Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

Project work (Computer Science)

Using Proximal Policy Optimization to play Bomberman

Authors	Jan Sonderegger Patrick Walter
Supervisor	Thilo Stadelmann

Date	21.12.2018

Zusammenfassung

Reinforcement Learning ist ein Teilgebiet der Künstlichen Intelligenz, welches seit Kurzem ermöglicht, einige komplexe Probleme zu lösen. Bei Reinforcement Learning beobachtet ein Agent den Zustand seiner Umgebung und führt darin eine Handlung aus. Die Umgebung wird daraufhin ihren Zustand ändern. Weiter erhält der Agent von der Umgebung eine Rückmeldung über sein Handeln in Form einer Belohnung oder Bestrafung. Dieser Ablauf wird ständig wiederholt und der Agent versucht seine Handlungen so zu verbessern, dass sich die Belohnung maximiert. Dies ist der grundlegende Ansatz von Reinforcement Learning. Diese Arbeit verfolgt das Ziel, einem Agenten mit diesem Ansatz das Spiel Bomberman beizubringen. In diesem Spiel treten vier Spieler gegeneinander an und versuchen sich durch geschicktes Platzieren von Bomben zu eliminieren. Durch eine Implementation des Proximal Policy Optimization (PPO) Algorithmus zusammen mit einem Convolutional Neural Network (CNN) versucht diese Arbeit dieses Ziel durch immer komplexer werdende Experimente zu erreichen. PPO ist ein vielversprechender, neuer Algorithmus, welcher in der Vergangenheit schon erfolgreich für Multiplayer-Spiele angewendet wurde. Die Resultate der Experimente zeigen, dass dieser Ansatz Potential hat das Spiel zu lernen. Es sind aber weitere Untersuchungen nötig, um dies abschliessend zeigen zu können.

Abstract

Reinforcement learning is a research field in artificial intelligence (AI), which was able to solve a number of complex tasks recently. In reinforcement learning, an agent observes the state of his environment and takes actions in this environment. The environment is going to change its state because of the action taken and will give feedback to the agent about his action. It does this either as a positive or negative reward. This sequence of observing, taking action an receiving feedback is repeated continuously and the agent tries to optimize his behaviour to gain the maximum reward. This is the fundamental idea behind reinforcement learning. The goal of this work is, to teach an agent to play Bomberman with this approach. Bomberman is a game played by four players, trying to eliminate each other by cleverly placing bombs on the board. With an implementation of the Proximal Policy Optimization (PPO) algorithm together with a Convolutional Neural Network (CNN), this work tries to reach the goal by conducting more and more complex experiments. PPO is a promising, new algorithm that was successfully used for other multi-player games. The results of the experiments show that this approach could potentially be able to learn Bomberman. However, further research is needed to definitely answer this question.

Preface

When we chose to work on an AI project we barely had any knowledge about the subject. We read some ideas behind it and were eager to learn more about the different approaches that can be used to solve AI problems. We thought that implementing an algorithm that could solve AI problems would surely challenge our software engineering skills. When we talked to our supervisor Thilo Stadelmann he confirmed that this project work would be a true challenge and would suit our interests.

Before we started to dive into reinforcement learning we talked to Joram Liebeskind and Silvan Wehner. They were both still students at the time and just completed their bachelor thesis about using reinforcement learning to play the game Bomberman. They provided us with helpful information about the project and gave useful answers to our questions. At this point we want to thank them for their time and for sharing the most important aspects about their thesis with us.

While we still collected information about the topic at the start of our project work Joram Liebeskind provided us with his updated and cleaned up code. Additionally he provided us with updated information as he worked for three more months on his approach and suggested to use a policy based algorithm. We pursued his suggestion and we really appreciated the updated code which helped us a lot to get started.

Even with this help we needed a lot of time to set up the experiments. And even after our first proof of concept experiment worked we had some problems getting the second experiment to work which cost us a lot of time. In the end we found some bugs in our code and had to rewrite a lot. Struggling with the implementation of the used technologies is the main reason why we could not complete as many experiments as we wanted. It was also at this point in our project where we learned the most about the used methods. We would not understand the ideas as good as we do now and would not be able to write our work in such detail. We want to thank Katharina Rombach who helped us fixing our implementation and who we could ask anytime when we struggled understanding a reinforcement learning concept.

We want to thank our supervisor Dr. Thilo Stadelmann. This work would not be as detailed and as processed as it is now without his help. He gave us additional information when we needed it, explained machine learning concepts if we did not understand them and motivated us to create the best possible version of this work. We also want to thank Janot Angehrn, Milovan Mijatovic and Urs Sonderegger for proofreading this work.

Zurich University
of Applied Sciences

School of Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirm that this work is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the project work have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date: Signature:

....

.....

.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all project works submitted.

Contents

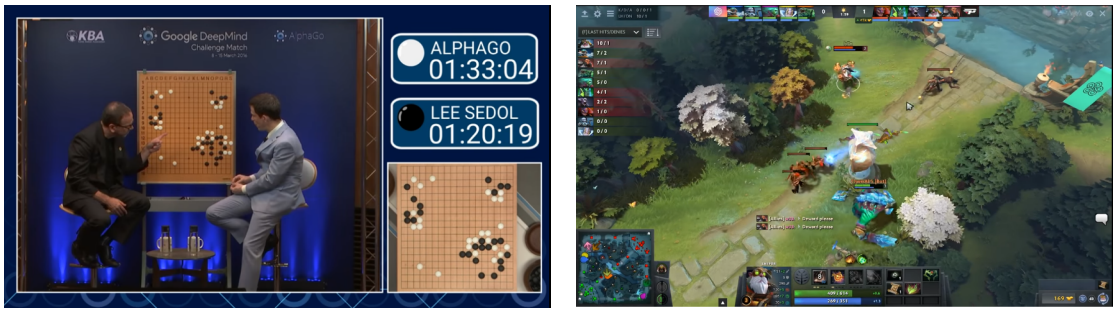
1. Introduction

1.1. Motivation

Artificial Intelligence (AI) is one of the most discussed research fields at the moment. DeepMinds publication [Rie13] in 2013, where they introduced an algorithm that was able to play a number of Atari games better than a human player, has drawn the attention to reinforcement learning as well. Their algorithm learned to play the Atari games by seeing raw pixel images of the game as input and receiving a reward for its performance. The algorithm did not require any previous knowledge about the rules of the games.

In 2015, DeepMind [SHM+16] published an article on an AI called AlphaGo that was able to beat the reigning European Go champion. Figure 1.1a shows a game of Go played between AlphaGo and human professional Lee Sedol. Go is a strategy board game where two players try to surround more area of the board than their opponent. Solving this problem was thought to be a milestone for AI because of the high number of possible game states and difficulty of evaluating board positions and moves. It was believed that reaching this milestone would take another decade.

OpenAI has launched the Project OpenAIFive which uses a reinforcement learning algorithm called PPO to master the esport game Dota2. Dota2 is a strategy game where two teams of five players compete against each other. OpenAIFive has beaten amateur teams in the past and also competed against professional teams at The International 2018[1]. Figure (1.1b) shows a screenshot taken from a game against professional human players at this event.



(a) Screenshot from the livestream of the Google Deepmind challenge match 5 against professional player Lee Sedol. [Dee16]

(b) Screenshots from the match Humans vs OpenAI at The International 2018. [Dot18]

Figure 1.1.: Description of the games Go and Dota2.

[1]The International 2018 was a Dota2 championship held in Vancouver in August 2018. (https://www.dota2.com/international/overview/?l=english)

The remarkable achievements mentioned above, show that a few complex problems were solved by using reinforcement learning. This motivates to apply such a reinforcement learning approach to a complex problem like a multiplayer game.

1.2. Problem Formulation

In their bachelor thesis Liebeskind and Wehner [Weh18] have used the Deep-Q-Network (DQN) algorithm to play Bomberman. Bomberman is a computer game for four players, released by Nintendo in 1985. The goal is to destroy the other players by dropping bombs on the board, the last man standing is the winner of the game.
As stated in their thesis, they were not able to proof that the task is solvable with DQN, and they suggested exploring other algorithms. In this thesis, we want to answer the following question: Can Bomberman be solved by a policy gradient method like the Proximal Policy Optimization (PPO) algorithm?

To answer this question, the PPO algorithm will be evaluated in the same Bomberman environment. For the evaluation, several parameters must be altered systematically by running different experiments.

The above stated question is successfully answered if the parameter values and a Neural Network can be found that are able to learn and defeat simple computer controlled players consistently.

2. Fundamentals

This chapter summarizes the foundational theory required to understand the paper. It does not cover the topics in all detail, for further information we suggest to read the provided sources.

2.1. Mathematical Notation

The following mathematical notations are a summarization of the most important symbols that are used in the equations. Every symbol is explained in the corresponding chapter.

Reinforcement Learning Basics

$s \in S$	States
$a \in A$	Actions
$r \in R$	Rewards
π	Policy
v	State-Value function
q	Action-Value function
γ	Discount factor
G	Expected future rewards function
R	Reward function
P	State transition probability

Policy Gradient

α	Step-size
θ	Policy parameter
ω	Value function parameter
∇	Gradient Symbol
J	Reward function
A	Advantage function

Convolutional Neural Network

w	weight
b	bias
σ	Activation Function

Probability Theory

$\hat{\mathbb{E}}$	Expectation
$\hat{\mathbb{P}}$	Probability

General

$min()$	Minimum function
$\sum$	Summation

2.2. Reinforcement Learning

The most similar approach to how humans learn is called reinforcement learning. We explore our world and learn what is good or bad behaviour. Reinforcement learning works quite similar. This chapter covers the basics of reinforcement learning.

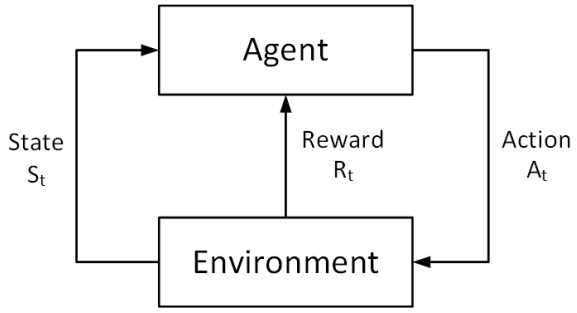


Figure 2.1.: The Reinforcement learning feedback loop. The agent gets the current state from the environment and determines what action to do. The environment gets that action and sends the new state. If the action led to a reward the environment sends it at the same time as the current state. Adapted from [Sil15, Lecture 1 S.23].

2.2.1. Difference Between Supervised and Reinforcement Learning

In supervised learning the agent gets a set of input data and the corresponding output. The agent now has to learn to produce the correct output from the input data. We can tell the agent if he predicted something right or wrong because we already know the correct output. At the end the agent has learned a function which can be applied to a input set to get a predicted output. If we would apply supervised learning to a game like Bomberman we would let a good human player play the game over a long period of time and would record each frame. A frame would be the input and the corresponding action that the players decides to make the output. There are a few problems with this approach. First the agent would never be better than the human player because the agent only learns to imitate the actions that the human player does. Also creating an input-output set would be complicated because the actions which are normally executed with an input device have to be recorded in sync with the frames on the screen.

Another type of learning is reinforcement learning. The agent explores the environment by itself and gets a positive or negative reward depending on how the agent interacts with the environment. Figure (2.1) shows this interaction between agent an environment in detail. The goal of the agent is to maximize the total number of rewards. In a game like Bomberman the agent would get a positive reward if it accidentally killes an enemy and would learn over time by playing multiple games that killing an enemy is a good thing. This method solves the two problems of learning to play a game with supervised learning. We do not have to create an input data set because we only define the rewards and not an output for an input frame. The agent does not imitate a human player and can therefore be better than a human and develop new strategies by himself. [RN10, Chapter 18.1]

2.2.2. Basic Concepts and Terms

To fully understand reinforcement learning the associated concepts and terms are explained in the following [Sil15, Lecture 1, 2]:

Agent An agent picks an action and executes it. Which action to pick is defined by the algorithm. In video games the playable character is the agent.

