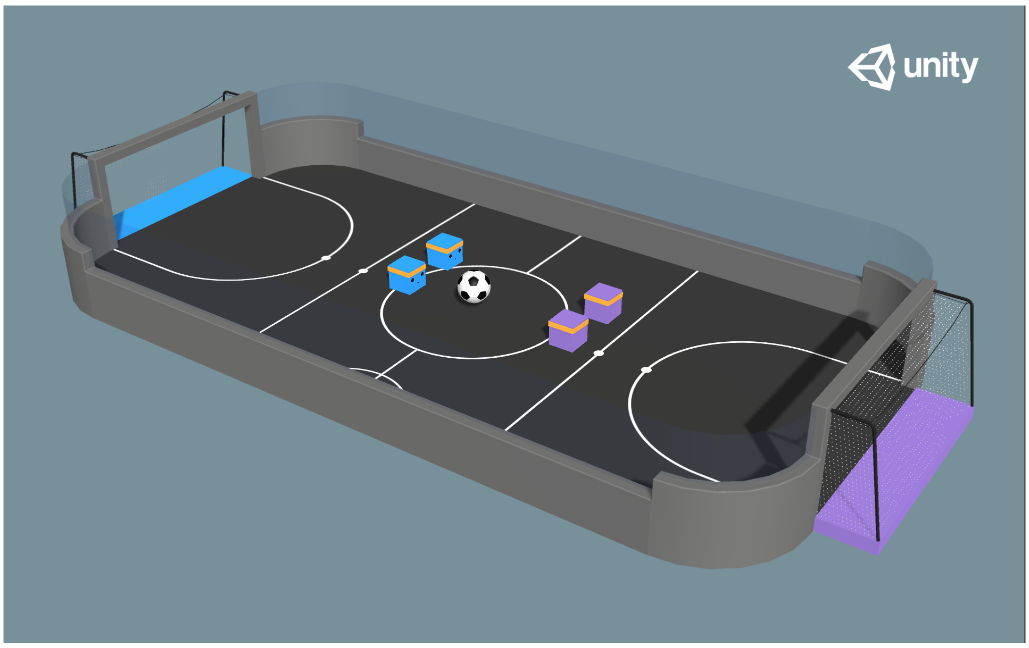
Project Manual - Bachelor Computer Science - Year 2
Project 2.1 - AI and Machine Learning
Controlling Agents in the Unity Game Engine

Dennis Soemers

2024-2025



Period 2.1, 2.2, and 2.3
Academic year 2024-2025
Department of Advanced Computing Sciences
Faculty of Science and Engineering
Maastricht University

1 Project Description

The central topic of Project 2.1 *(Computer Science programme, AI and Machine Learning module)* is to evaluate and analyse the performance of Machine Learning (ML) solutions for automatically controlling agents in a real-time video game engine. Along the way, you will gain experience combining your own code with large existing frameworks, libraries, and code written by others.

1.1 Background

Unity[1] is an engine for the development of 3D real-time video games, among other types. The Unity Machine Learning Agents Toolkit (ML-Agents) [1] extends the engine with a suite of Deep Reinforcement Learning (DRL) algorithms, as well as example environments in which to test them. Reinforcement Learning (RL) is a branch of ML concerned with training agents, which live in some environment in which they may take sequences of actions over longer time spans, to select their actions so as to optimise sums of "reward" signals. Agents may be tasked with balancing balls on top of themselves, navigating to a goal, playing soccer, and so on. See Figure 1 for screenshots of a selection of example environments that are included in ML-Agents.

