Project Summary

The United Unity Universe (U3) is a framework developed for building and iterating on reinforcement learning environments in Unity 3D, utilizing both Unity and Python. Informed by the DeepMind paper, "Using Unity to Help Solve Intelligence," the design choices of U3 are influenced by the conventions presented therein. With a target audience of machine learning researchers, U3 offers interfaces to facilitate the efficient integration of diverse code for the purpose of implementing new environments and designing experiments to test the boundaries of an agent's capabilities. U3 also includes pre-defined environments such as GridWorld and OpenXLand, and further encourages researchers to contribute their own custom environments or modifications to the existing codebase.

The first goal for the project is to recapitulate XLand and Ada. This step will serve as a benchmark for making sure everything is running smoothly. Further, having a working version of Ada will allow us to test its capabilities. Specifically, I am interested in using the Ada model as a foundational model in novel tasks, and testing its zero or few shot capabilities. For example, can Ada adapt to a reskinned environment (just changing textures/models)? Can it adapt to a new set of rules/objects? Can it be used as a foundational model on new rules/objects? Can it then adapt to more complex tasks that require a combination of rules (Like using an object to build a bridge across a gap)? Finally, can Ada be used as a foundational model for a completely different environment as long as the basic controller is the same (new tasks, new visuals, new rewards)?

Moving past XLand, we'll want to develop a novel environment for training agents. XLand is somewhat limited by two main factors. The first is that the design of the environment is very artificial, casting doubt on how well knowledge gained in it will be able to transfer to real world tasks downstream. The second is that the "adaptation" exhibited by Ada is primarily in the form of solving novel rule combinations (with the exception of the "Push, don't lift" task). If we find that Ada is unable to adapt outside of this rule-based regime (as defined by XLand) that will be a serious hindrance to any downstream tasks we might want to train the agent on. The worst case scenario is that Ada's "adaptation" is not really any sort of adaptation, but instead just a brute force search through possible XLand rulesets. One possible way to address these limitations would be to train the agent on many different environments (XLand, MineDojo, MiniGrid, HabitatAI, Avalon, etc). However, this would require us to set up many independent environments each with its own implementation and API. U3 can help alleviate this issue by allowing us to quickly develop new environments within a single, unified, framework. With the exception of MineDojo, most existing RL environments are simple enough to be easily implemented in Unity (once the precursor framework is in place). Having all implementations be in the Unity framework will allow for rapid iteration on the environments as problems arise. We will also be able to easily experiment with the trained agent using a standardized API. Finally, researchers have shown that multiagent training is a powerful approach to smoothly varying the difficulty of a task. U3 provides multi agent support, unlike many of the other environments mentioned above.

I would also like to design a novel environment. One limitation of simulated environments is that they are abstractions of the real world. MineDojo uses voxel based visuals, MiniGrid uses abstract shapes. It would be useful for real-world downstream tasks to train an agent on more ethologically relevant visual scenes. To this end, I propose an environment designed to mimic natural landscapes and natural tasks. The full details are outside the scope of this document, but the general idea would be to have an island with realistic graphics and possibly distinct biomes (forest, grassland, beach, etc), on which the agent is tasked to survive as long as possible. In contrast to XLand with its artificial rule sets this environment would require the agent to explore the island to find food and water. The environment would have many ways to obtain food and water and these would differ depending on the biome. The differing strategies would naturally lead to ecological niches for agents to exploit. By training a zoo of agents we can populate the island with ever stronger agents, and as more agents enter the environment more niches will be filled. Eventually, the agents would need to learn to adapt to the actions of other agents and seek out food and water in diverse settings as created by the actions of the other agents. This would in turn drive the agent to develop adaptive and generalist strategies. Importantly, the goal of the multi-agents in this environment is not to create direct competition, but instead to have a way to smoothly increase the difficulty of surviving on the island.

