Project Report

on

"Multi Agent Reinforcement Learning Simulations"

Submitted By:

Malam Vivek Kanji
[IU2254100006]

In partial fulfillment of the requirements for the degree of
Master of Science in Information Technology.

Department of Computer Science,
Indus Institute of
Information and Communication Technology (IIICT),
Ahmedabad, Gujarat.

January - April 2024

ज्ञानेन प्रकाशते जगत्
INDUS UNIVERSITY

CERTIFICATE

This is to certify that the project titled "Multi-Agent Reinforcement Learning Simulations" is the bonafide work carried out by **Malam Vivek Kanji [IU2254100006]**, a student of M.Sc-IT Semester - IV, during the academic year 2023-24, in partial fulfillment of the requirements for the award of the degree of Master of Science in Information Technology at Indus Institute of Information and Communication Technology (IIICT), Department of Computer Science, Ahmedabad, Gujarat.

.....

Dr. Akshara Dave
[Internal Guide]

.....

Bhavana Hotchandani
[Head of Department]

COMPANY PROFILE

Macrobian Games is one of the few organisations across the globe who has been actively involved in Gamification of Corporate Training and Education.

Company Statement:

Macrobian Games, a leading Unity 3D game development company in India, specializes in delivering high-quality games poised to disrupt the industry. With an average of 8 years of experience, our seasoned team offers comprehensive services to ideate, develop, and launch successful games within client budgets. Leveraging deep expertise in corporate training and gaming, we excel in creating engaging simulation-based experiences. Pioneers in technology adoption, we've crafted some of the first Augmented and Virtual Reality games. Since 2008, our games have garnered over 1M downloads worldwide. We provide staffing solutions and dedicated developers for prototype development and ongoing support. Specializing in gamification, hardcore, and casual games across platforms, we bring ideas to life affordably. Renowned for our quick responses and transparent communication, we're trusted by clients for efficient service delivery.

Our Proud Associates:

803 Landmark Building, 100 Feet Rd, Jodhpur Tekra, Ahmedabad 380015

info@macrobiangames.com

MACROBIAN GAMES

Date : April 5, 2024
Place : Ahmedabad, Gujarat.

PROJECT COMPLETION CERTIFICATE

To Whom It May Concern,

This is to certify that Mr. Malam Vivek Kanji has successfully completed the project under Macrobian Games Private Limited, Ahmedabad, as partial fulfillment of MSc IT (Master of Science in Information Technology) at Indus Institute of Information and Communication Technology (IIICT), affiliated with Indus University in Ahmedabad, Gujarat. The duration of the project was from January to April 2024.

He undertook the project titled 'Multi-Agent Reinforcement Learning Simulation,' where he implemented all the simulations and contributed significantly to the field of autonomous agents. The project was developed using Unity, C#, ML-Agents Toolkit, Python, PyTorch, TensorFlow, Tensorboard, Blender, Flask, and WebGL. We are satisfied with the work he has done and the efforts he has put into it.

Throughout the project duration, he has been found to be diligent, focused, and sincere towards his work schedule. His progress during the project work has been appreciable.

He has successfully completed the project work he undertook independently. We wish him success in his future endeavors.

Appreciating you,

Macrobian Games Pvt. Ltd.

Director

Amit Prajapati
[CEO | Macrobian Games]

Macrobian Games Private Limited
803 Landmark Building, 100 Feet Rd,
Jodhpur Tekra, Ahmedabad 380015
info@macrobiangames.com

THE AUTHOR

Researcher &
Data Scientist

Hello, I'm **Vivek Malam**, a Computer Science student from Indus Institute of Information and Communication Technology (IIICT), Department of Computer Science, Ahmedabad, Gujarat, specializing in Python programming and passionate about machine learning. I actively develop models, exploring the fascinating applications of algorithms. I've also dived into LangChain (LLM) and generative AI, expanding my understanding of Natural Language Processing.

Additionally, I've ventured into Unity game development, creating engaging 3D multiplayer experiences with AR/VR and XR Integration. I'm particularly interested in Meta XR, Spatial Computing and Simulation-based Agents, seeking to leverage these technologies to craft immersive and interactive virtual environments.

As a data enthusiast, I enjoy analyzing complex datasets to extract meaningful insights. I'm a dedicated researcher, constantly exploring new ideas to contribute to knowledge advancement. With my diverse skill set, I seek new challenges to apply my expertise, utilizing technology like machine learning and AI to solve real-world problems and create positive change.

I have also published research papers in international journals in the fields of NLP, Machine Learning, Blockchain, and more. I believe that research is my strength. I can collect and represent the most accurate data with expertise, thereby improving the capabilities of solving problems from different unique perspectives.

