Software Report

GitHub Link

This document outlines further details into the software development life cycle for the accompanying software. This programme was used to train, test, and gather the results which were presented in the thesis document. In particular, we present the evaluations that were factored into selecting a suitable toolkit, the dissertations made to improve system usability, and a summary of the installation and usage instructions (the GitHub repo contains a more comprehensive setup and usage guide).

1 SELECTION OF SOFTWARE SETUP

As reinforcement learning is a relatively new and rapidly advancing field, there is no defacto workflow. This led to an exhaustive search of various libraries, toolkits, environments, and algorithms to perform this experiment. Additionally, these evaluations would change depending on new library versions, which in some cases brought about significant API changes. This search also led to exploring and learning various computing and containerised platforms, such as WSL2 and Docker respectively, to ensure maximum compatibility. In selecting a set of suitable social dilemma environments, we explored open-source implementations of Clean Up, Prisoner's Dilemma, Harvest, Stag Hunt, Tiger-Deer, and Water World. These subsequently required evaluations of their respective environment frameworks and wrappers, including OpenAI Gym, PettingZoo, SuperSuit, and Lab2D. We then tested various algorithm abstraction frameworks such as Keras-RL, Keras-RL2, Stable-Baselines3, and Ray RLlib.

After thoroughly evaluating these various environments, platforms and libraries, we conducted this research with DeepMind's Melting Pot (built on top of Lab2D) for the chosen environment implementations of Clean Up and Prisoner's Dilemma. This is mainly due to DeepMind providing the highest quality implementations of these environments with a well-documented API. Furthermore, Melting Pot offers a standardised methodology for comparing and contrasting the efficacy of various MARL algorithms across various environments exhibiting different phenomena. This aligns perfectly with our chosen field of research. We chose to pair Melting Pot with Ray RLlib for the algorithm implementations. Ray RLlib provides an extensive list of granular algorithm implementations, specifically A3C, which is not offered by competing frameworks.

After examining this sensitive selection of technologies, it was determined that running experiments on a remote HPC would not be viable due to the complex setup procedure that would be required and having less granular control over the experiments. Instead, this research was conducted locally, on an appropriately equipped desktop using the WSL2 platform, and made deployable via a 'Devcontainer' containerised environment.

2 SOFTWARE DEVELOPMENT METHODOLOGY

To as great of an extent as possible, I attempted to implement an agile software methodology. This methodology is based on iterative and incremental software solution development accompanied by continuous feedback (see Figure 1). I used an adjusted version of my initial Gantt Chart, presented in the project proposal, to plan out the iterations for this project. This is shown in Figure 2.

Earlier iterations of this project involved analysing the various frameworks and environment implementations. Like most agile methodologies, I used my newly found gatherings about these frameworks to create simple reinforcement learning applications. These simple applications indicated the usability of these frameworks and environments for a much larger development effort. After