Desiderata

1. *Dynamic environment setup and probing*: ML researchers need the ability to customize and probe environments to evaluate task difficulty and select environments appropriate for the current policy. U3 enables complete environmental customization from the python side before starting an episode, allowing researchers to find or create an appropriate environment instance for training.
2. *Dynamic environment loading*: Saving and loading environments during testing phases or environmental setup can be useful for debugging and improving models. The U3 framework facilitates easy serialization of environments, with a focus on interpretability rather than speed. This framework also facilitates experimentation for testing how the trained agent reacts to certain situations in the environment.
3. *Multiple independent environments in a single Unity instance*: Running many instances of environments in the same Unity process reduces the cost of the Unity engine overhead. Our framework also allows for a single Unity instance to run distinct environments (such as a Gridworld and a 3D environment) simultaneously.
4. *Multiple agents in a single environment*: Multi-agent environments are useful both for multi-agent research but also for modifying task difficulty. U3 supports multiple agents in a single environment using the petting-zoo interface. The decision interval for each agent is not fixed, allowing for greater flexibility in training models.
5. *Python interface using Docker, gym/petting zoo, and a simple API for environmental manipulations*: U3 is designed to be a community tool, and as such, it uses existing standards for all interfaces into the framework. The environmental API is environment-specific, but the basic functionality such as serialization is done through a

standardized JSON format. Docker is used for the Unity instances to increase reproducibility and facilitate easier setup.

6. *Modular code design*: To make U3 accessible to as many researchers as possible, the framework encourages modular code using Unity components. U3 comes with several basic environments already defined, but users can create new objects and add new features to those objects using components. This means that community code can be mixed and matched to fit the unique requirements of each project with minimal coding.
7. *Tools for environmental initialization*: The ability to randomize environment layouts is important for creating diverse and challenging environments for training models. U3 provides a number of tools for environmental initialization, such as wave function collapse and compositional pattern-producing networks.
8. *Tools for human experiments:* In order to compare models to human-level baseline, or to gather expert trajectories from human players U3 provides a simple human interface both as a standalone application and as a web plugin.

Code Structure (WIP)

U3 is comprised of a collection of C# classes that provide basic functionality and callbacks at various points in the environment life cycle. The framework is managed by EnvironmentManager which deals with the python communication and creating/updating the parallel environment instances running in a single Unity instance.

To modify an existing environment you will define objects using EnvironmentObject and give them functionality using EnvironmentComponent and built-in Unity classes (such as Rigidbodies for physics).

To create new environments you can extend an existing EnvironmentEngine and make the necessary changes.

EnvironmentComponent: Basic unit of logic in U3. This defines a modular piece of code and variables that can give an object a particular function. For example, in the 3D environment there is a “Jump” component that endows an Agent with the ability to jump.

Serialization: EnvironmentComponent provides the base serialization functionality. By default, all Properties (those with both set and get functions) defined in the class are serialized. The set function should be coded in such a way to ensure that the state of the component is valid after being called. For example, if you need to modify an underlying Unity object (such as a Rigidbody) that logic needs to be taken care of by the set function. You can force a Property to be ignored by the serializer by adding the [NotSaved] attribute.

EnvironmentComponent also provides two callback functions to deal with serialization, OnPreGetState and OnPostLoadState, which are called before and after the serialization steps respectively. These can be used if more advanced serialization is required.

Callbacks: The following callbacks are common to all U3 scripts and can be used to inject functionality into the environment.

OnRunStarted() and OnRunEnded(): Called at the start and end of the environment's lifecycle. Note that this is based on the environment itself, unlike OnEpisodeStarted() and OnEpisodeEnded(), which are controlled by EnvironmentTask.

OnStepStarted() and OnStepEnded(): Called at the start and end of the python step. This is only called when at least one agent in the environment is making a decision request to python.

OnUpdate() and OnLateUpdate(): Called each frame. Can be used for non-simulation updates, such as graphics and animations. In training settings isn't best to use OnFixedUpdate instead.

OnFixedUpdate() and OnLateFixedUpdate(): Called each physics step. Any simulation-relevant updates should occur here. OnLateFixedUpdate is called after the physics step occurs.

EnvironmentObject: The basic unit of function in U3. This defines a single object in the environment, and is a collection of several different EnvironmentComponents that define the possible functions of that object.

EnvironmentEngine: The global workspace of U3. Here you can deal with interactions between objects, and overall logic that wouldn't make sense at the object level.

StartRun()
EndRun()

EnvironmentAgent: Gives an object a trainable set of instructions. Defines observations and actions that the object can take.

