DIAMBRA Arena: a New Reinforcement Learning Platform for Research and Experimentation

Alessandro Palmas - DIAMBRA - alex@diambra.ai

Abstract

The recent advances in reinforcement learning have led to effective methods able to obtain above human-level performances in very complex environments. However, once solved, these environments become less valuable, and new challenges with different or more complex scenarios are needed to support research advances. This work presents DIAMBRA Arena, a new platform for reinforcement learning research and experimentation, featuring a collection of high-quality environments exposing a Python API fully compliant with OpenAI Gym standard. They are episodic tasks with discrete actions and observations composed by raw pixels plus additional numerical values, all supporting both single player and two players mode, allowing to work on standard reinforcement learning, competitive multi-agent, human-agent competition, self-play, human-in-the-loop training and imitation learning. Software capabilities are demonstrated by successfully training multiple deep reinforcement learning agents with proximal policy optimization obtaining human-like behavior. Results confirm the utility of DIAMBRA Arena as a reinforcement learning research tool, providing environments designed to study some of the most challenging topics in the field.

Keywords: reinforcement learning, transfer learning, multi-agent, games

1. Introduction

In the past few years, advances combining deep learning (DL) with reinforcement learning (RL) reached a few outstanding milestones (Silver et al., 2016, 2017; Berner et al., 2019), showing that general algorithms can obtain top level performance even when facing complex tasks, while treating them as closed-boxes and without any problem-specific knowledge.

In order to compare different approaches, and to measure their rate of improvement, the research community needs a valid set of benchmarks. In RL the role of such benchmarks is played by software packages providing problems, also called environments, that are challenging for RL to solve. In recent years, appearance of such tools relevantly contributed to the rapid development of this domain, making available problems of different complexity and high scalability.

Since simulation environments are the fundamental building block on which the RL community relays for testing its algorithms, their quality is of critical importance. In spite of that, the literature focused on this highly relevant aspect is not as comprehensive and developed as the one concerning algorithms and methods (Juliani et al., 2018).

A large portion of current RL research platforms is based on popular video games or game engines such as Atari 2600, Quake III, Doom, and Minecraft. It has been a long time since artificial intelligence (AI) research started leveraging games to find meaningful, well posed and challenging problems, starting from chess and checkers (Shannon, 1950; Samuel, 1959) and the application of RL to the game of Backgammon (Tesauro, 1995). This trend continues today, as the AI research community shares the idea that creating algorithms able to reach human-level performances on these games, efficiently solving the same complex challenges, notably contributes to the pursuit of machine intelligence (Laird and VanLent, 2001).

As deep reinforcement learning (DeepRL) research finds new algorithms and above human-level performance is obtained, the benchmarks based on existing environments become less valuable and their impact as research drivers becomes marginal. There is a deep and entangled relation between algorithmic progress and new environments release, where the latter is key to favor the former. If on one side the research community is expected to advance the RL state-of-the-art developing better performing algorithms, who is in charge of delivering new high-quality environments is way less obvious. The task is in fact as challenging as algorithms development, requiring a relevant amount of time for design and implementation, as well as specialized domain knowledge. Proper care must be devoted to the creation of environments that pose meaningful challenges to learning systems, preferring properties that intercept main research interests to make them valuable benchmarks.

Figure 1: Snapshots of some DIAMBRA Arena environments
Within this context, this paper presents DIAMBRA Arena, a brand new software package featuring a collection of carefully selected, high-quality environments for RL research and experimentation. It has been developed with the goal of opening a high-quality curated stream of ever new environments, focusing on key aspects of RL research: competitive / cooperative multi-agents, agents transfer learning and generalization, hierarchical learning, human-agent cooperation / competition and human-in-the-loop training.

It also aims at allowing everyone willing to approach RL to have access to state-of-the-art environments and to be able to use them. This means to design and develop something that is not extremely demanding in terms of hardware requirements, that runs on the majority of operating systems (OSs), well documented and provided with a comprehensive collection of working examples.

Finally, it moves in the direction of applying on RL research, study and experimentation, the current gamification trend, which is being successfully adopted in different fields of science lately (Feger et al., 2018; Kalogiannakis et al., 2021; Gari et al., 2018; Hursen and Bas, 2019). Favoring exciting and rewarding experiences, this will be a key aspect in motivating and getting more people involved in the field.

The remainder of the paper is structured as follows: Section 2 discusses the state of the art of RL environments and provides an overview of relevant related work. Then, Section 3 presents DIAMBRA Arena software package and its most important technical details. In Section 4 it is shown how DeepRL agents have been successfully trained in the games provided, presenting their architecture, the adopted training strategy and achieved performances and results. In Section 5 one finds a discussion focusing on research directions that are enabled by DIAMBRA Arena, current limitations and the features roadmap planned for the near future. Finally, Section 6 is where conclusions about this work are presented.

2. Background and Related Work

RL problems, formulated in the standard form, assume there is an agent that is able to interact with an environment in iterative steps. Each iteration, the agent selects an action to execute, and it receives an observation and a reward from the environment. A RL algorithm aims at maximizing some measure of the agent total reward, as the agent interacts with the environment. In the RL literature, this formulation is typically formalized as a partially observable Markov decision process (POMDP).

In the episodic declination of RL, the agent experience is split into a series of episodes. Each episode starts from an initial state and the interaction proceeds until a terminal state is reached. The goal in episodic RL is maximizing the expectation of total cumulative reward per episode, and to achieve a high level of performance in the lowest number of episodes possible.