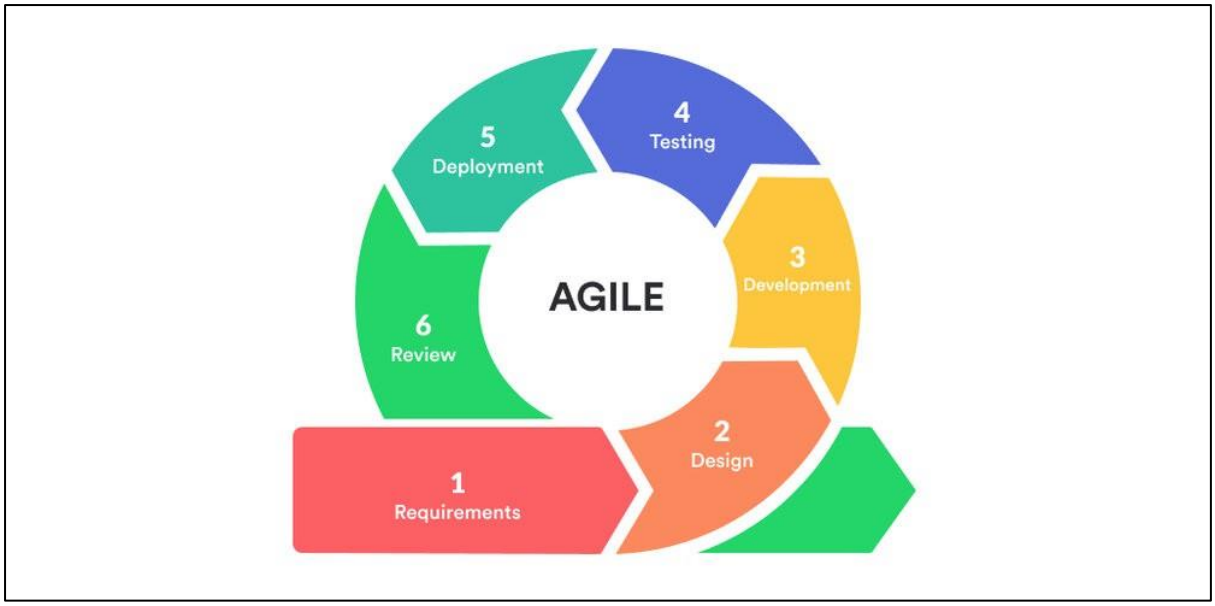

Figure 1: Agile development outline

performing this analysis and project-based evaluation, I concluded that Melting Pot and Ray RLlib are the best frameworks for my use case, as mentioned above.

However, it was arguably during this framework evaluation phase of the project, that I spent the majority of my time. Various open-source environment implementations were investigated to determine their efficacy in conducting our research on. However, these community-developed environments were rarely maintained to the frequency required to keep up with their backbone frameworks. Furthermore these backbone frameworks were diverse themselves, as there is no clear *defacto* workflow in reinforcement learning at this stage. Initially, OpenAI Gym environments were very appealing due to their well-documented API along with the largest relative developer community and online resources. However, Gym went through a major API refactoring that saw much of the older environments obsolete. Whilst these environments were being refactored, the various algorithm abstraction frameworks had conflicting dependencies between the backbone environment frameworks, in what was aptly described at this time as a "*dependency soup*". Thus, after much evaluation Melting Pot and RLlib was the best pairing as it offered stability, quality, and well-documented APIs to perform our experimentation.

The following iterations focused on designing a specification for the system – that is how I plan to modularise the system into various components. Naturally, we Object-Orientated Programming (OOP) to provide for this modular framework and promote reuse during development. The following iterations focused on testing the software to ensure that the chosen parameters were resilient for extended training durations.

Once all software checks were done, and we were satisfied with the experimental pipeline, we planned the order to train the environment and algorithms for the most efficient use of resources. This process is described in the "Methods" section of the accompanying research paper. A key extract from this section are the considerations which had to be made in light of WSL2 RAM allocation limitations. We overcame this hurdle by appropriately adjusting the offending algorithm's stopping criteria, whilst still maintaining a fair comparison.

After all the results were gathered, the software reached the end of its development life cycle, and subsequent iterations focused on enhancing the system's usability, which we achieved with various efforts detailed in Section 3.

3 USABILITY STRATEGY

Table 1: Usability strategy

Area	Description
Command Line Interface (CLI)	We provide the user with a command line interface to run the various scripts. This is the industry standard for SOTA research, as it is the most lightweight and fastest way to run experimentation.
Extensive function documentation	All functions are accompanied by a description of their purpose. Additionally, all function outputs and input arguments are described so there is no ambiguity for future maintenance.
Type hinting	As Python is a dynamically typed language, there is no explicit type declaration; however, with later Python releases, developers can add type hints to each input/output. We use type hinting wherever possible to ensure maximum code readability and easier debugging.
PEP8 code formatting standard	All code is formatted according to the PEP8 standard.
Object-Orientated design	The code is written following an OOP design. This modularity allows for easier troubleshooting, collaboration and reuse, amongst others. OOP also allows for a more intuitive interpretation of the code for any new maintainers.
Default values	All command line arguments have a default value, such that if the user were to omit a parameter when running the script, the programme would not seize.
Containerised deployment	As the software is married to Melting Pot, we piggyback off Melting Pot's containerised deployment method with no need to change any Docker configurations.
Report of testing metrics	After running a testing scenario, output a formatted report for the obtained social metrics.
Example commands	We provide sample commands for the user to run the script. This aids the onboarding process and helps the user get up and running with minimal need to search through the CLI parameters.
Extensive installation instructions	This software includes a comprehensive set of installation instructions that provides multiple ways to perform a step to accommodate the user's computing setup or general preferences.
Future-Proof technologies	We performed an exhaustive survey of all major reinforcement learning libraries and frameworks and landed on a technology stack that we have no reason to believe will become irrelevant. The chosen libraries are well-documented, with a healthy community of users and are backed by major cooperations like Google.