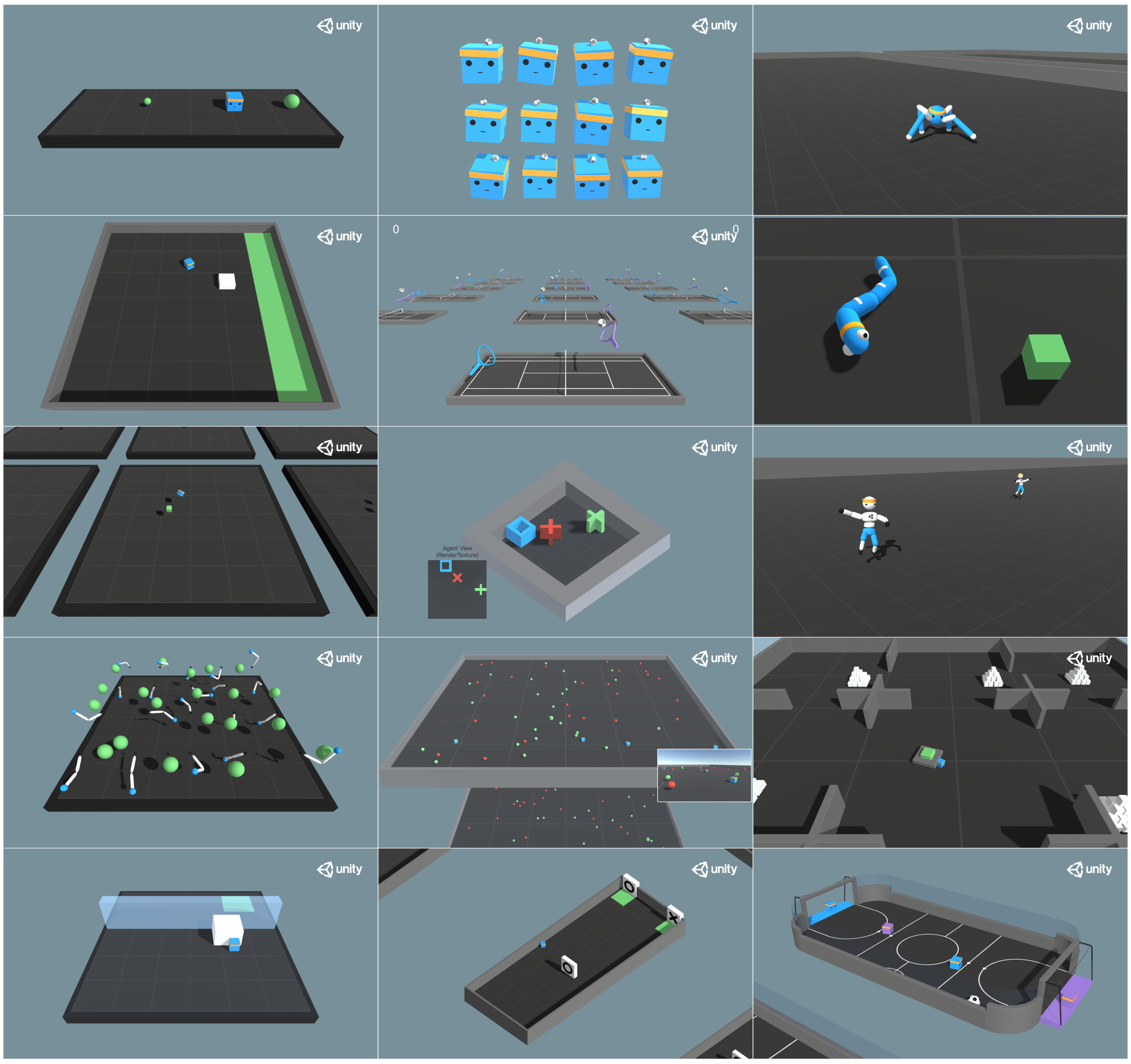


Figure 1: Example environments in Unity ML-Agents. Image source: [1].

[1] `https://unity.com/`

Combining these two tools (Unity and ML-Agents) will enable you to play around with state-of-the-art DRL techniques in real-time 3D simulations, without having to implement either of those from scratch yourself (which would be a very time-consuming task). However, you will have to learn how to effectively use large frameworks and codebases written by other people, possibly using one or more programming languages that you are not yet familiar with. These are important skills to pick up as, in the remainder of your lives as computer scientists and programmers, you will rarely if ever work solely with code that you wrote yourself from scratch.

1.2 Objective

Your main objective is to (i) apply and analyse the performance of state-of-the-art DRL algorithms for training agents in a real-time 3D video game engine, and (ii) build this in a clean, well-documented, public GitHub repository which you will be able to include in your programming portfolio.

In the first phase of the project, you will focus on making a plan (for the entire semester), ensuring that every member of the group sets up and understands the necessary tools and infrastructure, and figuring out how you can add or edit, compile, and run your own code within Unity and the ML-Agents environments. In the second phase, you will run training algorithms from ML-Agents on one or more example environments, analyse their performance, and implement your own new simulated sensors for agents to use as inputs to the training algorithms. Finally, in the third phase, you will more thoroughly analyse and profile the performance of the code, and investigate how scaling up the complexity of environments affects training and simulation speed.

Throughout all three phases, you should ensure to carefully and clearly document your work and provide instructions for any and all used tools and dependencies. The documentation should be written such that, if a new member were to join your team midway through any phase, they would be able to quickly and easily get started.