Environment The environment is the world that the agent explores. It takes actions from the agent and applies them to its current state. The new state is then sent back to the agent. If the player did something good or bad we also return a positive or negative reward respectively.

Action An action is something an agent can do in the environment. There are multiple actions to choose from and the agent has to decide which action to take in which state. In video games the actions are normally tied to a button press. Deciding to perform a specific action by pressing a specific button as a human player is the same as an agent that decides which action to take.

State The state is a description of the current situation in the environment. In video games the state may contain the position of the player, the position of obstacles and the position of enemies.

Reward The reward is either a positive or negative number that the agent gets from the environment. The number defines if the agent did something good or bad in the environment and how good or bad it was. The reward is not necessarily tied to the latest action the agent did but rather is a result from the actions that the agent has done. So the agent also has to decide which actions led to getting the reward. In a game like Bomberman the bomb is placed some seconds before it explodes. If the bomb kills an enemy when it explodes the agent may walk to the left. But the reason that the agent walked to the left is not the reason that he got a reward.

Policy The policy defines how the agent picks its actions. A deterministic policy is just a function $a = \pi(s)$ that defines which action a to take in state s. The policy can also be stochastic. $\pi(a|s)$ is the probability to take a specific action in a specific state.

State-Value function The state-value function defines how good it is to be in a particular state. Basically it is an expectation of the rewards the agent gets in the future based only on the current state.

$$v_\pi(s) = \hat{\mathbb{E}}_\pi[G_t|S_t = s] \tag{2.1}$$

G_t is the expected sum of future rewards that are discounted the more they are in the future.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.2}$$

$\gamma \in [0, 1]$ is the discount factor that disvalues rewards the more they are in the future. A discount can have different reasons. It may be because future rewards have a higher uncertainty or because the agent should prefer short-term rewards over long-term rewards. The lower the discount factor is the more the agent prefers short-term rewards.

Action-Value Function The action-value function is the expectation of future rewards when taking action a in state s and then following policy π.

$$q_\pi(s, a) = \hat{\mathbb{E}}_\pi[G_t|S_t = s, A_t = a] \tag{2.3}$$

2.2.3. Markov Decision Process (MDP)

To solve a problem with reinforcement learning it has to be a Markov Decision Process (hereinafter referred to as MDP). A MDP is defined as follows [Sil15, Lecture 2]:

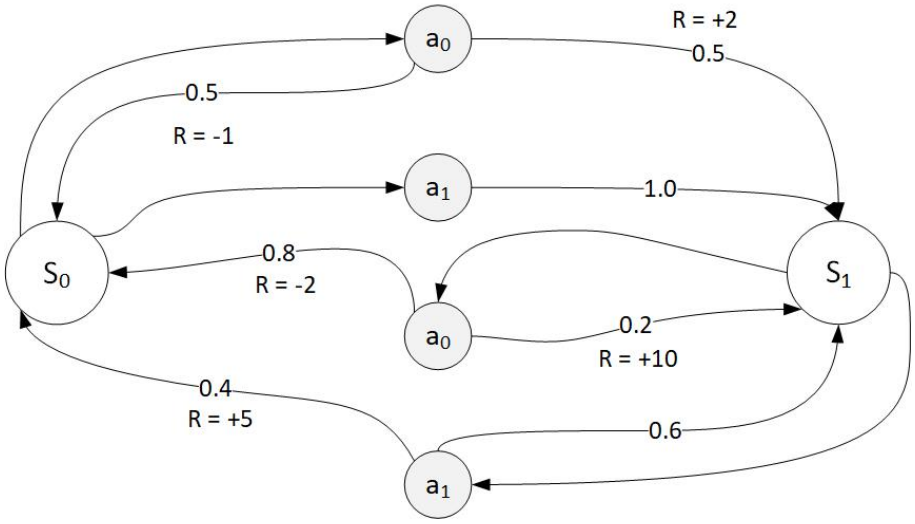


Figure 2.2.: A MDP with two states, two actions and four rewards.

Definition 1. A MDP is a tuple that contains five elements $< S, A, P, R, \gamma >$.

- S is a finite set of states.
- A is a finite set of actions.
- P is a state transition probability matrix where every entry defines how high the probability is to change from state s to a future state s' when taking action a: $P^a_{ss'} = \hat{\mathbb{P}}[S_{t+1} = s' | S_t = s, A_t = a]$.
- R is a reward function that returns the expected reward when taking action a and going to state s: $R^a_s = \hat{\mathbb{E}}[R_{t+1} | S_t = s, A_t = a]$.
- γ is the discount factor.

Additionally every state in the environment should have the Markov property.

Definition 2. A state S_t is Markov if the next state is independent from past states and only depends on the present state.

$$\hat{\mathbb{P}}[S_{t+1} | S_t] = \hat{\mathbb{P}}[S_{t+1} | S_1, ..., S_t] \tag{2.4}$$

A basic example of a MDP is shown in figure (2.2). In every state we can pick between two actions. Depending on the action we take there are different probabilities in which state we go after taking that action. We also get a reward if we reach a specific state through a specific action.

2.3. Policy Gradient

There are many different approaches for solving reinforcement learning problems. One popular approach is to learn the state or action value function. The DQN algorithm which was used by Liebeskind and Wehner falls into this category. [Weh18] Policy gradient is

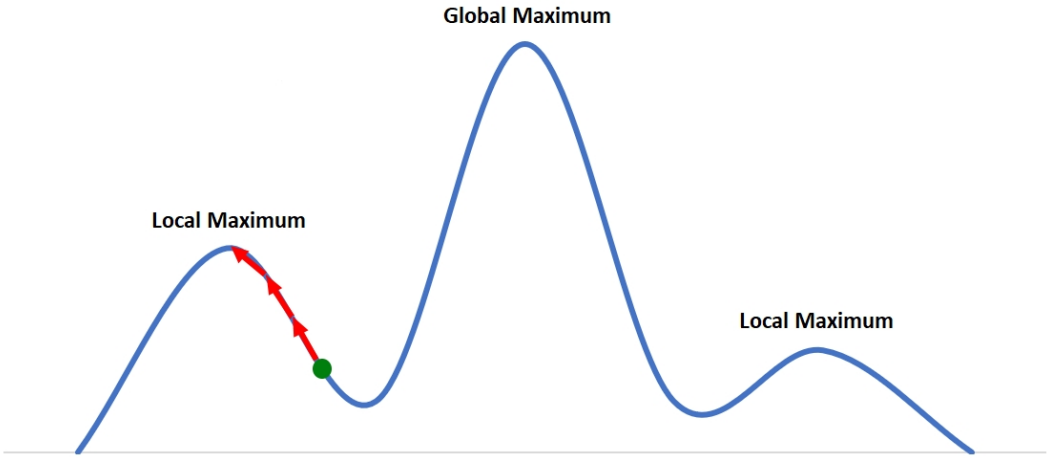


Figure 2.3.: A function with two local and one global maximum. The green point ascends to the local maximum and will never know about the global maximum.

another approach that optimizes the policy directly to get the best reward. A policy function $\pi_\theta(a|s)$ is used where the parameter θ is updated with the gradient $\nabla_\theta J(\theta)$:

$$\theta' = \theta + \alpha \nabla_\theta J(\theta) \tag{2.5}$$

The gradient is calculated using the Policy Gradient Theorem which is the foundation for many policy gradient algorithms. The proof of this theorem can be found in [SB17, Chapter 13.2].

$$\nabla_\theta J(\theta) = \hat{\mathbb{E}}_{\pi_\theta}[\nabla \ln \pi_\theta(a|s) q_\pi(s,a)] \tag{2.6}$$

Additionally α is used as the step size.

By ascending the policy like this the algorithm can find the local maximum but is not guaranteed to find the global maximum. It depends on where θ starts. If the policy update is too big the maximum can be missed. Figure (2.3) shows an example of local and global maximum in a function. [Res15] [Wen18]

2.3.1. Actor-Critic

While policy gradient algorithms learn the policy they can additionally learn the value function. Such algorithms are called action-critic algorithms. They consist of two models [Wen18]:

Critic Updates the value function parameter ω. The value function can either be a state-value function $v_\omega(s)$ or an action-value function $q_\omega(s,a)$.

Actor Updates the policy parameter θ in the direction the critic suggests.

A simple actor-critic algorithm can look like this [Wen18]:

Algorithm 1 Simple Actor-Critic Algorithm

Initialize s, θ, ω at random; sample $a \sim \pi_\theta(a|s)$
for $t = 1 \dots T$ **do**
  Sample reward $r_t \sim R(s,a)$ and next state $s' \sim \hat{\mathbb{P}}(s'|s,a)$
  Sample next action $a' \sim \pi_\theta(a'|s')$
  Update the policy parameters: $\theta \leftarrow \theta + \alpha_\theta q_\omega(s,a) \nabla_\theta \ln \pi_\theta(a|s)$
  Compute the correction for action-value at time t:
  $\delta_t = r_t + \gamma q_w(s', a') - q_\omega(s,a)$
  and use it to update the parameters of the action-value function:
  $\omega \leftarrow \omega + \alpha_\omega \delta_t \nabla_\omega q_\omega(s,a)$
  Update $a \leftarrow a'$ and $s \leftarrow s'$
end for

2.4. Proximal Policy Optimization (PPO)

Proximal Policy Optimization (hereinafter referred to as PPO) is a policy gradient algorithm designed for reinforcement learning. The algorithm was published in 2017 by Schulman et al. [JS17] and used in well-known projects such as OpenAI Five. [Ope18] OpenAI[1] has stated that PPO is much simpler to implement and tune than state-of-the-art approaches, while performing comparably or better. [Ope17b] Whereas other policy gradient algorithms do updates after every step taken with a policy, PPO does updates after collecting minibatches of samples. Furthermore, PPO does not allow excessively large policy updates. The algorithm uses a surrogate objective to prevent and penalize large policy steps. [JS17, Chapter 3]

2.4.1. Clipped Objective Function

As discussed in chapter 2.4, a global or local maximum can be missed due to large policy changes. PPO avoids large policy steps by using a clipped objective function. To determine how much the policy should be updated, PPO uses an advantage estimator. [JS17, Chapter 2]

The advantage function is defined by [MBM+16] as follows:

$$\hat{A}_t = q(a_t, s_t) - v(s_t) \tag{2.7}$$

As discussed in chapter 2.2 the action-value function q is defined as:

$$q(a_t, s_t) = r_t + \gamma r_{t+1} + \dots + \gamma^{n-1} r_{t+n-1} + \gamma^n r_{t+n} v(s_{t+n}) \tag{2.8}$$

With $q(a_t, s_t)$ plugged into (2.7), the advantage $\hat{A}_t$ is defined as:

$$\hat{A}_t = -v(s_t) + r_t + (\gamma\lambda) r_{t+1} + \dots + (\gamma\lambda)^{n-1} r_{t+n-1} + (\gamma\lambda)^n r_{t+n} v(s_{t+n}) \tag{2.9}$$

The advantage $\hat{A}_t$ is used in the objective proposed by [JS17]:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.10}$$

[1] OpenAI is a non-profit organization, that conducts research in AI. `https://openai.com/about/`

In equation (2.10), ϵ is a hyperparameter. $r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t,s_t)}{\pi_{\theta_{old}}(a_t,s_t)}$. The term $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)$ clips the probability ratio which removes the incentive for moving r_t outside of the interval $[1-\epsilon, 1+\epsilon]$. By taking the minimum of the unclipped and clipped objective $r_t(\theta)\hat{A}_t$, the cliping only has an impact on positive policy changes and not for negative changes. [JS17] Figure (2.4) visualizes this explanations.