Around year 2012, the first benchmarks have started to arise in standard form, with the aim of creating a common test-bed to measure algorithms performances and to stimulate RL research. Since then, a few very notable environments and software packages have been developed and used to demonstrate tremendous improvements and outstanding breakthroughs in the RL domain, proving how important their role is in driving advancements in this field. Here is a list of the most relevant ones currently used in RL research, all publicly available:

- **Arcade Learning Environment (ALE)** (Bellemare et al., 2012): provides an interface to hundreds of Atari 2600 game environments with distinctive features. It has been and is still widely adopted as rigorous test-bed for evaluating and comparing approaches and algorithms in RL, representing the first important effort made in this standardization direction.

- **OpenAI Gym and OpenAI Retro** (Brockman et al., 2016; Nichol et al., 2018): OpenAI Gym includes a collection of benchmark environments that expose a common interface, providing different categories of problems like classic control and toy text, algorithmic, Atari and 2D and 3D robots. OpenAI Retro creates RL environments from various emulated video games.

- **DeepMind Lab** (Beattie et al., 2016): first-person 3D game platform, it can be used to study autonomous artificial agents learning complex tasks in large, partially observed, and visually diverse worlds. Powered by a fast and widely recognized game engine, features a simple and flexible API that enables creative task-designs.

- **Starcraft II Learning Environment** (Vinyals et al., 2017): based on the game StarCraft II, it represented a new grand challenge for RL, providing a more difficult class of problems than those considered in most previous research. Its main features are: multi-agent setting with multiple players interacting, imperfect information and partial observability, large action space, large state space, and delayed credit assignment requiring long-term strategies over thousands of steps.

- **Unity ML-Agents and Obstacle Tower** (Juliani et al., 2018; 2019): Unity ML-Agents is open source project enabling to create simulated environments using the Unity Editor, and interact with them via a Python API. It includes a set of example environments together with state of the art RL, imitation learning and self-play algorithms in both symmetric and asymmetric games. Obstacle Tower is a high fidelity, 3D, 3rd person, procedurally generated environment. It provides both low-level control and high-level planning problems in tandem, learning from pixels and a sparse reward signal, as well as performance evaluation based on unseen instances of the environment.

- **VizDoom** (Kempka et al., 2016): based on the classical first-person shooter video game, Doom, uses raw visual information employing the first-person perspective in a semi-realistic 3D world. It is lightweight, fast, and highly customizable via a convenient mechanism of user scenarios.
3. DIAMBRA Arena

This section presents DIAMBRA Arena software package and its most important technical details. It starts providing a high level overview, then it describes its technical details in terms of its basic usage, available settings, observation and action spaces, the reward wrappers and environment wrappers. Next subsection describes its advanced features, such as human-in-the-loop training and multi-agent and self-play, and the final subsection compares DIAMBRA Arena with other RL environments in terms of features, speed and memory footprint is discussed in Section 3.4.

3.1. Overview

DIAMBRA Arena is a software package featuring a collection of high-quality environments for RL research and experimentation. It provides a standard interface to popular arcade emulated video games, offering a Python API fully compliant with OpenAI Gym format, that makes its adoption smooth and straightforward.

It is accompanied by a comprehensive documentation (DIAMBRA, 2022a) and its repository (DIAMBRA, 2022b) comes with a collection of examples covering main use cases of interest that can be run in just a few steps. It supports all major OSs (Linux, Windows and MacOS) and can be easily installed via Python PIP, as described in the installation section of the documentation. It is completely free to use, the user only needs to register on the official website.

The first version of the software focuses on fighting games, creating a robust and consistent package. All of them are episodic RL environments, with discrete actions that represent gamepad buttons. The observations to be used for control are composed by a screen pixels buffer plus additional data made of RAM values representing game elements such as characters health values or characters stage side. The problem can thus be framed as “control from pixels” only, or as a more general one, taking advantage of RAM numerical values too. It is worth noticing that RAM values usage is guaranteed to be fair, meaning that they only provide redundant information that can be retrieved from the game screen alone, and no “hidden” state is contained in it.

Every environment supports both single player (1P) as well as two players (2P) mode. The former is the “classic” arcade race for clearing the game achieving the score record that can be used for standard RL. The latter adds three new, orthogonal dimensions to these environments, making them suitable for research and experimentation in the domains of competitive multi-agent as well as for human-agent cooperation/competition, and allowing to make use of self-play for training.

In addition, it can be easily set up to explore human-in-the-loop training, covering applications like human assisted rewards and the natively supported imitation learning, for which it provides tools to record human expert demonstrations and a specific class to seamlessly load and use them for agent training.

All interfaced games have been selected among the most popular and successful fighting retro-games. They have been chosen so that, while sharing the same fundamental mechanics, they provide slightly different challenges, with specific features such as different number of characters to be used at the same time, how to handle combo moves, possibility to recharge health bars or not, and similar.

Whenever possible, games are released with all hidden/bonus characters unlocked. For every released title, extensive testing has been carried out, being the minimum requirement for a game to be released in beta. After that, the next internal step is training a DeepRL agent to play, and possibly complete it, making sure the 1P mode is playable with no bugs up until game end. This is the condition under which titles are moved from beta to stable status.