1.3 Software

For this project, you are required to use:

- Git, for version control, with your repository/repositories hosted on GitHub with public visibility.
- The Unity game engine (the “Unity Personal” free license will suffice, but you may also try to get the “Unity Student” license).
- The Unity Machine Learning Agents Toolkit (ML-Agents) (it is recommended to use the fork and branch from `https://github.com/DennisSoemers/ml-agents/tree/fix-numpy-release-21-branch`, as this includes an important fix for installation to an otherwise stable release branch).

You are free to use any additional libraries or software as you see fit, but *you will be expected to motivate your choice for selecting them.* Unity traditionally uses `C#` as its main programming language, and ML-Agents is built using `Python`, which makes it likely you will have to use one or both of these languages.

2 Project Tasks

The following three subsections describe in more detail what the minimum requirements are for each phase, and how you will be assessed. The final assessment of the project as a whole is a pass or a fail, based on whether or not the minimum requirements were satisfied, but no more fine-grained grade. However, if you are able, we still strongly recommend more deeply exploring any aspect of the project that you are interested in and exceeding minimum requirements. While you cannot be rewarded for this with a higher grade, you will be able to showcase your work in your public GitHub profile, which many of your potential future employers will carefully look at when making hiring decisions!

2.1 First phase

Your deliverables for the first phase are:

1. **A written project plan**, in which you write a detailed plan for the steps to take over the duration of the project. The plan should include at least a description of the work you aim to do, a rough time schedule, and an assessment of risks and how to work around them. The maximum number of pages is 10, but using fewer pages is also fine.
2. **At least one publicly visible GitHub repository**, with relevant documentation, to contain your work. You will continue iterating on this page throughout the entire project.

Throughout this phase, if you aren't already, you will have to become acquainted with (among other things):

- Using git and GitHub for version control. Every individual member of your group should have a GitHub account, and should understand what the following terms mean in the context of git and GitHub: commit, pull, push, branch, fork.
- Basic use of the Unity game engine. You will have to install it, install the ML-Agents package, open example scenarios from ML-Agents, and play around in them.
  - There is documentation for all this online, but it may not always be perfectly clear or accurate. Please ensure that any installation and setup instructions are 100% clear in your own repository.
- Installation of Python, and packages for it. The ML-Agents package is written in Python, and has many other Python packages as dependencies (in many cases requiring specific versions). For other work (for example, the *Introduction to AI* course), you may also need or want specific versions of Python packages (or Python itself) installed, which may differ from the versions that ML-Agents requires! Therefore, we strongly recommend using *virtual environments*,[2] such that you may have different versions installed in different virtual environments.

In the first phase, you should also already try implementing some modifications that requires code changes (not in-editor changes to variables or level layout) to agent, learning, and/or environment code in one or more ML-Agents example environments. For this phase, there is no specific requirement as to *what* change you implement, as long as you can do one. The purpose of this is for you to figure out how best to structure your entire project and installation. For example, you will have to figure out whether you can easily implement code changes in a completely separate repository, or whether this is better done directly inside the ML-Agents codebase itself. If the latter is the case, you may prefer starting out with your own GitHub repository as being a fork of the official one, rather than starting from an empty repository. If you do this, you should still completely overhaul and update the documentation that shows on the front page of your repository, such that it reflects your work and how it differs from and is related to the original repository.

[2] `https://docs.python.org/3/library/venv.html`

2.2 Second phase

Your focus in the second phase will be on running RL algorithms to train agents, and building a new, custom sensor type to use as input for training algorithms. For the most part, you will be able to treat the different algorithms that are already implemented in ML-Agents as black boxes, without necessarily having to understand all of their inner workings. However, you may have to develop an understanding of which algorithms are applicable or not applicable to which types of scenarios (e.g., some algorithms may not work in multi-agent settings, whereas others are specifically designed for multi-agent settings).

There are many parameters that may be adjusted to customise how training algorithms work,[3][4] and it is recommended to already experiment and play around with this a bit in the second phase (in preparation of the third phase). Aside from open-ended experimenting with already existing code, there is a concrete requirement for your own new code addition: the implementation of a new input type[5] for the training algorithms, for use in the "Soccer Twos" environment. By default,[6] the input in this environment is constructed from 11 ray-casts in a 120-degrees cone in front of an agent, 3 ray-casts behind the agent. However, as it is unrealistic for football players to have eyes in the back of their head, we would like you to change this: the agents should only use information from ray-casts in front of them. Having some idea of what is behind or to the side of an agent can be important though, and you are required to think of a realistic way to implement this. A non-exhaustive list of ideas that you may consider implementing:

- Include memory of observations from (forwards-directed) ray traces of previous frames. Note that you must implement this in your own code, not through a simple parameter change of existing code.
- Simulate sound by including observations of nearby (moving) objects. Note that, in this case (without vision), it would likely be impossible to hear whether an observed agent is a teammate or an opponent.
- Decouple the movement direction from the vision cone, simulating an agent that can move in one direction whilst looking in a different direction. The vision angle may either be an additional action of the agent, determined randomly or according to a handcrafted heuristic, or move according to some regular pattern.