4 INSTALLATION

In this section, we summarise the installation procedure. Refer to the README.md file accompanying the software submission for an in-depth guide.

In short, the installation procedure requires the user to install Melting Pot and merge the contents of this project.

There are two main ways to go about this:

1. Use a Devcontainer (x86 only) and merge the additions and changes.
2. Build the required libraries from source and merge the additions and changes.

4.1 DEVCONTAINER

Melting Pot provides a pre-configured development environment (devcontainer). This allows for an easy, containerised deployment of the software.

Note: The following steps assume you already have Docker installed on your system.

1. Download Melting Pot v1.0.4. You can do this in two ways:
    a. Download from the release page and extract contents
    b. Clone the repository into a container using the devcontainer VSCode extension.
2. Launch the *devcontainer* using the VSCode Containers extension
3. Merge the additions and changes made by my software
4. Download the models
    a. Download the Melting Pot models
    b. Download my trained A3C, PPO, and R2D2 models from this Google Drive link.
5. (Optional) Install *ffmpeg* for environment recording

To enable CUDA support, download and install the Nvidia container toolkit.

4.2 MANUAL INSTALL

1. (Optional) Activate a virtual environment, e.g.:
2. Install *dmlab2d* from the wheel files
3. Install Melting Pot
4. Install Ray RLlib dependencies
5. Merge the additions and changes made by my software
6. Download the models
    a. Download the Melting Pot models
    b. Download my trained A3C, PPO, and R2D2 models from this Google Drive link.
7. Download the models
8. (Optional) Install *ffmpeg* for environment recording
9. (Optional) Install `ffmpeg` for environment recording

5 HOW TO USE

Once again, for a more extensive guide on how to use the software, refer to the *README.md* file accompanying the software submission.

This software is comprised of two main scripts:

1. `*training_and_eval.py*` - runs the training and evaluation of a substrate using a particular algorithm.
2. `*scenario_testing.py*` - tests the trained focal policies in a testing scenario and reports back on various custom social metrics.

Each of these scripts is designed to be run from the command line, as is the standard for SOTA approaches.

`*training_and_eval.py*` takes the following command line arguments:

- `-- *experiment_name*`: A custom name for the experiment.
- `-- *substrate*`: The substrate to use for training and evaluation ('clean_up' or 'prisoners_dilemma_in_the_matrix')
- `-- *algorithm*`: The algorithm to use for training and evaluation ('A3C', 'PPO', 'R2D2')
- `-- *use_policy_sharing*`: Whether to use policy sharing or not (True / False)
- `-- *num_iterations*`: The number of iterations to train for
- `-- *local_dir*`: The directory to store the experiment results in
- `-- *use_gpu*`: Whether to use GPU training or not (True / False)
- `-- *record_env*`: Whether to record the environment or not (True / False)

`*scenario_testing.py*` takes the following command line arguments:

- `--*scenario_name*`: The name of the scenario to test.
- `--*algorithm*`: The name of the algorithm to load and test.
- `--*use_policy_sharing*`: (True / False) Whether to use policy sharing
- `--*substrate*`: The name of the base substrate to test (clean_up or prisoners_dilemma_in_the_matrix)
- `--*experiment_path*`: The path to the experiment directory where you saved the trained model.