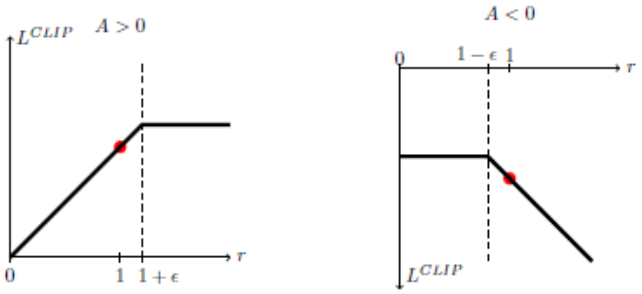


Figure 2.4.: This figure shows a plot of the surrogate function L^{CLIP} for one single term. The plot on the left shows the function for a positive value of $r_t(\theta)$, the plot on the right for a negative value. [JS17]

2.4.2. KL Penalty Coefficient

An alternative to the clipped objective function is using a penalty on KL divergence. The KL divergence $KL(a,b)$is a measure for the distance between two statistical populations a and b.[2] [KL51] Using the KL penalty, the objective function becomes:

$$L^{KLPEN}(\theta) = \hat{\mathbb{E}}[r_t(\theta)\hat{A}_t - \beta KL(\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)) \tag{2.11}$$

Equation 2.11 shows that the divergence between the old policy π_{old} and current policy π are calculated. This is possible since the policies are stochastic. The parameter β is a coefficient that can be adapted. As in the previous chapter, $r_t(\theta)$ denotes the probability ratio $r_t(\theta) = \frac{\pi_\theta(a_t,s_t)}{\pi_{\theta_{old}}(a_t,s_t)}$.
Schulman et al. show in their experiments, that the KL penalty performed worse compared to the clipped surrogate objective. [JS17]

[2] In the PPO paper, there is no statement regarding the exact implementation of the KL-Divergence. We suggest following the provided source for the detailed mathematics.

2.4.3. The Algorithm

The PPO algorithm is described as follows: [JS17]

Algorithm 2 PPO

for *iteration* $= 1, 2, \ldots$ **do**
  for *actor* $= 1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{old}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, ..., \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{old} \leftarrow \theta$
end for

As the algorithm (2) above indicates, PPO uses the same policy π_θ for multiple timesteps T and calculates the advantage $\hat{A}$ for every step before evaluating the performance. This is called a minibatch of sample points. PPO uses these sample points to calculate the surrogate L that has been discussed in the previous chapter to calculate the policy update $\theta_{old} \leftarrow \theta$.

For the usage of PPO with a neural network that shares parameters between the policy π_θ and the value function $v(s_t)$, [JS17] suggest that an error term $L_t^{VF}(\theta) = (v_\theta(s_t) - V_t^{target})^2$ should be added to the objective L. Furthermore, [MBM+16] suggested to add an entropy bonus S to the objective to ensure sufficient exploration. With these terms combined with L^{CLIP} discussed in equation (2.10), we obtain the following objective:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}[L_t^{CLIP} - c_1 L_t^{VF}(\theta) - c_2 S[\pi_\theta](s_t)] \tag{2.12}$$

where c_1 and c_2 are coefficients.[JS17] These coefficients and the cliprange are values that can be modified to tune the learning progress.

2.5. Convolutional Neural Network (CNN)

Convolutional Neural Networks (hereinafter referred to as CNN) are biologically inspired models often used for pattern recognition. [Kan03, Chapter 1] CNNs consist of multiple neurons organized in a number of layers. In this chapter, we discuss neurons and the different layers that were used in this work.

2.5.1. Neurons

Neurons are single nodes in a Neural Network consisting of a high number of these neurons. Each neuron can take multiple input values which are applied to an activation function to produce one output value. Since some input values might be of more significance than others, every input value is multiplied by a weight w_i. Furthermore, every neuron has a bias b_i. This value can be interpreted as a negative threshold value and indicates how fast a neuron will fire. [Nie15, Chapter 1]

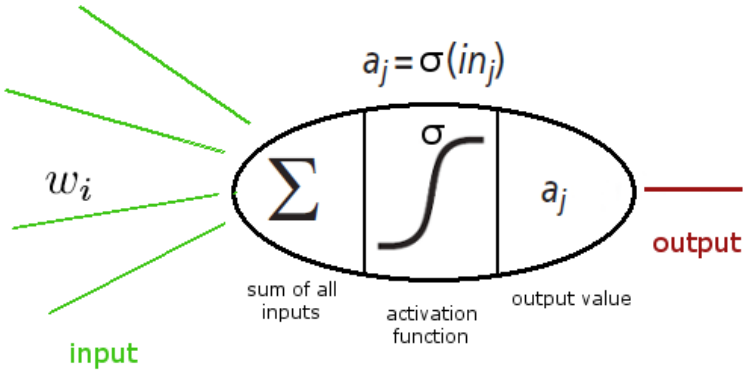


Figure 2.5.: A simple model of a neuron with the sigmoid function σ as activation function. (Adapted from [RN10, Chapter 18.7, p. 728].)

$$z = (\sum_{i=1}^{n} w_i x_i + b) \tag{2.13}$$

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{2.14}$$

$$\sigma(z) = \begin{cases} z, & \text{if } z \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.15}$$

Equation (2.13) shows, how the weight w and bias b are used to transform the input x. The result of this equation is used as an input in an activation function $\sigma(z)$ to compute the value of the neuron. Both functions (2.14) and (2.15) are non-linear activation functions. Function (2.14) is called the sigmoid function, function (2.15) is called ReLu. [Nie15, Chapter 1]

2.5.2. Convolutional Layers

These neurons are organized in different layers. A CNN can consist of multiple layers that are connected to their previous and following layer. The exact number of layers depends on the chosen architecture.
Convolutional layers take images as inputs. The core idea behind convolutional layers are local receptive fields. [Ben95] With these receptive fields, the local correlation between pixels are taken into account: The receptive fields are slid through the input image and create the input value of a neuron in the following layer as illustrated in figure (2.6). One receptive field might find horizontal edges in the image, while a second field discovers vertical edges or other features.[Nie15]

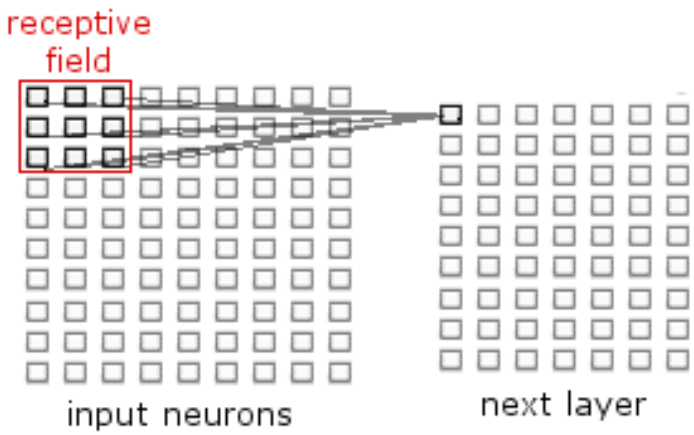


Figure 2.6.: A 3x3 receptive field applied to a 10x10 input image. (Adapted from [Nie15].)

Figure (2.6) indicates that the receptive field is moved by one pixel at a time. In practice, this means the filter has a stride length of one. However, a receptive filter can be moved by more than one pixel, hence the stride length can be greater than one.[Nie15]
Furthermore, figure (2.6) shows that after applying a filter to a nXn dimension layer, the result has a smaller dimension. This might not always be desired, therefore padding can be used. Zero values will be appended to the input image increasing the dimension and therefore also increasing the dimension of the result.

2.5.3. Fully-Connected Layers

In contrast to a convolutional layer, the neurons of a fully connected layer are connected to all the neurons of the previous layer. Usually, the last layer in a CNN is a fully-connected layer, with as many neurons as the CNN has possible outputs. All the neurons of the last convolutional layers are therefore connected to the fully-connected layer and determine the value of the output of the CNN. [Nie15, Chapter 6]

3. Existing Work

[Gra18] presents an architecture that is used in a 3D environment with multiple agents. The agents were able to outperform human players but also learned human like behaviour in the environment. Similar to Pommerman the environment also changes each time a game is over. While they also used a policy gradient method the algorithm is not yet published to the OpenAI baselines. If we wanted to use this algorithm we would have to rebuild it with the information provided by the paper. Also the algorithm is optimized for partially observable environments. Because of this two reasons we decided against using it but it showed us that policy gradients methods can be successful in a multiagent environment.

[Weh18] is a bachelor thesis which was also written at the ZHAW in the semester before this work. Liebeskind and Wehner could not solve Pommerman with their double DQN approach. While they did not succeed they provided helpful information about what methods did and did not work and which methods might work in the future. They also had the same goal as this work, solving Pommerman using a reinforcement learning approach. Because they used an algorithm where they learned the value function we wanted to differ from that and used a policy gradient method. They stated that the CNN architecture is recommended for future work which is why it is also used in this work.

[Rie13] showed that it is possible for an algorithm to learn playing 2D environments with a better performance than humans can. They used a CNN that received the raw pixel as an input and used an reinforcement learning approach to solve this task. While the approach was successful further publications had a higher focus on learning the policy directly. They released [MBM$^+$16] were they showed that an actor-critic algorithm could outperform the original approach.

4. Methods

This chapter introduces the technologies and configurations that are used in the experiments. It summarizes the environment and its properties and features. Additionally the processing of the observations and the used implementations for the PPO algorithm and the CNN are described.

4.1. Pommerman Environment

For this work we used Pommerman as the environment for our experiments. In this chapter we want to introduce the environment. Additionally we want to compare it with Bomberman, the game its based on.

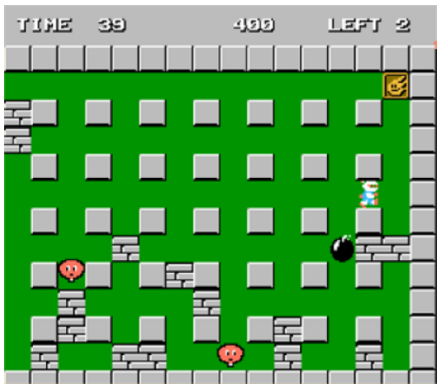
4.1.1. The Original Bomberman Game

The game Bomberman was first released in 1985 on the Nintendo Entertainment System. The goal of the game is to destroy every enemy on the board which in turn unlocks the door to proceed to the next stage. There are unbreakable and randomly placed breakable walls on the board. Additionally the enemies and the door are also randomly placed on the board. The player is able to place bombs on the board to destroy the enemies or the breakable walls. Behind these walls is either nothing, a power-up or the door to the next stage. The player starts with one bomb at a time. The amount of bombs that can be placed at the same time can be increased by a power-up. The blast strength of a bomb is also one in the beginning which means that it will damage every wall or enemy that is in a adjacent field to the bomb. The strength can also be increased with a power-up. With each stage the game gets more and more difficult as new enemy types spawn. The player has a total of three lives and if he loses all of them he has to start all over from stage one without any power-ups. There are many new official Bomberman titles and many clones today but the focus is much more on the versus-mode which was first released in Bomberman II. [Sof85] [Sof92]

4.1.2. The Pommerman Environment

In Pommerman each player starts in a corner of a squared board with 11x11 fields. The board contains destructible blocks and indestructible blocks which are randomly placed on the board. The players goal is to kill each other. To do so the player can place bombs on the board to destroy the destructible blocks and also to kill the other players. If a block is destroyed it may contain a power-up which gives the player that picks it up additional powers. Similar to the original Bomberman the player starts with one bomb with a blast strength of one. The amount of bombs and the blast strength can be increased with a power-up.

Pommerman was developed specifically as an environment for multi-agent learning. The team behind it is called Playgrounds and hosts regular competitions. This year they held a FFA competition on June 3rd where each agent in a game fought against each other. On November 21st they held a Team competition where the agents would play in a team of two against another team in a partially observable environment. [Eld]



(a) Screenshot of the original Bomberman game.

(b) Screenshots of 4 agents playing Pommerman.

Figure 4.1.: Comparison between the original Bomberman game and the Pommerman environment.

4.1.3. State

Every agent gets the following information as the state from the environment [Pla]:

Board The board is represented as a 2D array which has the same size as the board. The representation of every field of the board has a corresponding value that is saved in the array. Every possible representation and the corresponding value is listed in table (4.1).