In the Midway Evaluation at the end of this phase, you will give **a live demonstration of your work and explain it in an in-person presentation of up to 20 minutes (including at least 5 minutes for questions)**.

2.3 Third phase

In the third phase, you should thoroughly analyse the performance of one or more RL algorithms from ML-Agents in one or more environments. You should evaluate how performance is affected by changes in any (not necessarily all) of the following:

- Parameters of the learning algorithms (learning rates, neural network sizes, number of concurrent copies of the environment, etc.).
- Types of inputs/sensors used.
- Whether training is done in-editor or from a compiled executable.

[3] `https://unity-technologies.github.io/ml-agents/Training-ML-Agents/`

[4] `https://unity-technologies.github.io/ml-agents/Training-Configuration-File/`

[5] `https://unity-technologies.github.io/ml-agents/Learning-Environment-Design-Agents/`

[6] `https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Learning-Environment-Examples.md#soccer-twos`

The word "performance" here should be interpreted as referring to various aspects, such as amount of wall time required before an agent (consistently) reaches a certain reward threshold, but also CPU, GPU, and/or memory usage, and framerate of the simulations. It is strongly recommended to try using Unity's built-in Profiler and getting some experience with such tools.

All of your experiments should be well-documented and easily reproducible. Parameters should be changed using configuration files, or, at the bare minimum, well-documented command-line arguments. If someone wanted to reproduce your experiments (which they should be able to from the information provided in your GitHub page), it should not be necessary for them to manually change any lines of code in between experiments.

In this phase, you will provide us with **a written report explaining your work (max 10 pages, excluding appendices and references)**. You will also give **a short, live demonstration and presentation, and discuss the work with us**. A total of 30 minutes will be reserved for the demonstration, presentation, and discussion, but we expect a sizeable portion of this (at least 15 minutes, preferably more) to be discussion: you only need to prepare a relatively short presentation alongside the demonstration.

References

[1] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

A Tips for Installation of Dependencies

Regardless of which operating system (OS) you use, you will require at least four of the following five dependencies. Only Visual Studio (A.2) may be unique to Windows. Some of the other tips below may also only be applicable to or precisely correct for Windows, but general recommendations will apply to, or be similar for, all OSes. You will often be able to find solutions for any trouble you run into by browsing the web.

A.1 Git

You will need to install, and become acquainted with, the `git` version control system. You will also need to create a GitHub account, and become familiar with GitHub.

A.2 Visual Studio

In principle you will not require most of Visual Studio (except if you choose to use it as your IDE of choice for any of the involved programming languages), but a part of the installation process of `ml-agents` (A.5) may require builds tools that come with Visual Studio. For Windows users, we recommend downloading Visual Studio from `https://visualstudio.microsoft.com/downloads/` (the *Community* edition). In its installer, make sure to select at least *Python development* (under *Web & Cloud*), and *Desktop development with C++* (under *Desktop & Mobile*). The following additions may also be useful, especially if during the project you end up deciding you'd like to use Visual Studio for any programming in C# for Unity:

- *.NET desktop development.*
- *Windows application development.*
- *Game development with Unity.*

A.3 Python

A.3.1 Python Versions

You will need to install Python. Make sure to pay attention to version numbers! The official documentation of `ml-agents` say that version 3.10.12 is required. However, at least for Windows users, version 3.10.11 is the latest version (among 3.10.x versions) which is still easily installable via a Windows installer, so you may wish to give this one a try anyway. Version 3.10.12 will be more challenging to install on Windows, as it must be built from source.

A.3.2 Virtual Environments

After installing Python, it is strongly recommended to create a virtual environment (using, e.g., `"<python installation directory>/python.exe" -m venv "<ml-agents installation directory>ml-agents/venv"`, and to activate that virtual environment whenever you install any further python packages for this project, or whenever you work on this project.

A.4 Unity

Download and install the Unity game engine from `https://unity.com/`.

A.5 ml-agents

Make sure *not* to clone ml-agents from the official repository, but rather from a specific branch on a fork of our own, using: `git clone --branch fix-numpy-release-21-branch https://github.com/DennisSoemers/ml-agents.git`. This is a stable branch, including a fix for an important installation error reported in `https://github.com/Unity-Technologies/ml-agents/issues/6102` (fix not available from the official repository as of the time of this writing).