An Evaluation of Unity ML-Agents Toolkit for Learning Boss Strategies

by

Lucas A.E Pineda Metz

Thesis of 60 ECTS credits submitted to the School of Technology,
Department of Computer Science
at Reykjavík University in partial fulfillment
of the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

September 2020

Examining Committee:

Dr. Yngvi Björnsson, Supervisor
Professor, Reykjavík University, Iceland

Dr. David James Thue, Co-supervisor
Assistant Professor, Carleton University, Canada

Kári Halldórsson, Examiner
Adjunct, Reykjavik Univeristy, Iceland

An Evaluation of Unity ML-Agents Toolkit for Learning Boss Strategies

Lucas A.E Pineda Metz

September 2020

Abstract

Accompanying the growing pace of AI research for video games is the development of new benchmark environments. One of the most recently introduced environments is Unity's Machine Learning Toolkit (ML-Agents Toolkit). With this toolkit, Unity allows its users (researchers or game developers) to incorporate state-of-the-art Reinforcement Learning or Imitation Learning algorithms or one's own Machine Learning algorithms to train a learning agent. On this project I used one of the Reinforce Learning algorithms (Proximal Policy Optimization ; PPO) alone and in combination with two Imitation Learning algorithms (Generative Adversarial Imitation Learning ; GAIL and Behavioral Cloning ; BC) provided with Unity's Machine Learning Toolkit. These were used to teach a learning agent how to optimize its policy in order to maximize its reward by learning how to better choose from a set of attacks to win in a fight against a simpler non-learning agent. The project has two focuses: a) To compare the learning provided by the different algorithms included in Unity's toolkit, and additionally compare the use of the Imitation Learning algorithms as complements of the Reinforce Learning algorithm, and; b) Test the usability of the ML-Agents Toolkit by creating a learning environment to train an agent and compare my experience implementing and training such an agent with the information provided by Unity's documentation. To achieve this, I conducted three case studies, one providing a demonstration file containing an optimal policy, one with a sub-optimal policy, and a third one with a mix of both. For all study cases, the learning was done considering four combinations of learning algorithms: a) PPO alone; b) PPO in combination with GAIL; c) PPO in combination with BC, and; d) all learning algorithms. The overall result of the three case studies showed a successful learning by the agent, regardless of the learning algorithms considered. From the Imitation Learning algorithms, GAIL showed difficulties to learn policies which involved several complex actions, whereas BC greatly increased the rate of learning. The results of the project show the advantages and limitations of the use of Imitation Learning algorithms for learning behaviours, the importance of the demonstration provided for the Imitation Learning algorithms, and further discusses the usefulness of entropy as a complementary variable to consider when assessing the success rate of the learning process.

Keywords: ML-Agents Toolkit, Unity, learning agent

Titll verkefnis með þöæéíó

Lucas A.E Pineda Metz

september 2020

Útdráttur

Aðferðir sem byggja á vélrænu gagnanámi eru að verða sífellt vinsælari við lausn margskonar vandamála. Tölvuleikjaiðnaðurinn hefur ekki farið varhluta af þeirri þróun. Nýlega kynnti Unity, framleiðandi einnar vinsælustu leikjavélar heims, viðbót við hugbúnaðinn sinn, "ML-Agents Toolkit", sem á að auðvelda leikjaframleiðendum að nýta vélrænar gagnanámsaðferðir við þróun leikja sinna, þar með talið nýlegar hermi- og styrkingarlærdómsaðferðir.

Í þessu verkefni er skoðað hversu vel þesssar lærdómsaðferðir eru fallnar til þess að læra að stýra tölvuagent í svokölluðum "Boss Battle" leikjum, en það er gerð leikja þar sem spilarar þurfa að sigrast öflugan tölvustýrðum agent (e. Boss). Við berum saman lærdómsaðferðirnar, en þær eru "Proximal Policy Optimization (PPO)", "Behavioral Cloning (BC)", og "Generative Adversarial Immitation Learning (GAIL)", bæði þegar þær eru keyrðar stakar og saman. Við metum aðferðirnar út frá bæði skilvirkni og hversu notendavænar þær eru. Við keyrðum þrjár mismunandi sviðsmyndir þar sem notast var við mismunandi þjálfunargögn. Helsu niðurstöður eru þær að allar lærdómsaðferðirnar lærðu að stýra agentinum, en þó misvel. Einnig kom í ljós að gerð þjálfunargagnanna fyrir hermilærdómsaðferðirnar hafði töluverð áhrif á skilvirkni þeirra.

Efnisorð: ML-Agents Tookit, Unity

An Evaluation of Unity ML-Agents Toolkit for Learning Boss Strategies

Lucas A.E Pineda Metz

Thesis of 60 ECTS credits submitted to the School of Technology,
Department of Computer Science
at Reykjavík University in partial fulfillment of
the requirements for the degree of
Master of Science (M.Sc.) in Computer Science

September 2020

Student:

.....

Lucas A.E Pineda Metz

Examining Committee:

.....

Dr. Yngvi Björnsson

.....

Dr. David James Thue

.....

Kári Halldórsson

.....
date

.....
Lucas A.E Pineda Metz
Master of Science

No se lo dedico a nadie.

Acknowledgements

I want to dedicate this work to my family that helped me get to this moment and fulfill another chapter in my life, specially to my brother (alegrate que te mencioné) that helped me at every step of the writing.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past two decades, applications of Artificial Intelligence (AI) in virtual environments have rapidly increased. For the case of video game environments, the use of AIs includes, e.g.: Procedural Content Generation, used to create in-game objects such as flower shapes [1] or patterns for new spaceships weapons [2]; the creation of games [3][4]; the generation and adaptation of game mechanics [5], and; authoring character behavior with examples such as Façade [6] and Versu [7]. Additionally to their use to create content, AIs can also be used to play it and test this content, as seen e.g. in General Video Game Playing [8].

The continued use and research on AI has transformed what was a group of individual studies into a well established field of research. During the past decade, different AI methods (e.g. Artificial Neural Networks, Evolutionary Algorithms, Reinforcement Learning) were introduced to video game benchmarks. Some well known benchmarks in this field include Doom with ViZDoom using Deep Q-learning [9], and Pac-Man [10] with NeuroEvolution of Augmenting Topologies [11][12], Monte Carlo Tree Search [13] and Q-learning [14]. These virtual environments were developed as platforms to enable the design, creation, and testing of Machine Learning (ML) algorithms. Such rich environments can be used to challenge an algorithm to learn how to finish a certain task and, as such, these platforms were made to be easily demonstrated and observed.

One kind of rich environment are commercial video games. Commercial video games as a learning environment are greatly beneficial since they have existing agents (e.g. characters, non-playable characters (NPC)) which can be turned into learning agents, a rich environment in which micro or macro goals (e.g. getting a high-score, and finishing a quest, respectively) can be set for an agent to optimize how to complete them and obtain a maximum reward, and with fully or partially observable states (e.g. characters stats such as health, attacks and traits, and an NPC which the health cannot be observe, respectively) of the environment. The major factor preventing the use of commercial video games as learning environments is that they are not open source, thus leaving researchers without the option to edit source code to implement the necessary learning algorithms. Additionally, the existing learning algorithms (e.g. Ape-X DQfD [15], Deep Q-Network [16], Asynchronous Advantage Actor-Critic [17]) have either reached super human level of playability on emulated video games or the environments are too simple (e.g. Arcade Learning Environment (ALE)[18]) to generate new challenges when testing the learning algorithms as stated on the literature [19][20][21], or are extremely complex and require a high computational power (e.g. OpenAI Five [22][23]). To solve these problems, recent research enabled the development of new testing environments such as Gym Retro [24], Google Research Football

[25], Retro Learning Environment [19] and Unity's Machine Learning Toolkit (ML-Agents Toolkit; [26]). In this dissertation I will be talking about Unity's ML-Agents Toolkit, since it is the testing environment chosen for experimentation.

Unity is relatively easy to use, and allows for creating one's environment, instead of relying on a pre-existing one like the case of ALE. Additionally, Unity works in collaboration with researchers and video game companies to develop new benchmarks [27], and to promote and improve the implementation of the ML-Agents Toolkit [28]. Since Unity's ML-Agents Toolkit is fairly new and is attracting interest, it is important to test the implementation of the toolkit and different aspect of it, for this reason, one of the project focuses is the usability of Unity's ML-Agents Toolkit to make an agent learn to optimize its behavior to fulfill/accomplish a given task.

Unity's ML-Agents Toolkit has two families of learning algorithms, RL and Imitation Learning (IL), each consisting of two algorithms, Proximal Policy Optimization (PPO) and Soft Actor Critic (SAC) for RL, and Behavioural Cloning (BC) and Generative Adversarial Imitation Learning (GAIL) for IL. While PPO and SAC are mutually exclusive (i.e. only one of them can be chosen), GAIL and BC can be used in combination with PPO or SAC and with each other. This complementarity led Unity to promote that the use of IL algorithms helps to speed up the learning process of RL algorithms, especially when having sparse-reward environments [26]. However, little research testing this hypothesis exists. Thus, the other focus of this project is to compare using an RL algorithm as a stand-alone learning algorithm to the same learning algorithm in combination of two IL algorithms (see Section 2.3). For this, I created an environment modeled from video game boss battles. A boss battle is a trial in which a player fights an opponent. Here, the Boss (learning agent) has to learn a behavior/policy that optimizes the use of a given set of abilities, in order to maximize the rewards obtained from the environment.

The structure of the dissertation is as follows:

- Section 2 consist of a brief introduction to Unity, RL, and the ML-Agents Toolkit.
- Section 3 defines the learning environment and both agents (learning and non-learning).
- Section 4 contains the specifics on the learning of policies, as well as results of this learning.
- Section 5 includes an interpretation on the results obtained during the learning experiments.
- Section 6 includes final remarks in the use of Unity's ML-Agents Toolkit and outlook for this project.

Chapter 2

Background

2.1 Unity

Unity is a game engine released in 2005 [29]. Due to its free license, large amount of assets, and vast variety of supporting documents and tutorials, several independent game developers adopted the use of Unity, making it one of the most popular game engines. Even though the engine was designed for game development, the opening of Unity Lab promoted research in various fields which can be applied to games (e.g. Virtual Reality (VR), AI). In 2017 Unity released the first version of their ML-Agents Toolkit as an add-on for the engine to allow game developers and researchers to experiment with different ML algorithms. The toolkit is still in active development, and is expected to become a part of the main engine in the near future.

2.2 Reinforcement Learning

Reinforcement Learning refers to learning a task in a trial-and-error manner. For this, the agent is set in a known learning environment for a finite amount of steps of time. At each step t, the agent is present in a state of the environment (s_t) where it can observe and select an action (a_t). Performing a selected action will place the environment in the next state (s_{t+1}), and the agent will receive a reward (r) from the environment. This is repeated until reaching a terminal state, and the goal of the agent is to maximize the rewards obtained.

The learning environment consists of an environment modeled from the real world (e.g. a house) in a fictional world (e.g. video games), where a learning agent is going to be set-in to learn a task (e.g. an agent finishing a Pac-Man level). It is important that the modeled learning environment is designed in such a way that: the agent will be able to observe information from it and perform one or several actions that will affect the environment; provide feedback (i.e. reward) to the agent based on the actions it performed, and; that the states of the environment at any time can be modeled through state/action pairs to predict the next state of the environment and the applicable agent's action to understand if the agent is achieving its goal.

The agent is a participant in the environment with the abilities to observe different aspects of it, and to perform actions in said environment for which it will receive feedback. The abilities of the agent allow it to interact with the environment to receive rewards, and the goal of the learning agent is to optimize its policy to get the maximum reward posible from the environment. The rewards are positive or negative

signals that the agent can receive frequently or sparsely. E.g. a learning agent set to play a level of Mario in which the agent has to maximize the reward obtained through finishing the level as fast as possible (sparse positive reward) and getting as many coins as it can (frequent positive reward), and the agent obtains a negative reward either by dying when contacting an enemy (frequent reward) or the time it has for finishing the level reaches zero before the agent gets to the end of the level(sparse reward).

Some RL method approaches to obtain an agent policy that maximizes the reward obtained from the environment include:

- Policy Gradient [30]: Policy gradient methods learn a parameterized policy ($\pi(a|s;\theta)$, probability of taking action a at state s given parameter θ) that selects the actions of the agent by learning the parameter (θ) of the policy using an stochastic gradient ascent $\nabla_\theta \sum_a \pi(a|s;\theta)R(s)$, were $R(s)$ is the discounted cumulative reward obtained from s and forward.

- Actor-critic [31]: This method learns a policy in the same manner as policy gradient. Additionally, it learns an approximation of the state-value function $V(s_t)$. The calculation of the state-value function $V(s_t, w)$, where w is a weight vector calculated with another method (e.g. Monte Carlo), is referred to as *critic* and its functionality is to reduce the variance of the gradient policy. While the policy update is referred to as *actor*, the overall policy-gradient method is referred to as actor-critic [31].

- Q-learning [32]: This method optimizes the policy by estimating the function Q for an optimal policy, where function Q is the expected discounted reward for executing an action a at state s while following a behavior policy.

Video games have been an interesting domain for researching RL because users can easily model an environment and learning agents. An example of this is the Arcade Learning Environment or Mario AI [33], two game emulators where the environments are Atari games and Mario, respectively. These environments have well defined states that can be observed by the agent and a player modeled by an agent that can perform simple actions and receive rewards to win the game. The drawback of such environments is their unchanging nature (e.g. no variation on the amount of enemies a level presents), the low and fixed amount of actions that an agent can perform (e.g. the game *Space Invader* only has three actions) and the environment representation are done in 2D images. These factors make this environment trivial for new learning agents such as Ape-X DQN [34] or Rainbow [35]. Because of this, new environments such as Unity's ML-Toolki were created. In these new environments, variability can be added to the environment, as well as a variable amount of actions. Additionally, these new tools allow for the creation of 3D environments.

Some RL method applied to video games include:

- Deep Q-Network (DQN)[16]: A method which uses Q-learning as a basis where a convolutional neural network calculates the approximated Q value. An important factor of DQN, is the replay buffer (s_t, a_t, r_t, s_{t+1}) used for randomly sampling from past experiences when updating the network. This was the first learning algorithm to obtain super-human level scores on 39 games on ALE.

- Trust Region Policy Optimization (TRPO)[36]: A policy gradient method that maximizes an objective function based on the policy estimator constrained by

the distance between $\pi_{\theta old}$ and π_θ (Formula 2.1).

$$\begin{aligned} \underset{\theta}{\text{maximize}} \quad & \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}},a\sim q}\big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}Q_{\theta_{\text{old}}}(s,a)\big] \\ \text{subject to} \quad & \mathbb{E}_{s\sim\rho_{\theta_{\text{old}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s)] \leq \delta \end{aligned} \tag{2.1}$$

Where q is a sampling distribution over $Q_{\theta_{\text{old}}}$-values, and $[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s)]$ is the distance measure between the $\pi_{\theta old}$ and π_θ given the KL divergence. TRPO was tested by playing 7 games present in ALE: Beam Rider, Breakout, Enduro, Pong, Q*bert, Seaquest and Space Invaders.

- Asynchronous advantage actor-critic (A3C)[17]: A variation of actor-critic method that uses multiple agents learning in parallel, where each of them calculate a parameterized policy and a value function and store the information in memory. The stored information of the multiple agents is sampled by a convolutional neural network that outputs a single policy $\pi(a_t|s_t;\theta)$ and value function $V(s_t;\theta_v)$. A3C was tested over 57 Atari 2600 games in ALE.

2.3 Machine Learning Toolkit

The Unity Machine Learning Toolkit (ML-Agents Toolkit) [21] is an open-source package project that can be added to an existing Unity project, enabling the creation of environments for training intelligent agents. This add-on was created as a way to promote the creation of new complex environments for researching new types of algorithms or state-of-the-art algorithms, for testing these algorithms in new environments, and to enable game developers to implement agents to serve multiple purposes during development.

An agent is any game object present in the environment which is created by adding the following scripts: a) Agent, a Unity script to define the game object as an agent of ML-Agents Toolkit; b) Behavior Parameters, a component of the agent that functions as the "brain" of said agent (see Section 2.3.1), and; c) Decision Requester, communication script to link the *Academy* and the agent.

The *Academy* is a class structured within the ML-Agents Toolkit. This class is responsible for several actions, e.g. instantiating the agents, keeping track of the amount of steps that ticked and the amount of episodes run since the beginning of the training.

The ML-Agents Toolkit has four main features:

1. Learning Environment. This represents the scene created in Unity where the agent will be set. Here, the agent can make observations (see Section 3.2.2) of the environment, perform actions (see Section 3.2.3), and receive rewards (see Section 3.2.4) for doing a specific task.

2. Low-level Python API. An external component used to connect the learning trainers to the environment agent.

3. Communicator. This feature belongs to the *Learning Environment* and handles the communication between the brain of the agent and the low-level Python API.

4. Python Trainer. An external feature used to define learning algorithms. The final product given by the trainer is a Neural Network (NN) model, also referred to as a behavioral model, which controls the behavior of the agent.

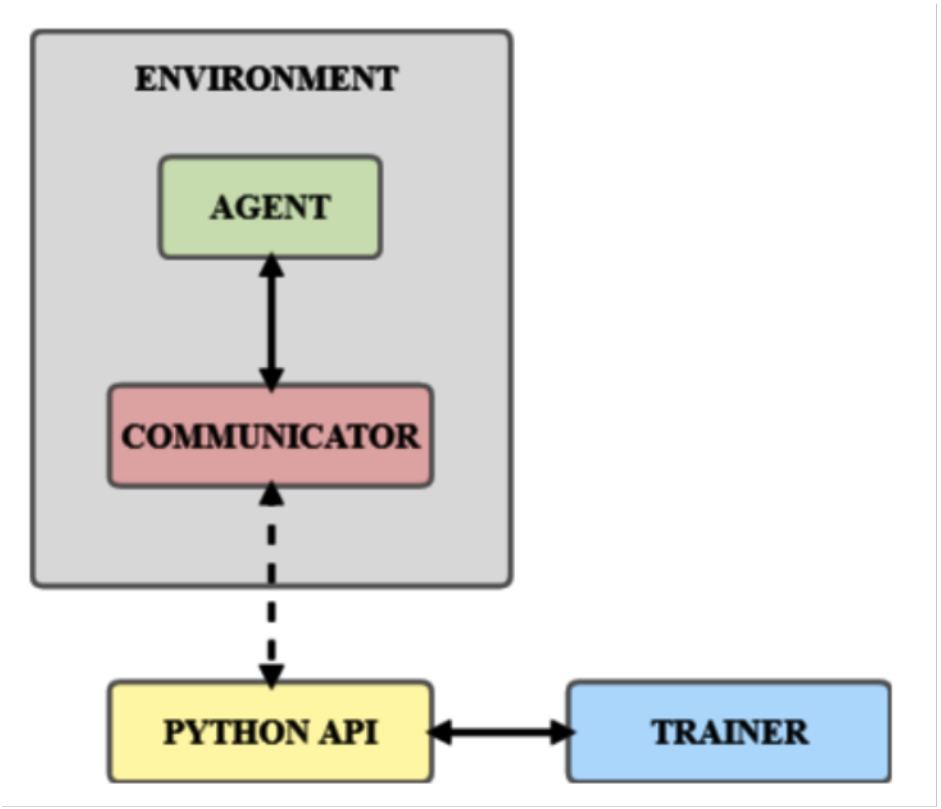

Figure 2.1: High-level representation of the ML-Agents Toolkit

2.3.1 Brain

The brain of the agent is in the *Behavior Parameters* script attached to a game object (as seen on Figure 2.2). The functionality of the brain is to communicate the observations and rewards to the trainer's API and tell the agent which action to perform.