3.2. Technical Details

3.2.1. Basic Usage and Settings

DIAMBRA Arena usage follows the standard RL interaction framework: the agent sends an action to the environment, which processes it and performs a transition accordingly, from the starting state to the new state, returning the observation and the reward to the agent to close the interaction loop. Figure 2 shows this typical interaction scheme and data flow.

![Figure 2: Environment interaction framed under the standard RL POMDP paradigm](image-url)
The shortest snippet for a complete basic execution of the environment consists of just 11 lines of code, and is presented in Listing 1 below.

```python
import diambra.arena
env = diambra.arena.make("doapp")
obs = env.reset()

while True:
    actions = env.action_space.sample()
    obs, rew, done, info = env.step(actions)
    if done:
        obs = env.reset()
    break

env.close()
```

Listing 1: DIAMBRA Arena basic usage

A trained agent is expected to replace random action sampling with the result of its policy prediction based on the observation.

All environments share the list of options presented in Table 1 allowing to handle many different aspects of them, controlled by key-value pairs in a Python dictionary. The high level presentation reported here is detailed in depth in the documentation.

### 3.2.2. Observation Space

Environment observations are composed by two main elements: a visual one (the game frame) and an aggregation of quantitative values called RAM states (stage number, health values, etc.). In the default mode, both of them are exposed through an observation space of type `gym.spaces.Dict`.

Figure 3 (on the left) shows an example of Dead Or Alive++ observation where some of the RAM states are highlighted, superimposed on the game frame.

An alternative configuration is also available, triggered by the hardcore setting: when set to True the observation space is composed only by the game frame, discarding all additional numerical values, thus it is of type `gym.spaces.Box`. Additional details can be found in the documentation.

### 3.2.3. Action Space(s)

Actions of the interfaced games can be grouped in two categories: move actions (up, left, etc.) and attack ones (punch, kick, etc.). DIAMBRA Arena provides four different action spaces: the main distinction is between discrete and multi-discrete ones. The former is a single list composed by the union of move and attack actions (of type `gym.spaces.Discrete`), while the latter consists of two sets combined, for move and attack actions respectively (of type `gym.spaces.MultiDiscrete`).

For each of the two options, there is an additional differentiation available: if to use attack buttons combinations or not. This option is mainly available to reduce the action space size as much as possible, since combinations of attack buttons can be seen as additional attack buttons.

The complete visual description of available action spaces is shown in Figure 3 (on the right), where all four choices are presented via the correspondent gamepad buttons configuration.

When run in 2P mode, the environment is provided with an action space of type `gym.spaces.Dict` populated with two items, identified by keys "P1" and "P2", whose values are either `gym.spaces.Discrete` or `gym.spaces.MultiDiscrete` as described above. Additional details on the action spaces are reported in the documentation.

### 3.2.4. Reward Function

The reward is defined as a function of characters’ health values so that, qualitatively, damage suffered by the agent corresponds to a negative reward, and damage inflicted to the opponent corresponds to a positive reward. The quantitative, general, and formal reward function definition defined by Equation 1:

\[
R_t = \sum_{i}^{N_c} (\hat{H}_t^i - \hat{H}_t^i - (\hat{H}_t^i - \hat{H}_t^i))
\]

where:

- \(\hat{H}\) and \(\hat{H}\) are health values for opponent’s character(s) and agent’s one(s) respectively;
- \(\tau\) and \(\tau\) are used to indicate conditions at "state-time" and at "new state-time" (that is before and after environment step);
- \(N_c\) is the number of characters taking part in a round. Usually is \(N_c = 1\) but there are some games where multiple characters are used, with the additional possible option of alternating them during gameplay, like Tekken Tag Tournament where 2 characters have to be selected and two opponents are faced every round (thus \(N_c = 2\)).

For details on Gym spaces, see [https://github.com/openai/gym/tree/master/gym.spaces/](https://github.com/openai/gym/tree/master/gym.spaces/)
The lower and upper bounds for the episode total cumulative reward are defined in Equations 2. They consider the default reward function (Equation 1) for game execution with continue option set equal to 0.0 (no continue allowed).

\[
\min \sum_{t}^{0:T} R_t = -N_r \left( (N_s - 1) (N_r - 1) + N_r \right) \Delta H
\]

\[
\max \sum_{t}^{0:T} R_t = N_s N_r \Delta H
\]

where:

- \(N_r\) is the number of rounds to win (or lose) in order to win (or lose) a stage;
- \(T_s\) is the terminal state, reached when either \(N_r\) rounds are lost (for both 1P and 2P mode) or game is cleared (for 1P mode only);
- \(t\) represents the environment step and for an episode goes from 0 to \(T_s\);
- \(N_s\) is the maximum number of stages the agent can play before the game reaches \(T_s\);
- \(\Delta H = H_{\text{max}} - H_{\text{min}}\) is the difference between the maximum and minimum health values for the given game; usually, but not always, \(H_{\text{min}} = 0\).

For 1P mode \(N_s\) is game-dependent, while for 2P mode \(N_s = 1\), meaning the episode always ends after a single stage (so after \(N_r\) rounds have been won / lost be the same player, either P1 or P2).

For 2P mode, P1 reward is defined as \(R\) in Equation 1, and P2 reward is equal to \(-R\) (zero-sum games). Equation 1 describes the default reward function. It is of course possible to tweak it at will by means of custom reward wrappers (see Section 3.2.5 below). Additional details can be found in the official documentation.