6 PROPOSED TIMELINE

Table 2: Tabular task outline

Task	Start Date (MM/DD/YY)	End Date (MM/DD/YY)	Duration (Days)
Task 1: Proposal			
Task 1.1: Final Submission	5/20/22	6/03/22	14
Task 2: Environment			
Task 2.1: Environment suite	6/04/22	6/15/22	11
Task 2.2: Test existing implementations	6/16/22	6/27/22	11
Task 2.3: Implementation	6/28/22	7/09/22	11
Task 3: Algorithms			
Task 3.1: Implementation Research	7/10/22	7/14/22	4
Task 3.2: A3C Configuration	7/15/22	7/19/22	4
Task 3.3: R2D2 Configuration	7/20/22	7/24/22	4
Task 3.4: PPO Configuration	7/25/22	7/29/22	4
Task 4: Experimental Setup			
Task 4.1: Algorithm integration	7/30/22	8/01/22	2
Task 4.2: Hyperparameter optimization	8/02/22	8/05/22	3
Task 4.3: Experimental metrics	8/06/22	8/08/22	2
Task 4.4: Result gathering	8/09/22	8/13/22	4
Task 4.5: Result validation and comparison	8/14/22	8/17/22	3
Deliverables			
Recommended development end date	8/18/22	9/15/22	28
End to End Draft Paper	9/16/22	9/30/22	14
Final Paper and Software submission	10/01/22	10/28/22	27
Project Presentation	10/29/22	11/10/22	19

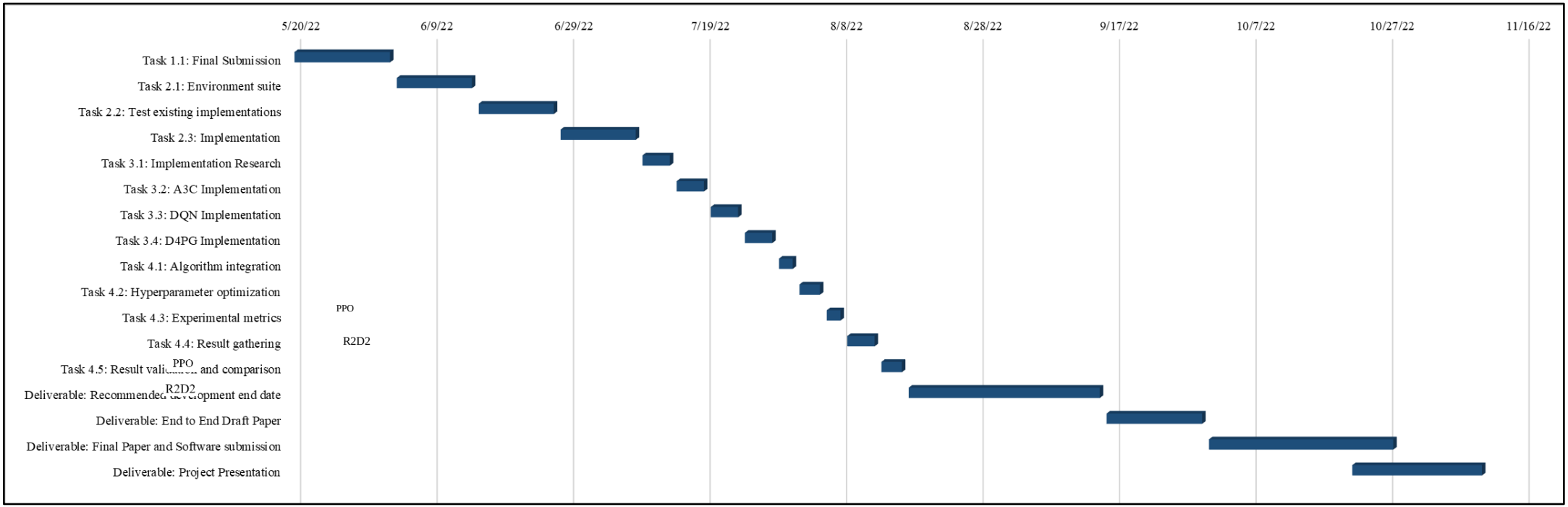

Figure 2: Gantt Chart encompassing project deliverables and milestones