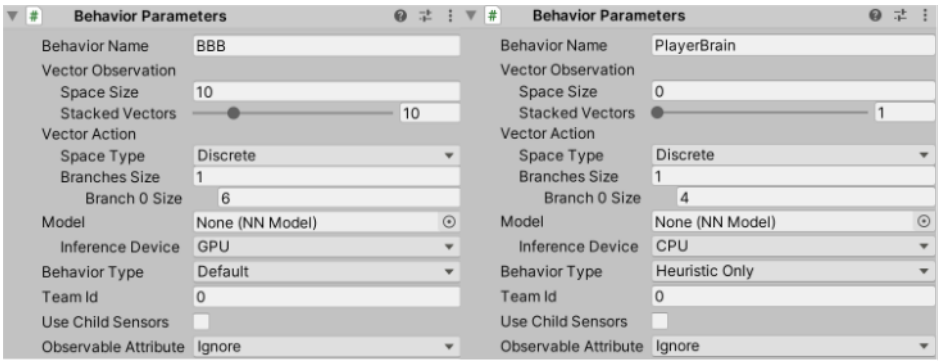

Figure 2.2: Brain example, on the left "Boss" brain and on the right "Player" brain.

Composition of a brain:

- Behavior Name: A name given to the set of hyper parameters used by the trainer algorithm.
- Vector Observation: The amount of variables that the agent can observe from the state of the environment.
- Vector Action: Array of float numbers or integer numbers depending if it's a continuous or discrete vector type. Use by the agent to choose the actions.
- Model: Corresponds to what NN is using the brain for determining the actions to apply. It is also referred to as an agent behavior.
- Behavior Type: The brain will select which action to perform according to the behavior type selected. There are three types of behavior:
  - Inference: The actions selected by this type are decided by the model created by the algorithm after training. The agent receives the action from the model with the *OnActionReceived* function.
  - Heuristic: The actions are hand coded inside the function Heuristic of the agent's script and then pass to the function *OnActionReceived*.
  - Default: The actions are selected and sent to *OnActionReceived* by the model being created and changed during training.
- The remaining three concepts are *Team ID*, *Use Child Sensors* and *Observable Attribute* but are not considered since they escape the scope of this project.

2.3.2 Training

A training session starts after the agent behavior is set as *Default* and the environment is executed (see Section 3.4). During these sessions, the agent makes observations, receives rewards, and performs actions in order to optimize a policy. The optimal policy is learned using one or more state-of-the-art algorithms included in the ML-Agents Toolkit. The ones within the scope of this project are:

- Proximal Policy Optimization (PPO). This is a RL algorithm which optimizes a policy by using a stochastic ascent to optimize a surrogate objective function [37]. PPO is based on TRPO; instead of maximizing the objective function with a hard constraint (Formula 2.1) PPO penalizes changes on the policy that move $r_t(\theta)$ away from 1, where $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$. This is done by Formula 2.2

$$L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)] \tag{2.2}$$

  Where ϵ is the exploration hyper parameter, the first term is the minimum of the unclipped objective and the second term is a clip function that modifies the surrogate objective by clipping the probability ratio, thus, removing the incentive of moving outside of the range $[1-\epsilon, 1+\epsilon]$.
- Soft Actor Critic (SAC). This is a RL algorithm which maximizes the expected return and entropy needed to get an optimal policy [38]. This is achieved by parameterizing a Gaussian policy and a soft Q-function by collecting experiences from the environment and a replay pool to update both functions.

- Generative Adversarial Imitation Learning (GAIL). This is an IL algorithm that confronts two NNs to generate an optimal policy, one being the generative neural network (NN) and the second an expert discriminator [39]. The policy is optimized by the generative NN which tries to generate a policy such that the discriminator cannot distinguish if the policy is drawn from the demonstrations provided or if it is created by the generative NN. The discrimination is done by comparing the generative NN occupancy measure policy and the expert occupancy measure policy. Based on the difference on these parameters, the expert NN penalizes the generative NN (i.e. reduces the obtained reward).

- Behavioral Cloning (BC). This is an IL algorithm used in combination with PPO, SAC or GAIL to obtain a policy which mimics the one stored in a demonstration file. This is done by pre-buffering the set of actions from the demonstration file which is used by the algorithms to update their policies.

The environment is complemented by a configuration file that contains the *Behavior name* which is used to distinguish to which agent each behavior belongs to. The file also contains the learning algorithms used, and the set of hyper parameters of the different learning algorithms. Furthermore, for GAIL and BC a demonstration file containing sets of environment states and actions is included in the environment.

In RL information from the environment is taken and used to optimize policies, whereas in IL information is taken from the demonstration file to optimize the policies. Thus, PPO and SAC can be used without complementing algorithms and can be used as the base learning algorithms. From these, PPO is considered the default one by Unity if no trainer is specified on the configuration file. PPO, SAC and GAIL can be used as stand-alone learning algorithms for the agent. In contrast, BC is the only trainer which has to be used in combination with a second algorithm. This is due to how BC works by storing information into its memory, which can be accessed and used by other training algorithms, i.e. it works as a complement to PPO, SAC and GAIL. Due to GAIL and BC being IL algorithms, they are best used to compliment RL algorithms. As such, the aim of this work is to compare the learning performance when using PPO alone, or when combining it with GAIL and/or BC.

Additional to the differences in terms of how each algorithm does the learning, the rewards system varies depending on the algorithm. The ML-Agents Toolkit assigns different types of rewards for the PPO, SAC and GAIL. While PPO and SAC are assigned extrinsic rewards, meaning the reward that the algorithm receives comes from the environment (see Section 3.2.4), GAIL is assigned an intrinsic reward, meaning the reward received is based on the discriminating value given by the expert discriminator to signal if the generative NN is fooling the expert or not.

The training algorithms are built on top of the open source library TensorFlow and the output of the trainers is stored as a behavioral model file. During the training, several information (e.g. environment steps, reward values, entropy) is recorded for later analysis and shown through TensorBoard. This information is useful to analytically interpret what the agent is doing and if it is training correctly.

2.3.3 Demonstration Recorder

The ML-Agent Toolkit comes with a script used to record episodes. This recorder generates a file containing: demonstration name, total number of steps run during the

recording, number of episodes recorded, mean reward obtained at each episode, the observation of the agent, the vector action of the agent and the type of vector action.

2.4 Summary

In this chapter, I presented the necessary background material, including ML-Agents Toolkit structure and the different learning algorithms that will be used in this thesis. The next chapter will deal with how these concepts will be applied in a learning experiment combining PPO with ILs, which will be used by a learning agent to optimize its policy to defeat a simpler non-learning agent.

Chapter 3

Method

The idea is to test the performance of the base learning algorithms that come with the Unity ML-Agents Toolkit by having an agent learn the best strategy to follow to win a fight with another simpler non-learning agent. For this, I created an environment with two agents. While both agents have a fixed amount of attacks/actions with their own stats (e.g. Damage or Cooldown), each agent has a distinct functionality.

Our main agent, defined as "Boss" or learning agent, has the ability to learn new sequences of actions in order to win and optimize the rewards it gets. The changes done to old sequences (i.e. the learning) are based on the information the Boss gets from the observations of the environment (see Section 3.2.2). Contrastingly, the "Player" or simple agent, lacks the ability to learn, observe, receive rewards, and it instead follows a fixed policy which remains unchanged throughout all episodes.

In this chapter, I detail the different aspects of the environment (e.g. state space), the actions that the agents can perform and the different observations that they can make, how training is structured, and the different rewards that the Boss can get.

3.1 Agent Environment

Inside Unity I created an environment for testing the different training algorithms provided by the ML-Agent Toolkit. As shown in Figure 3.1 our environment consists of:

1. An agent defined as "Boss". This is our subject of interest, and the one who will do all the learning. The Boss has attack abilities and it will remain at its starting position for the duration of the episode.

2. An agent defined as "Player". In contrast to the Boss, this agent has a simple behavior, lacks the capacity to learn new actions, and has the ability to attack and move within the environment.

3. A rectangular arena where the agents exist and move. This area is $5m^2$ and is designed to allow a user to view the domain of the fight.

4. A Graphical User Interface (GUI) that shows the current health of the agents. The left bar corresponds to the Boss and the right bar belongs to the Player

The environment works using *Fixed Updates*, a concept inside Unity's game engine to indicate the frequency in which the environment is updated. The function *FixedUp-*

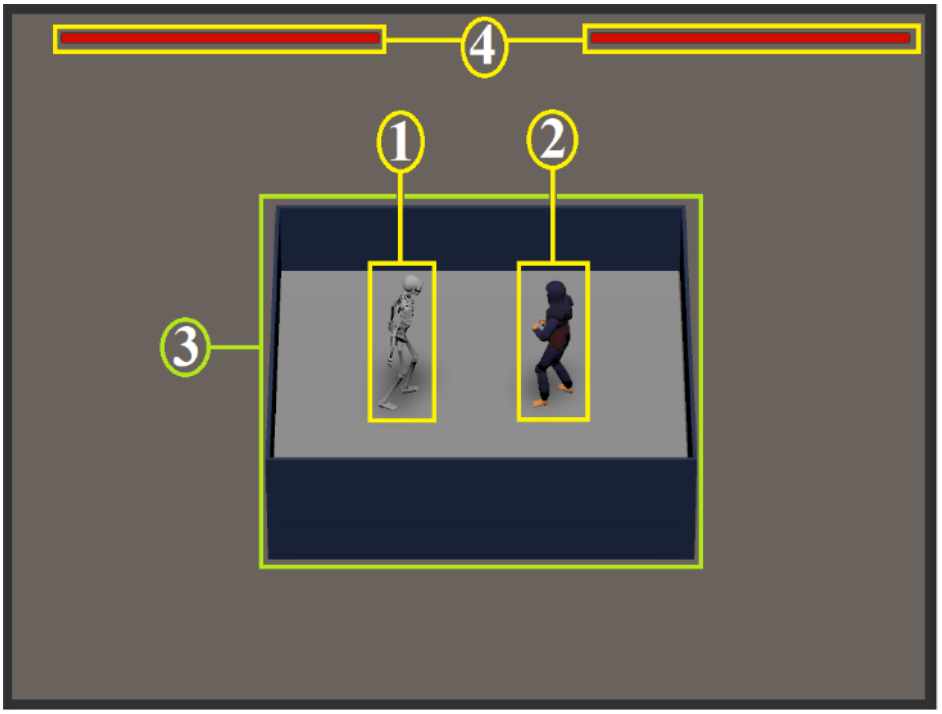

Figure 3.1: Illustration of the basic work environment. 1) Boss agent; 2) Player agent; 3) Area where the agents are restrained; 4) GUI element showing the agents' "health" left bar corresponds to the Boss and right bar to the Player.

date is called every frame of the running project. The time that passes between frames is 0.02 seconds.

3.2 Learning Environment

The environment is designed to reflect a video game boss battle where either a player must fight and defeat a stationary boss, or the boss must win the fight against the player. This project focuses on the point of view of the Boss, that has: a) a set of actions (Section 3.2.3) that can be performed in the environment and will affect the Player in different ways; b) observations (Section 3.2.2) that the agent receives to sense information from the environment state (Section 3.2.1) and learn the consequences of the actions it selected at a given time, and; c) reward (Section 3.2.4) gotten at the end of each fight based on its own performance.

3.2.1 State Space

The environment state is computed in a frame-by-frame fashion and consists of a set of variables regarding the three-dimensional position of both agents, actions performed/available, Boss status, and agents' health (see further below). The relevant information included in the state of the environment is:

- Agent health. Measured with a variable of type float. This is given by a script which keeps track of this value that can vary between the values 0 and 100. The learning agent can observe its opponent's health at every state of the environment (see 3.2.2).

- Agent Position. Both agents have access to the three-dimensional vector variables of their own position and that of their counterpart, which they use to calculate the distance between each other.
- Distance between agents. The distance is saved as a float variable which is calculated at the end of every frame. This variable is used to determine the effectiveness of the attacks (See Section 3.2.3).
- Action Selected. This is a scriptable object variable used by the Boss to keep track of the selected action. The Player can also use this to react to the Boss' selected action (see Section 3.2.3).
- Arena size. Distance set by the user with the Unity editor using a variable of type float. The distance will affect the range of the attacks available to both agents, since the ranges are measured as portions of the arena size. In this context, an attack can damage the target if it's in a distance equal or below to a quarter of the arena size.
- Frame second. A variable of type float used to indicate the environment time in seconds. This is calculated in a frame-by-frame fashion from the beginning of an episode (see Section 3.4). All time dependent parameters are calculated based on this variable, i.e. *Action Time* see Section 3.2.3 for further details.
- Boss vulnerability check. A boolean type variable, that indicates if the Boss is in a state of vulnerability or not. See Section 3.2.3 for further details. This is tracked by the same script that handles the health value. Both agents can see if the Boss is in its vulnerable state or not.
- Boss vulnerability time. Measured in seconds and saved on a variable of type float. It indicates when the Boss will stop being in its vulnerable state. Only the Boss has access to this variable.
- Time until next attack. This is a variable of type float which indicates the time, in seconds, until the next action of an agent can be executed. The time is calculated using *Frame Second* plus *Delay Time* (see Section 3.2.3). Until this given time is reached, the agent is unable to attack. The value of this variable is unique to each agent and can only be known by the corresponding agent.
- Attack availability. Every attack has a boolean variable that is used to know if the attack is ready to be used or not. This state will change according to the attacks' cooldown (see Section 3.2.3), after the cooldown has passed the attack will be ready to use again.

3.2.2 Observable Traits

Observable traits are represented by the variables present in the environment which change during the course of an episode. This information is used by the trainer to learn the optimal policy for a given problem. The observations are made at the beginning of each step.

These observations are crucial for the learning of the agent since they are going to affect how well the trainer algorithms learn. The information fed to the agent has to be sufficient to accomplish the task at hand, which in this case is to find the best

strategy to maximize the rewards obtained at the end of the episode. The Boss agent has ten different observations:

1. Current health of the Boss. This is represented by a variable of type float, that corresponds to the State Space variable *Agent health*. This variable measures the health that the Boss has at each frame of any given episode.

2. Current health of the Player. This is represented by a variable of type float, that corresponds to the State Space variable *Agent health*. This variable measures the health the Player has at each frame over the episodes.

3. Distance to target. Variable of type float, calculated at the end of each frame by using the State Space variable *Agent Position*. This is used to determine if an attack will be effective or miss.

4. Boss vulnerability. A true or false statement given by the State Space variable *Boss vulnerability check*. It is used to learn which attack gives the vulnerability status to the Boss. In combination with *Current health*, it is used by the Boss to learn that during this state it will receive more damage.

5. Boss vulnerability time. Indication of the frame at which the vulnerability state ends, taken from the State Space variable *Boss vulnerability time*. This observation is used by the Boss to learn how long the vulnerable state will last.

6. Attack readiness. A true or false statement indicated by the State Space variable *Attack availability*. It is used to know when an attack can be performed. There is one attack readiness observation for each attack.

Since the Player behavior is based on a scripted behavior, i.e. it does not learn during the training sessions. The agent does not receive any observations from the environment but instead has hand coded conditions.

3.2.3 Action Space

During an episode both agents have the ability to choose an attack from the pool of available actions given (see Table 3.1-3.2), but only the Player agent can move inside the arena (see Table 3.3).

Stat \ Name	Null	Light	Heavy	Poison	Stun	Gust
Activation Time (s)	-	0	0.06	0	0.02	0.02
Delay Time (s)	-	0.02	0.2	0.02	0.04	0.06
Range	-	1	1/3	1	1/3	1
Damage (%)	-	3	25	-	-	-
Attack Cooldown (s)	-	0.2	0.8	0.28	0.44	0.64
Status Effect	-	-	Stun	Poison	Stun	Push
Status Duration (s)	-	-	0.04	0.1	0.08	-
Status Damage (%)	-	-	-	0.4	-	-

Table 3.1: Set of available attacks for the Boss agent.

Every attack has the following properties:

Stat \ Name	Null	Light	Medium	Heavy
Activation Time (s)	-	0	0.02	0.04
Delay Time (s)	-	0	0.02	0.02
Range	-	1/4	1/4	1/4
Damage (%)	-	1	2	5
Attack Cooldown (s)	-	0.1	0.2	0.25
Status Effect	-	-	-	-
Status Duration (s)	-	-	-	-
Status Damage (%)	-	-	-	-

Table 3.2: Set of available attacks for the Player agent.

- ID. It corresponds with the index within the *Action vector* selected by the brain.
- Action Time. Variable measured in seconds representing the waiting time between selecting an action and its effect being applied in the environment.
- Delay Time. Time measured in seconds which is used to calculate the *Time until Next Attack* state of the environment.
- Range. The range in which the attack is effective and will not miss. Measured as a fraction of the arena size.
- Damage. How much of the current health's target is going to diminish. Measured as a percentage of the target's maximum health.
- Attack Cooldown. The amount of time that it needs to pass until the attack is ready again. Changes the state variable *Attack Availability* to true once the time has passed.
- Status Effect. Indicates if the attack applies a negative status to the target.
- Status Duration. Indicates the duration, in seconds, of the status effect applied to the target.
- Status Damage. How much damage is dealt by the status effect to the current health's target. Measured as a percentage of maximum target's health.

Before the attack is applied in the environment, the following conditions must be met: The attack needs to be ready, the *Time until Next Attack* must have been reached, and the target needs to be within the attack range. Each action has a range of effect within which damage can be dealt to the target. when an attack is used but the other agent is out of range, the attack will not deal any damage, this is known as missing an attack. I also denote the attacks as effective or ineffective based on landed and missed attacks.