3.2.5. Wrappers

DIAMBRA Arena comes with a large number of ready-to-use wrappers and examples showing how to apply them. They cover a wide spectrum of use cases, and also provide reference templates to develop custom ones. In order to activate wrappers one has just to add an additional dictionary to the environment creation method, having properly populated it. A summary of available wrappers is presented in Table 2 additional details can be found in the official documentation.

3.3. Advanced Features

3.3.1. Human-in-the-Loop Training

Guiding the learning process by leveraging human input is desirable as taking advantage of humans domain expertise boosts learning, and helps the agent learning to behave as humans would expect. Different approaches have been studied by researchers (Zhang et al., 2019) such as humans providing evaluative feedback to the agent (Knox and Stone, 2008), humans manipulating the agent’s observed states and actions (Abel et al., 2016), or learning to imitate expert trajectories (Ho and Ermon, 2016).

The success of the former two families of strategies strongly depends on how the human interfaces with the agent during learning, that is typically very difficult, if not impossible, with the majority of RL platforms and environments, since it requires low-level access to the environment mechanics that is rarely available.

DIAMBRA Arena can be easily set up to support an interactive and collaborative learning process between the human and

| Setting | Description |
|---------|-------------|
| Player | Allows to select single player (1P) or two players (2P) mode. In 1P mode, the same parameter is used to select on which side to play, P1 (left) or P2 (right) |
| Step ratio | Defines how many steps the game (emulator) performs for every environment step. This value has a direct impact on how frequently actions are sent to the game |
| Frame shape | Resizes game frame to prescribed height and width, keeping it RGB or making it grayscale |
| Continue game (1P mode only) | Defines if and how to allow “Continue” when the agent is about to face the game over condition |
| Difficulty (1P mode only) | Selects game difficulty |
| Character(s) | Selects the character(s) the agent will use |
| Characters outfit | Defines the number of outfits to draw from at character selection |
| Action space | Specifies the action spaces to be used |
| Attack buttons combination | Specifies if to include attack buttons combinations or not as available actions |
| Hardcore | Limits the observation to the screen pixels discarding additional RAM states |

Table 1: Environment settings
the agent. The great flexibility given by environment wrappers, allows to easily include human contribution in the training process. It is possible for a human to have real-time access and interact with the agent during training in order to, for example, pause the scene, and control of the agent via keyboard or gamepad commands.

Imitating expert trajectories is a significantly more common approach in research. DIAMBRA Arena provides advanced tools to leverage the imitation learning technique (Hussein et al., 2017). Very useful to speed up / bootstrap learning, usually in the very early training stages, it requires to store human gameplay in order to use it to train the agent policy, typically adopting either the supervised learning approach (behavioural cloning) or using the human to guide exploration in the "classic" RL setting. Recordings must therefore store both observations and actions sent by the human player at least in the former case, and rewards too in the latter.

The package comes with a wrapper dedicated to this purpose and one specific example showing how to set it up. In order to activate it, one has just to add an additional dictionary to the environment creation method.

Human players are requested to play using a USB gamepad, that is interfaced via a custom class named DiambraGamepad. Two main settings needs to be provided: file_path and user_name. The former is the local absolute path where to save recorded trajectories, provided as a string, while the latter is a string variable that can be used to differentiate between users in case recordings comes from multiple players.

**Behavioral cloning approach.** Having a dataset (the collected player experience) with features (environment observations, for example the game frame) and targets (action(s) selected by the human player), one can train the agent policy network as typically done in supervised learning classification problems. The result would be an agent whose behavior replicates the one of the player, that’s where the term behavioral cloning comes from.

**Guided exploration approach.** Since rewards are stored alongside observations and actions, recorded trajectories can be used while remaining inside the RL paradigm: the human player has here the role of “guidance” in the exploration phase, providing the algorithm with a “meaningful” experience of the environment (from a human perspective) in the form of trajectories. This is expected to significantly speedup training, optimizing exploration towards zones of greater relevance in the observation/action domain space.

```python
import diambra.arena

settings = {
    "traj_files_list": ["path/to/recFile1".
        "path/to/recFile2".
        ...] .
    "total_cpus": 2}

env = diambra.arena.ImitationLearning(**settings)
obs = env.reset()

while True:
    obs, rew, done, info = env.step(0)
    if done:
        obs = env.reset()
        break

env.close()
```

Listing 2: DIAMBRA Arena imitation learning example

In order to do so, the software package provides a dedicated class named ImitationLearning with the specific purpose of loading trajectories recorded with the wrapper provided, and stepping through them. **Listing 2** shows a code snippet with all basic instructions required to run it.

It should be noted how in this case the action selection step is not needed anymore since the “history” of environment transi-
tions is already written in stored human player experience files.

3.3.2. Multi-Agent and Self-play
As already mentioned, all environments can be run in both single player and two players mode, the latter making DIAMBRA Arena, de-facto, a software package that can be used for research in the fields of competitive multi-agent and human-agent competition.

This feature also allows to implement self-play (Baker et al., 2020; Bansal et al., 2018), that consists, roughly speaking, in training an agent by making it play (or fight) against itself. In fact, supporting the 2P mode, the library allows the same agent (or two different agents) to play on both sides, one against the other.

This advanced technique offers many advantages: the algorithm always faces an opponent of similar skill level, enabling more efficient learning; it can be adopted also when no single player mode is available for a given game; it gives the agent the freedom to explore and find optimal strategies beyond those known by expert human players.

