Requirement 7 Report - Software Capabilities

What programs, applications, APIs, and/or libraries are commonly used for Minecraft ML? Include download/installation measures, relevance to our goal, and version numbers if applicable.

Many Minecraft ML projects fit into three broad categories of software stacks: MineRL, corporate-developed AI stacks, and personal project stacks. By far, the most shared machine learning software stack used for Minecraft is the MineRL project, which has its requirements documented in its own section in the research. MineRL is a publicly accessible interface between Minecraft and a machine learning model, which means that there are a variety of different projects that use it, including many published on arXiv. Many large corporations, such as OpenAI [1] and NVIDIA, have also developed Minecraft ML projects. Because these are projects sponsored by corporations, much of their software is not replicable for a senior design project. However, even corporate projects have used the MineRL infrastructure, such as NVIDIA's use of MineDojo, a project derivative of MineRL [2].Additionally, several Minecraft ML projects that do not use MineRL develop their own personal tools for interfacing with the game and for running their desired ML model. One example is a Minecraft ML Software Stack developed by a YouTube creator Poppro [3], for which they created two separate Github projects for their attempt at the problem, PyreNet for running Neural Networks in C++ and MCNE for simulation.

How does Overwolf interface with Minecraft? To what extent and detail does it provide in-game data and events? What types of game data is Overwolf unable to retrieve?

The Overwolf Platform provides an overwolf.games.events API, which notifies when something interesting happens while playing a target game. It's available features that are relevant to our project are [4]:

- gep_internal - Local + Public version number.
- game_info
    - scene - Name of current scene.
    - player_X - List of players in the current game.
    - mc_version - Current Minecraft version active.
- match_info
    - server - Full server id.
    - general_stats - Current general statistics.
    - items_stats - Current item statistics.
    - mobs_stats - Current mob statistics.
    - location - Player's in-game location.
    - facing - Direction the player is facing.

The API supports both vanilla and modded game version, specifically:

- Vanilla
    - 1.12.2 - All events except chat, statistics, scene, and player_x.
    - 1.16 - 1.16.1 - All events except chat and statistics.

    - 1.16.2 - All events except chat.
    - 1.17 - 1.20 - Full support.
- Modded
    - Forge 36.0.0 - 36.2.39
    - Forge 39.0.0 - 44.1.8
    - Fabric 0.14.0 - 0.14.13
- Addons (mods) event is supported for MC 1.8 to 1.19 for Forge and Fabric.

Overwolf also offers a number of other useful APIs, a select few with high relevance to our project are [5]:

- overwolf.io - File I/O, check existence, monitor files.
- overwolf.replays - Capture screenshots or create video media.
- overwolf.utils - Get system information, open URLs, and send keystrokes to a game.
- overwolf.web - API to open a local HTTP web-server.

Since Overwolf runs by detecting a game instance from specific launchers, it may not be possible to integrate it into a MineRL gym environment. However, it can be linked to a user's Minecraft client, which is able to join MineRL gym environments in certain versions of MineRL [6].

How does MineRL interface with Minecraft? How much time is required in training a model? What hyperparameters and optimizations are necessary to maximize the efficiency of training a model?

MineRL interfaces with Minecraft directly using an OpenAI Gym environment. MineRL itself provides two spaces where it interfaces with the game: an Observation Space and an Action Space. The Observation Space takes in certain metrics about the current game state (such as an RGB image of the game, angle, and inventory), and the Action Space is a set of different actions (attacking, moving, changing slots, etc.) that the AI can take in response. Depending on the version of MineRL used, the possible interactions between the game and MineRL may be more or less diverse. The time to train a model is highly dependent on the context of the training, and the specific model itself. MineRL yearly competitions give teams a very constrained time to train their model (4 days of training time), but researchers using MineRL may spend much longer training their models. The Align-RUDDER project, for example, had a total wall-clock time on 128 CPU cores of 10,360 hours, or around 14 months of training [7]. MineRL itself does not specify a single model that is allowed to be used, so the hyperparameters needed will vary depending on model choice. Optimization choices will also vary depending on model choice, but one general optimization method that might be applicable to multiple model choices is Proximal Policy Optimization [8], an optimization method that mixes environmental data collection with training a gradient for a specified objective. This has been used for previous MineRL projects, such the Align-RUDDER project in the reference papers within the MineRL documentation [7].

Overwolf and MineRL are built on different languages (Javascript/Typescript, and Python respectively), so how can cross-language communication be achieved? What architecture components are necessary to realize it?

- Python Subprocess
    - Run JS scripts from Python using a subprocess and perform actions over stdin / stdout.
- Message Broker / Local Server-Client Communication
    - Python or JS code listens to the broker or starts a listening server, while the other sends messages.
- Overwolf.io / File Updates
    - Overwolf API that offers file I/O and options to monitor files for changes.
- Removing Overwolf
    - Various libraries exist for Python for generating a GUI, which is the main function Overwolf fulfills.

What are the advantages and disadvantages that different Minecraft ML software stacks have?

Only two of the three common solutions (MineRL / Corporate / Self-Developed) to Minecraft ML listed in this section are realistically attainable for our project, as we lack the hardware resources and number of developers needed to use software comparable to GPT-4, as an example, for our project. This comparison will then cover software stacks that use MineRL against independently developed solutions.

- MineRL-based software stacks
    - Advantages:
        - Pre-built infrastructure, can be trained almost immediately
        - Well-documented, used by academics
        - Existing pool of training data and objectives
    - Disadvantages:
        - Only available for older versions of Minecraft
            - MineRL 1.0 runs on Minecraft 1.16.5
                - More than 2 ½ years old and 2 years out of date
            - 0.4.4 and 0.37 on Minecraft 1.11.2
                - More than 6 ½ years old and 6 years out of date
        - Built for a competition setting
            - MineRL's features may not align with some goals
- Independently-developed software stacks
    - Advantages:
        - Self-developed
            - only assets needed are those that align with our goals
        - Customizable
            - May be able to support a learning style MineRL can't
    - Disadvantages:

- Front-loads project with software development
    - Poppro’s software stack took 2 years to develop
- Procuring large datasets would be a requirement
- May exceed the scope of a senior design project

References

[1] B. Baker et al., "Learning to play Minecraft with Video PreTraining," *OpenAI*, https://openai.com/research/vpt (accessed Sep. 21, 2023).

[2] Horrocks, Nathan, "Building Generally Capable AI Agents with MineDojo," *Nvidia Developer*, https://developer.nvidia.com/blog/building-generally-capable-ai-agents-with-minedojo/ (accessed Sep. 21, 2023).

[3] Poppro, "My Computer Taught Itself to Play Minecraft", *YouTube*, https://www.youtube.com/watch?v=IQ7sK6PezJo (accessed Sep. 21, 2023).

[4] Overwolf, "Minecraft Game Events", *Overwolf*, https://overwolf.github.io/api/live-game-data/supported-games/minecraft (accessed Sep. 22, 2023).

[5] Overwolf, "Overwolf API Overview," *Overwolf*, https://overwolf.github.io/api (accessed Sep. 22, 2023).

[6] Overwolf, "Game IDs", *Overwolf*, https://overwolf.github.io/api/games/ids (accessed Sep. 22, 2023).

[7] V. Patil et al., "Align-RUDDER: Learning From New Demonstrations by Reward Redistribution," *arxiv.org*, https://arxiv.org/abs/2009.14108 (accessed Sep. 21, 2023).

[8] J. Shculman et al., "Proximal Policy Optimization Algorithms," *arxiv.org*, https://arxiv.org/abs/1707.06347 (accessed Sep. 21, 2023).