As stated at the beginning of this section, only the Player can move. When the Player is not idle or taking advantage of the Boss' vulnerable state, the movement speed has a unique value. All movements done are intended to either keep the Boss within the Player's attack range or to elude the Boss' *Heavy* attacks. Additionally, the Player will run towards the Boss to take advantage of its vulnerable state. By reading

Stat \ Name	Boss	Player
Type	Idle	Idle, Walking & Running
Walking Speed (cm/seg)	-	2.0
Running Speed (cm/seg)	-	5.8

Table 3.3: Movement actions of each agent

the state space *Action selected* of the Boss, the Player can detect that a *Heavy* attack is about to come, making the Player run away from the range of the attack. While the Boss does not have any means of movement, he can push the Player backwards instead.

The status effects that the Boss can apply are:

- Poison damage. The target receives a total damage of 2% (0.4% per second) of its max health during the duration period of the status effect.
- Stun. The target is unable to act while stunned.
- Push. The target is pushed backwards a distance of 1/6 of the arena dimensions.

Using his *Heavy* attack will render the Boss vulnerable during the *Time until Next Attack* period, making the attacks of the Player to do an additional 3% of damage. For this setup the *Heavy* attack will hit the target Player only if it is stunned before performing the action, otherwise the attack will always miss.

The actions *Heavy* attack and *Stun* are considered hard actions, while the rest are considered easy actions. Thus, the combined use of *Stun* and *Heavy* attack can be regarded as a hard set of actions.

3.2.4 Reward System

The agent can get two possible rewards, positive or negative. Both rewards are given at the end of the episode if one of the agents is killed. If the end of an episode is reached because > 500 steps were used, the learning agent will not receive any reward. Based on the rewards obtained in an episode, the trainer algorithm optimizes its policy according to the rewards gotten from the environment through the episodes.

Rewards are calculated by adding a fixed value to the ratio between the current health and the maximum health times a fixed factor (Formula 3.1 & 3.2). Positive rewards are calculated based on the health of the Boss (Formula 3.1) and given when the Boss wins the fight, whereas the negative rewards are calculated based on the health of the Player (Formula 3.2) and are given when the Boss loses the fight. The reward value fluctuates between the ranges -1 and -0.5 for the negative and the ranges 0.5 and 1 for the positive.

$$R = 0.5 + BossCurrentHealth/BossMaxHealth * 0.5 \tag{3.1}$$

$$R = -0.5 - PlayerCurrentHealth/PlayerMaxHealth * 0.5 \tag{3.2}$$

The first fixed factor in the equations is used to ensure the rewards always give a positive/negative value, whereas the second fixed factor is used as an indicative bonus of how good of a win or lose a situation was. Calculating rewards in this fashion,

i.e. including two fixed factors, causes the reward variability to remain relatively low which, in turn, makes the learning of the agent more efficient. This was found during the testing and development of the agent and environment structures. These tests showed that having a simple reward with one fixed factor and another variable which depends on the agent's behavior gives better results than having multiple rewards and one major reward at the end of the episode. Furthermore, having a simpler system facilitates data analysis.

3.3 Agent Task

The objective of the learning agent is to optimize its behavior to get the maximum amount of positive reward, which is done by strategizing over the six actions given. The battle strategy consists of learning when an action can be performed (i.e. the attack conditions must be met), the amount of attacks to be done over a fight, and the effects that the actions apply to the environment. An extract of how an optimal fight looks is provided in Appendix A Figure A.1.

3.4 Training Structure

The training is divided into individual episodes which consists of the steps taken by both agents in each frame. These steps are ticked by the *Academy*. During the episodes played in the training, several events occur:

- An agent selects one action from his vector of available actions for every step of the episode.
- The training algorithm processes the information generated by the state observations, chosen actions, and obtained rewards. Based on these data, the training algorithm optimizes its policy in order to pick the best possible sequence of actions, thus maximizing the rewards to be obtained from the environment.
- The information gathered during training is saved for later analysis. This happens at every step in a frame-by-frame fashion.
- All the changing state variables get updated according to the effects applied by the actions performed by the agents.

The episode starts when the *Academy* uses the *OnEpisodeBegin* function. This function sets the environment to its starting position, i.e. it locates the agents at their initial positions and sets environment variables as well as specific agent values. Once the episode starts and until the end of it, the agents select and perform their actions in a frame-by-frame fashion. The episode comes to an end when one of three possible outcomes is reached:

1. When the current health of the Player ≤ 0. This is what I refer to as the Boss killing the Player, which is considered as winning a fight.
2. When the current health of the Boss ≤ 0. This is what I refer to as the Player killing the Boss, which is considered as losing a fight.

3. Over 500 steps are used.

In the case the health of both agents reaches zero at the same time, the Boss is declared the winner of the fight.

When an episode concludes the agent will get a reward (see Section 3.2.4) and the *Academy* calls the *OnEpisodeBegin* function as at the beginning, that resets all values to start a new episode. This process is then repeated until the end of the training.

3.5 Summary

In this chapter, I presented the learning environment to be used in the evaluation and different aspects of it (e.g. states space, observations), the agents considered for the evaluation setup, as well as their abilities. Additionally, other aspects considered for evaluating the learning, such as positive/negative rewards, and the structure of the training sessions were also described. The following chapter describes in detail the evaluation of the learning and the results obtained of the three case studies.

Chapter 4

Experimental Evaluation

4.1 Evaluation Setup

To evaluate the performance of the different learning algorithms presented on Section 2 and the combination of them, I run a series of training sessions. Every session runs until the agent completes 800k steps, the testing sessions and tuning of the hyper parameters show that the behavior of the agent becomes stable and predictable after the 800k mark. While the training is being performed, the ML-Agents Toolkit saves information that will be used to analyze the final performance of the agent. In addition to the data recorded by the toolkit, I saved additional data in two different files that contain the following information from the episodes:

1. Fight Logs. One text file for each agent per episode played. These files include information about: The current time of the episode, the distance between the agents, the attack performed in that frame, the damage done by the attack, the time in which it would be ready again, and the health of the agent. These files are created for both agents.

```
0,02;3,75;-;-;-;100
0,04;3,71;bp;0;0,32;100
0,06;3,67;-;-;-;100
0,08;3,63;bh;0;-;100
```

Figure 4.1: This is an example of how the Fight Log file looks like.

2. Attack summary. One text file per training session. In here, I record the amount of *Light* attacks performed, the amount of successful *Heavy* attacks, the amount of unsuccessful *Heavy* attacks, the amount of *Poison* attacks performed, the amount of successful *Stun* attacks, the amount of unsuccessful *Stun* attacks, the amount of *Gust* attacks performed and who won the fight. Every line on the file corresponds to one episode. This file in combination with the *Fight Logs* can be used to determine the differences between all final policies (see Sections 4.2 and 5)

The training sessions are divided into three case studies which use four trainers for evaluating the learning process: a) PPO; b) PPO + GAIL; c) PPO + BC, and; d)

```
8;0;5;9;3;2;3;Boss
6;0;4;9;4;3;5;Boss
9;0;5;8;6;1;5;Boss
12;0;5;10;3;3;4;Boss
```

Figure 4.2: This is an example of how the Attack Summary file looks like.

PPO + GAIL + BC. The SAC learning algorithm was not used because of performance issues with the hardware used. While all cases use the same trainers, the demonstration file used for training with GAIL and BC is different for each case. All trainings were run using an Alienware 17 R3 laptop with the following specifications:

- Processor: Intel(R) Core(TM) i7-6820HK CPU @ 2.70GHz
- GPU: NVIDIA(R) GeForce(R) GTX 980M with 8GB GDDR5
- RAM: 16GB (2x8GB) DDR4 2133MHz
- Storage: 512GB PCIe SSD.

The first case study uses a demonstration file with episodes in which the Boss only does the best course of action throughout the fight, finishing the episode without getting damage and obtaining the maximum reward possible. I refer to this set of actions as the optimal fight or optimal policy, and the file is referred to as the optimal file.

The second case study uses a demonstration file with episodes in which the agent follows the action given by a behavioral model created during training using only PPO. The actions done in this fashion will always result in a sub optimal fight due to a sub optimal policy, since the Boss never wins without suffering some damage.

For the last case study, I used a demonstration file filled with a mixed behavior. The episodes use two model behaviors, first it would do three fights with the PPO model behavior and then three fights with the optimal behavior. This is repeated until the end of the demonstration recording.

The demonstration files created follow that structure to test the ability of both learning algorithms to produce a favorable result when trying to imitate a behavior with hard actions, a behavior with only easy actions and a combination of both. Additionally, this allows to check the overall impact of the demonstration file. Each demonstration file includes 40 episodes.

The following Tables 4.1, Table 4.2 and Table 4.3 depict the hyper parameters used for each algorithm during the training sessions. All values were based on the recommendation given by Unity and the testing done.

4.2 Results

I base the performance analysis of the different RL algorithms and combinations used, on three different data collected by the tool and the two manually created files mentioned in Section 4.1.

Hyper parameter	Value
Batch Size	360
Buffer Size	4320
Learning Rate	0.0001
Beta	0.00001
Epsilon	0.15
Lambd	0.95
Num Epoch	10
Learning Rate Schedule	Linear
Hidden Units	12
Number of Layers	6
Gamma	0.99

Table 4.1: Hyper parameters for setting PPO

Hyper parameter	Value
Encoding Size	256
Learning Rate	0.0001
Gamma	0.99
Demonstration File	Optimal / Suboptimal / Mix

Table 4.2: Hyper parameters for setting GAIL

Hyper parameter	Value
Steps	100000
Samples per Update	0
Number Epoch	10
Batch Size	360
Demonstration File	Optimal / Suboptimal / Mix

Table 4.3: Hyper parameters for setting BC

For evaluating the learning, three aspects given by the tool were considered. The first aspect is the mean cumulative reward obtained for every episode played. With this, I can determine if the agent is successfully learning the task, and the rate at which the task is being learned. I consider a task to be learned successfully when the mean reward value reaches and maintains for the remaining duration of the session a reward > 0.75, and; the median reward value > 0.75. The median reward value is calculated considering all rewards obtained in a training session.

The second aspect taken from the tool is called entropy, which can be used to observe changes in the policy during the session run and how randomly the agent chooses its actions. The policy is considered to choose actions randomly when the median entropy > 1.2. The median reward value is calculated considering all rewards obtained in a training session.

The third aspect taken from the tool is called GAIL Estimates. The estimates correspond to the expected occupancy of the two policies, one being the expected value of the generative NN and the second the expected value of the expert NN. This value is important to determine if the generative NN is fooling or not the expert NN. I

consider that the generative NN is fooling the expert NN when the difference between the expected values is < 0.35.

Another aspect to consider is the randomness in the actions performed at the beginning of the training. Every implementation starts doing random actions to test their outcome. Each random action is performed with the same amount of repetitions, e.g. if action A is done 8 times, then actions B-E will also be done 8 times each.

Additionally to the above mentioned aspects, for each case study and agent, the average use of all available action was calculated based on the last ten episodes. These were used in an Attack selection analysis in order to compare the attacks selected by the different agents with the optimal fighting policy. For doing the comparison the fight was divided into three sections, defined based in the average length of the episodes: a) beginning, for the first case this section consisted of the first 90 steps, whereas for the second and third cases, this section consists of the first 80 steps; b) middle, for the first case this section expanded from the 90th to the 180th step, whereas for the second and third the middle section was from the 80th to the 160th step, and; c) final, for all cases, this section consisted of the steps remaining. Using this approach, all comparisons are assured to have the three fight segments with equal length (beginning, middle, final), regardless of the fight duration. For the optimal fight policy (fight done by an user), the segments assigned were similar to those considered in each case study. During the first case study, the optimal fight was shorter than those made by the learning agents. Thus, the final stage was never reached when considering the optimal fight policy, i.e. fights came to an end in the middle stage of the fight.

4.2.1 Case Study N°1

For the first case, the demonstration file consisted of the best policy possible. While the agent's learning was successful whenever GAIL was absent, the inclusion of GAIL resulted in poor learning (Figure 4.3). In the case of PPO + GAIL this occurred after the 280k steps mark, whereas in the PPO + GAIL + BC this happened after the 430k steps mark (Figure 4.3).

During the first 110k steps, GAIL, BC, and the combination of both improve the learning rate of PPO. This is related to the action selected by the policy while training. Every trainer starts in the same fashion, selecting random actions, and observing the impact these have over the environment and the reward gotten from using them. This is reflected by the high entropy of the training (Figure 4.4), which shows how random the selection of the actions was. Overall, the median entropy of successful learning was < 1.2, whereas for unsuccessful learning it was > 1.2 (Table 4.5).

Using PPO in combination with GAIL (PPO + GAIL) helped in the learning and to get the maximum reward, even when reaching a sub-optimal policy. However, after the 430k steps mark GAIL became unstable when trying to imitate the hard actions (see Section 3.2.3) in the demonstrations file, which GAIL is unable to replicate.

The inability of GAIL to replicate the hard actions creates a conflict between the NNs which compete against each other. This conflict results in the expert not being fooled by the policy which, in turn, causes the generative NN responsible for the policy to get locked in a state in which it repeatedly tries to imitate a behavior unsuccessfully. The conflict results in the agent to lose fights (i.e. in more and higher negative rewards) or to win with little to no health (i.e. less and low positive rewards). This is shown in Figure 4.5 where the differences between the expected policy values

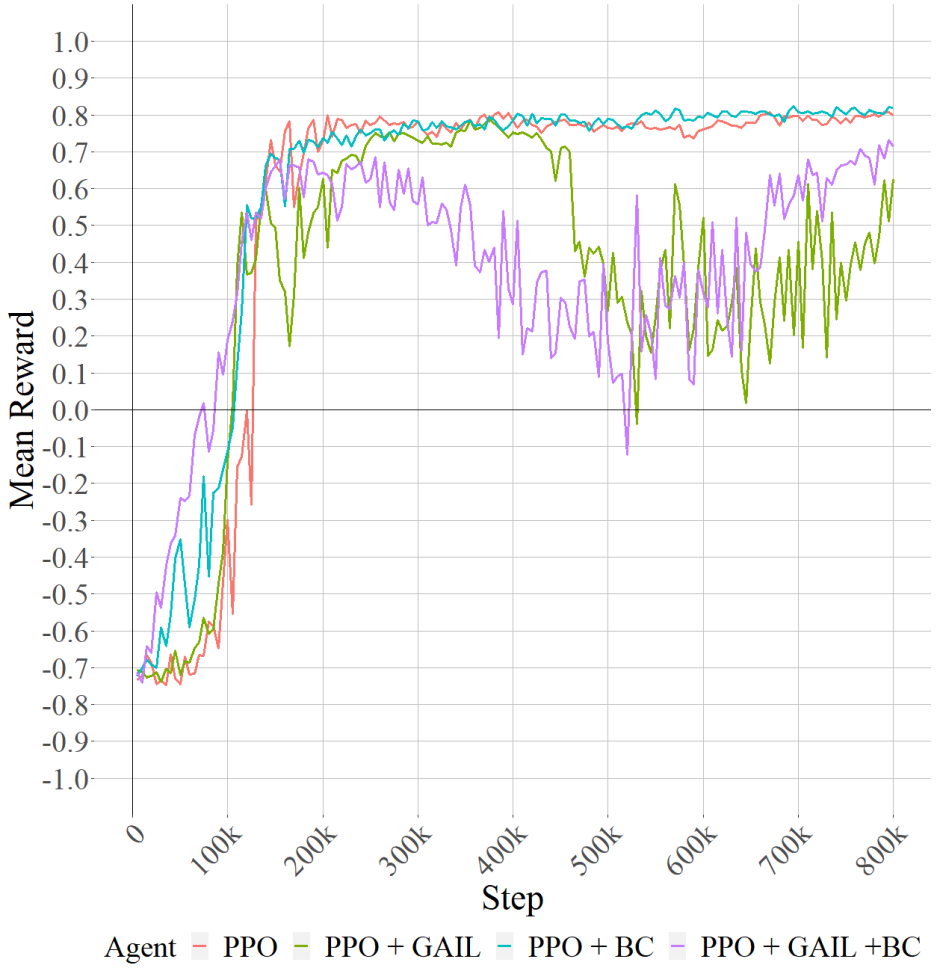

Figure 4.3: Mean rewards that the agent got using four different trainers combination

have a large difference between them (>0.35), resulting in the discriminator penalizing the generative NN, thus reducing the obtained reward.

The addition of BC to PPO (PPO + BC) enhances the initial training in an unstable fashion. This instability comes from the agent trying to copy the hard action with mixed results, which hinders in a small amount the learning and rewards obtained by the agent. After the 100k BC stops working, making both, the reward's increment and agent's learning to resemble those of the case where PPO worked alone from the start of the training. However, after half the training has passed, PPO does slightly better thanks to the initial boost and the experience it provides for correctly doing some of the hard actions. The logs (*Attack Summary* and *Fight Logs*) showed PPO alone to never learn how to use the *Heavy* attack or the *Stun*, whereas PPO with BC learns how to use *Stun*. While only one hard action is learned, this allows the agent to finish the fight receiving less damage.

Combining GAIL and BC improves the start of the training and makes the reward earning more stable. While GAIL helps PPO imitate the hard actions, BC does the same for the hard actions. Once BC stops functioning at the 100k steps mark, GAIL suffers the same problem as observed when used alone (case PPO + GAIL). This problem is compensated by the experience provided by BC which enables the agent to perform hard actions and combine them. However, due to GAIL's instability, hard actions are not always successfully imitated.