Bomb Blast Strength A 2D array that is the same size as the board. It is representing the bombs blast strength in the agents view. Everything outside of its view is represented as fog.

Bomb Life A 2D array that is the same size as the board. It is representing the bombs life in the agents view. Everything outside of its view is represented as fog.

Additional Information The following additional information are stored in another array:

Position	The current position of the agent as x and y coordinates.
Ammo	The amount of bombs the player can place.
Blast Strength	The number of fields that are covered in flames horizontally and vertically when the bomb explodes.
Can Kick	Predicates if the agent can kick a bomb or not.
Teammate	A number between 9 and 13 to inform the agent about its teammate. The corresponding agent to the value can be found in table (4.1).
Enemies	Three numbers in a list that represent the agents enemies. Every value in the list is also between 9 and 13.

The following table contains every representation that can appear on a field, the corresponding value and a short description about the representation [Pla]:

Representation	Value	Description
Passage	0	A block which agents can pass. It is displayed as a gray concrete block.
Rigid Wall	1	A block which can not be destroyed by a bomb and is not passable. It is displayed as a red brick block.
Wooden Wall	2	A block which a player can not pass. It is destroyed by a bomb if the block is in the blast length of the bomb. It is displayed as a brown wooden block.
Bomb	3	Can be placed on a passage by a player. Makes the passage impassable for 10 timesteps and will explode afterwards.
Flames	4	Is displayed for 3 timesteps after a bomb explodes. If a player touches it the player will die.
Fog	5	Makes the environment partially observable by covering an area around the player with fog.
Extra Bomb Power-Up	6	Adds another bomb to the players inventory.
Increase Range Power-Up	7	Increases the range of every bomb that explodes after the power-up is picked up.
Can Kick Power-Up	8	When touching a bomb the player now moves the bomb in the same direction as the player walks.
AgentDummy	9	In the FFA game mode the AgentDummy is used as the teammate. In the team mode the AgentDummy is used as the third enemy.
Agent0	10	The first player.
Agent1	11	The second player.
Agent2	12	The third player.
Agent3	13	The fourth player.

Table 4.1.: Every representation on the board with the corresponding value.

4.1.4. Actions

An agent can do the following actions in the environment [Pla]:

Action	Value	Description
Stop	0	This action lets the agent stay on its position. If an agents does not respond in 100ms this action is automatically executed.
Up	1	Moves the agent one field up.
Down	2	Moves the agent one field down.
Left	3	Moves the agent one field to the left.
Right	4	Moves the agent one field to the right.
Bomb	5	Places a bomb on the agents current position.

4.2. Properties of the Environment

Environments can vastly differ from each other. To categorize them we can use the following properties defined in [RN10, Chapter 2.3.2]:

Fully observable vs. partially observable If the state of the environment is known at any point in time to the agent it is a fully observable environment. If this is not the case the environment is partially observable. In a partially observable environment the agent has to keep track of the environment internally. As current reinforcement learning algorithms are tested in video games the visual input is often the key criteria to determine if the environment is fully observable or not. If the game displays the whole board without hiding any information the game is fully observable. In the original Bomberman game the player can not see the full board because the board scrolls depending on the players position. If the player is on the left side of the board he does not know about the enemies who are on the right side. On the other hand the Pommerman board is fully visible to the player and is therefore fully observable. The Pommerman environment can also be configured so its only partially observable to the agent but because this is never the case in our experiments we will consider the Pommerman environment as fully observable.

Single agent vs. multiagent If only one agent interacts with the environment we consider it as a single agent environment. If multiple agents interact with the environment it is a multiagent environment. Therefore a video game that only offers a singleplayer mode is a single agent environment. In the case of Bomberman only one player can move a character on the board. Enemies are not controlled by another person. Pommerman was specifically designed for multi agent learning as it can be played by 4 agents at the same time.

Deterministic vs. stochastic If the next state of the environment is only determined by the current state and the action of the agent, it is a deterministic environment. Otherwise it is stochastic. In multi-agent environments we ignore the uncertainty of other agents actions even though they are unknown to us. In Bomberman the enemies walk in a certain path that is defined at the start of the game and only changes if a player or a bomb is nearby. Also the board only changes by actions of the player. Therefore it is a deterministic environment. So is also Pommerman because even though we do not know about the actions of the other agents, we defined that this fact is ignored.

Episodic vs. sequential In an episodic environment only the current perception is relevant for the agent to pick an action. In a sequential environment the agent requires a memory of past actions to determine the next action. In Bomberman and in Pommerman an agent gets the current state of the environment and determines the next action. It is not influenced by other actions done before.

Static vs. dynamic If the environment changes while the player is thinking about an action, then the environment is dynamic. Otherwise it is static. In a static environment the player does not have to worry about the passage of time. In Bomberman and in Pommerman the player has to take an action in time because the game will not wait for the player to pick an action. If the player waits too long, the controlled character will just stand still which means that the player decided to do nothing. Therefore both environments are dynamic.

Bomberman	Pommerman
Partially observable	Fully observable
Single-agent	Multi-agent
Deterministic	Deterministic
Episodic	Episodic
Dynamic	Dynamic
Continuous	Discrete

Table 4.2.: The properties of the Bomberman and Pommerman environment summarized.

Discrete vs. continuous A discrete environment is given when there are clearly separated states, time intervalls and a finite set of actions. Otherwise the environment is continuous. In Bomberman the location of the player is defined by its x and y coordinate. Also the time moves as fast as the real time. Therefore the environment is continuous. In Pommerman however there are defined time steps and there is a finite number of states. Therefore Pommerman is discrete.

As summarized in table (4.2) the Pommerman environment is easier to learn for agents because the agent does not have to keep track of the environment state as it is fully observable. Also a finite number of time intervalls, states and actions is much easier. The only thing that is harder for the agent is that Pommerman is an multiagent environment. Because of that, we decided to perform the experiments with only one agent. For more advanced experiments the Pommerman environment can also be turned into a partially observable environment.

4.3. Experiment Setup

At this point, we would like to discuss the concrete implementations and concepts we used for this work.

4.3.1. Observation Processing

The Pommerman environment provides observations as an array of numbers. This array contains three different encodings of the board for obstacles, bombs and explosions with additional values such as the agents positions, agents that are alive, etc. Our approach is to transform the information to imitate image information. Therefore, we transform the 1D array into a 3D array with dimension ($boardheightXboardwithX4$). The original boardsize is 11x11 while we used a 8x8 for our experiments to make it less complex. The result after the transformation can be interpreted as an image with 4 channels, one for colors (RGB[1]) and an alpha channel. Because it can be interpreted as an image, we can then use CNNs to analyze the observations.

[1]RGB stands for Red, Green and Blue. These are the values used to encode any given other color.

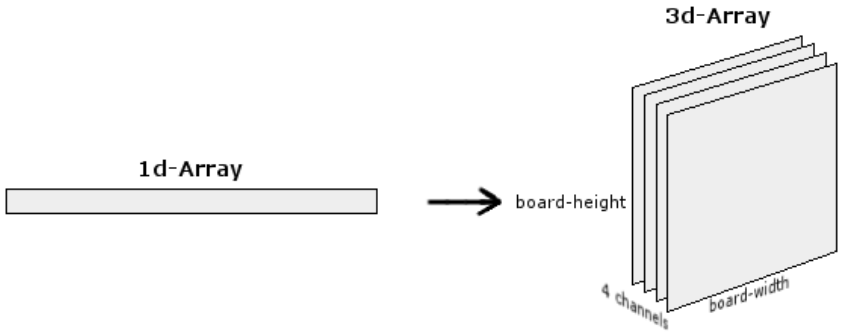


Figure 4.2.: Visualization of the observation transformation. The 1D-array should be considered excessively longer compared to the 3D-Array, since it contains almost the same amount of data. For aesthetic reasons, it is displayed shorter here.

4.3.2. Implementations

For the experiments we used the OpenAI baselines [Ope17a] implementation of the PPO algorithm, since PPO is relatively easy to configure as stated by OpenAI. [Ope17b] The OpenAI baseline provides implementations of several machine learning algorithms. However the repository is still subject to changes as the commit history [2] shows. The baseline is not stable and there still might be minor bugs in the code. There are a number of predefined Neural Networks inside the baseline and it also provides methods that can be used to configure convolutional and fully connected layers for own NNs as desired. The baseline uses Tensorflow [3] as a framework to build the networks.

The PPO impelentation With OpenAIs PPO implementation, we built an agent that learns to solve tasks that slowly increase in complexity. The implementations follows the description of the algorithm in chapter 2.4.3. To calculate a policy update, it uses a clipped surrogate objective. The alternative KL penalty is only used as a measure for the calculated policy update.
The implementation can be tuned by the hyperparameters described in table (4.3)

The CNN implementation As described before, we found that the image can be interpreted as an image. Since CNN are frequently used for image processing, it is predestined for this task.
In their bachelor thesis, Liebesking and Wehner have also used a CNN architecture. Since the network architecture has already been used in this environment, we decided to follow their specification.
The network consist of two convolutional layers followed by a fully connected layer

[2] See the history at https://github.com/openai/baselines/commits/master
[3] Tensorflow is an open source machine learning framework for high performance numerical computation. https://www.tensorflow.org/

Name (in OpenAIs code)	Description
totaltimesteps	The total number of steps an agent takes during the experiment.
nsteps	Number of steps taken by each actor for a single update (Represented by T in the algorithm description in ch 2.5.2).
ent_coef	The policy entropy coefficient, described as c_2 in equation (2.12).
lr	The learning rate of the algorithm, which determines how much the agent learns per update.
vf_coef	Value function loss coefficient, described as c_1 in equation (2.12).
max_grad_norm	The gradient norm clipping coefficient.
gamma	The discounting factor γ for the rewards as shown in equation (2.9).
lam	The advantage estimator discounting factor λ as shown in equation (2.9).
cliprange	The clipping parameter ϵ as described in equation (2.10).
nminibatches	Number of training minibatches per update(2.10).

Table 4.3.: The hyperparameters as named in OpenAIs PPO implementation.

as shown in figure (4.3). The receptive fields are all 3x3 sized. As described above, the input image in our case is 8x8 in size. Because of the small input dimension, we use padding, so our image keeps the 8x8 dimension while applying the 3x3 receptive fields.

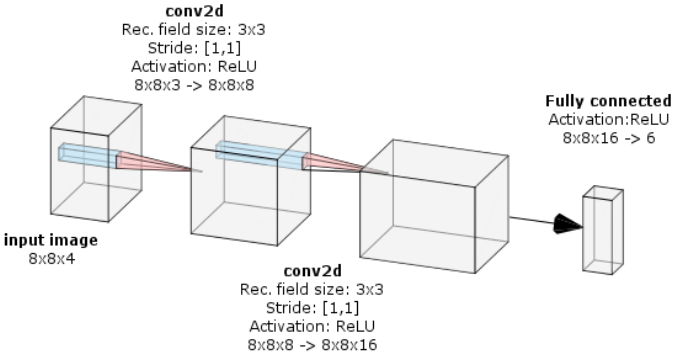


Figure 4.3.: A 3D visualization of our CNN. The first convolutional layer uses eight different receptive fields transforming the 8x8x3 input to a 8x8x8 layer. Accordingly, the second layer uses 16 receptive fields transforming the 8x8x8 input to a 8x8x16 layer.

5. Experiments

In this work, we conducted two types of experiments to proof that our setup is able to solve the Pommerman environment. Our approach is to start with a basic type of experiment with only our agent and zero obstacles while gradually increasing complexity towards the complexity of a Bomberman game with multiple agents and obstacles. For this, we have conducted a number of experiments of which we discuss the five most important experiments.

5.1. Measurement Criteria

In the following section, the experiments will be interpreted using plots and these criteria:

mean reward:	This plot displays the mean reward the agent has collected per timestep.
average length of episode:	In this plot, the average number of timesteps per episode is displayed.
clipping:	indicates how much the clipped objective function influenced the policy.
KL-divergence penalty:	This plot shows the approximate KL-divergence as an indication of how big the policy updates were.[1]

Not all plots are equally meaningful for all the experiments. For the sake of completeness we decided to display all the plots for all the experiments discussed. All the plots can be found in original size in the appendix.