3.4. Environments Comparison
This section aims at comparing DIAMBRA Arena with the most important tools currently available in the literature that provide similar capabilities. Two main aspects are considered: features exposed by the software and its performances in terms of speed and memory footprint. The former is fundamental for measuring how broadly it can be applied, determining the range of research topics it enables to study. The latter is strictly linked with one of the most important limitations of current RL, sample complexity: the amount of data required by RL algorithms to achieve optimal performances is still very large, thus requiring very efficient environments that allow to generate the needed experience quickly and to perform a large number of experiments in reasonable time.

3.4.1. Features
Table 3 compares different environments in terms of the most important characteristics for problems in this domain: type of observation spaces, type of action spaces, if they provide ready-to-use wrappers, and if they provide multi-agent and advanced features.

Regarding the observation spaces, DIAMBRA Arena covers all options provided by other tools but the full RAM state. This has been an explicit design choice, aiming to push the research towards studying algorithms able to learn in a more human-like condition, so avoiding the option of directly read the complete RAM states. Nonetheless, it can be easily added, if relevant, in future work.

In its current form, DIAMBRA Arena does not provide a continuous action space. This is mainly related to the nature of interfaced games, that are meant to be played with digital controllers (gamepads). Also in this case, extension to games featuring such type of input (for example mouse cursor position) will be subject of future work.

DIAMBRA Arena is one of the few environments providing ready-to-use wrappers. This is a key element in speeding up the implementation of an interface with third party RL libraries, such as Stable Baselines or Ray RLlib.

Lastly, a very relevant aspect is that DIAMBRA Arena allows to study many advanced topics, such as competitive agent-agent, human expert demonstration recording, imitation learning, transfer learning and cross-task generalization. In addition, it also natively covers human assisted reward and competitive human-agent settings, unique features not provided by the others, to the best knowledge of the author.

3.4.2. Performances
Table 4 provides a comparison for the different environments in terms of execution speed, expressed in frames per second (FPS), and memory footprint for a single environment instance, measured in megabytes (MB). All environments have been run in the same machine, a standard low-medium level desktop also used to train DeepRL agents that is described in a later section.

Only most relevant environments are reported in the table, in particular those similar in terms of observation spaces, action spaces and advanced features. By looking at the values reported, one notes that both performance measures for DIAMBRA Arena are similar and around the same order of magnitude of the other RL environments. If compared with the most simple ones (ALE, VizDoom, Retro), the speed is between 2 to 6 times slower, while the memory footprint is between 1.5 to 4 times larger. This slightly lower efficiency is a fair price to pay to have the vast set of features DIAMBRA Arena provides that are not available in the other environments, as shown in the previous subsection. In fact, when compared with an environment of similar complexity (Unity Obstacle Tower), it becomes clear how faster DIAMBRA Arena is (about 10x), and how reasonable the memory footprint turns out to be.

4. Trained DeepRL Agent

In order to validate and confirm that the environments provided can be learned, many tests have been performed, training multiple DeepRL agents on different games, with different setups in terms of game settings, wrappers used, observation and action spaces.

This section describes in detail how these agents have been trained to play different games in single player mode, maximizing the total cumulative reward, where the immediate reward is defined by Equation 1.

4.1. Problem Framing and Algorithm

To provide useful context for the discussion, in what follows the agent trained on Dead Or Alive++ will be considered. A reward normalization wrapper has been applied to it, using a scaling factor equal to \( K = 0.5 \), the game has a \( \Delta H = (H_{\text{max}} - H_{\text{min}}) = 208 \), resulting in a normalization term equal to \( K \Delta H = K(H_{\text{max}} - H_{\text{min}}) = 0.5 \times 208 = 104 \). It has 8 stages, the last always against the same final boss and the second last always against the same character that depends on the one used by the agent. Two rounds are needed to win a stage, and a single character is used. Therefore, with respect to quantities in Equations 2
Table 3: Environments features comparison

| Environment          | Observation spaces                  | Action spaces                  | Ready-to-use wrappers | Multi-agent ready | Advanced Features ready |
|----------------------|-------------------------------------|--------------------------------|-----------------------|-------------------|------------------------|
| DIAMBRA Arena        | Raw pixels (2D/3D), Raw pixels (2D/3D) + Vector data | Discrete, Multi-discrete       | ✓                     |   | Human expert demonstration recording, Imitation learning, Human assisted learning, Transfer learning, Cross-task generalization |
| Unity ML-Agents      | Vector data                         | Discrete, Multi-discrete, Continuous | X                  | Competitive agent-agent | Imitation learning, Transfer learning, Cross-task generalization |
| StarCraft LE         | Raw Pixels (2D), Features layers, Vector data | Compound                       | ✓                     | Competitive agent-agent | Human expert demonstration recording |
| VizDoom              | Raw pixels (3D)                     | Discrete                        | X                    | Competitive agent-agent | Human expert demonstration recording |
| Unity Obstacle Tower | Raw pixels (2D/3D)                  | Discrete, Multi-discrete        | X                    |   | Transfer learning, Cross-task generalization |
| DMLab                | Raw pixels (3D), Vector data        | Discrete, Continuous            | X                    |   | Transfer learning, Cross-task generalization |
| ALE                  | Raw pixels (2D), RAM state          | Discrete                        | X                    |   |   |
| Gym                  | Raw pixels (2D), Vector data, RAM State | Discrete, Multi-discrete, Continuous | ✓                   |   |   |
| Retro                | Raw pixels (2D), RAM state          | Discrete, Multi-discrete        | X                    |   |   |
| MuJoCo               | Vector data                         | Continuous                      | X                    |   |   |

Table 4: Environments performances comparison

| Environment          | Speed [FPS] | Memory [MB] |
|----------------------|-------------|-------------|
| DIAMBRA Arena        | 0.5k        | 140         |
| Unity Obstacle Tower | 0.06k       | 150         |
| Retro                | 1.1k        | 86          |
| DMLab                | 1.2k        | 40          |
| VizDoom              | 1.5k        | 35          |
| ALE                  | 3k          | 70          |

the normalization term defined above, one has $N_c = 1$, $N_s = 8$, $N_e = 2$, resulting in episode total cumulative reward bounds equal to $\min \sum_{t=1}^{T_e} R_t = -18$, $\max \sum_{t=1}^{T_e} R_t = 32$.