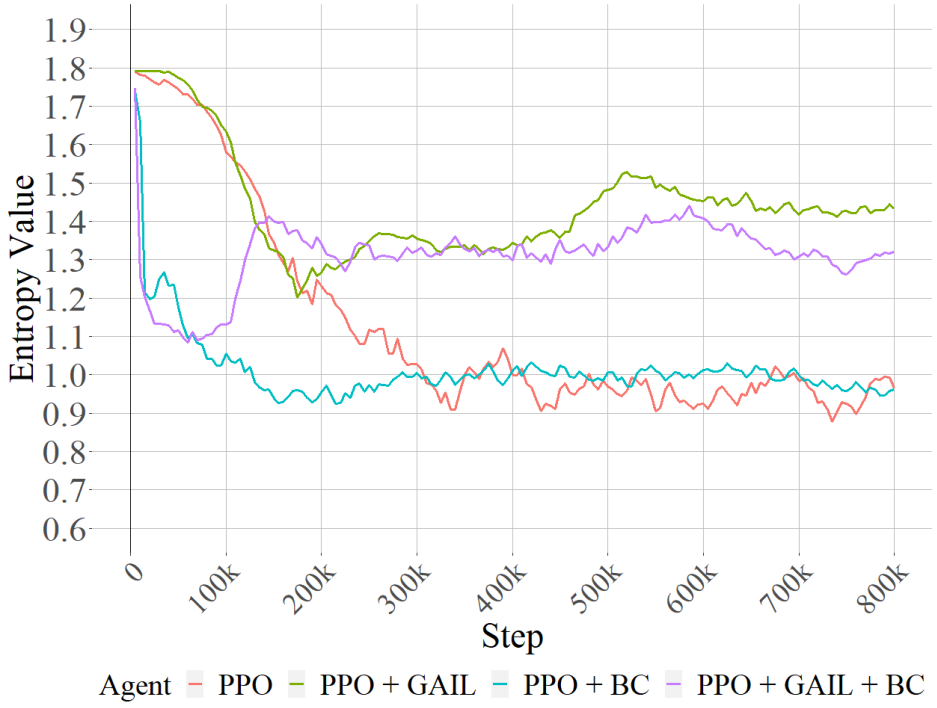

Figure 4.4: Entropy

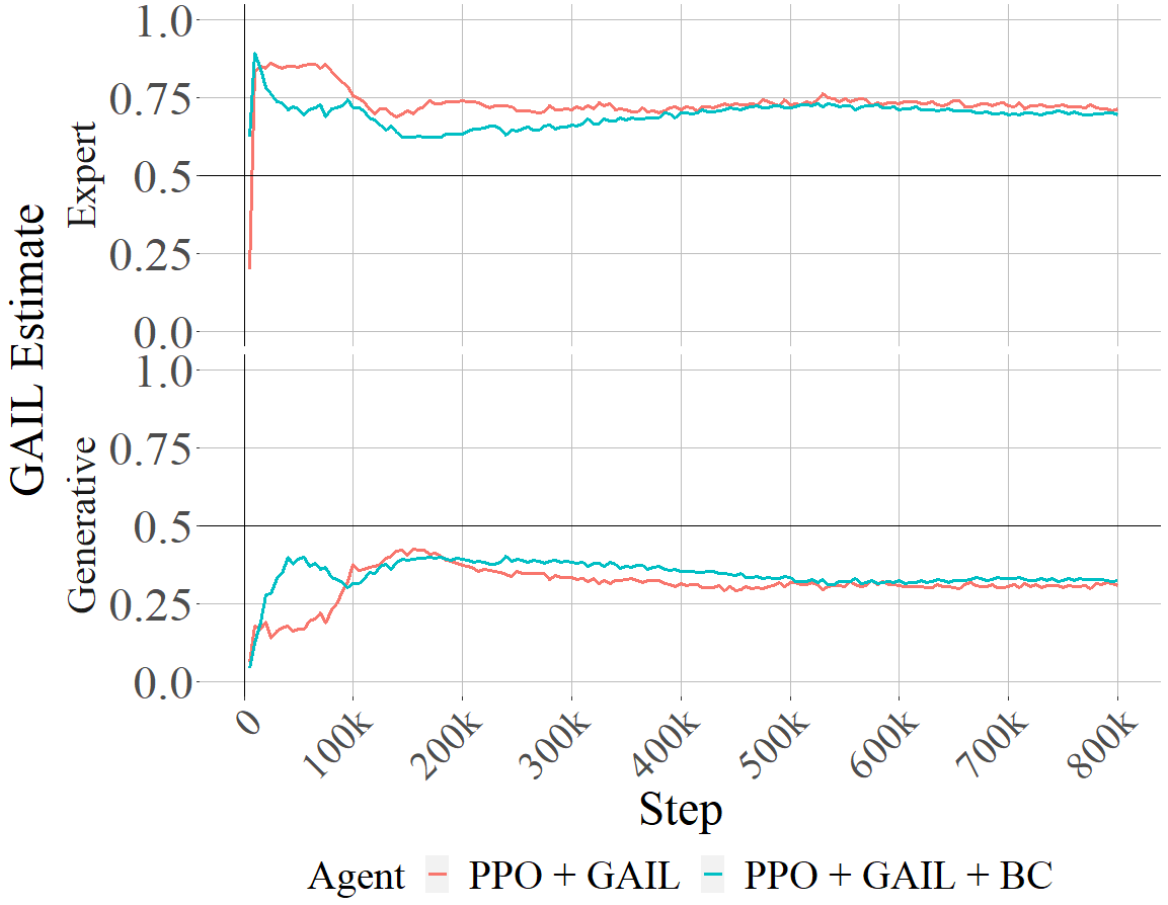

Figure 4.5: Expected cost function values for the generative NN and the expert discriminator for the agent when using PPO + GAIL and PPO + GAIL + BC

In terms of attack selection, all agents selected attacks in a similar fashion as the one used in the optimal policy. This, however, only applied for the beginning and the middle of the fight. PPO and PPO + BC were unable to learn complex actions, and avoided their use at the end of the fight. Learnings using GAIL attempted to learn complex actions, albeit being unsuccessful in achieving this.

4.2.2 Case Study N°2

For the second case, the demonstration file consisted of a sub optimal policy. For this case study, the agent's learning was successful with all learning algorithm combinations (Figure 4.7). Furthermore, the implementation of BC showed a clear improvement in the speed of learning (Figure 4.7).

Contrasting the results of Case Study N°1, in Case Study N°2 the use of GAIL does not improve the overall start of the learning of PPO. Furthermore, here the beginning of the training is hindered by two factors which are related to each other. The first one being the randomness (Figure 4.8) at the beginning of the test, where a higher focus is given to the use of hard actions. Unable to do the hard actions, results in the agent losing most fights until the 150k steps mark, which resembles the observations of case study N°1, where the inability of the generative NN to fool the discriminator negatively affects the training, Figure 4.9 shows the difference between the expected policy occupancy value of the expert and the generative NN is > 0.3 until the 150k steps mark, where this difference is reduced. This means that after the 150k steps mark, the generative NN is successfully fooling the expert. The decreased difference between expected policy occupancy relates to the agent avoiding using hard actions by only copying the actions from the demonstration file. This results in the agent winning more fights in a consistent manner.

Since the demonstration file lacks the hard sets of actions, GAIL does not enter a state of "internal fighting". In turn, the agent optimizes the attacks used to increase its remaining health at the end of a fight. This results in a slight improvement of the positive rewards given to the agent.

While GAIL alone fails to improve the optimization at the beginning of the training, the use of BC for training the agent (PPO + BC and PPO + GAIL + BC) clearly improved the initial learning. This is due to BC making the agent to avoid the use of hard actions all together. The use of BC is enhanced when combined with GAIL which improves the use of easy actions.

In this case study, the entropy value of a successful learning varies between 0.7 and 1.0 (Figure 4.8). These values are similar to the entropy values found in the successful learning with PPO and PPO + BC in case study 1, which varied around 1.0 and had median entropy < 1.2 (Figure 4.4; Table 4.5).

In terms of attack selection, all agents selected attacks in a similar fashion as the one used in the optimal policy. This, however, only applied for the beginning and the middle of the fight; whereas for the final stages of the fight, all agents could not learn the use of complex actions, and avoided their use. In contrast to the first case study, GAIL did not attempt to do complex actions, due to these being absent in the demonstration file.

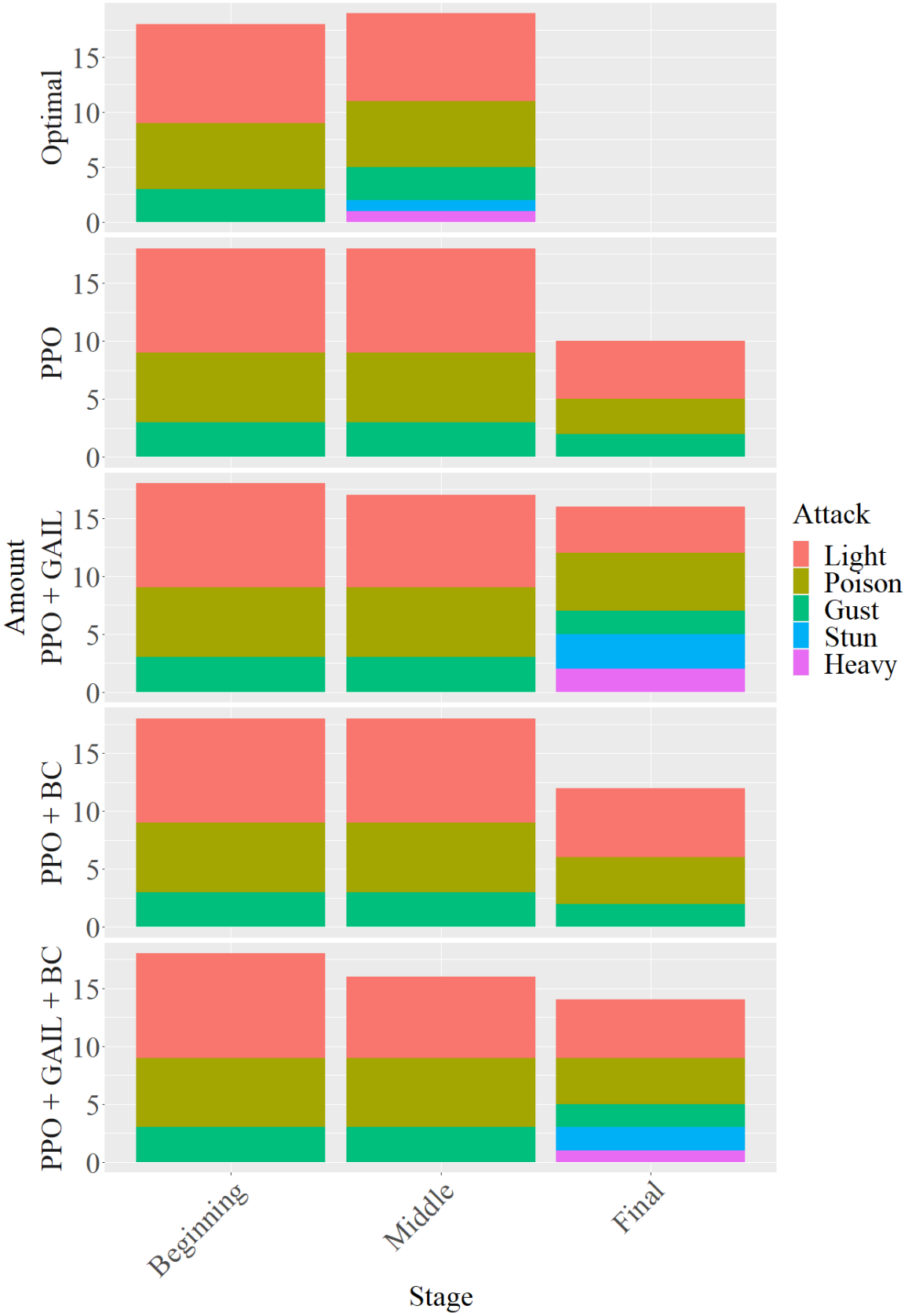

Figure 4.6: Average number of attacks used by all learning agents during the last 10 episodes. Episodes are divided into three fight stages: beginning, middle and final.

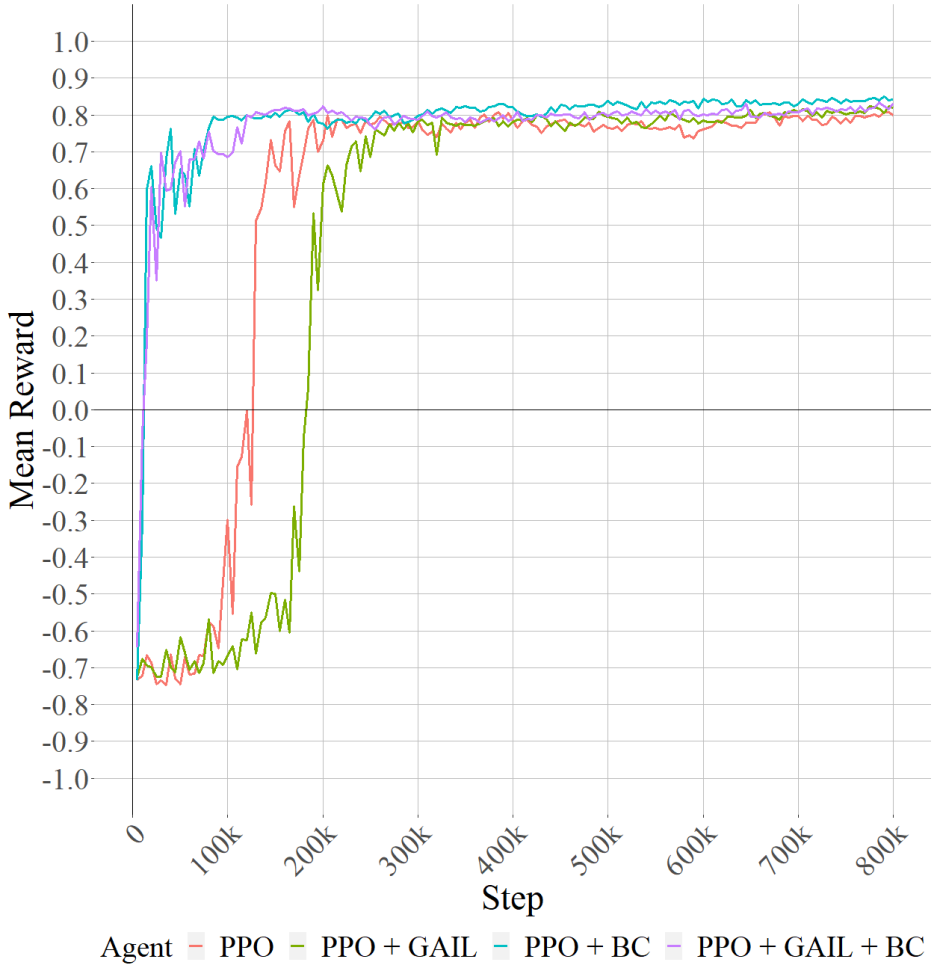

Figure 4.7: Mean rewards that the agent got using four different trainers combination

4.2.3 Case Study N°3

For the third case, the demonstration file consisted of a combination of optimal and sub optimal policies. For this case study, the agent's learning was successful with all learning algorithm combinations, and no clear differences between learnings was found (Figure 4.11).

PPO + GAIL performs slightly better than only PPO for the first 50k steps. The later improvement observed between this point and the 80k steps mark comes from the agent being able to learn how to use the *Stun* attack, while avoiding the use of the *Heavy* attack. Between the 80k and the 125k steps mark, the reward varies between -0.4 and -0.1 because the agent must still learn that the use of many (5-6) *Stun* attacks is counterproductive. Once the agent learns that the right amount of *Stun* attack to use is 1-2 times, its fighting performance starts to improve over time. Additionally, having the expert discriminator giving a negative reward to the generative NN, makes the generative NN take more time in learning the right amount of attacks.

The end result with GAIL is slightly better than the obtained when only training using PPO. This relates to the ability GAIL has to better use the easy actions and one of the hard actions. Having a mixed behavior in the demonstration only affects a small section at the start of the training. Furthermore, this behavior does not result in the internal conflict GAIL showed in the first case study.

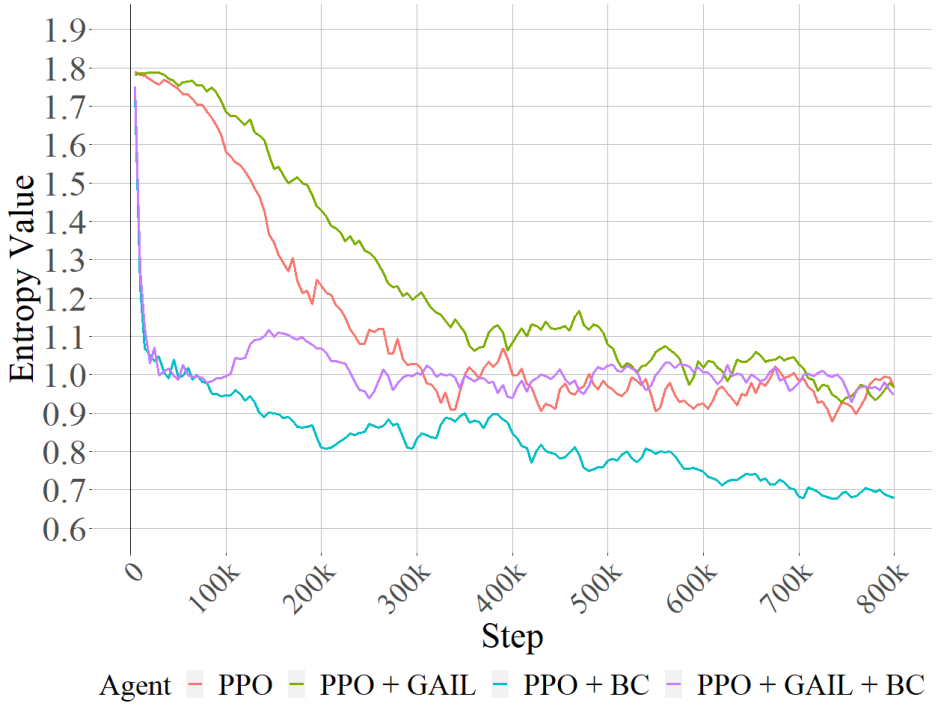

Figure 4.8: Entropy

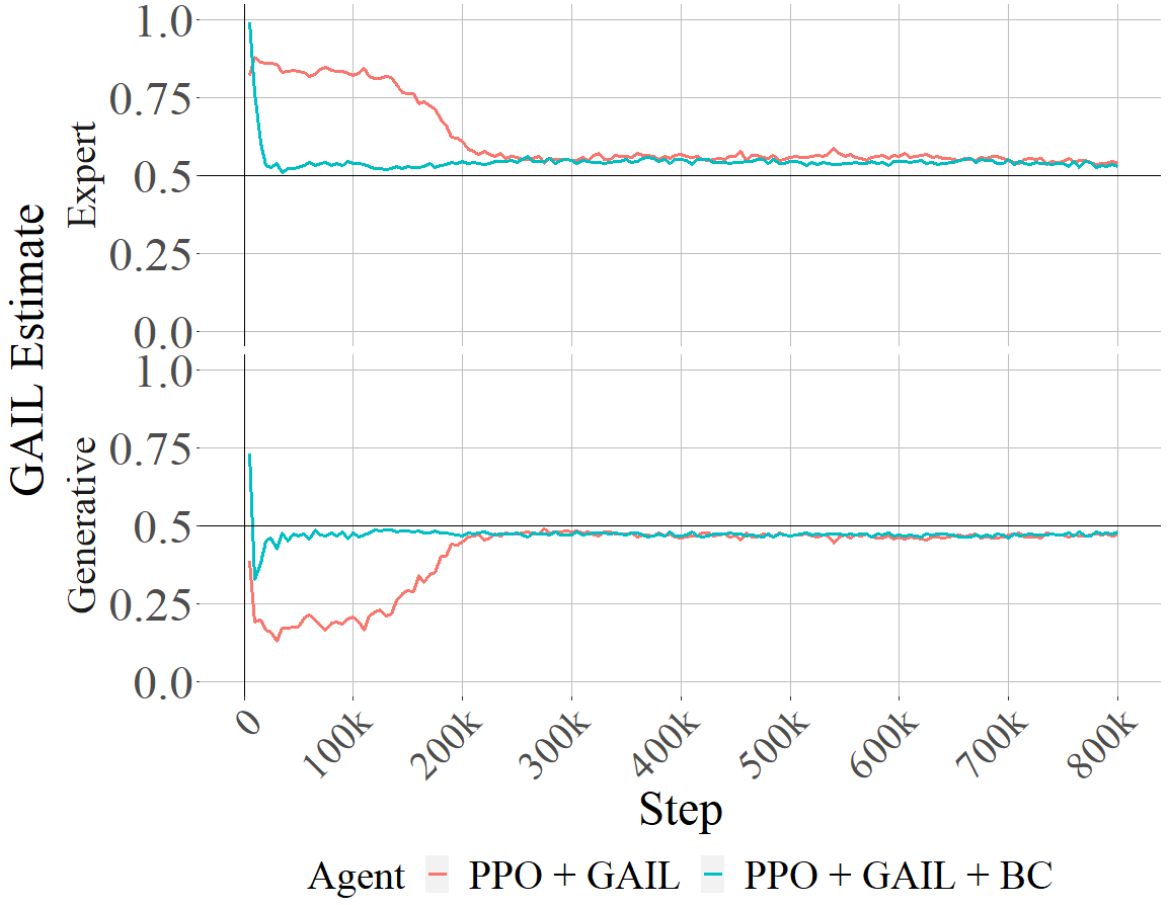

Figure 4.9: Expected cost function values for the generative NN and the expert discriminator for the agent when using PPO + GAIL and PPO + GAIL + BC