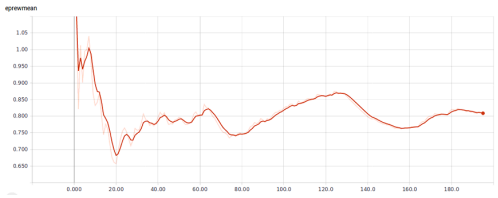
5.2. Experiment 1: Find the Box

With the first experiment we want to show that the setup is working and can be tuned to reach better results. This should lay the foundation for more complex experiments.

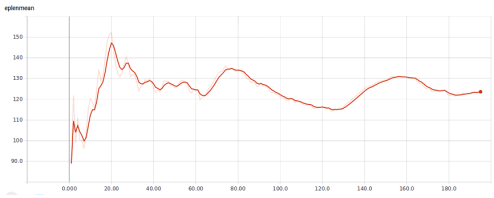
[1] As explained in chapter 2.4.2 and 3.3.2.

5.2.1. Experiment 1a: Proof of Concept

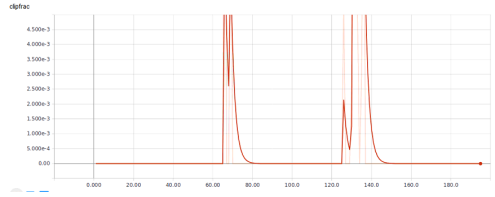
Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found
Allowed moves	up,down,left,right,stop
Hyperparameters	nsteps=512 nminibatches=4 ent_coef=0.0 lr=3e-4 vf_coef=0.5 max_grad_norm=0.5 gamma=0.99 lam=0.95 cliprange=0.2 total_timesteps=100000
Objective	With this experiment, we want to proof that our experiment setup is working. The hyperparameters are all left on default values set by OpenAIs baseline. We only ajusted the total_timesteps to and the nsteps so that our experiment will terminate a bit sooner.



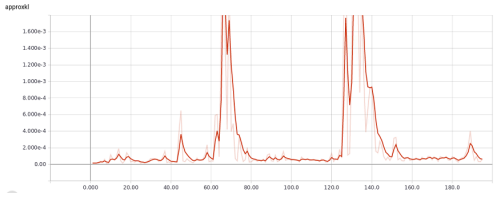
(a) Plot of the mean reward per policy update



(b) Plot of the average episode length per policy update.



(c) Plot of clipping per policy update.



(d) Plot of the KL divergence (penalty) per policy update.

Figure 5.1.: Plots of the first run of experiment 1. (Full sized plots can be found in appendix.)

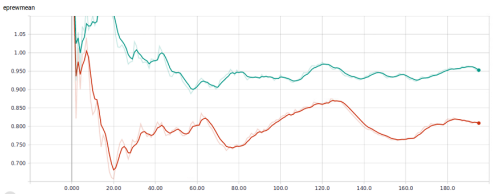
Result: The mean reward function in figure (5.1a) shows that the agent is able to improve his behaviour. On the other hand, it is obvious that the agent is not steadily improving. The performance drops as the agent makes bigger adjustments to its policy as figure 5.1c suggests. Visual analysis also showed that when multiple boxes appeared in one region of the board, the agent faces problems when the box finally appears in a different region.

Discussion: From the result of the experiment, we draw the following conclusions: Firstly, the performance drops significantly when the policy updates are exceptionally high. Therefore, we suggest that by reducing the policy update, the performance will stabilize. We tried to proof this hypothesis by dropping the cliprange-parameter to 0.1 in the next experiment. Since, in contrast to our expectations, this did have a negative effect on the performance, we will not discuss this experiment in detail here.[2]
Secondly, with parameter gamma=0.99 and lambda=0.95 PPO would estimate the advantage of an action taken 70 steps before finding the box with $100 * (0.99 * 0.95)^{70} \approx 1.36495$ [3]. Ideally, in a board with size 8x8 the box should be found in less steps. Therefore, these parameter values lead to overestimating actions in long episodes.

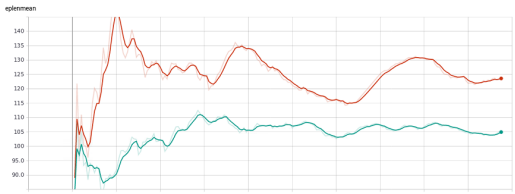
5.2.2. Experiment 1b: Increasing Entropy and Decreasing γ to 0.8

Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found
Allowed moves	up,down,left,right,stop
Hyperparameters	nsteps=512 nminibatches=4 ent_coef=0.1 lr=3e-4 vf_coef=0.5 max_grad_norm=0.5 gamma=0.8 lam=0.95 cliprange=0.2 total_timesteps=100000
Objective	As discussed in the previous experiment, we want to decrease the parameter value of γ to a more sensible value. We estimate that the agent should not take longer than 15 steps to reach the goal. Therefore we decrease gamma to 0.8 because $100 * (0.8 * 0.95)^{15} \approx 1.6$. The purpose of this experiment is to proof, that this increases the performance.

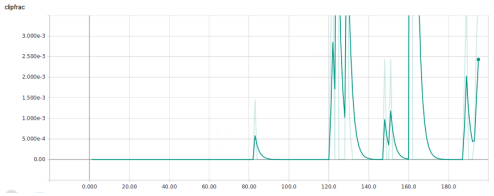
[2]The experiment can be found in appendix F.
[3]See chapter 2.5.2 equation (2.7)



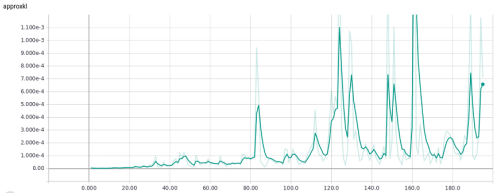
(a) Plot of the mean reward per policy update compared to the previous experiment (red).



(b) Plot of the average episode length per policy update compared to the previous experiment (red).



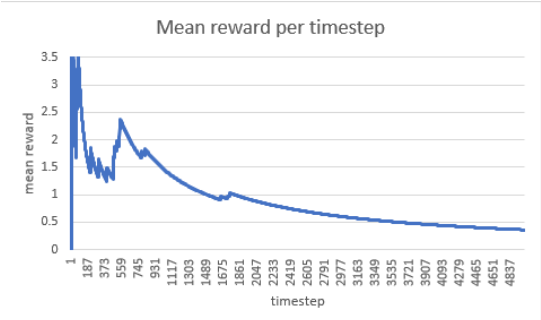
(c) Plot of clipping per policy update.



(d) Plot of the KL divergence (penalty) per policy update.

Figure 5.2.: Plots of experiment 2 with decreased γ. (Full sized plots can be found in appendix.)

Result: The mean reward (fig 5.2a) has surpassed the value of the previous experiment by far. More than expected before the experiment. However, there is no steady improvement visible. For a more meaningful conclusion, we evaluated the trained policy. While figure (5.3b) seems to to confirm the result from the training session with a mean reward of approximately 0.8 points per timestep, the evaluation in figure (5.3a) clearly indicates otherwise with a mean reward below 0.5 points per timestep.



(a) Mean reward plot of the first evaluation run of experiment 1b.

(b) Mean reward plot of the second evaluation run of experiment 1b.

Figure 5.3.: Comparison of two evaluation runs of experiment 1b.

Discussion: While the result after training seemed promising in figure (5.2a), the evaluation has shown that the performance is not stable. When observing the agent during evaluation, it seemed as if he had problems finding boxes on the left hand side. The results clearly suggest that the performance must be further stabilized. One hypothesis was, that when a number of boxes appear in one region of the board in a short period of time, the agent learns to favour walking to this corner before learning to go where the box really

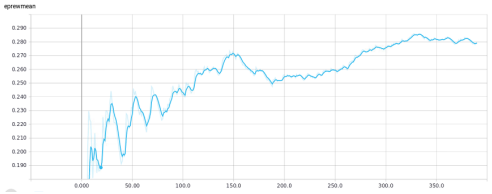
is. We have conducted experiments with a lower learning rate to overcome this effect. Unfortunately, these experiments were not successful.
In retrospect, after conducting experiment 5.3.3 running the experiments for more timesteps seemed to be an interesting possibility as well. The experiment has not been conducted by the end of this work due to lack of time.

5.3. Experiment 2: Destroying Wooden Boxes

With this experiment, we give the agent the ability to lay down bombs. The goal for the agent is now to find the wooden box and destroy it by planting a bomb next to it. This additional action only, provides an increase in complexity to the task. With bombs the agent can kill himself and end an episode without success, which was impossible in the previous example. Furthermore, using a bomb is essential for successfully completing the task, but can also kill the agent. This requires the agent to learn to use the bombs properly which is a complex task.

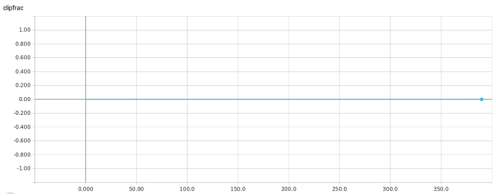
5.3.1. Experiment 2a: Applying Previous Findings

Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found
Allowed moves	up,down,left,right,stop,bomb
Hyperparameters	nsteps=256 nminibatches=4 ent_coef=0.1 lr=3e-4 vf_coef=0.5 max_grad_norm=0.5 gamma=0.8 lam=0.95 cliprange=0.2 total_timesteps=100000
Objective	In this experiment, we want to apply the findings from our previous experiment. Since we have seen that this configuration worked with the previous experiment, we expect to get a good foundation for future experiment with this attempt. Since an episode might end without any reward in this experiment, the mean reward will be significantly lower compared to the previous experiment.

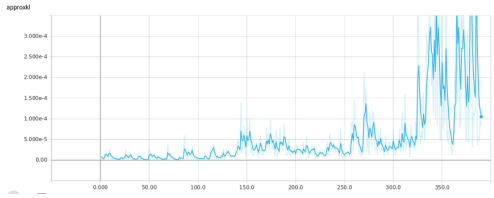


(a) Plot of the mean reward per policy update.

(b) Plot of the average episode length per policy update.



(c) Plot of clipping per policy update.



(d) Plot of the KL divergence (penalty) per policy update.

Figure 5.4.: Plots of the first run of experiment 2. (Full sized plots can be found in appendix.)

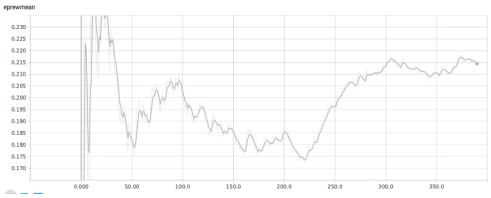
Result: The mean reward function in figure (5.4a) indicates that the agent is still steadily improving. As expected the mean reward is lower than in the previous example. This is due to the high number in deaths caused by bombs.
By comparing the agents behaviour in the beginning, middle and end of training, we found that the agent does destroy the boxes more often over time. The number of deaths caused by using bombs excessively is still to high for a satisfying performance.

Discussion: The results indicate, that the agent is improving over time. However, there is huge potential in learning to avoid deaths. By not dying early, the agent would be more likely to find boxes and therefore have more valuable episodes for training. An obvious solution to achieve this would be, to give the agent negative rewards when he dies. By adding this punishment, we expect that the agent might learn to not use bombs at all since deaths are more likely to happen than destroying boxes. We proofed our expectation by running an experiment with negative reward and found that our theory was correct.[4] By decreasing the policy updates with a smaller learning rate, learning to not use bombs could be prevented.
Furthermore, we think that due to the deaths, the agent does not get enough successful episodes. Especially in episodes where the box is far away, the agent is very likely to die before reaching the box. Increasing the total_timesteps parameter should lead to more successful episodes and therefore better performance.