The selected game has four different difficulty levels, they do not affect other game settings. The native game frame resolution is $480 \times 512 \times 3$ and there are four different outfits for each character (see Figure 4).

Figure 4: Kasumi available outfits in Dead Or Alive++

Environment settings have been selected so that the agent learns to play with a specific character (Kasumi), as both P1 and P2 (it will randomly start the episodes in one of the two positions with equal probability), while using both the first two outfits for the selected character, sending actions once every 6 game steps (actions frequency 10 Hz, at fixed points in "time").
Settings are summarized in Table 5.

Table 5: Environment settings used during training

| Setting                  | Value |
|--------------------------|-------|
| Player                   | "Random" |
| Step Ratio               | 6    |
| Frame Shape              | [128,128,1] |
| Continue Game            | 0.0  |
| Difficulty               | 3→4 of 4 |
| Character                | "KASUMI" |
| Character Outfit         | 2    |
| Action Space             | "DISCRETE" |
| Attack Buttons Combination| False |
| Hardcore                  | False |

“Difficulty level has been set equal to 3 at the beginning of training, and has been raised to 4 when the agent was able to complete the game in the majority of evaluation episodes using the initial difficulty value.

The selected observation space is made of the latest game frame plus the RAM states. For training this agent only some elements of the latter have been selected, specifically: ownponent health, ownponent side, stage number and actions.

The discrete action space was selected, with no attack buttons combination, for a total of 12 actions.

In addition to the previously mentioned reward normalization wrapper, a number of additional ones have been applied: latest 4 frames are stacked together with no dilation; latest 12 actions are stacked together; observation scaling is applied; and no reward clipping, no-op reset nor actions sticking are applied. All settings are reported in Table 6.

Table 6: Wrappers settings used during training

| WRAPPER SETTINGS                  | Value   |
|-----------------------------------|---------|
| Observation Scaling               | True    |
| Frame Stacking with Dilation      | [4,1]   |
| Action Stacking                   | 12      |
| Reward Normalization              | True    |
| (Scaling factor $K = 0.5$)        |         |
| Reward Clipping                   | False   |
| No-Op Reset                       | 0       |
| Actions Sticking                  | 1       |

The proximal policy optimization (PPO) algorithm (Schulman et al. 2017) has been used, leveraging the open source RL library Stable Baselines (Hill et al. 2018). The PPO2 model implemented therein has been interfaced with DIAMBRA Arena by means of a specific environment wrapper, mainly related to the management of observations format. A custom deep neural network has been designed for the policy and value networks, and provided to the PPO2 class as model to train. All details about networks model architecture, training strategy with hyperparameters and performances and results are described in the following three subsections.

4.2. Model Architecture

The policy and the value networks share all layers up to the latent space, where they bifurcate in two different tails. The shared part is composed by two different data processing pipelines, both performing information extraction, one from frames (frame encoder) and the other one from the RAM states (RAM states encoder). The architecture of the two networks is described below, and also detailed in Table 7 and in Figure 5.

Table 7: Model architecture details

| PART    | LAYER TYPE | DETAILS                     |
|---------|------------|-----------------------------|
| Frame   | In         | 128 × 128 × 4 tensor        |
| Frame   | Conv       | 8 × 8 kernel, 32 filters, relu |
| Frame   | Conv       | 4 × 4 kernel, 64 filters, relu |
| Frame   | Conv       | 3 × 3 kernel, 64 filters, relu |
| Frame   | FC/Out     | 256 neurons (LATENT)        |
| RAM states | Input     | 161 × 1 × 1 tensor         |
| RAM states | FC        | 64 neurons, tanh           |
| RAM states | FC/Out    | 64 neurons (LATENT), tanh  |
| Concat  | In         | 256 × 1 × 1 + 64 × 1 × 1 tensor |
| Concat  | Out        | 320 × 1 × 1 tensor         |
| Value   | In         | 320 × 1 × 1 tensor         |
| Value   | Out        | 1 value                    |
| Policy  | In         | 320 × 1 × 1 tensor         |
| Policy  | Out        | 12 values (actions)        |

The frame encoder is almost the same as the one used in the Nature paper by Mnih et al. (2015), usually referred as Nature CNN. It is composed by three convolution layers plus a fully connected one; the input tensor is of size 128×128×4 and a total of 256 latent features are generated by the final fully connected layer, which receives 9216 values obtained from flattening third convolution layer output tensor (12 × 12 × 64). It uses relu as activation function.

The RAM states encoder is composed by two fully connected layers, generating 64 additional latent features. It uses tanh as activation function.

Latent features extracted by the two encoders are concatenated together obtaining a 1D tensor of size 320.