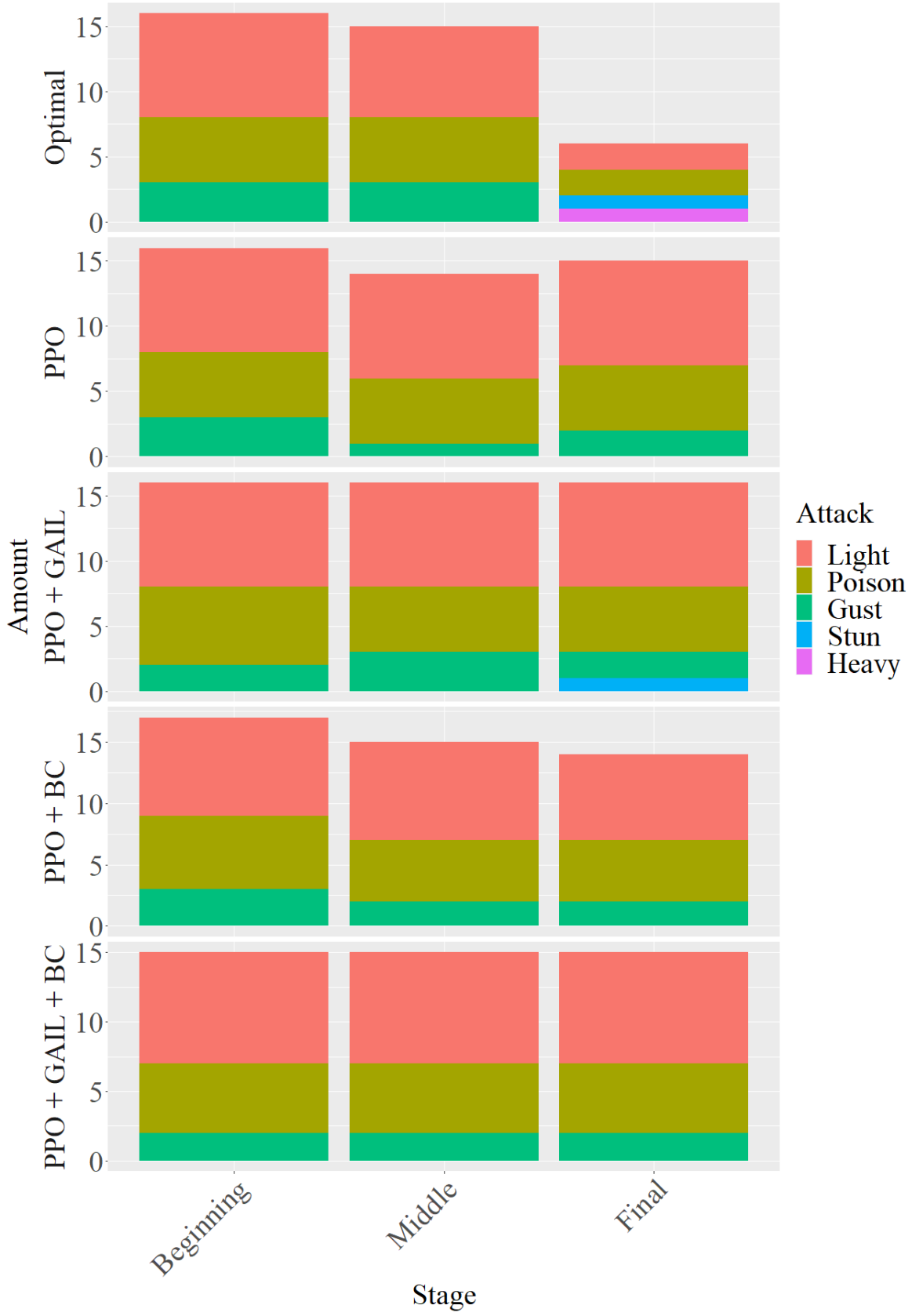

Figure 4.10: Average number of attacks used by all learning agents during the last 10 episodes. Episodes are divided into three fight stages: beginning, middle and final.

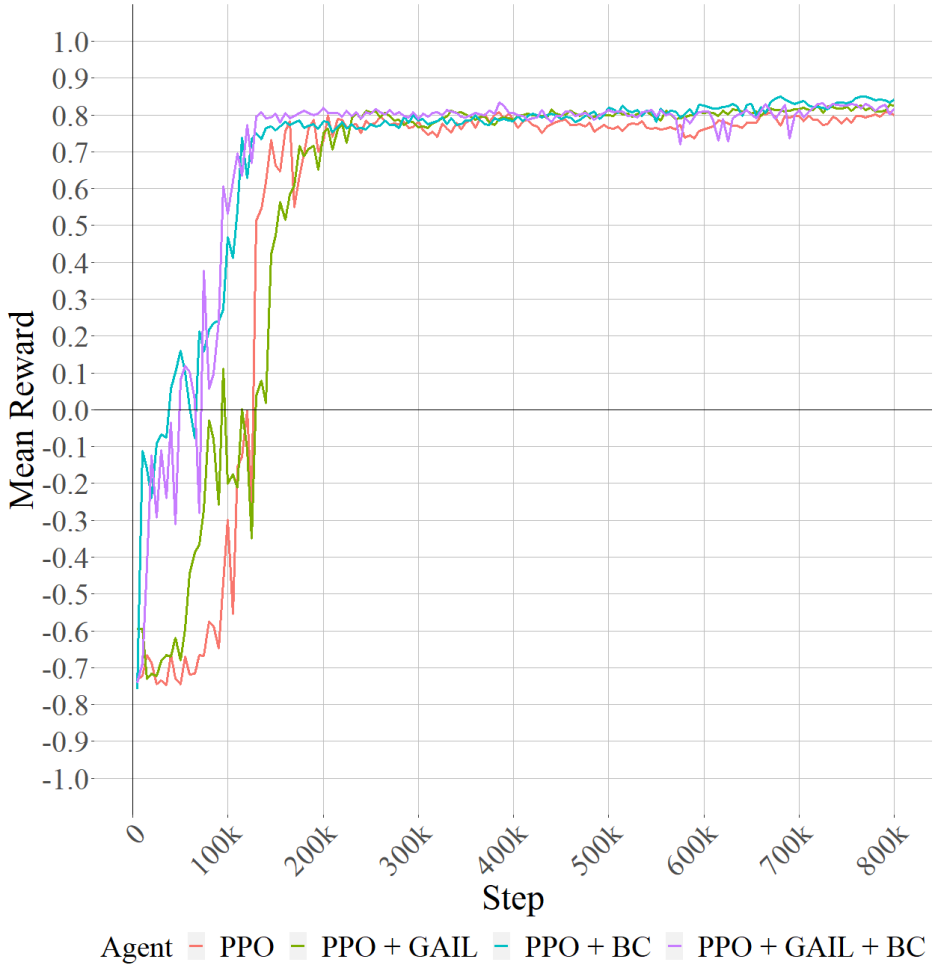

Figure 4.11: Mean rewards that the agent got using four different trainers combination

Both PPO + BC and PPO + GAIL + BC showed a similar behavior. They show a promising start on the first 10k – 20k steps, followed by an increase similar to that observed in the experiment with PPO + GAIL. This shows the effect of BC, which allows the agent to learn in a faster and more steady fashion the right amount of hard and easy actions it needs to do. Furthermore, the use of BC also allows the agent to perform the hard set of actions, which explains some of the bigger differences in terms of rewards and that the generative NN in GAIL does not present any problem in fooling the expert NN, as observed in Figure 4.12.

Similar to Case Study N°2, the version that starts with PPO + BC shows a slight loss of performance after BC stops working. However, the end results of the PPO + BC combinations are better than those observed for the PPO + GAIL + BC training. This can be observed in the *Fight Logs* and the *Attack Summary*. While PPO + GAIL + BC continues on trying to do only a few hard actions, PPO + BC avoids their use altogether (Figure 4.13). Furthermore, PPO + BC showed a lower delay for the use of actions. This can be observed in the *Fight Logs*, which showed that PPO + GAIL + BC took an average of 0.2 to 0.4 second frames between the indication of the attack being ready and it being used, whereas PPO + BC only took an average of 0 to 0.2 second frames (Figure 4.14).

For this case study, the entropy of a successful learning varied between 0.9 and 1.1 (Figure 4.15), further increasing the entropy range for a successful policy, in comparison

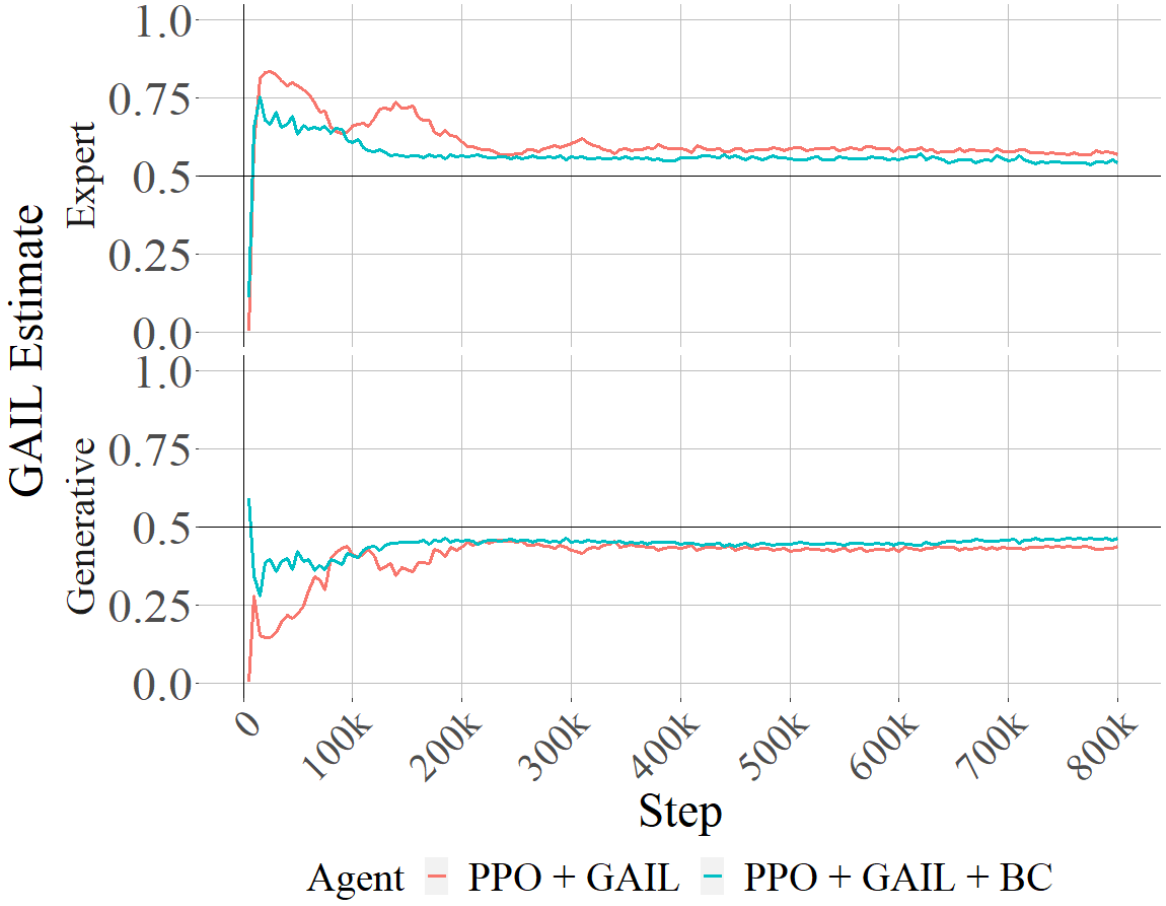

Figure 4.12: Expected cost function values for the generative NN and the expert discriminator for the agent when using PPO + GAIL and PPO + GAIL + BC

PPO + BC	PPO + GAIL + BC
23;0;0;16;0;0;7;Player	23;0;0;16;0;0;7;Player
23;0;0;16;0;0;7;Player	22;0;1;17;1;0;7;Player
23;0;0;17;0;0;7;Player	23;0;0;16;0;0;7;Player
23;0;0;16;0;0;7;Player	23;0;2;16;1;0;8;Player
23;0;0;16;0;0;7;Player	23;0;0;16;0;0;8;Player

Figure 4.13: A snippet of the amount of attacks performed by the agent when using PPO + BC and PPO + GAIL + BC

to Case Study N°1 and N°2. As observed for successful learning in other cases, median entropy of all trainings of this case study were < 1.2 (Table 4.5).

In terms of attack selection, all agents selected attacks in a similar fashion as the one used in the optimal policy. This, however, only applied for the beginning and the middle of the fight; whereas for the final stages of the fight, all agents could not learn the use of complex actions. PPO and PPO + BC avoided their use, whereas GAIL shows a mixed behaviour similar to the one described on the demonstration file, attempting to follow the action pattern in the same manner as recorded in the demonstration file, i.e. in episodes with complex actions GAIL attempted to perform

PPO + BC	PPO + GAIL + BC
0;3,75;-;-;-;100 0,02;3,75;bp;0;0,3;100 0,04;3,71;bl;3;0,24;100 0,24;4,143334;bl;3;0,44;100 0,26;4,103334;-;-;-;100 0,28;4,063334;-;-;-;100 0,3;4,023334;bp;0;0,58;100 0,42;3,783334;-;-;-;100 0,44;3,743334;bl;3;0,64;100 0,46;3,703334;-;-;-;100 0,56;3,503335;-;-;-;100 0,58;3,463335;bp;0;0,86;100 0,6;3,423335;-;-;-;100	0;3,75;-;-;-;100 0,02;3,75;-;-;-;100 0,04;3,71;bp;0;0,32;100 0,12;4,383334;-;-;-;100 0,14;4,343334;bl;3;0,34;100 0,16;4,303334;-;-;-;100 0,32;3,983334;-;-;-;100 0,34;3,943334;bp;0;0,62;100 0,36;3,903334;bl;3;0,56;100 0,38;3,863334;-;-;-;100 0,56;3,503335;-;-;-;100 0,58;3,463335;bl;3;0,78;100 0,6;3,423335;-;-;-;100 0,62;3,383335;-;-;-;100 0,64;3,343335;-;-;-;100 0,66;3,303335;bp;0;0,94;100

Figure 4.14: A snippet of the attack frequency of the agent when using PPO + BC and PPO + GAIL + BC

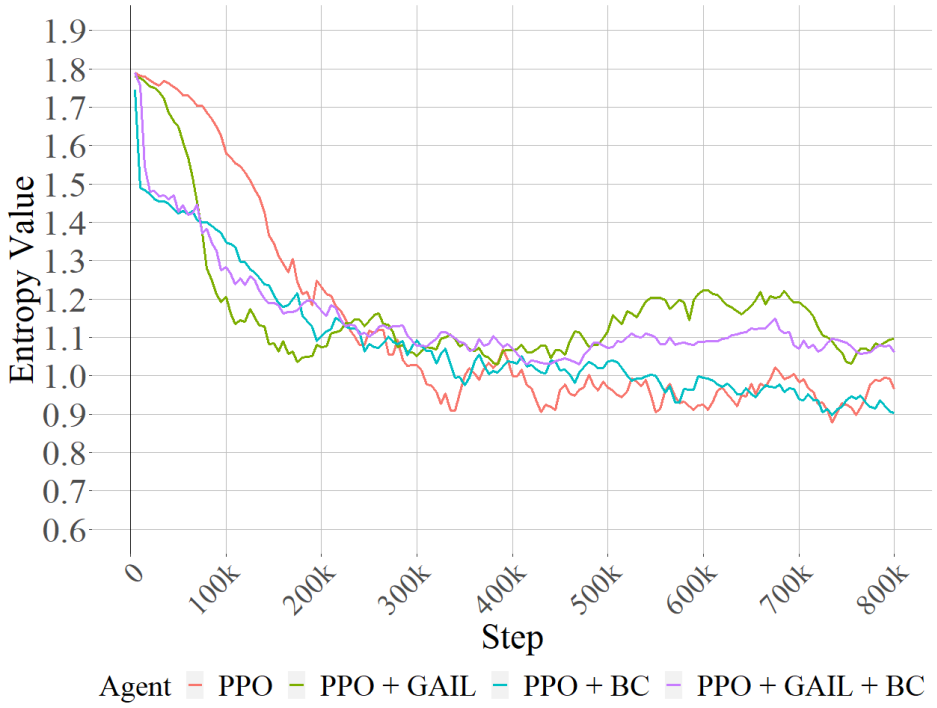

Figure 4.15: Entropy

them, whereas when complex actions are absent, GAIL does not attempted to execute them.