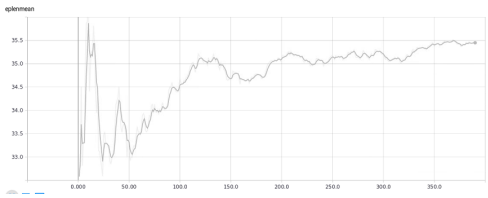
[4]This experiment can be found in appendix G.

5.3.2. Experiment 2b: Negative Reward with lr=1e-5

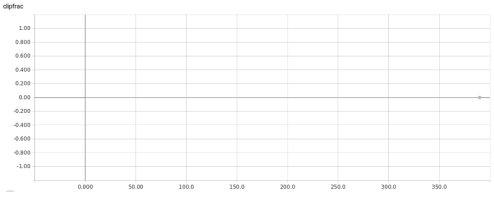
Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found, -1 if agent dies
Allowed moves	up,down,left,right,stop,bomb
Hyperparameters	nsteps=256 nminibatches=4 ent_coef=0.1 lr=1e-5 vf_coef=0.5 max_grad_norm=0.5 gamma=0.8 lam=0.95 cliprange=0.2 total_timesteps=100000
Objective	In previous experiments, we proofed that negative reward can lead to not using bombs. Our hypothesis is, that a lower learning rate can prevent this effect by gradually changing the policy to use less bombs.



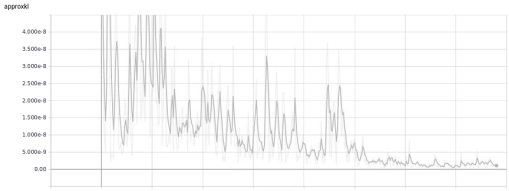
(a) Plot of the mean reward per policy update.



(b) Plot of the average episode length per policy update.



(c) Plot of clipping per policy update.

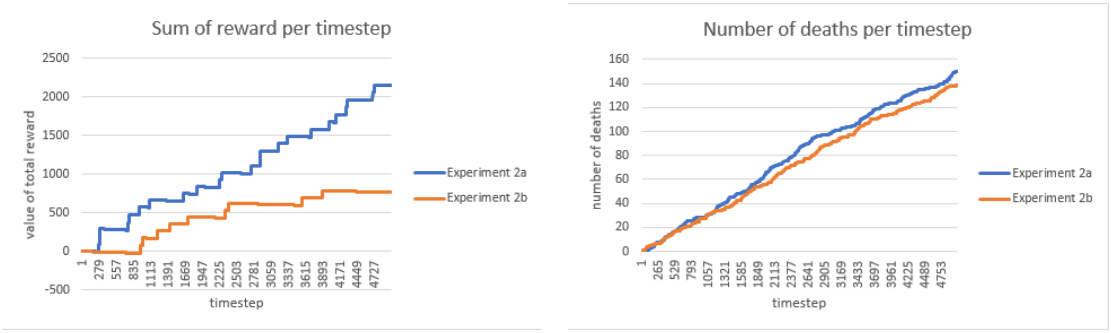


(d) Plot of the KL divergence (penalty) per policy update.

Figure 5.5.: Plots of the negative reward experiment with smaller learning rate. (Full sized plots can be found in appendix.)

Result: The mean reward function (figure 5.5a) does increase towards the end of the training, while in previous experiments with negative rewards the mean reward moved steadily towards zero. This is proof that the agent did not stop to use bombs despite the negative result.
Due to the negative reward for deaths in this experiment, the two mean reward plots are not comparable. Therefore, the two trained policies resulting from experiment 2a and

2b have been evaluated under equal conditions. This means although in experiment 2a the policy was trained without negative reward, there was a negative reward for deaths during evaluation to make the rewards comparable. Furthermore, the number of deaths have been counted as well.



(a) Comparison of the total reward per timestep. Experiment 2a in blue and 2b in orange.

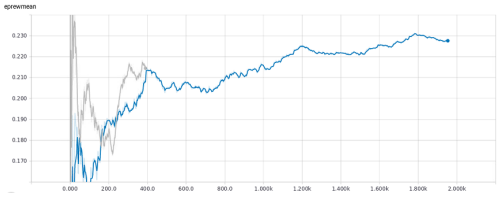
(b) Comparison of the number of deaths per timestep. Experiment 2a in blue and 2b in orange.

Figure 5.6.: Result of the evaluation of experiments 2a & 2b.

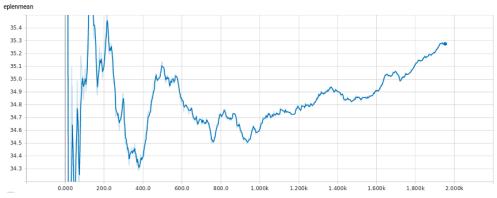
Discussion: We have done multiple experiments to find a good value for the learning rate. This experiment clearly proofs our theory, that negative reward can work with a smaller learning rate. The mean reward tends to improve steadily after half of the experiment. The evaluation has shown that this change did not have a positive impact on the reward. Figure (5.6a) clearly shows that experiment 2a was performing better in terms of reward. However, figure (5.6b) also shows that the agent did die less. Since an episode can only end with a huge positive reward or a small negative reward, the difference in rewards has to be because of longer episodes in experiment 2b. Therefore, the agent has learned to dodge bombs and survive longer which should enable him to reach wooden boxes in more remote locations. In conclusion this means that the negative reward does lead to longer episodes as desired. Unfortunately, the longer episodes do not automatically lead to better behaviour.

5.3.3. Experiment 2c: Negative reward with lr=1e-5 for 500'000 steps

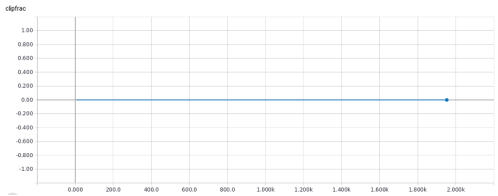
Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found, -1 if agent dies
Allowed moves	up,down,left,right,stop,bomb
Hyperparameters	nsteps=256 nminibatches=4 ent_coef=0.1 lr=1e-5 vf_coef=0.5 max_grad_norm=0.5 gamma=0.8 lam=0.95 cliprange=0.2 total_timesteps=500000
Objective	The past experiment has shown that that the reward was increasing towards the end of the experiment. With the evaluation, we can draw the conclusion, that the agent is able to survive longer. We believe that the longer lives will have a positive effect over time. As soon as the agent has learned to dodge bombs, he has more time to learn how to find and destroy the wooden box. To proof this theory, this experiment does not change any parameter except for the total_timesteps. We increase this parameter to 500'000 timesteps.



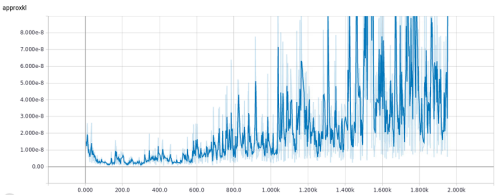
(a) Plot of the mean reward per policy update.



(b) Plot of the average episode length per policy update.

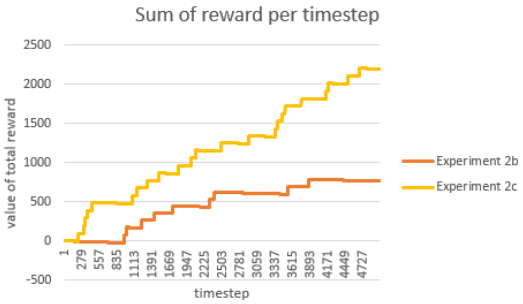


(c) Plot of clipping per policy update.

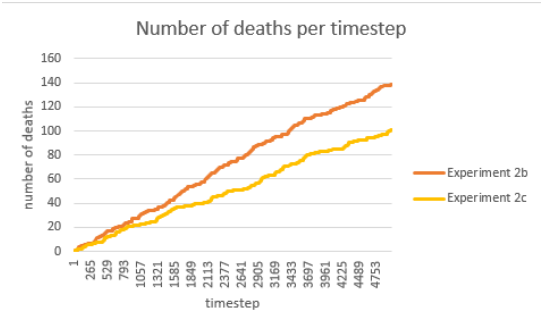


(d) Plot of the KL divergence (penalty) per policy update.

Result: As in the previous experiments, the mean reward (figure 5.7a) is increasing steadily and surpasses the last experiment. This is a strong indication that the longer episodes do improve the performance. This experiment was evaluated as well to compare to the previous experiment 2b. The comparison of rewards in figure (5.8a) clearly indicated that longer training episodes truly have a positive effect on the performance. Experiment 2c has reached almost the same amount of total reward as experiment 2a (compare figures (5.6a) and (5.8a)) while dying less than in experiment 2b as figure (5.8b) shows.



(a) Comparison of the total reward per timestep. Experiment 2b in orange and 2c in yellow.



(b) Comparison of the number of deaths per timestep. Experiment 2b in orange and 2c in yellow.

Figure 5.8.: Result of the evaluation of experiments 2b & 2c.

Discussion: This experiment does not change the learning process itself as other experiments before. This experiment was conducted to proof that negative reward does have a positive impact in the long run. Both figures (5.8a) and (5.8b) indicate that this is the case. The mean reward function during training in figure (5.7a) shows relatively steady progress which is promising as well. While there is still room for improvement, this experiment could serve as a promising starting point for further experiments.

6. Conclusion

With this work we can not definitely answer the question if PPO is able to solve a game like Bomberman. The experiments conducted in this work are not complex enough to draw this conclusion. On the other hand, the experiment have shown, that the set up has the ability to improve. To claim that PPO is not able to solve Bomberman at all would also seem unsubstantiated. However, the work leads to a number of conclusions and pointers that might lead to good results in future works.

Further experimenting with parameter values
Firstly, there are still hyperparameters that have not been explored in detail. In this work, we have systematically altered a number of parameters and took OpenAIs default values as a starting point. The experiments suggest that the decrease of the gamma value and small learning rate in combination with negative rewards were positive adjustments.

CNN architecture
The architecture of our CNN is also not guaranteed to be optimal for this problem. We did not experiment with different architectures, filter sizes etc. We are convinced that investigating the possibilities in this area would be beneficial to solve Bomberman or any other complex problem for that matter.

Rare Rewards
Experiment 2 has shown that learning can be difficult with rarely occurring rewards. Especially for further experiments with power ups and multiple opponents, we imagine that different rewards should be combined to improve learning.

Experiment log
To systematically experiment with parameters, an experiment log has been extremely helpful. It is essential to log parameter values and ideas or a hypothesis that belong to an experiment. As the number of experiments increases, it becomes more and more likely that important conclusions or ideas would go lost without an up-to-date experiment log.

7. Future Work

Since the fundamental question of this work can not be answered definitely, there are a number of further approaches to find a final answer.

Continuing with current set up
The experiments presented in this work are not conclusive. As stated in chapter 5.3.3 experiment 2c would be a good base for further experimenting with hyper-parameters. Developing a more complex experiment with i.e. multiple opponents would be an interesting next step.

Exploring other NN architecture
As discussed in the previous section, the CNN architecture used in this work is not guaranteed to be optimal. Exploring possibly more suitable network architectures could be a promising approach to progress with this work. Fortunately, OpenAI baseline is easily extendable with custom Neural Networks.

Using the newest PPO implementation
Since the PPO algorithm was only published recently and OpenAIs baseline has been updated frequently during this work, it might be beneficial to use the newest implementation of PPO. Because of the many updated it is possible that the implementation used in this work still contains bugs.

Other Algorithms
Reinforcement learning and artificial intelligence in general are both fields that are actively researched at the moment. Therefore, we believe that it is possible that other interesting algorithms are discovered. If that is the case it could be interesting to explore these algorithms as well.

8. Acronyms

AI	Artificial Intelligence
CNN	Convolutional Neural Network
DQN	Deep-Q-Network
FFA	Free For All
KI	Künsliche Intelligenz
MDP	Markov Decision Process
PPO	Proximal Policy Optimization
RGB	Red Green Blue

Bibliography

[Ben95] Yann LeCun, Yoshua Bengio. Convolutional networks for image, speech and time-series. `http://yann.lecun.com/exdb/publis/pdf/lecun-bengio-95a.pdf`, 1995. (Online; accessed 02-December-2018).