Contact Details:

Gmail : viv3k.19@gmail.com

Linked In : www.linkedin.com/in/viv3k19

GitHub : https://github.com/viv3k19

CANDIDATE DECLARATION

I, **Vivek Malam**, solemnly affirm that the research and development presented in this Multi-Agent Reinforcement Learning Simulations (MARL Simulations) represent my original work. This project is submitted in fulfillment of the requirements for the completion of Semester 4, Software Development Project, as part of my pursuit of a Master of Science degree in Information Technology.

The project was conducted under the supervision and guidance of Dr. Akshara Dave, Assistant Professor at Indus University, serving as the Internal Guide. Additionally, I received invaluable insights and support from Amit Prajapati, CEO of Macrobian Games, who served as the External Guide throughout the duration of this project.

I assert that all aspects of this project comply with academic integrity standards and have not been plagiarized or misrepresented in any form. Any external sources referenced in this work have been duly acknowledged and cited.

I understand the significance of academic honesty and take full responsibility for the content presented herein. Furthermore, I am prepared to defend the methodologies, findings, and conclusions outlined in this project during any evaluation or examination process.

Malam Vivek Kanji
[IU2254100006]

ACKNOWLEDGEMENTS

I would like to extend my heartfelt gratitude to all those who have contributed to the success of the Multi-Agent Reinforcement Learning Simulation Project. Without their support, guidance, and encouragement, this project would not have been possible.

First and foremost, I am immensely grateful to my Internal Guide, Dr. Akshara Dave, Assistant Professor at Indus University, whose expertise, guidance, and unwavering support helped shape the perfect outline for my project. Her valuable insights and constructive feedback were instrumental in steering the project in the right direction.

I would also like to express my sincere appreciation to my External Guide, Mr. Amit Prajapati, CEO of Macrobian Games, Ahmedabad. I am deeply thankful for his interest in my project and for providing approval to conduct the project under Macrobian Games. His mentorship and industry insights were invaluable throughout the project journey.

Special acknowledgements are due to the IIICT Department of Computer Science for providing us with the wonderful opportunity to submit this project work for the partial fulfillment of Masters of Science in Information Technology. Their support and resources significantly contributed to the successful completion of this endeavor.

I am extremely grateful to all individuals who have directly or indirectly contributed in various capacities, including providing resources, assisting in research, conducting literature surveys, offering valuable suggestions, correcting errors, and participating in testing. Your collective efforts have played a crucial role in ensuring the perfection of this work.

LIST OF FIGURES

LIST OF DIAGRAMS

INDEX

INTRODUCTION

In the domain of machine learning (ML), reinforcement learning (RL) has attracted considerable attention from the scientific community due to its ability to address a wide range of tasks using a simple architecture, without requiring prior knowledge of problem dynamics. RL has found applications in various domains such as finance, robotics, natural language processing, and telecommunications. Central to RL systems is the agent, which operates within an environment that models the task it seeks to accomplish. Across these applications, RL agents interact with the environment through trial and error, receiving rewards (reinforcements) for their actions. This mechanism, reminiscent of human learning, guides agents towards improving their future decisions to optimize future rewards. However, many real-world challenges cannot be adequately addressed by a single active agent interacting with the environment; the solution lies in multi-agent systems (MAS), where multiple agents learn concurrently to solve a task by interacting with the same environment.

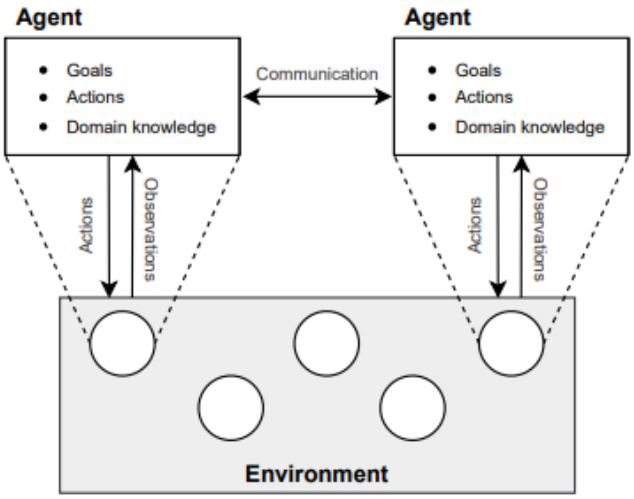
MASs serve various purposes in fields such as traffic control, network packet routing, energy distribution, robotic systems, economic modeling, and analysis of social dilemmas. Consequently, researchers have been striving to extend existing single-agent RL algorithms to multi-agent approaches in recent years. However, empirical evaluations have demonstrated that directly applying single-agent RL to multiple agents cannot lead to optimal solutions because the environment is no longer static from each agent's perspective. An action by one agent can result in different rewards based on the actions of other agents. This challenge, known as the non-stationarity of the environment, is the primary obstacle to overcome in developing efficient multi-agent reinforcement learning (MARL) algorithms.