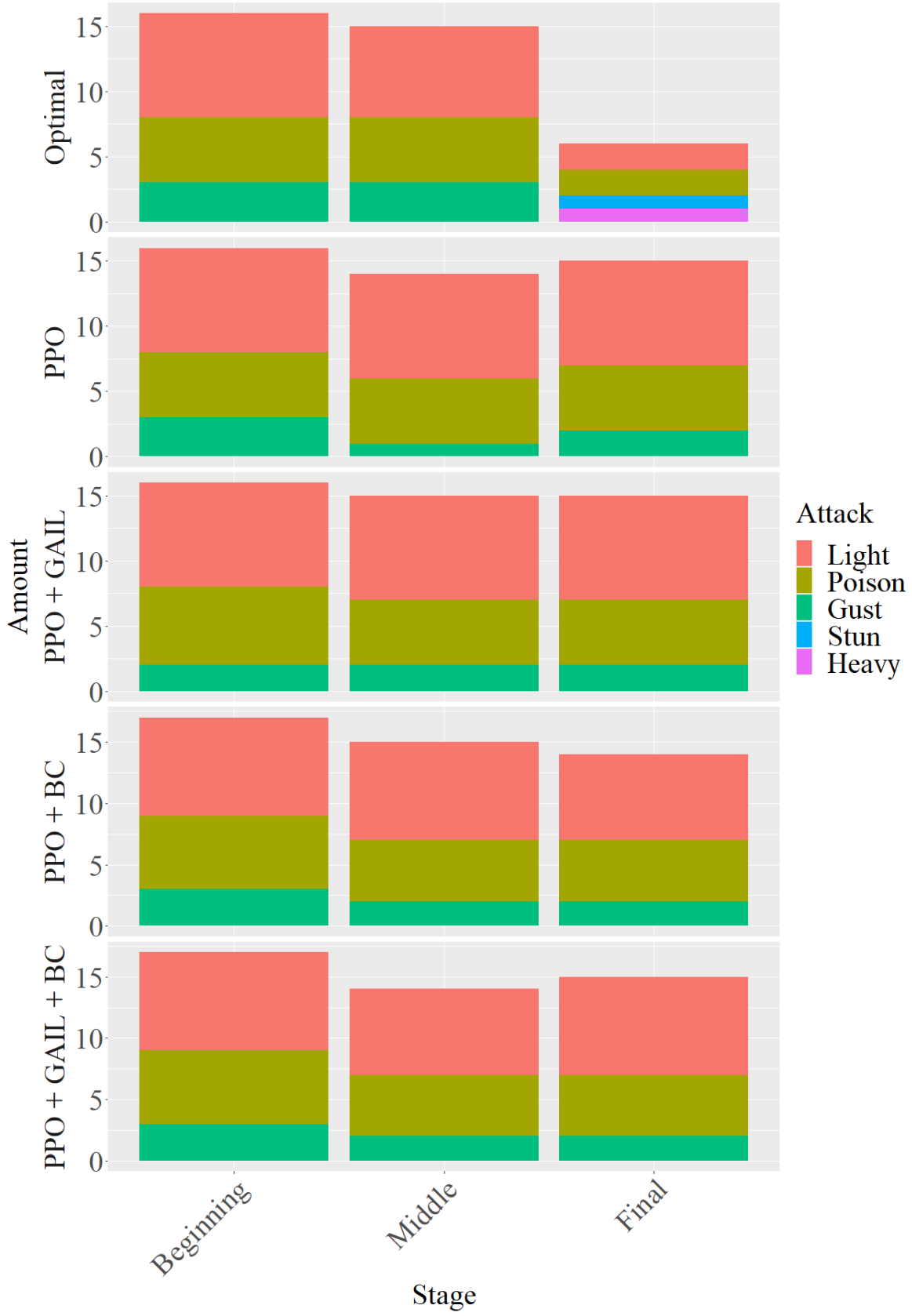

Figure 4.16: Average number of attacks used by all learning agents during the last 10 episodes. Episodes are divided into three fight stages: beginning, middle and final.

4.3 Runtime Performance

Due to the hardware used for this experiment the learning algorithm SAC could not be used. SAC requires a large amount of samples, as stated in the literature [38], which the hardware could not handle to perform, this, in turn, resulted in Unity to stop working and crashing. On the other hand, the different combinations of learning methods (PPO, PPO + GAIL, PPO + BC, PPO + GAIL + BC) performed without problems and reached the required 800k steps mark. Overall, reaching this mark, thus finishing the sessions, took an average of 2.00 hours ± 0.17 hours.

4.4 Summary

Overall, the learning agent or Boss was successful (i.e. median reward > 0.75) in learning a strategy to win against a simple agent with a fixed behavior, albeit the results were not successful for all case studies (Table 4.4). Unsuccessful learning was observed only for the first case study when the learning agent used GAIL. In general the use of BC improved the learning during its active period between 0 and 100k steps.

Agent	Study Case N°1 Reward (± SE)	Study Case N°2 Reward (± SE)	Study Case N°3 Reward (± SE)
PPO	0.772 ± 0.039	0.772 ± 0.039	0.772 ± 0.039
PPO + GAIL	0.436 ± 0.033	0.778 ± 0.046	0.796 ± 0.035
PPO + BC	0.776 ± 0.033	0.818 ± 0.012	0.790 ± 0.021
PPO + GAIL + BC	0.470 ± 0.025	0.799 ± 0.011	0.801 ± 0.024

Table 4.4: The median reward value obtained (± Standard Error) considering all rewards obtained in a training session for each case study.

Agent	Study Case N°1 Entropy (± SE)	Study Case N°2 Entropy (± SE)	Study Case N°3 Entropy (± SE)
PPO	0.992 ± 0.021	0.992 ± 0.021	0.992 ± 0.021
PPO + GAIL	1.427 ± 0.010	1.122 ± 0.021	1.128 ± 0.013
PPO + BC	0.994 ± 0.007	0.809 ± 0.009	1.028 ± 0.013
PPO + GAIL + BC	1.321 ± 0.006	0.998 ± 0.005	1.100 ± 0.010

Table 4.5: The median entropy value obtained (± Standard Error) considering all entropies obtained in a training session for each case study.

In the next chapter, I will discuss the results obtained in the evaluation of each case and point out different interesting aspects observed in the results.

Chapter 5

Discussion

The users of Unity and their ML-Agents Toolkit have greatly increased in the recent years. However, most of the effort in terms of learning, has been focused in physics-based policies, e.g. movement of joints and extremities, body rotation. This project focused on action choosing instead of physics-based policies, thus, inconsistencies with previous results on learning algorithm performance have been found. Additionally, since most research with Unity and the ML-Agents Toolkit has been focused in learning of physics-based policies, only scarce support material was found to support the preparation of experiments and interpretation of results obtained within the frame of this project. The following discussion aims to tackle these topics.

5.1 Policy Learning: Demonstration Files, Learning Agent and Entropy

5.1.1 The Effects of Demonstration Files and a Comparison Between Learning Agents

The training sessions showed the importance of the contents of the demonstration files and the use of ILs. While BC showed that it can improve the performance of PPO regardless of what demonstration files were provided, GAIL performance was negatively affected when the file only had episodes with hard actions and sets of hard actions (optimal policy). When provided with this policy, the agent could not replicate the complex actions. This was especially observed when the actions had to be performed in a specific point of the fight (i.e. to end the fight). Having actions which the agent is unable to perform affected the expert discriminator of GAIL, facilitating the task of distinguishing if state/action pairs were drawn from the environment or from the demonstration file. This caused the expert NN to heavily penalize the generative NN with a negative reward, which then caused the generative NN to always try to imitate hard actions, regardless of when they were used, and never be able to successfully perform the actions. GAIL has been described as a high performance algorithm to learn complex actions [39]. However, this has been proven for the learning of physics-based control tasks, e.g. movement of joints and extremities. Thus, our results would suggest that while GAIL is efficient to learn physics-based control tasks, it suffers difficulties to learn how to optimize the selection of actions. This could potentially be improved by e.g. the use of BC as a complement to GAIL, the combined use of several demon-

stration files, or the increase of state/action pairs. Proving these hypotheses, however, will require further experimenting.

The best example of the importance of the contents of the demonstration file could be observed in the learning of the Boss in the third case study. Here, a demonstration file with a combination of two policies (sub-optimal and optimal) consisting of a low amount and frequency of the *Heavy* attack was used (see Figure 5.1). Furthermore, the two policies included in the mixed policy demonstration file were alternating every three full cycles; i.e. the file had three copies of the sub-optimal policy, followed by three copies of the optimal policy. The use of this demonstration file helped the generative NN to fool the expert, especially when the behaviour of the agent resembles a suboptimal behavior, as seen in Figure 5.1. When this occurs, the discriminator can be easily fooled, and the optimal behaviour ends up being completely disregarded by the generative NN. This poses an interesting result on how the sampling for calculating the cost function is done by the algorithm, and how it optimizes its function, even if this means disregarding the learning of complex actions.

When the Boss was learning with BC as the only compliment for PPO, the only variation between trainings was observed in terms of the speed at which the algorithm reached a successful state. The growth speed correlates directly to the amount of hard actions a demonstration file has, and the frequency they appear over, as observed in Figure 5.1. To show this, first it is needed to understand the hard action set up of each policy (Table 5.1):

Policy	Hard actions (Heavy/Stun)	% of episodes with hard actions (Heavy/Stun)
Optimal	80 (40/40)	100 (100/100)
Sub Optimal	24 (0/24)	30 (0/75)
Mix	50 (18/32)	62.5 (45/80)

Table 5.1: Amount and frequency of hard actions in each demonstration file used.

The learning curves were clearly affected by the ILs which complemented PPO, with contrasting results between the learning using PPO + BC (Figure 5.2) with that of the learning using PPO + GAIL (Figure 5.3). Learning using PPO + BC was, depending on the policy, as followed: a) The agent that had the maximum amount of hard actions showed a slow learning pace, reaching a constant positive reward at the 110k step mark, just after BC stops working; b) The learning of the agent was fastest with the lowest amount of hard actions, reaching a constant positive reward at already at the 15k steps mark and a successful state at the 80k steps mark (i.e., before BC stopped working), and; c) The agent with the medium amount of hard actions showed an intermediate learning speed, reaching a constant positive reward at the 70k steps mark. Overall, the use of BC drastically improved the learning speed of the agent, and appeared especially effective for the learning of easy actions. In contrast, the cases where only GAIL complemented PPO (PPO + GAIL) required, at least, 110k steps. When using the optimal policy, the learning agent reached a constant positive reward at the 110k steps mark, albeit a successful state was not fully reached. The use of the sub-optimal policy showed learning to occur at a slower pace, reaching a constant positive reward after the 185k steps, and the successful state at the 270k steps mark. Finally, the learning agent that used the mixed policy had an intermediate learning pace, reaching a constant positive reward at the 130k steps mark, and a successful state at the 230k steps mark.

Optimal Policy	Sub Optimal Policy	Mix Policy
17;1;0;13;1;0;6;Player	24;0;0;15;0;1;7;Player	24;0;0;15;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;6;Player	23;0;0;16;0;1;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	24;0;0;15;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	24;0;0;16;0;0;7;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;6;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	24;0;0;16;0;0;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	24;0;0;16;0;1;7;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;6;Player	23;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	22;0;0;17;0;1;7;Player	23;0;0;16;0;1;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;8;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	24;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	23;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;0;7;Player	23;0;0;16;0;1;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	24;0;0;15;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;17;0;1;7;Player	17;1;0;13;1;0;6;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	24;0;0;16;0;1;7;Player
17;1;0;13;1;0;6;Player	24;0;0;16;0;1;7;Player	24;0;0;16;0;0;7;Player
17;1;0;13;1;0;6;Player	23;0;0;16;0;1;7;Player	24;0;0;16;0;1;7;Player

Figure 5.1: A snippet of the *Attack Summary* of each demonstration file used on the training.

Overall, GAIL was slower than BC, but was apparently more efficient to learn the hard actions using the mixed policy. The contrasting learning speeds might be the result of BC having all required observations for learning from the start, whereas GAIL needs to do most of the observations needed itself [39]. Another observation which can be drawn from these results is that GAIL is better than BC for learning complex actions and combinations of them, fact mentioned in the corresponding literature [39]. However, there are records of GAIL being unable to learn and solve complex behaviours and puzzles [40]. This problem was also reflected in the learning curve obtained when using the optimal policy, which shows the instability GAIL might suffer when learning complex behaviours. The results of this project show this instability to translate into

internal conflicts between GAIL's NNs, which hampers the learning procedure, a fact shown by the training being unsuccessful (e.g. Figure 4.3).

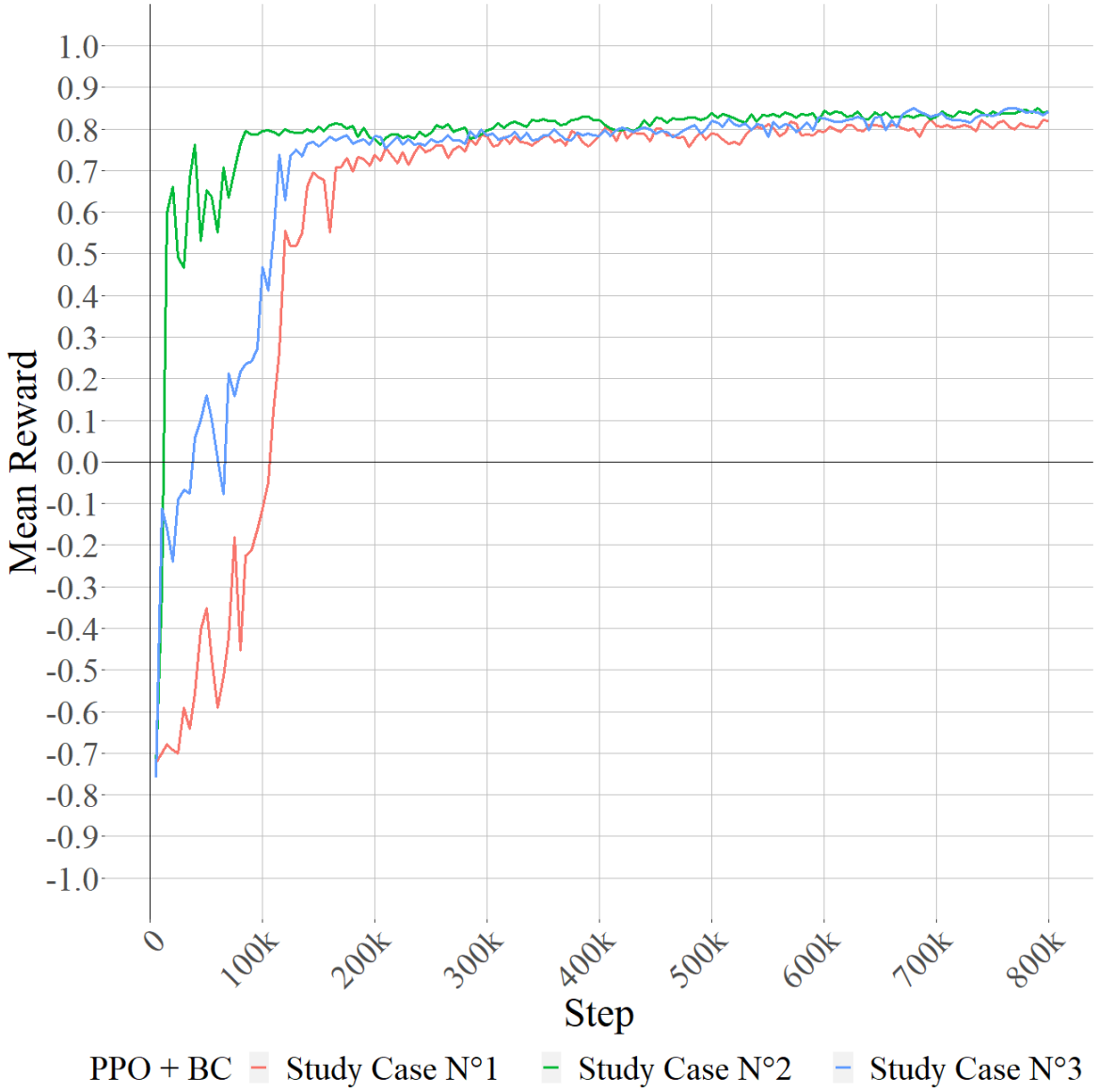

Figure 5.2: Mean rewards that the agent got in the three case studies using PPO + BC.

One aspect that appears to regulate the learning speed of GAIL is the amount of episodes which include complex actions in their demonstration files. The amount of complex actions modifies the attempts in each episode and the frequency during the training session of GAIL to perform these hard actions. This can be seen when analyzing the attacks used on the last 10 episodes (see Figure 5.4). In the first case, with a demonstration file where hard actions are always present (Table 5.1), GAIL attempted to perform both hard actions at the final stage of the fight (as given in the demonstration file) in every episode. Furthermore, GAIL also attempted to use hard actions at other stages of the episodes, although it didn't learn how to perform them. Performing the actions in this fashion, translates in higher chances of receiving lower positive rewards, thus lowering the chances of successfully learning a policy. In the second case study, where the demonstration file consists of a low amount of hard

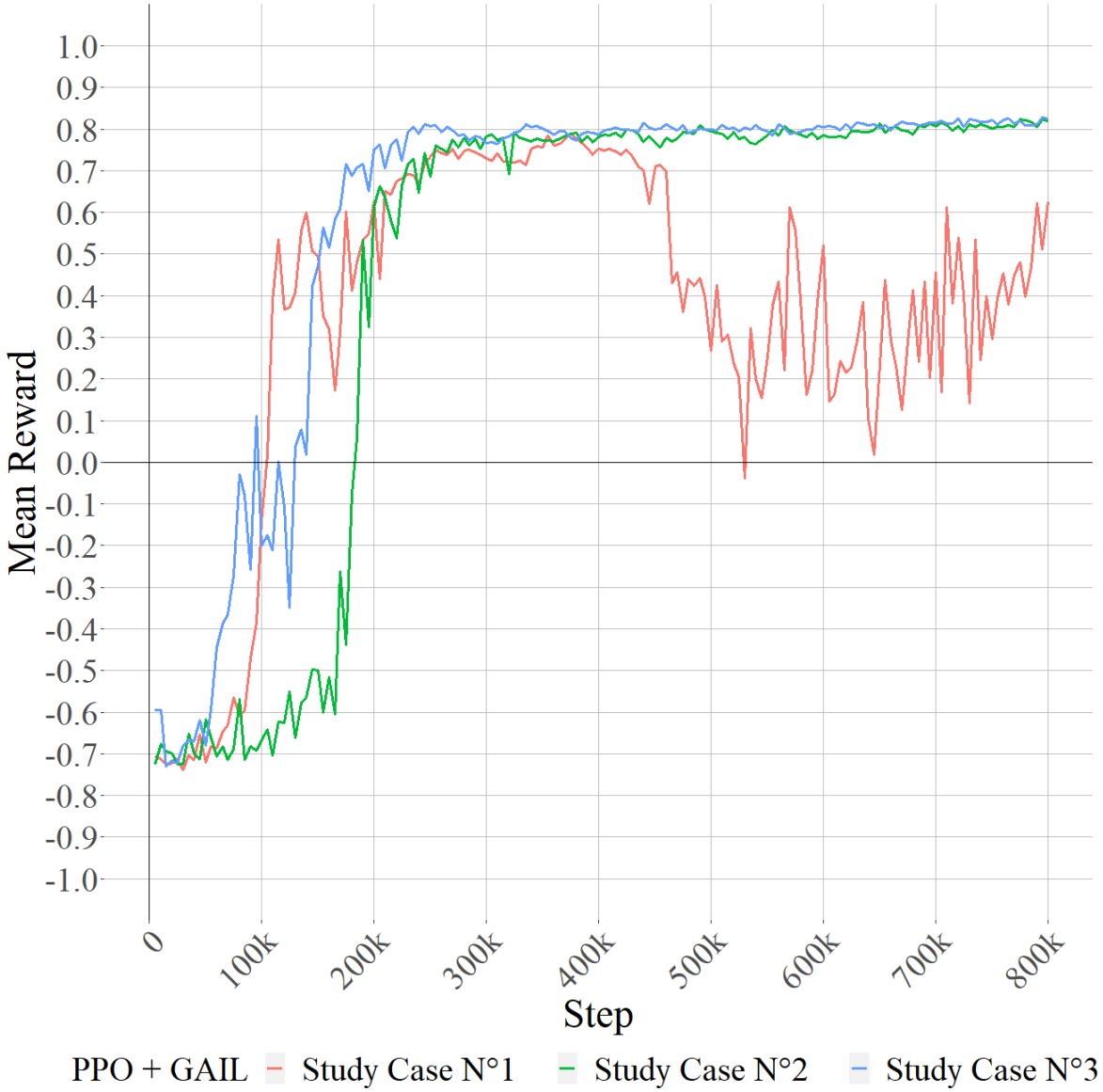

Figure 5.3: Mean rewards that the agent got in the three case studies using PPO + GAIL.