[Dee16] Deepmind. Match 5 - google deepmind challenge match: Lee sedol vs alphago. `https://www.youtube.com/watch?v=mzpW10DPHeQ`, 2016. (Online; accessed 14-December-2018).

[Dot18] Dota2. Humans vs openai5 the international 2018 main event. `https://www.youtube.com/watch?v=nGhkCuQloXI`, 2018. (Online; accessed 14-December-2018).

[Eld] Cinjon Resnick, Denny Britz, David Ha, Jakob Foerster, Wes Eldridge. Pommerman website. `https://www.pommerman.com/`.

[Gra18] Max Jaderberg, Wojciech M. Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C. Rabinowitz, Ari S. Morcos, Avraham Ruderman, Nicolas Sonnerat, Tim Green, Louise Deason, Joel Z. Leibo, David Silver, Demis Hassabis, Koray Kavukcuoglu, Thore Graepel. Human-level performance in first-person multiplayer games with population-based deep reinforcement learning. 2018.

[JS17] Prafulla Dhariwal Alec Radford Oleg Klimov John Schulman, Filip Wolski. Proximal policy optimization algorithms. *CoRR*, abs/1707.063472, 2017.

[Kan03] Masakazu Matsugu, Katsuhiko Mori, Yusuke Mitari, Yuji Kaneda. Subject independent facial expression recognition with robust face detection using a convolutional neural network. `http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/sparse/matsugo_etal_face_expression_conv_nnet.pdf`, 2003. (Online; accessed 02-December-2018).

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 03 1951.

[MBM+16] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016.

[Nie15] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[Ope17a] OpenAI. Openai baseline. https://github.com/openai/baselines, 2017. (Online; accessed 08-October-2018).

[Ope17b] OpenAI. Openai blog ppo. https://blog.openai.com/openai-baselines-ppo/, 2017. (Online; accessed 08-December-2018).

[Ope18] OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018. (Online; accessed 08-December-2018).

[Pla] Playgrounds. Pommerman readme. https://github.com/MultiAgentLearning/playground/tree/master/pommerman. (Online; accessed 20-December-2018).

[Res15] Marcello Restelli. Policy gradient. http://home.deib.polimi.it/restelli/MyWebSite/pdf/rl7.pdf, 2015. (Online; accessed 14-December-2018).

[Rie13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller. Playing atari with deep reinforcement learning. 2013.

[RN10] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, third edition edition, 2010. ISBN: 978-0-13-604259-4.

[SB17] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition edition, 2017.

[SHM+16] David Silver, Aja Huang, Christopher J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *Nature*, 529:484–503, 2016.

[Sil15] David Silver. Advanced topics in machine learning. http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html, 2015. (Online; accessed 02-December-2018).

[Sof85] Hudson Soft. Bomberman. US Version, Nintendo Entertainment System, 1985.

[Sof92] Hudson Soft. Bomberman 2. US Version, Nintendo Entertainment System, 1992.

[Weh18] Joram Liebeskind, Silvan Wehner. Using reinforcement learning to play pommerman. Master's thesis, Zurich University of Applied Science, 2018.

[Wen18] Lilian Weng. Policy gradient algorithms. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html, 2018. (Online; accessed 14-December-2018).

Appendices

A. Plots experiment 1a

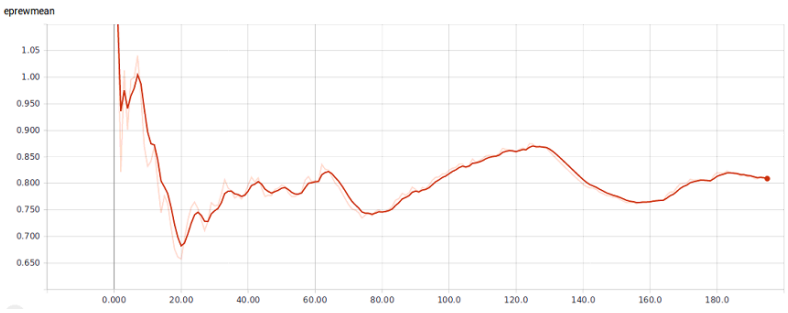


Figure A.1.: Mean reward plot experiment 1a.

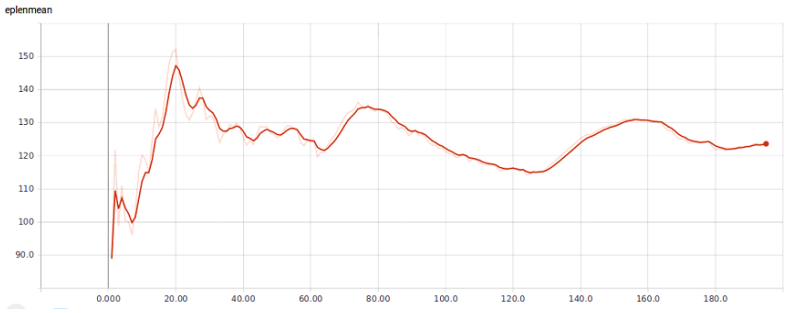


Figure A.2.: Mean length plot experiment 1a.

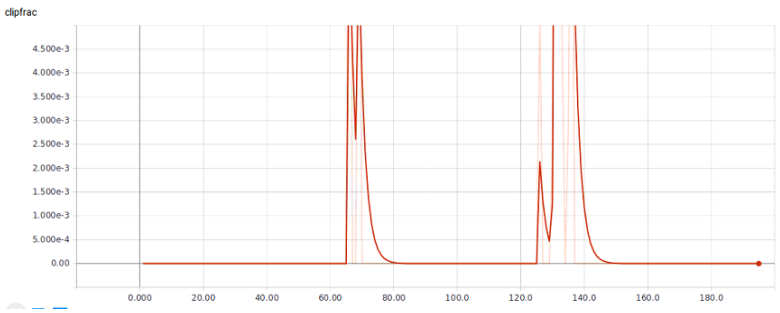


Figure A.3.: Clipping plot experiment 1a.

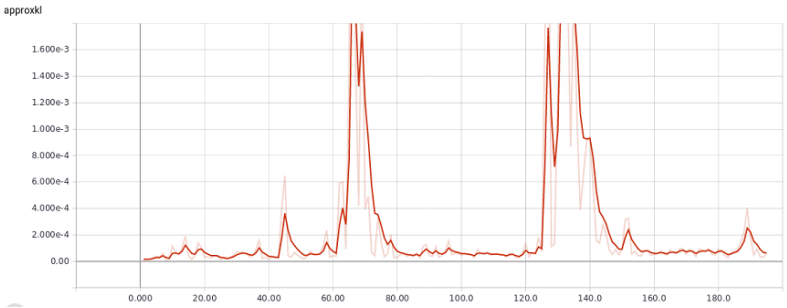


Figure A.4.: KL divergence plot experiment 1a.

B. Plots experiment 1b

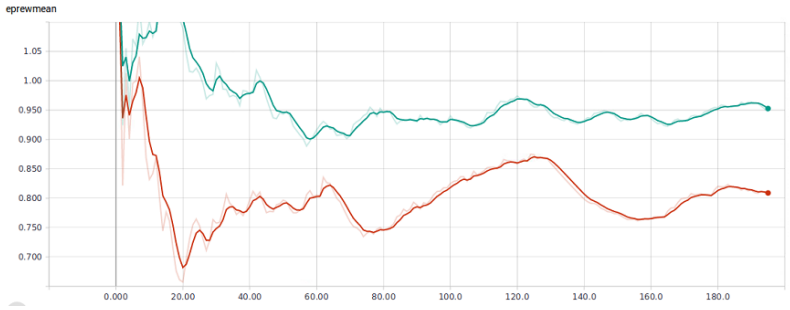


Figure B.1.: Mean reward plot experiment 1b.

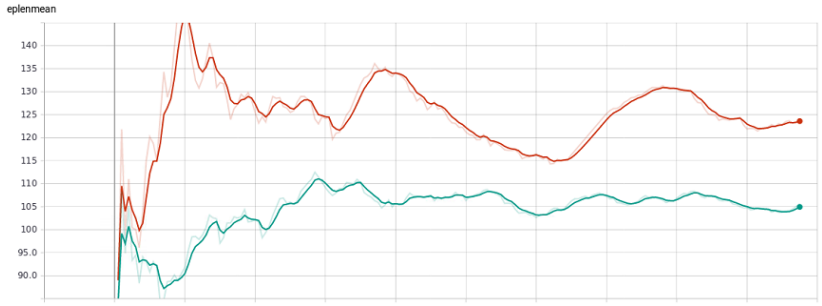


Figure B.2.: Mean length plot experiment 1b.

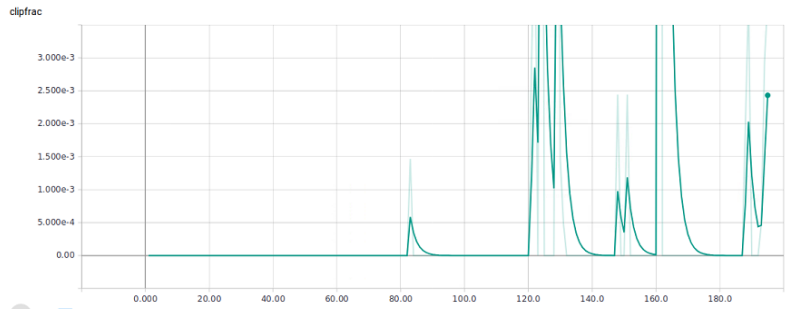


Figure B.3.: Clipping plot experiment 1b.

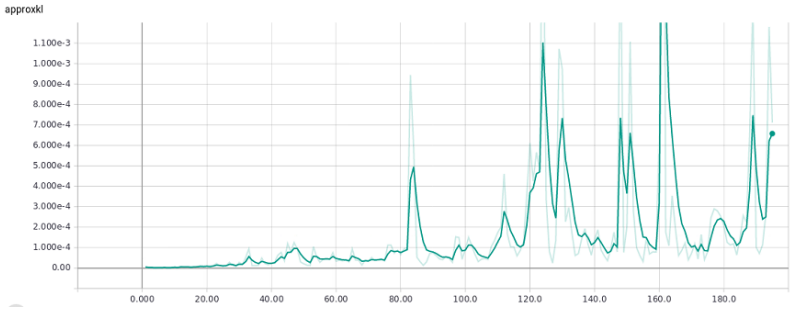


Figure B.4.: KL divergence plot experiment 1b.

C. Plots experiment 2a

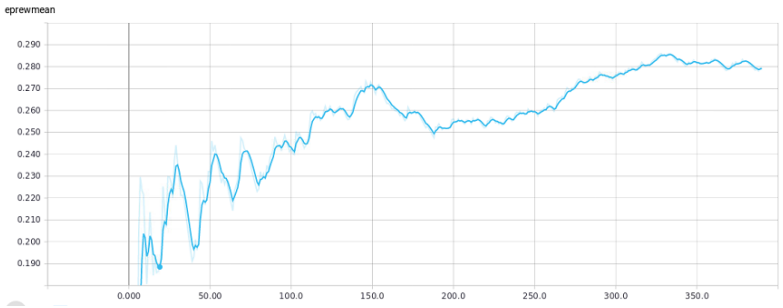


Figure C.1.: Mean reward plot experiment 2a.

Figure C.2.: Mean length plot experiment 2a.

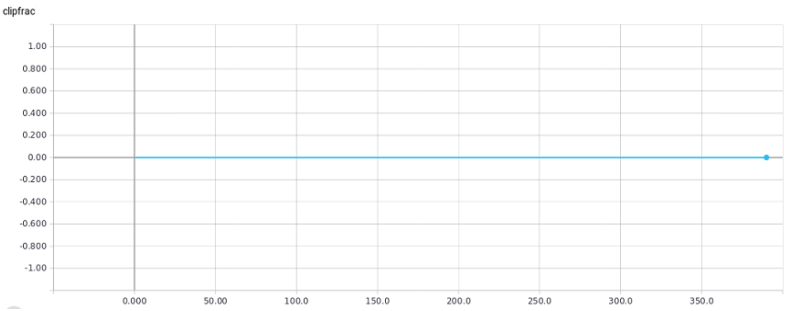


Figure C.3.: Clipping plot experiment 2a.