OnEpisodeStart()
OnEpisodeEnd()

EnvironmentSensor: Defines the types of observations available to an Agent. Each agent can have several sensors.

Discrete, Continuous or Camera
TODO: Add logic to add sensors programmatically.

EnvironmentTask: Defines the reward schedule of an Agent. Each agent must have 1 task.

AddReward()
EpisodeStart()
EpisodeEnd()
TODO - Add multidimensional rewards

Callback timeline

This section describes the timeline of callbacks into the various U3 elements listed in the previous section, along with important Unity events such as physics updates and python communication.

Environment lifetime:

```
InitializeEnvironment

Environment run:
OnRunStarted

    Python step:
    OnStepStarted
        (If IsEndOfTurn -> OnEndTurn in GridEnvironment)
        -> ShouldRequestDecision
        -> CheckDecisions
    OnDecisionRequested
    OnActionReceived (Repeat while blocking)
        -> ShouldBlockDecision
    OnStepEnded

    Environment step:
    OnUpdate
        OnStartTurn (GridEnvironment only)
            OnFixedUpdate
                -> Physics update
            OnLateFixedUpdate
        OnEndTurn (GridEnvironment only)
    OnLateUpdate

OnRunEnded
```

Graveyard

Project Summary

Desiderata

The objective of U3 is to provide ML researchers with an easy to use framework for quickly building and iterating on environments (3D, 2D or other) in Unity 3D. Much of the design choices have been influenced by the DeepMind paper: Using Unity to Help Solve Intelligence. We adapt their nomenclature and conventions when applicable. The default ML Agents library provided by Unity is targeted for users that wish to add ML to their games, and thus differs from U3 in some key aspects:

Dynamic environment setup and probing. One of the key requirements of modern day environments is the ability to probe randomly seeded environments for task difficulty given the current policy. This can be done either by explicitly testing a policy on the environment seeds, or by estimating task difficulty using environmental complexity metrics. U3 is designed to allow complete environmental customization from the python side before starting an episode, allowing an appropriate environment instance to be found or created before training.

Dynamic environment loading. The U3 framework is designed to facilitate easy serialization of the environment for saving and loading. The serialization is optimized for interpretability, not speed. Thus, we suggest its use only during testing phases, or during environmental setup, not during training.

Multiple independent environments in a single Unity instance. In order to reduce the cost of the Unity engine overhead we support running many instances of environments in the same Unity process. These environments can even be entirely different environments (such as XLand or GridWorld), as each runs with its own independent physics engine.

Multiple agents in a single environment. Multi-agent environments are also supported by simply adding a new agent object to the environment. The policy of these agents can be determined by the python side (training, or using a fixed policy). The decision interval for an agent is not fixed, and U3 can accommodate multiple agents with different decision intervals.

Python interface using Docker, gym/petting zoo and a simple API for environmental manipulations. Ultimately U3 is designed to be a community tool. The goal of the framework is to facilitate rapid prototyping and iteration. Thus, all interfaces into the framework use existing standards. The environmental API is necessarily environment specific, but the basic functionality such as serialization is done through a standardized JSON format. Docker is used for the Unity instances to increase reproducibility and facilitate easier setup.

Modular code design. We want content developed in U3 to be as accessible as possible. To this end, the framework encourages modular code using Unity components. U3 comes with several basic environments already defined, such as a simple first person 3D environment, and a top down gridworld environment. To make modifications to these environments users can create new objects and add new features to those objects using components. Components are designed not to be self-contained, such that they can be added and removed from an object without any changes to the code. This means that community code can be mixed and matched to fit the unique requirements of your project with minimal coding!

Tools for environmental initialization. We provide a number of tools for randomizing environment layouts, such as wave function collapse and compositional pattern-producing networks.

Code Structure

Callback timeline

Structures

EnvironmentComponent: Basic unit of logic in U3. This defines a modular piece of code and variables that can give an object a particular function.

- OnRunStart()
- OnRunEnd()
- OnCollision()
- DoStepActions()
- DoAction()
- Set Properties

EnvironmentObject: The basic unit of function in U3. This defines a single object in the environment, and is a collection of several different EnvironmentComponents that define the possible functions of that object.