actions (Table 5.1), GAIL avoided the use of hard actions. It could be thus inferred, that having a low amount of hard actions could result in GAIL being unable to learn an optimal policy due to this behaviour of avoiding their use. In the third case study, GAIL performed the hard actions in a similar fashion as the demonstration file, every three episodes performed at least once the hard actions but never multiple times, as shown in the first case. This resulted in learning a sub optimal policy at the cost of not learning an optimal policy. It would be intriguing to understand if, in the hypothetical case that GAIL is indeed able to learn the optimal policy: would GAIL still follow a mixed pattern, or would it choose only to follow one policy? While GAIL and BC show clear differences in terms of learning speeds, it has been proposed that BC represents a compliment for GAIL [39]. This can be observed in most trainings (Figure B.1 in Appendix B), where the use of BC improves the learning of GAIL, albeit without being better than BC alone. The only exception is when considering the optimal policy, where the use of GAIL + BC results in a faster learning than with

either of them on its own. However, while BC clearly improves the performance of GAIL, it appears it does not solve the problems GAIL has with policies plagued with complex actions, as it is observed in the learning curve with the optimal policy. An interesting outlook from this, is to study the potential effect of a longer use of BC in combination with GAIL, and if this would help GAIL to learn the optimal policy considered in the project, and avoid the internal conflict between the expert NN and generative NN.

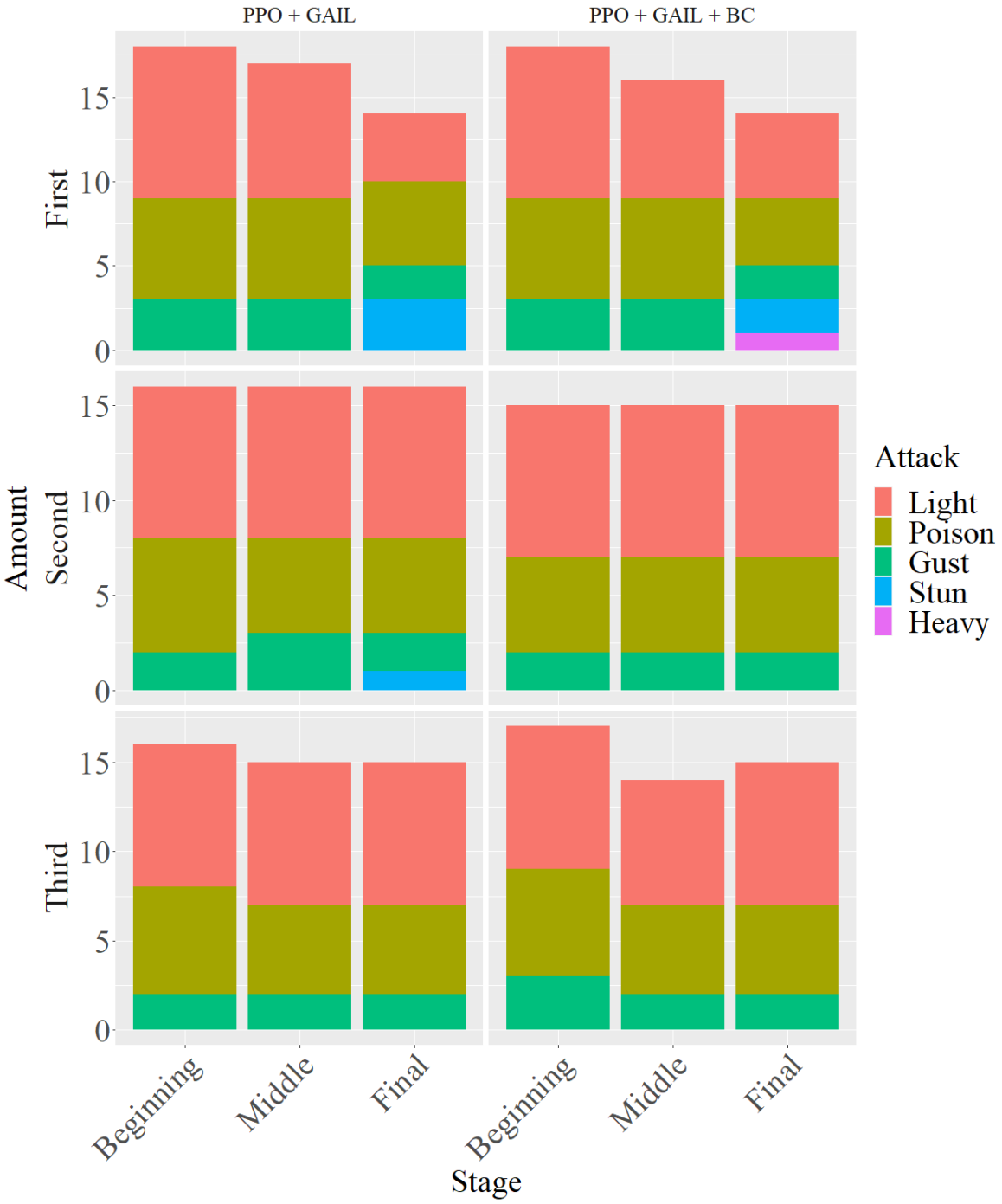

Figure 5.4: Comparison of the attacks performed at each fight stage during learning where GAIL was present.

5.1.2 Importance of the Entropy as a Learning Success Indicator

For simplification purposes in the comparison in terms of the study cases, three training examples were drawn from all cases, these represent: the best observed learning (PPO + BC with a sub optimal policy in the demonstration file), the worst observed learning (PPO + GAIL with an optimal policy in the demonstration file), and the base case which represents an intermediate learning (PPO).

From these cases, a clear relationship between the reward obtained and the entropy can be observed, which can be used to draw two general observations:

1. The success of the learning experiment is tightly related to the entropy. All learnings were successful whenever the entropy: a) decreased and remained below 1.2, and; b) the overall median entropy value < 1.2. In this context, the best and intermediate learnings resulted in median entropy values < 1.2, and median reward > 0.75; whereas the unsuccessful learning had a median entropy of 1.427 $\pm$ 0.010 (Table 4.5) and median reward of 0.436 $\pm$ 0.033 (Table 4.4)

2. The general trend and shape of the curves for both entropy and rewards appeared similar. This implies that entropy can be used as a predictor of the performance (approximated obtained reward), and learning stability.

A way of testing this relation is to correlate the reward and entropies obtained in the trainings. For the three cases, the correlations resulted in strong and significant anticorrelations (all Pearson correlation statistics $<$ -0.75; all p-values $<$ 0.001; see Table C.1 in Appendix C), thus supporting the above mentioned observations. This would imply that the random selection of actions plays an important role in the training of an agent. In this context, the correlations considering the examples with PPO + GAIL and PPO + BC both gave strong and significant anticorrelations (all Pearson correlation statistics $<$ -0.90; all p-values $<$ 0.001). Thus, the use of ILs appears beneficial for training as long as no internal conflict occurs (as observed using GAIL in Case Study N°1), since they tend to reduce the entropy of a training, i.e. reduce the randomness involved in selecting actions.

5.2 Usability Issues of Unity's ML-Agents Toolkit

Unity is regarded as one of the most popular game engines, mainly due to the vast library of assets, documentation, and tutorials (see Section 2). The ML-Agents Toolkit is remarkably easy to use and to set-up in order to conduct learning of policies. However, the ML-Agents Toolkit suffers from poor documentation, and lack of up-to-date tutorials (most individual developers doing YouTube tutorials use old versions of the toolkit). An additional problem is found when reading the overview material of the ML-Agents Toolkit [26], where it is mentioned that when the approach for using ML and RL is not academic, the learning algorithms (e.g. PPO) can be treated as black boxes. Based on this, any user should be able to set and train an agent successfully by only changing the trainer's related parameters (i.e. the hyper parameters). However, this claim is only partially true. While each hyper parameter is partially described in a way that any user (even non-experts) can understand it, the descriptions also include specific RL language, which can only be understood and interpreted by experts.

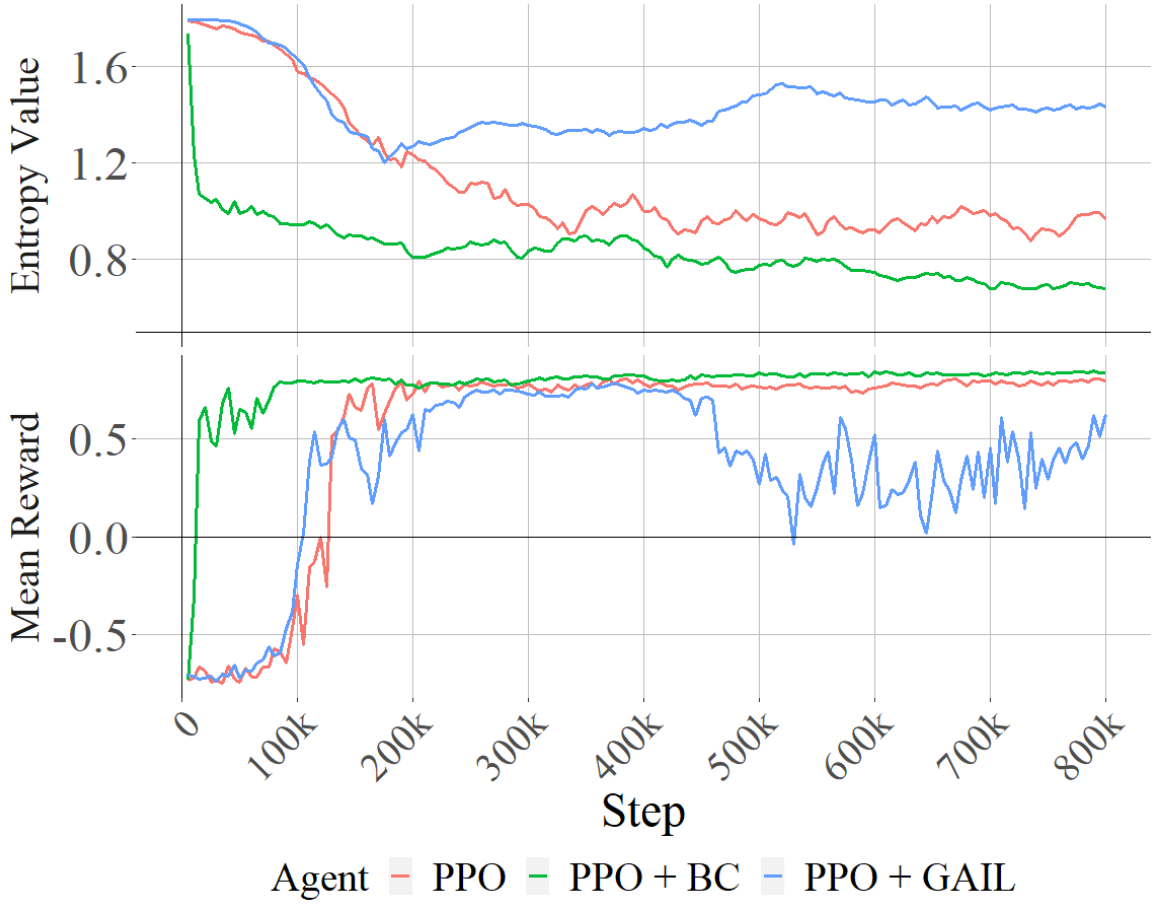

Figure 5.5: Comparison between the mean rewards and the entropy values gotten by the learning agent using PPO, PPO + BC and PPO + GAIL + BC.

This can result in a successful deployment and training of an agent, since the user will be able to set an agent in the environment and start its training by only setting the learning parameters. However, the information found in the documentation [41][42] is insufficient if the user needs to correctly understand if the training is being successful or not, and if the failure of the training is due to the agent hyper parameters, rewards, or a combination of them. The combination of colloquial and technical language, further results in a higher difficulty for a user (without any knowledge of RLs) to successfully deploy and train an agent. This problem is significantly higher when considering GAIL and the results of its use. For GAIL, the documentation [42] only describes the results with high level technical language, which requires users to first read research publications [39] in order to understand the documentation provided by Unity.

Another aspect that complicates the usage of IL algorithms for training comes from the little to no documentation provided regarding the contents of the demonstration file. Considering the importance of the demonstration file, as shown in the discussion of this project, a clear and easy to understand documentation is highly needed to interpret the learning process. The demonstration file is only mentioned in Unity's *Imitation Learning* introduction [26], and *Recording Demonstration* section for the agent composition documentation [43]. Here, the only mentioned information is in regards to the contents of the files (observations, actions, and rewards), and that a user is required to record "minutes to hours" of training, without providing further details

or explanations. This kind of information is relatively vague, especially compared to other literature on imitation algorithms, where a specific and concrete parameter such as the amount of state/action pairs is clearly given [39].

Other difficulties encountered while conducting the case studies were related to the environment updates, which can potentially result in difficulties to interpret the results of the learning process. The learning environments constructed using the ML-Agents Toolkit work in a frame-by-frame fashion by calling the function *FixedUpdate*. In Unity, all classes can use the function *FixedUpdate* to update at each frame their logic and values, with the class *Academy* from the ML-Agents Toolkit working before all classes. This results in every logic update from the learning environment (e.g. calculating the distance between the agents) being attached to a function of the class *Agent* (e.g. *OnActionReceived*) and not to the class supplementary functions of the agent (e.g. a void function to calculate the *Time until next attack*) nor supplementary classes of the environment (e.g. a *Game Manager* class that calculates the distance between agents). The way that the learning environment updates is not mentioned in the documentation provided, and failing to understand how it works can cause learning problems for the agent that are easily misattributed to poor coding, the actions system, the observations taken, and/or the reward structure. This problem is greatly aggravated if the learning environment updates its logic using Unity's *Update* function since the learning environment is sped up by default (e.g. in the default case 19 *FixedUpdate* are done while only 1 *Update* is performed). The fact that the learning environment is sped up and its velocity can be controlled is also not mentioned by Unity's documentation.

Chapter 6

Conclusion & Future Work

The project was successful in presenting an AI agent that learns to choose from a set of actions in order to win a fight against a non-learning agent with a fixed behavior. Furthermore, the aim of comparing different learning algorithms was achieved. This comparison showed the difficulty that some algorithms have to learn specific actions (e.g. GAIL learning hard actions) and, in some cases showing results which contradict the established literature. In turn, this showed the importance of new environments and various combinations of actions for testing and creating new ML algorithms.

In general, Unity's ML-Agents Toolkit showed promising results. However, the poor documentation and lack of tutorials, two of the main aspects that makes their game engine popular with independent game developers, were lacking. While they use established terminology, key methodological aspects (e.g. detailed explanations regarding behavioral cloning) are lacking. This fact made the experimenting and interpretation part of this project more difficult. Thus, an improvement in the supporting material for Unity's ML-Agents Toolkit is in dire need.

While the learning was complex because of the hard actions that had to be performed, there exists the possibility of adding complexity to the environment by: a) Having the Boss fight against multiple instances of the Player; b) Giving the Player agent the ability of blocking and/or dodging the attacks of the Boss; c) Adding different movement abilities to the Player, and d) Including a Human Player to train the Boss against different Player play styles. While these would represent an appealing way to further continue this work on learning algorithms, the complexity and time requirements for implementing them in Unity would require more time than that allocated for this project.

One last conclusion regards the need to further improve the comparative work between learnings. This includes the improvement of hardware to allow the inclusion of SAC in this type of experiments, as well as the inclusion of other ways of comparing the performance of the learning. This could be improved by the use of metrics (e.g. mean learning speed, time needed to reach a successful state, rate of learning) and multi runs of the same learning. These metrics and repetitions could be considered in comparative statistical analysis, to observe if learnings are significantly different or not, instead of simply comparing learning curves. While these will require better hardware and longer computational time, they would enable us to further test hypotheses, improve the existing learning exercises and the confection of new learning exercises.

Bibliography

[1] S. Risi, J. Lehman, D. B. D'Ambrosio, R. Hall, and K. O. Stanley, "Combining search-based procedural content generation and social gaming in the petalz video game.", in *In 8th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.*, 2012.

[2] E. J. Hastings, R. K. Guha, and K. O. Stanley, "Automatic content generation in the galactic arms race video game", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 1, no. 4, pp. 245–263, 2009.

[3] M. P. Eladhari, A. Sullivan, G. Smith, and J. McCoy, "Ai-based game design : Enabling new playable experiences", 2011.

[4] M. Cook, M. Eladhari, A. M. Smith, G. Smith, T. Thompson, J. Togelius, and A. Zook, "Ai-based games: Contrabot and what did you do?", in *In Proceedings of FDG*, 2015.

[5] A. Zook and M. O. Riedl, "Generating and adapting game mechanics", in *In Proceedings of FDG Workshop on Procedural Content Generation.*, 2014.

[6] M. Mateas and A. Stern, "Architecture, authorial idioms and early observations of the interactive drama facade", Jan. 2002.

[7] R. Evans and E. Short, "Versu—a simulationist storytelling system", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 6, no. 2, pp. 113–130, 2014.

[8] J. Levine, C. B. Congdon, M. Ebner, G. Kendall, and et al., "General video game playing.", in *In Dagstuhl Follow-Ups.*, 2013.