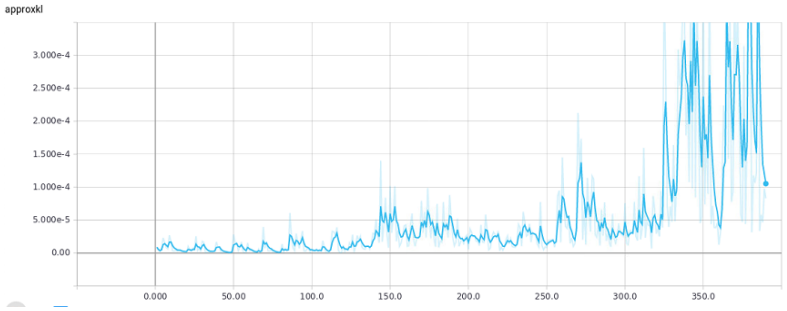


Figure C.4.: KL divergence plot experiment 2a.

D. Plots experiment 2b

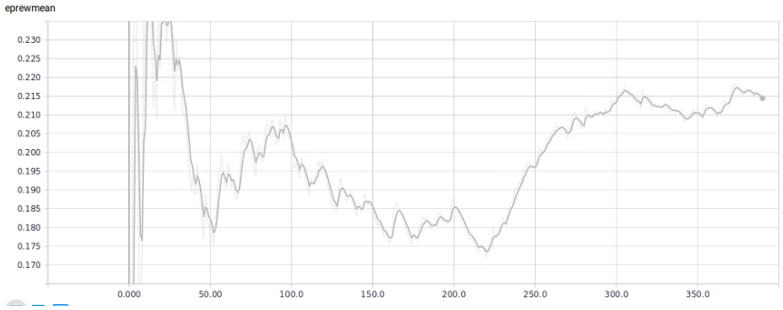


Figure D.1.: Mean reward plot experiment 2b.

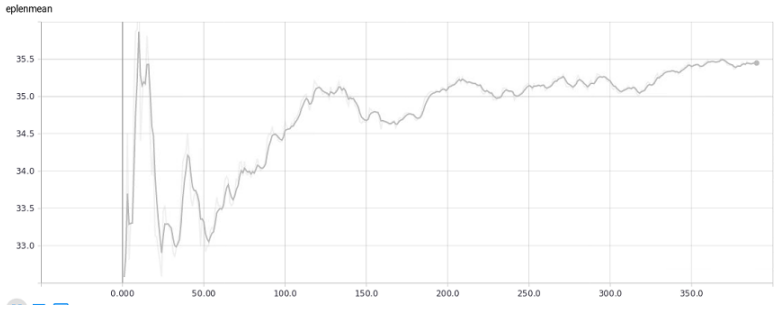


Figure D.2.: Mean length plot experiment 2b

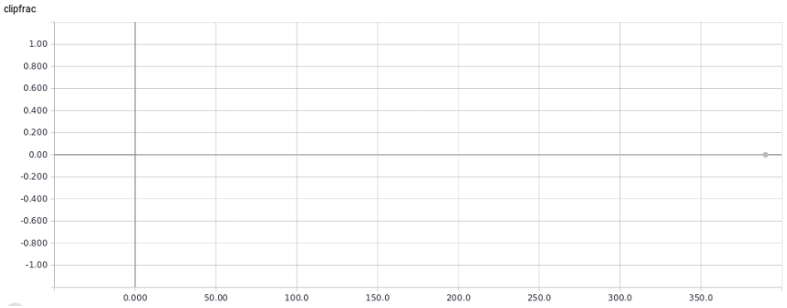


Figure D.3.: Clipping plot experiment 2b.

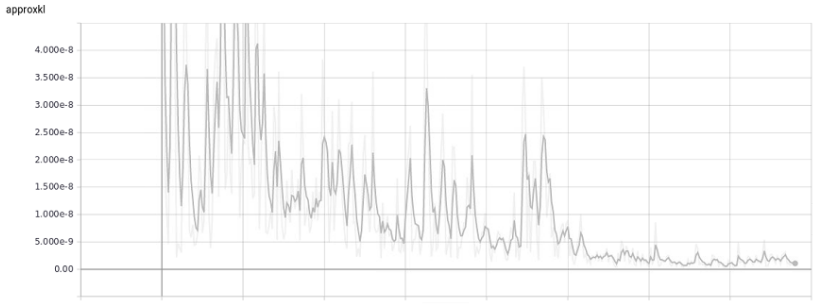


Figure D.4.: KL divergence plot experiment 2b.

E. Plots experiment 2c

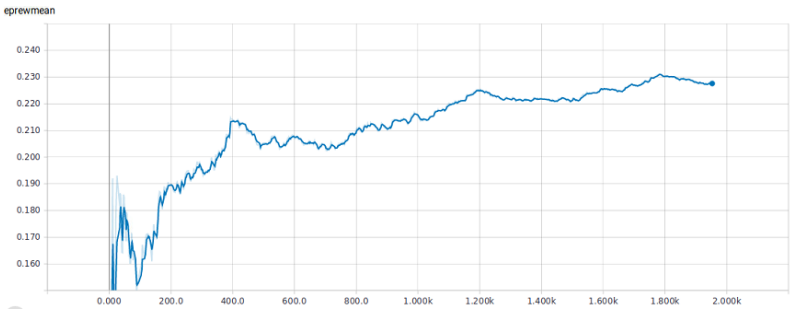


Figure E.1.: Mean reward plot experiment 2c.

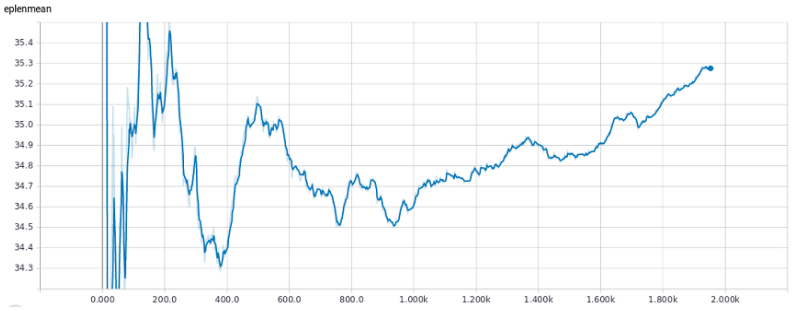


Figure E.2.: Mean length plot experiment 2c.

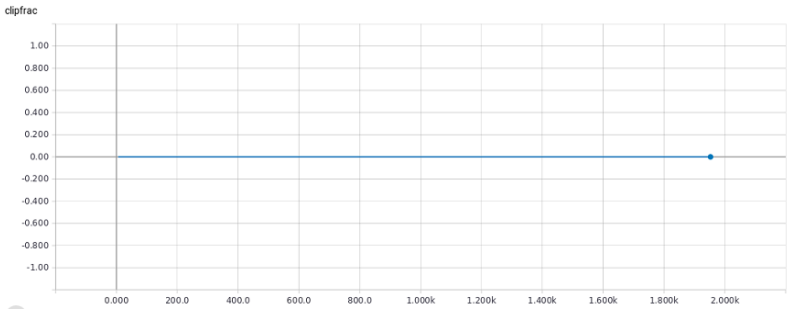


Figure E.3.: Clipping plot experiment 2c.

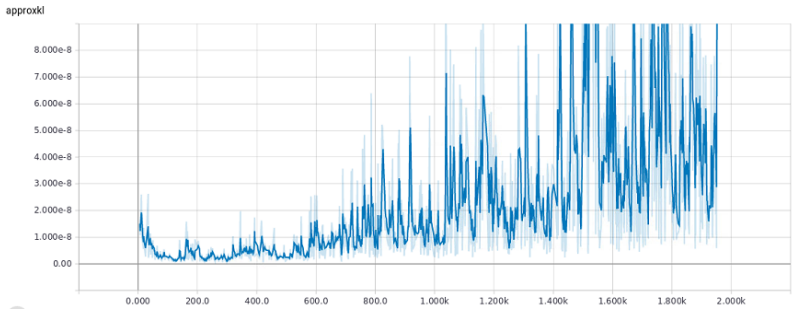
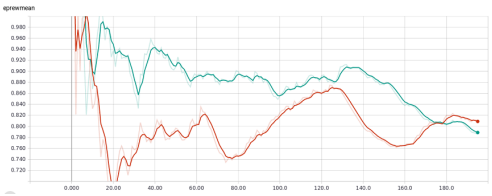


Figure E.4.: KL divergence plot experiment 2c.

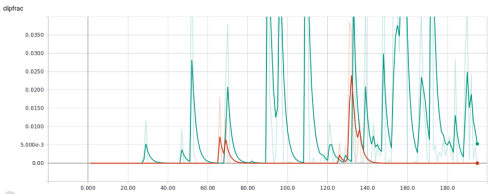
F. Experiment with smaller clip range

Since this experiment was not successful, we only discuss the most important aspects. The experiment was conducted after experiment 1a in chapter 5.2.1 and tried to stabilize the mean reward by narrowing the clip range to 0.1. The other parameters were left the same as in experiment 1a:

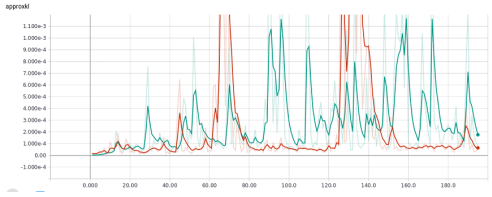
Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found
Allowed moves	up,down,left,right,stop
Hyperparameters	nsteps=512 nminibatches=4 ent_coef=0.0 lr=3e-4 vf_coef=0.5 max_grad_norm=0.5 gamma=0.99 lam=0.95 cliprange=0.2 total_timesteps=100000



(a) Plot of the mean reward per policy update.



(b) Plot of clipping per policy update.



(c) Plot of the KL divergence (penalty) per policy update.

Figure F.1.: Selected plots of the experiment with cliprange 0.1.

As the figures clearly show, the performance did not increase nor stabilize. The policy update did clip more often, but as the KL divergence penalties show it did not lead to smaller policy updates.

G. Negative reward without any adjustments

Since this experiment was not successful, we only discuss the most important aspects. The experiment was conducted after experiment 2a in chapter 5.3.1. The goal was to proof or disproof our hypothesis that the agent might learn to not use bombs with negative reward. Therefore, we added negative reward to experiment 2a without changing any parameters:

Environment	board-size: 8x8, no boxes except one wooden box in random locations
Reward	+100 reward if the box was found, -1 if agent dies
Allowed moves	up,down,left,right,stop,bomb
Hyperparameters	nsteps=256 nminibatches=4 ent_coef=0.1 lr=3e-4 vf_coef=0.5 max_grad_norm=0.5 gamma=0.8 lam=0.95 cliprange=0.2 total_timesteps=100000

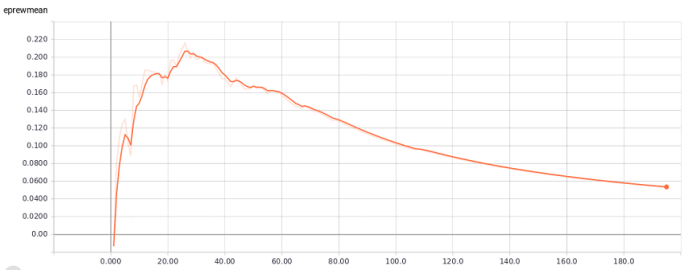


Figure G.1.: Mean reward of experiment 2a configuration with negative reward

The reward function shows a smooth decrease after a certain time. This is because the reward is not changing anymore, there would be local peaks or drops otherwise. The only reasonable explanation for this would be that the agent is not using bombs and will never reach a state that would give him positive or negative reward.