The policy and value network tails are both made of a single fully connected layer connecting the 320 latent features to, respectively: the set of 12 available actions followed by a softmax block for action selection (policy net), and the single output value (value net).

4.3. Training Strategy

A fair amount of tests has been performed to identify a training strategy able to obtain good results. The resulting con-
configuration consist of: 128 × 128 frame resolution, grayscale frame depth, 4 frames stack, 12 actions stack, 16 parallel environments, 128 steps per update, batch size of 256, 4 training epochs per update, discount factor of 0.94, learning rate schedule $2e^{-4} \rightarrow 2e^{-6}$, clipping factor schedule 0.15 → 0.025.

Additional details are presented in Table 8 and Figure 6. They show the list of hyperparameters values with their description, and the training process block diagram. Three clusters of hyperparameters can be identified related to observation space, rollouts and training updates.

The first four hyperparameters (frame resolution, frame depth, frame stack and actions stack) have a direct impact on the size of the observation space. Different tests have been carried out, especially for frame resolution and frame stack, reaching values up to [256 × 256] and 6 respectively. The selected values, reported in Table 8, allow to obtain a good performance while maintaining a reasonable training time, granting a 6x speedup with respect to the largest configuration tested.

The next three hyperparameters (parallel environments, environment steps per update and batch size) are strictly related to rollouts generation. They define the amount and the diversification of the experience collected by the agent, and also in this case, a trade-off is needed in order to limit RAM and GPU memory requirements. Having many environments running in parallel is particularly important to diversify the collected experience samples, minimizing their correlation, especially for standard on-policy methods (as PPO) that cannot leverage (optionally prioritized) memory replay buffers. The batch size plays a role too in this regard, while the number of environment steps per update must be chosen with care, taking into account in particular rewards sparsity, and making sure the value is high enough to collect informative experience samples, that is samples where collected rewards have relevant effects.

The last four hyperparameters (training epochs number, discount factor, learning rate and clipping factor) influence more low level aspects of training updates. Two key aspects to note of the adopted training strategy are: A) a linearly decreasing schedule, function of training steps, for both the learning rate and the clipping factor to favor fine convergence towards an optimal policy; B) the selected value for rewards discounting
(0.94), which is again a parameter to be chosen with care. In fact, it determines how far back in time the rewards are propagated, thus it must be set taking into account rewards sparsity, past actions importance, and game specific dynamics.

The block scheme representing the complete training process (Figure 6) describes in detail the infrastructure: it consists of a single workstation in which 16 rollout workers collect training experience in parallel, store experience collected in 128 environment steps and discount rewards across them. 10 evaluation workers evaluate agent performances at regular intervals during training. Agent model parameters are shared between the two groups of workers. Rollout data samples are moved in batches to the single GPU to leverage CUDA computing, and model updates are performed accordingly to hyperparameters setup.

4.4. Performances and Results

Two different hardware setups have been used to perform training tests. Table 9 provides details in terms of components and training speed. They can be considered low-to-medium level home desktop configurations in terms of computing power. Agent performance as a function of training steps is presented in Figure 7, where the 10-episodes averaged total cumulative reward for the PPO agent is plotted together with the random agent one. After 25M training steps, the trained agent average reward has reached a value of around 12, notably larger than the random agent one, a clear demonstration it is learning how to play in order to maximize episode total cumulative reward. Visual comparison between the random agent and the trained one demonstrates, even more than the chart, how well the agent policy learned a playing strategy at least very good, if not optimal.

A video comparison showing the agent at three different training stages (10M steps, 25M steps and 50M steps) can be found at this link. There are a few elements clearly showing agent improvement, two that can be easily noted are: A) the trained agent stops performing attack moves when the opponent is far and out of reach. Instead, it starts to wait for the opponent not only to be close enough, but even to stand up when laying on the ground after being hit; B) the trained agent timely performs counter-moves when appropriate, evading opponent’s attacks.

An identical approach has been successfully adopted to train the same DeepRL agent in the other games and omitted here for conciseness. Visual results are publicly available for Street Fighter III and Tekken Tag Tournament at this link and this link respectively.

Achieving these results within a training time of less than two and a half days, using a medium-low level standard home desktop, confirms these environments, while featuring games with complex mechanics, can be successfully used by everyone.

5. Discussion

The presentation of DIAMBRA Arena (Section 3) and DeepRL Agent training results (Section 4), provides many interesting insights. One important advantage of the software

---

2 DeepRL Agent in Dead Or Alive: https://www.youtube.com/watch?v=2IxsMAAEBU
3 DeepRL Agent in Street Fighter: https://www.youtube.com/watch?v=dv72POqyGQc
4 DeepRL Agent in Tekken Tag: https://www.youtube.com/watch?v=XEJ9qfmzv0k

| Hyperparameter          | Value            | Details                                                                 |
|------------------------|------------------|-------------------------------------------------------------------------|
| Frame size             | 128 x 128        | Game frame warping resolution                                           |
| Frame depth            | 1 Channel        | Game frame warping color compression                                    |
| Frame stack            | 4                | Number of most recent frames experienced by the agent stacked together and used as input to the Policy and Value Networks |
| Actions stack          | 12               | Number of most recent actions executed by the agent stacked together and used in the RAM states as input to the Policy and Value Networks |
| Parallel environments  | 16               | Number of environments collecting training rollouts in parallel          |
| Environment steps per update | 128         | Number of environment steps run per update                              |
| Batch size             | 256              | Number of training steps over which each Adam Optimizer update is computed |
| Training epochs per update | 4          | Number of training epochs run per update                                |
| Discount factor        | 0.94             | Discount factor used in return discounting, constant through training    |
| Learning rate          | $2 \times 10^{-4} \rightarrow 2 \times 10^{-6}$ | Learning rate used by Adam Optimizer update, linear scheduler function of training steps |
| Clipping factor        | 0.15 $\rightarrow$ 0.025 | Clipping factor used by PPO surrogate objective clipping, linear scheduler function of training steps |