[9] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, "Vizdoom: A doom-based ai research platform for visual reinforcement learning", in *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, 2016, pp. 1–8.

[10] P. Rohlfshagen, J. Liu, D. Perez-Liebana, and S. M. Lucas, "Pac-man conquers academia: Two decades of research using a classic arcade game", *IEEE Transactions on Games*, vol. 10, no. 3, pp. 233–256, 2018.

[11] M. Wittkamp, L. Barone, and P. Hingston, "Using neat for continuous adaptation and teamwork formation in pacman", in *2008 IEEE Symposium On Computational Intelligence and Games*, 2008, pp. 234–242.

[12] K. Oh and S. Cho, "A hybrid method of dijkstra algorithm and evolutionary neural network for optimal ms. pac-man agent", in *2010 Second World Congress on Nature and Biologically Inspired Computing (NaBIC)*, 2010, pp. 239–243.

[13] S. Samothrakis, D. Robles, and S. Lucas, "Fast approximate max-n monte carlo tree search for ms pac-man", *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 3, no. 2, pp. 142–154, 2011.

[14] H. Handa, "Detection of critical situations by cmac+q-learning for pacman agents", in *2009 International Conference on Networking, Sensing and Control*, 2009, pp. 124–129.

[15] T. Pohlen, B. Piot, T. Hester, M. Gheshlaghi Azar, D. Horgan, and et al., *Observe and look further: Achieving consistent performance on atari*, 2018. eprint: `arXiv:1805.11593`.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, and et al., *Human-level control through deep reinforcement learning*. Nature, 2015, vol. 518, pp. 529–533.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, and et al., "Asynchronous methods for deep reinforcement learning", ser. Proceedings of Machine Learning Research, vol. 48, PMLR, Jun. 2016, pp. 1928–1937.

[18] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents", *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, Jun. 2013.

[19] N. Bhonker, S. Rozenberg, and I. Hubara, *Playing snes in the retro learning environment*, 2016. eprint: `arXiv:1611.02205`.

[20] M. C. Machado, M. G. Bellemare, E. Talvitie, J. Veness, M. J. Hausknecht, and M. H. Bowling, "Revisiting the arcade learning environment: Evaluation protocols and open problems for general agents", *ArXiv*, vol. abs/1709.06009, 2018.

[21] A. Juliani, V. Berges, E. Teng, A. Cohen, J. Harper, and et al., *Unity: A general platform for intelligent agents*, 2018. eprint: `arXiv:1809.02627`.

[22] OpenAI, *Openai five*, `https://blog.openai.com/openai-five/`, 2018.

[23] J. Raiman, S. Zhang, and F. Wolski, *Long-term planning and situational awareness in openai five*, 2019. eprint: `arXiv:1912.06721`.

[24] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, *Gotta learn fast: A new benchmark for generalization in rl*, 2018. eprint: `arXiv:1804.03720`.

[25] K. Kurach, A. Raichuk, P. Stańczyk, M. Zając, and et al., *Google research football: A novel reinforcement learning environment*, 2019. eprint: `arXiv:1907.11180`.

[26] *Ml-agents toolkit overview*. [Online]. Available: `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md`.

[27] A. Juliani, A. Khalifa, V. Berges, J. Harper, E. Teng, and et al., *Obstacle tower: A generalization challenge in vision, control, and planning*, 2019. eprint: `arXiv:1902.01378`.

[28] *Millions of natural-feeling, procedurally generated monsters*. [Online]. Available: `https://unity.com/products/machine-learning-agents`.

[29] "Unite 2007 - keynote", in *Unite 2007 Developer Conference*, 2007. [Online]. Available: `https://unite.unity.com/archive/2007`.

[30] R. S. Sutton and G. Barto, *Reinforcement Learning: An Introduction*. 2018, pp. 321–326.

[31] ——, *Reinforcement Learning: An Introduction*. 2018, pp. 331–332.

[32] c. J. Watkins and P. Dayan, "Technical note: Q-learning", in *Machine Learning*, vol. 8, Springer, 1992, pp. 279–292.

[33] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 mario ai competition", in *IEEE Congress on Evolutionary Computation*, 2010, pp. 1–8.

[34] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, and et al., *Distributed prioritized experience replay*, 2018. eprint: `arXiv:1803.00933`.

[35] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, and et al., *Rainbow: Combining improvements in deep reinforcement learning*, 2017. eprint: `arXiv:1710.02298`.

[36] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization", in *Proceedings of Machine Learning Research*, vol. 37, 2010, pp. 1889–1897.

[37] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, *Proximal policy optimization algorithms*, 2017. eprint: `arXiv:1707.06347`.

[38] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, and et al., *Soft actor-critic algorithms and applications*, 2018. eprint: `arXiv:1812.05905`.

[39] J. Ho and S. Ermon, "Generative adversarial imitation learning", in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Curran Associates, Inc., 2016, pp. 4565–4573. [Online]. Available: `http://papers.nips.cc/paper/6391-generative-adversarial-imitation-learning.pdf`.

[40] A. Nichol, *Competing in the obstacle tower challenge*. [Online]. Available: `https://blog.aqnichol.com/2019/07/24/competing-in-the-obstacle-tower-challenge/`.

[41] *Training configuration file*. [Online]. Available: `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Training-Configuration-File.md`.

[42] *Using tensorboard to observe training*. [Online]. Available: `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md`.

[43] *Agents*. [Online]. Available: `https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Learning-Environment-Design-Agents.md`.

Appendix A

Optimal Fight

Figure A.1 shows the attacks performed by the Boss if it follows the optimal policy. This same behavior is the one used for creating the demonstration files for the first and third case study shown in Section 4.

```
0,02;3,75;bg;0;-;100
0,04;3,71;bg;0;0,68;100
0,06;4,503334;-;-;-;100
0,08;4,463334;-;-;-;100
0,1;4,423334;bp;0;0,38;100
0,12;4,383334;bl;3;0,32;100
0,14;4,343334;-;-;-;100
0,16;4,303334;-;-;-;100
0,18;4,263334;-;-;-;100
0,2;4,223334;-;-;-;100
0,22;4,183334;-;-;-;100
0,24;4,143334;-;-;-;100
0,26;4,103334;-;-;-;100
0,28;4,063334;-;-;-;100
0,3;4,023334;-;-;-;100
0,32;3,983334;bl;3;0,52;100
0,34;3,943334;-;-;-;100
0,36;3,903334;-;-;-;100
0,38;3,863334;bp;0;0,66;100
0,4;3,823334;-;-;-;100
0,42;3,783334;-;-;-;100
0,44;3,743334;-;-;-;100
0,46;3,703334;-;-;-;100
0,48;3,663334;-;-;-;100
0,5;3,623334;-;-;-;100
0,52;3,583334;bl;3;0,72;100
0,54;3,543334;-;-;-;100
0,56;3,503335;-;-;-;100
0,58;3,463335;-;-;-;100
0,6;3,423335;-;-;-;100
0,62;3,383335;-;-;-;100
0,64;3,343335;-;-;-;100
0,66;3,303335;bp;0;0,94;100
0,68;3,263335;bg;0;-;100
0,7;3,223335;bg;0;1,34;100
0,72;4,016668;-;-;-;100
0,74;3,976668;-;-;-;100
0,76;3,936668;bl;3;0,96;100
0,78;3,896668;-;-;-;100
0,8;3,856668;-;-;-;100
0,82;3,816669;-;-;-;100
0,84;3,776669;-;-;-;100
0,86;3,736669;-;-;-;100
0,88;3,696669;-;-;-;100
0,9;3,656669;-;-;-;100
0,92;3,616669;-;-;-;100
0,94;3,576669;bp;0;1,22;100
0,96;3,536669;bl;3;1,16;100
0,98;3,496669;-;-;-;100
1;3,456669;-;-;-;100
1,02;3,416669;-;-;-;100
1,04;3,376669;-;-;-;100
1,06;3,336669;-;-;-;100
1,08;3,296669;-;-;-;100
1,1;3,256669;-;-;-;100
1,12;3,216669;-;-;-;100
1,14;3,176669;-;-;-;100
1,16;3,136669;bl;3;1,36;100
1,18;3,096669;-;-;-;100
1,2;3,056669;-;-;-;100
1,22;3,016669;bp;0;1,5;100
1,24;2,976669;-;-;-;100
1,26;2,936669;-;-;-;100
1,28;2,896669;-;-;-;100
1,3;2,856669;-;-;-;100
1,32;2,816669;-;-;-;100
1,34;2,776669;bg;0;-;100
1,36;2,736669;bg;0;2;100
1,38;3,530003;-;-;-;100
1,4;3,490003;-;-;-;100
1,42;3,450003;bl;3;1,62;100
1,44;3,410003;-;-;-;100
1,46;3,370003;-;-;-;100
1,48;3,330003;-;-;-;100
1,5;3,290003;bp;0;1,78;100
1,52;3,250003;-;-;-;100
1,54;3,210003;-;-;-;100
1,56;3,170003;-;-;-;100
1,58;3,130003;-;-;-;100
1,6;3,090003;-;-;-;100
1,62;3,050003;bl;3;1,82;100
1,64;3,010003;-;-;-;100
1,66;2,970003;-;-;-;100
1,68;2,930003;-;-;-;100
1,7;2,890003;-;-;-;100
1,72;2,850003;-;-;-;100
1,74;2,810003;-;-;-;100
1,76;2,770003;-;-;-;100
1,78;2,730003;bp;0;2,06;100
1,8;2,690003;-;-;-;100
1,82;2,650003;bl;3;2,02;100
1,84;2,610003;-;-;-;100
1,86;2,570003;-;-;-;100
1,88;2,530003;-;-;-;100
1,9;2,490003;-;-;-;100
1,92;2,450003;-;-;-;100
1,94;2,410003;-;-;-;100
1,96;2,370003;-;-;-;100
1,98;2,330003;-;-;-;100
2;2,290003;bg;0;-;100
2,02;2,250003;bg;0;2,66;100
2,04;3,043336;-;-;-;100
2,06;3,003336;-;-;-;100
2,08;2,963336;bp;0;2,36;100
2,1;2,923336;bl;3;2,3;100
2,12;2,883336;-;-;-;100
2,14;2,843336;-;-;-;100
2,16;2,803336;-;-;-;100
2,18;2,763336;-;-;-;100
2,2;2,723336;-;-;-;100
2,22;2,683336;-;-;-;100
2,24;2,643336;-;-;-;100
2,26;2,603336;-;-;-;100
2,28;2,563336;-;-;-;100
2,3;2,523336;bl;3;2,5;100
2,32;2,483336;-;-;-;100
2,34;2,443336;-;-;-;100
2,36;2,403336;bp;0;2,64;100
2,38;2,363336;-;-;-;100
2,4;2,323336;-;-;-;100
2,42;2,283336;-;-;-;100
2,44;2,243336;-;-;-;100
2,46;2,203336;-;-;-;100
2,48;2,163336;-;-;-;100
2,5;2,123336;bl;3;2,7;100
2,52;2,083336;-;-;-;100
2,54;2,043336;-;-;-;100
2,56;2,003336;-;-;-;100
2,58;1,963336;-;-;-;100
2,6;1,923336;-;-;-;100
2,62;1,883336;-;-;-;100
2,64;1,843336;bp;0;2,92;100
2,66;1,803336;bg;0;-;100
2,68;1,763335;bg;0;3,32;100
2,7;2,556669;-;-;-;100
2,72;2,516669;-;-;-;100
2,74;2,476669;bl;3;2,94;100
2,76;2,436669;-;-;-;100
2,78;2,396669;-;-;-;100
2,8;2,356669;-;-;-;100
2,82;2,316669;-;-;-;100
2,84;2,276669;-;-;-;100
2,86;2,236669;-;-;-;100
2,88;2,196669;-;-;-;100
2,9;2,156669;-;-;-;100
2,92;2,116669;bp;0;3,2;100
2,94;2,076669;bl;3;3,14;100
2,96;2,036669;-;-;-;100
2,98;1,996669;-;-;-;100
3;1,956669;-;-;-;100
3,02;1,916669;-;-;-;100
3,04;1,876669;-;-;-;100
3,06;1,836668;-;-;-;100
3,08;1,796669;-;-;-;100
3,1;1,756669;-;-;-;100
3,12;1,716668;-;-;-;100
3,14;1,676668;bl;3;3,34;100
3,16;1,636668;-;-;-;100
3,18;1,596668;-;-;-;100
3,2;1,556668;bp;0;3,48;100
3,22;1,516668;-;-;-;100
3,24;1,476668;-;-;-;100
3,26;1,436668;-;-;-;100
3,28;1,396668;-;-;-;100
3,3;1,356668;-;-;-;100
3,32;1,316668;bg;0;-;100
3,34;1,276668;bg;0;3,98;100
3,36;2,070002;-;-;-;100
3,38;2,070002;-;-;-;100
3,4;2,030002;bl;3;3,6;100
3,42;1,990002;-;-;-;100
3,44;1,950002;-;-;-;100
3,46;1,910002;-;-;-;100
3,48;1,870002;bp;0;3,76;100
3,5;1,830002;-;-;-;100
3,52;1,790002;-;-;-;100
3,54;1,750001;-;-;-;100
3,56;1,710001;-;-;-;100
3,58;1,670002;-;-;-;100
3,6;1,630001;bl;3;3,8;100
3,62;1,590001;bs;0;-;100
3,64;1,550001;bs;0;4,08;100
3,66;1,510001;-;-;-;100
3,68;1,510001;bh;0;-;100
3,7;1,510001;bh;0;-;100
3,72;1,510001;bh;0;-;100
3,74;1,510001;bh;25;4,54;100
Player
```

Figure A.1: Snippet of the optimal fight as seen in the *Fight Logs* created. Where every column separated by semicolons corresponds to: Frame second, distance between agents, abbreviated attack name, damage dealt, time in which will be ready to be used again, and the Boss health. The final line of the file shows the losing agent for that episode.

Appendix B

All GAIL, BC and GAIL + BC Rewards

Figure B.1 shows the mean rewards obtained per case study for the learning agent when using PPO + GAIL, PPO + BC and PPO + GAIL + BC.

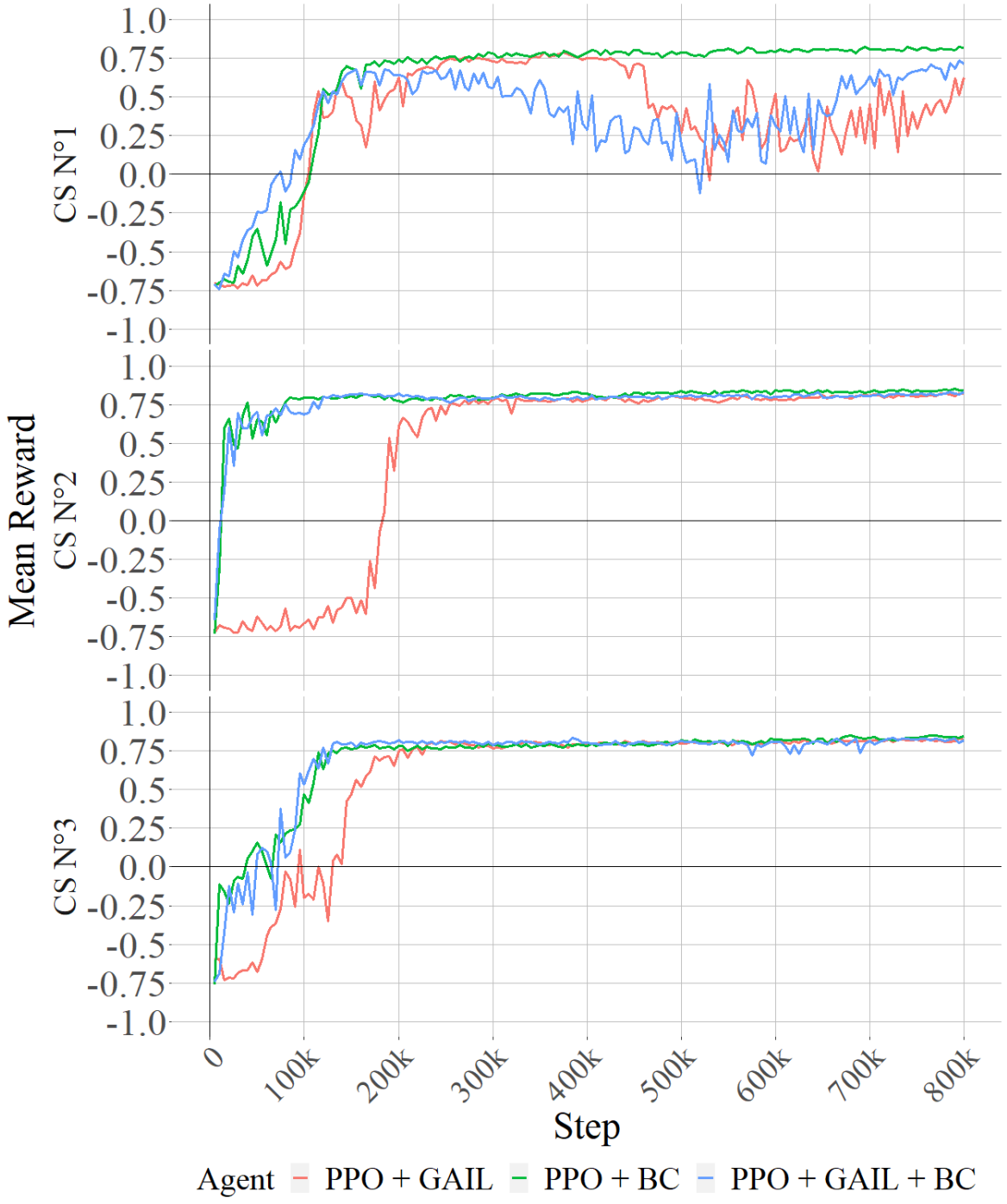

Figure B.1: Comparison for each Case Study (CS) between the mean rewards obtained when the learning agent used PPO + GAIL, PPO + BC and PPO + GAIL + BC.

Appendix C

Reward Entropy Correlation

Table C.1 shows the results of the Pearson correlation between mean rewards obtained and entropy in the best (PPO + BC with the sub optimal policy in the demonstration file), intermediate (PPO), and worst (PPO + GAIL with the optimal policy in the demonstration file) training cases. All correlations were significant at $p < 0.001$.

Agent	Pearson Correlation Statistic - CS N°1	Pearson Correlation Statistic - CS N°2	Pearson Correlation Statistic - CS N°3
PPO	-0.925	-0.925	-0.925
PPO + GAIL	-0.908	-0.932	-0.825
PPO + BC	-0.740	-0.790	-0.884
PPO + GAIL + BC	0.339	-0.773	-0.933

Table C.1: Pearson correlation results for each learning agent per Case Study (CS) between mean rewards obtained and entropy.