10 we can see the significant increase in the number of times the light agent selected the top row illumination throughout the simulation progress.  
After 900 episodes, the light master chooses to light the correct row more than 90% of the time. Note that the value of the greedy epsilon here is decreasing to 0 at the 900th episode.

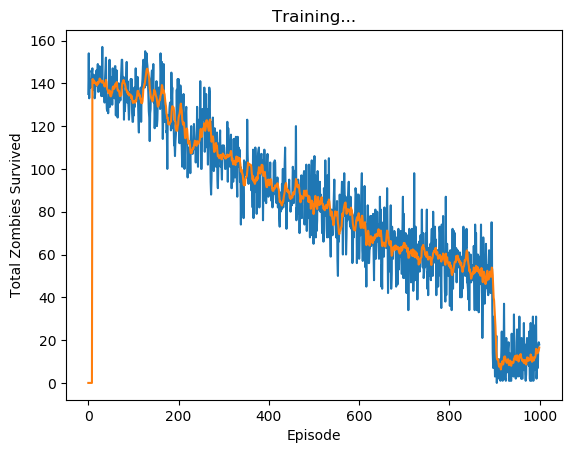


Figure - Total zombies survived vs. the episodes (blue) with its moving average (orange)

In Figure 11 we can clearly see the learning process of the light agent, from the first episodes with 140 zombies survived (out of possible 195 over the episode, it's approximately two thirds), it managed to eliminate over 90% of the zombies by the 1000th episode, Note that the noise remains thanks to the stochastic coin we flip every time step which means that zombies can still survive after getting hit.

## Learning based DDQN

After testing the DDQN algorithm we've implemented on both the agents separately, now we let them learn in parallel and compete with each other.

This chapter will review the results of the agent's success under two variables:

* Fixed board size of 9x9 with different sizes of the light-mark.
* Fixed light-mark with varying board sizes.

### Zombie vs Light – fixed bored size

We are going to review the results of light marks over the range: 3,4,5 and draw conclusions accordingly.

First, lets review the actions distribution over the different scenarios:

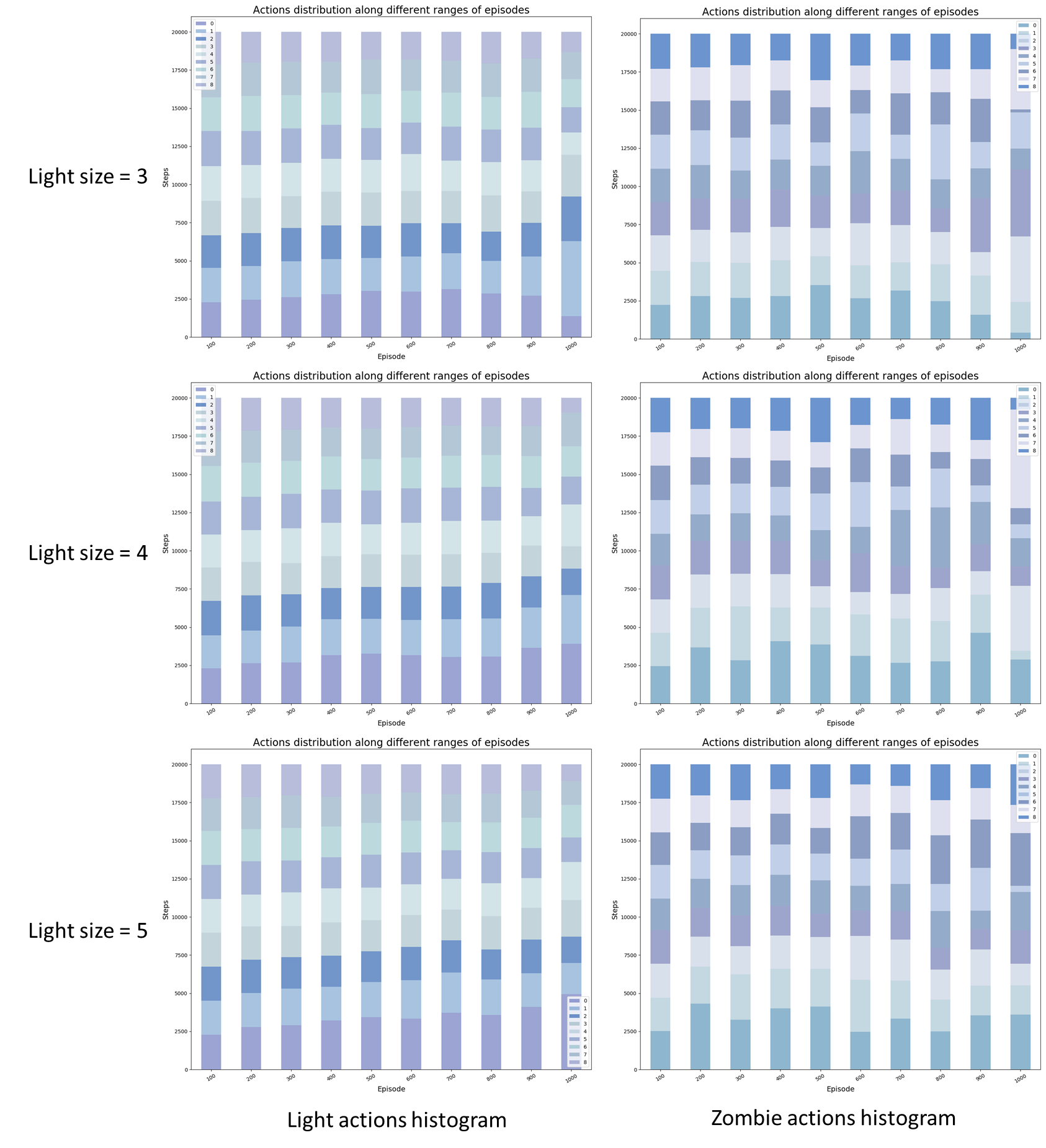


Figure - agents actions over varying light sizes

In Figure 11 we can see that the actions distribution of the light master is approximately unite over the possible actions, with an exception with actions: 0,1. The light agent prefers to mark its light over the first rows. As of zombie agent, its decisions are more unique, we can see that with small light size it chooses to send zombies from mainly three-four entries. However, with light size of five the actions over the last 100 episodes distribute unite except two or three (actions 1,2 and 5)

The reward values for the scenarios:

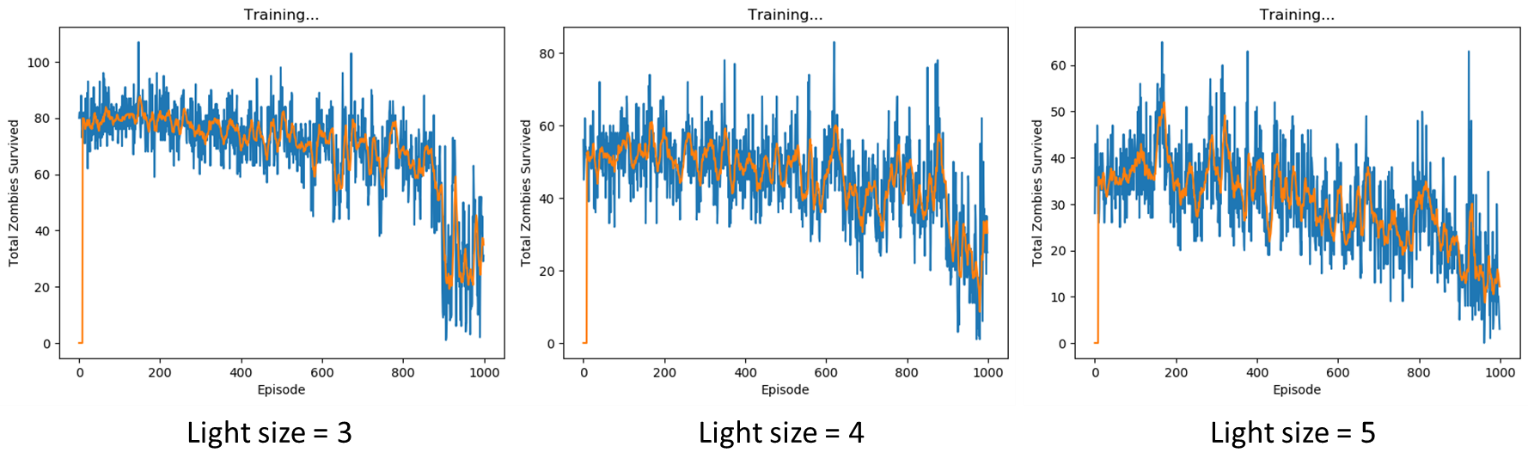


Figure - reward over time with varying light sizes

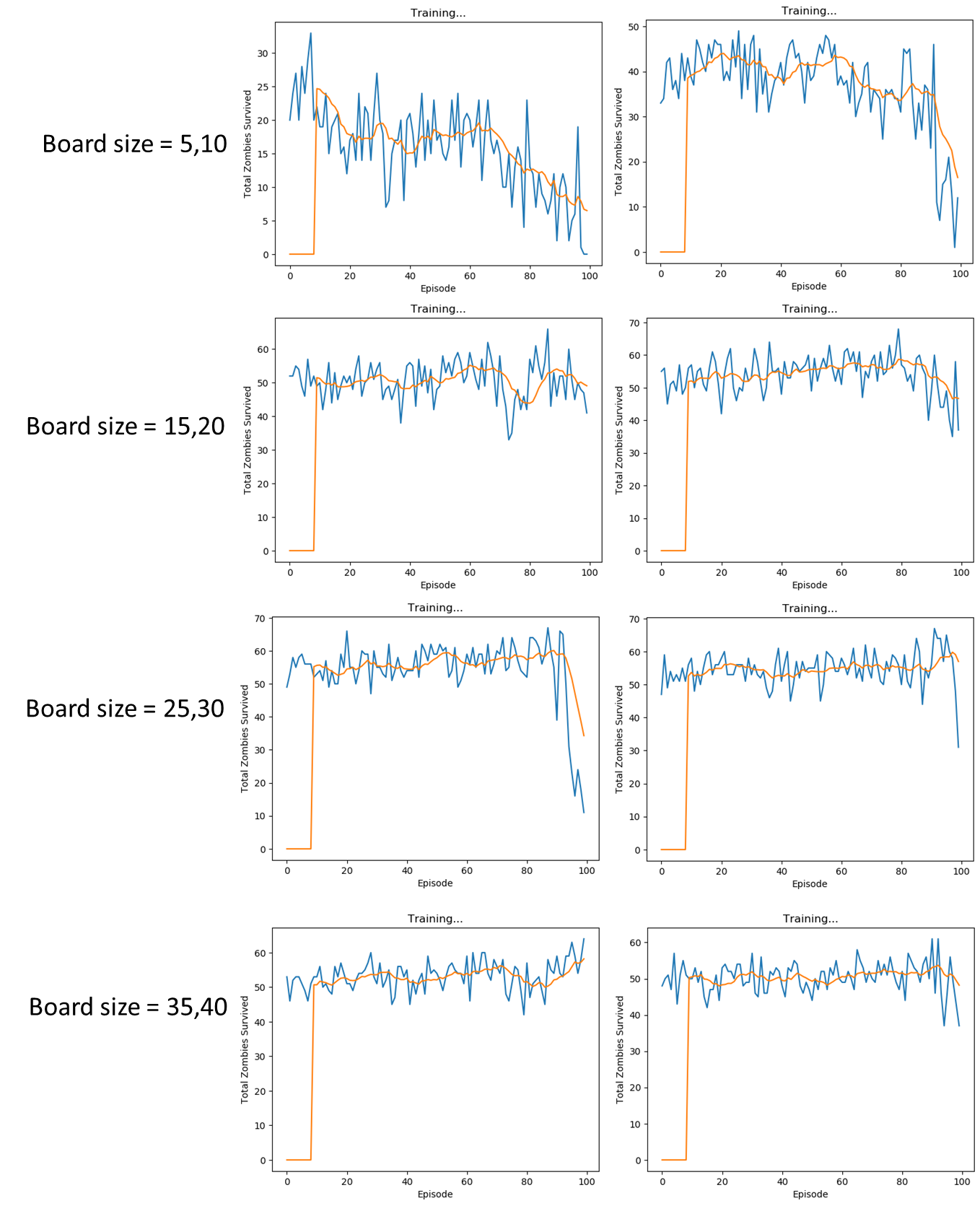


Figure - agents rewards with fixed light size of 3 over 100 episodes

## REFERENCES

[1] Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994* (pp. 157-163). Morgan Kaufmann.

[2] Omidshafiei, S., Pazis, J., Amato, C., How, J. P., & Vian, J. (2017, August). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 2681-2690). JMLR. org.

[3] Peshkin, L., Kim, K. E., Meuleau, N., & Kaelbling, L. P. (2001). Learning to cooperate via policy search. *arXiv preprint cs/0105032*.

[4] Dutech, A., Buffet, O., & Charpillet, F. (2001, August). Multi-agent systems by incremental gradient reinforcement learning. In *International Joint Conference on Artificial Intelligence* (Vol. 17, No. 1, pp. 833-838). LAWRENCE ERLBAUM ASSOCIATES LTD.

[5] Wu, F., Zilberstein, S., & Chen, X. (2012). Rollout sampling policy iteration for decentralized POMDPs. *arXiv preprint arXiv:1203.3528*.

[6] Liu, M., Amato, C., Anesta, E. P., Griffith, J. D., & How, J. P. (2016, March). Learning for decentralized control of multiagent systems in large, partially-observable stochastic environments. In *Thirtieth AAAI Conference on Artificial Intelligence*.

[7] Omidshafiei, S., Pazis, J., Amato, C., How, J. P., & Vian, J. (2017, August). Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70* (pp. 2681-2690). JMLR. org.

[8] Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2007, October). Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 64-69). IEEE.

[9] Matignon, L., Laurent, G. J., & Le Fort-Piat, N. (2007, October). Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 64-69). IEEE.

[10] Conitzer, V., & Sandholm, T. (2007). AWESOME: A general multiagent learning algorithm that converges in self-play and learns a best response against stationary opponents. *Machine Learning*, *67*(1-2), 23-43.

[11] Gaina, R. D., Couëtoux, A., Soemers, D. J., Winands, M. H., Vodopivec, T., Kirchgeßner, F., ... & Perez-Liebana, D. (2017). The 2016 two-player gvgai competition. *IEEE Transactions on Games*, *10*(2), 209-220.

[12] Vodopivec, T., Samothrakis,dz S., & Ster, B. (2017). On Monte Carlo tree search and reinforcement learning. *Journal of Artificial Intelligence Research*, *60*, 881-936.