Table 9: Hardware specs and training numbers

| Setup | CPU | RAM | GPU  | ENV steps/day | Training time |
|-------|-----|-----|------|---------------|---------------|
| #1    | i5 4 cores / 8 threads | 16Gt | GTX 1050 4Gt | ~2.6M | ~10.2 Days 24/7 |
| #2    | AMD Ryzen 9 12 cores/24 threads | 16Gt | GTX 1080Ti 11Gt | ~10.2M | ~2.36 Days 24/7 |

Figure 7: Mean episode reward plot as a function of environment training steps, with random agent score as a reference baseline

The package is the high degree of customization in terms of action spaces, observation space and environments mechanics. For example, it is possible to make the tasks purely visual (using the screen buffer alone as observation) or to consider a selection of (fair) RAM values too as available features, to easily extend the actions set or to use multiple characters.

The software comes with ready-to-use wrappers of different kind, covering observation, action and reward wrapping classes. These are very important elements when it comes to interfacing environments with RL libraries, taking advantage of already implemented ones results in a major speedup.

DIAMBRA Arena guarantees environments accessibility and usability for a very broad audience. Thanks to minimal requirements in terms of computing power, even CPU-only training is a viable option. The comparison with similar tools currently available in the literature, carried out in Section 3.4 demonstrated a comparable performance in terms of speed and memory footprint, while providing more advanced environments enabling research on more advanced and complex RL topics. In addition, Section 4 showed only a few days are needed to train end-to-end a DeepRL agent with a low-medium level workstation. It can be easily interfaced with third party libraries, especially for RL training, as done in this work with Stable Baselines.
In its current form, DIAMBRA Arena provides fighting games as RL tasks, covering many different interesting options for modern RL research that are discussed below. Moreover, additional types of environments are already planned for the near future.

**Agent generalization.** One potential problem of the majority of RL benchmarks is that they are not ideal for testing generalization between similar tasks. This may cause falling into the pitfall of "training RL algorithms on the test set", measuring model performance on the same environment(s) it was trained on. In order to prevent this, RL environments need to provide means by which they can replicate the concept of "train" and "test" split, as typically done for supervised learning datasets. Few-shot learning, cross-task generalization, exploration-maximization, are all topics that would relevantly benefit from such a feature. DIAMBRA Arena already supports this need, providing different tasks with similar scope and mechanics; it allows to run episodes from game start to end, as a sequence of different stages; and allows to select different game difficulty levels. These are all features that can be used to test agent generalization capabilities. To stress this aspect even more, the future addition of new games will favor environments consisting of many similar tasks sampled from a single task distribution creating opportunities to learn how to explore on some levels and transfer this ability to other levels.

**Curriculum and transfer learning.** Typically, RL challenges and tasks are approached with the aim of solving them from scratch. One of the most interesting directions of current research considers sequences of tasks, training the algorithm on one task after the other. These sequences are made of tasks with an increasing level of difficulty, and are meant to be solved in order. As for agent generalization, also these applications are already supported and more effort will be put to extend features in this specific direction.

**Human-in-the-loop training.** Functionalities currently available in the software package allow to easily provide feedback to the agent and to modify the environment during training, thus supporting exploration in this field of research. Also in this case, new types of environments are planned for development in the near future, with the aim of making human-in-the-loop training even more accessible for research and experimentation.

**Multi-agent.** Current version of DIAMBRA Arena already supports multi-agent applications in the competitive flavor (both agent-agent and agent-human). One of the future development directions is to add new environments in which also the cooperative flavor is available, for both agent-agent as well as human-agent settings. Additional directions are being discussed, also touching the so called "value alignment problem", one of the aspects involved in typical reasonings around existential concerns for AI.

**Real-world operation.** The need to validate RL algorithms in the real world is of paramount importance, and would represent a major achievement that could lead to the realization of a great potential. Almost certainly, it will involve aspects related to agent-human cooperation/competition, this may require robust and scalable training where agents play against (or in cooperation with) humans. The aim of developing tools and infrastructure to enable the study of agent-human interaction at scale is also in the road map.

6. Conclusions

This paper presented DIAMBRA Arena, a novel software package for RL research and experimentation. It provides high-quality environments, complemented by a very broad set of tools enabling many different lines of research that focus on major challenges the scientific community is facing, as standard RL, competitive multi-agent, human-agent competition, human-in-the-loop training, imitation learning and self-play. Supporting all major OSs, and implementing the standard OpenAI Gym Python API interface, its adoption is easy and straightforward. Multiple DeepRL agents have been trained in the provided environments, using PPO algorithm through Stable Baselines library, obtaining optimal performances and human-like behavior. Results achieved confirm DIAMBRA Arena utility as a reinforcement learning research platform, providing environments designed to study the most challenging topics in the field.

Acknowledgments

The author thanks Marco Meloni for his insightful comments and suggestions for editing an earlier draft of this manuscript.