EnvironmentEngine: The global workspace of U3. Here you can deal with interactions between objects, and overall logic that wouldn't make sense at the object level.

- StartRun()
- EndRun()

EnvironmentAgent: Gives an object a trainable set of instructions. Defines observations and actions that the object can take.

- OnEpisodeStart()
- OnEpisodeEnd()

EnvironmentSensor: Defines the types of observations available to an Agent. Each agent can have several sensors.

- Discrete, Continuous or Camera
- TODO: Add logic to add sensors programmatically.

EnvironmentTask: Defines the reward schedule of an Agent. Each agent must have 1 task.

- AddReward()
- OnEpisodeStart()
- OnEpisodeEnd()
- TODO - Add multidimensional rewards

Callbacks:

Environment lifetime:

InitializeEnvironment

Environment run:

```
OnRunStarted

    Python step:
    OnStepStarted
        (If IsEndOfTurn -> OnEndTurn in GridEnvironment)
        -> ShouldRequestDecision
        -> CheckDecisions
    OnDecisionRequested
    OnActionReceived (Repeat while blocking)
        -> ShouldBlockDecision
    OnStepEnded

    Environment step:
    OnUpdate
        OnStartTurn
            OnFixedUpdate
                -> Physics
            OnLateFixedUpdate
        OnEndTurn
    OnLateUpdate

OnRunEnded
```

Timeline

Run Python
Create Unity instance through UnityEnvironment in Python

  Pass in environment parameters via JSON string

  If AutoStartEpisode is false, then the environment will idle until **InitializeEnvironment** is called.

While idling:

  Can call functions to query the statistics of different instantiations of the environment.

    Use this to perform data selection.

InitializeEnvironment (JSON params): sets the initialization of the environment, and of JSON properties to initialize.

  Calls **EngineComponent.Initialize** on all components.

  Python can query and manipulate the environment during this step

  Standard environment manipulations through **SetProperty** and **DoAction** are enabled.

StartRun: Begins the episode to start training.

  Calls **EngineComponent.OnRunStarted** before first Step().

    Also makes a snapshot of the default value of the environment before first Step().

    Note that a Environment Run can consist of many Agent Episodes.

Academy.Step():

  ShouldRequestDecision() Called on each Agent in the environment.

  If true, call RequestDecision()

    This triggers **CollectObservations()**

  Once all observations are collected, send to Python and get action

  OnActionReceived() Called on each Agent

  Once all Agents have acted, call **DoStepActions()** for all other EnvironmentObjects.

EnvironmentComponent.DoStepActions(): The behavior of a single object over one time step.

  Note that this logic can be done on an object-by-object level by overriding the Object's function, or on a global level by overriding the EnvironmentEngine's function.

  Logic here should be restricted to calls to **EngineComponent.DoAction**, and **SetProperty**. This ensures that the environment stays self-consistent, and saveable/loadable.

  Python injections to the environment will be processed here, before the normal calls to **DoAction**

EndRun(): Closes the current episode, returning the environment to an Idle state.

  Calls **EngineComponent.OnRunEnded()** on all components

New objects can be created during initialization or runtime by **EnviornmentEngine.CreateObject(JSON params)**.

  EngineComponent.Initialize(JSON params) is called first on the newly created Object, then **EngineComponent.OnCreated(JSON params)** is called before the first environmental step.

The Object is returned inline, and is expected to be set up and in a valid state after these two functions are called.

EnvironmentEngine.SaveEnvironmentState() creates a JSON representation of the current state of the environment.

If **EnviornmentEngine.UseInitialState** is enabled then this save will be a diff from the initial state, saving space and avoiding redundant information (at the expense of processing overhead).

It is suggested that you do not use this setting in training environments.

EnvironmentEngine.LoadEnvironmentState(JSON environment) reloads the environment from a previous save state.

EngineComponent.DoAction(????)

Note that Action values should be restricted to basic types and classes deriving from EnvironmentComponents.

EngineComponent.SetProperty Properties are public and capitalized. They must be set up in such a way as to always leave the object in a valid state. (i.e. they will update any internal Unity classes as required)

Note that Properties should be restricted to basic types and classes deriving from EnvironmentComponents.