Even after achieving convergence, such algorithms typically exhibit satisfactory performance in terms of policy quality and convergence speed only when a limited number of agents are involved. Therefore, scalability to a large number of agents is a crucial consideration when designing algorithms applicable to real-world problems.

In this documentation, we provide an introduction to multi-agent reinforcement learning, focusing on outlining the framework, environment models, and adapting relevant methodologies for reinforcement learning techniques for multi-agent settings.

Here, we present various MARL algorithms addressing the challenges of non-stationarity and scalability mentioned earlier. We also explore partially observable environments, which are more common in MAS than in single-agent settings and are essential for developing algorithms applicable to real-world problems. Additionally, we offer an overview of common benchmarking environments used to evaluate RL algorithm performances. This survey serves as a foundational guide to multi-agent reinforcement learning, discussing the principal challenges and solutions in the field, and describing typical MARL applications. Although research into MARL is still in its early stages and often lacks theoretical foundations, it has demonstrated promising advancements in its applications. It could be considered a novel approach to achieving systems capable of assisting humans in complex tasks, such as operating in hazardous environments, and demonstrating general artificial intelligence.

Imagine a scenario where autonomous agents with independent decision-making capabilities interact in a shared environment to achieve specific objectives, they may share common goals (e.g., a fleet of robots in a warehouse) or have conflicting ones (e.g., trading agents in a virtual market), autonomously discerning optimal interactions over time, acquiring skills, coordinating actions, and possibly developing a shared language to efficiently achieve their goals.



[Figure.1: agent-to-environment behavior]

Multi-agent reinforcement learning (MARL) algorithms aim to learn optimal policies for a group of agents within a multi-agent system. Analogous to their single-agent counterparts, these policies are learned through a trial-and-error process to maximize the agents' cumulative rewards or returns. We delineate a basic schematic of the MARL training loop: a set of n agents selects individual actions, collectively constituting the joint action. This joint action alters the environment's state according to its dynamics, with agents receiving individual rewards and observations about the new state. This iterative process persists until a terminal criterion is met (such as one agent winning a chess game) or indefinitely. A complete iteration of this loop from the initial to terminal state is termed an episode. Data generated from multiple independent episodes, comprising observations, actions, and rewards, are utilized to iteratively refine the agents' policies.

“Factorization” has been widely studied in the realm of multi-agent reinforcement learning. This class of learning problems is complicated due to the large combined action and observation spaces. In the fully centralized and decentralized approaches, we find the problem of spurious rewards and a phenomenon we call the “lazy agent” problem, which arises due to partial observability.

To prevent the “lazy agents” from receiving high rewards even though they did nothing, this paper proposed a novel network called the Value-Decomposition Network, which learns to decompose the team value function into agent-wise value functions.

This network aims to learn an optimal linear value decomposition from the team reward signal by back-propagating the total Q gradient through deep neural networks representing the individual component value functions. This additive value decomposition is specifically motivated by avoiding the spurious reward signals that emerge in purely independent learners. The implicit value function learned by each agent depends only on local observations, and thus is more easily learned.

Safe Multi-agent reinforcement learning (safe MARL) has increasingly gained attention in recent years, emphasizing the need for agents to not only optimize the global return but also adhere to safety requirements through behavioral constraints. Some recent work has integrated control theory with multi-agent reinforcement learning to address the challenge of ensuring safety. However, there have been only very limited applications of Model Predictive Control (MPC) methods in this domain, primarily due to the complex and implicit dynamics characteristic of multi-agent environments. To bridge this gap, we propose a novel method called Deep Learning-Based Model Predictive Control for Safe Multi-Agent Reinforcement Learning (DeepSafeMPC). The key insight of DeepSafeMPC is leveraging a centralized deep learning model to predict environmental dynamics accurately. Our method applies MARL principles to search for optimal solutions. Through the employment of MPC, the actions of agents can be restricted within safe states concurrently. We demonstrate the effectiveness of our approach using the Safe Multi-agent MuJoCo environment, showcasing significant advancements in addressing safety concerns in MARL.

HISTORY OF MARL

The concept of multi-agent systems traces its origins to the early days of distributed artificial intelligence and cognitive science. Over time, this concept has evolved from rudimentary agent interactions to sophisticated multi-agent frameworks capable of addressing intricate real-world problems.

Implementations of MARL in realtime:

Emergent Tool Use From Multi-Agent Autocurricula:

Emergent Tool Use From Multi-Agent Autocurricula has been observed in the field of Multi-Agent Reinforcement Learning (MARL). This system explores how agents engage in multi-agent competition, particularly through the game of hide-and-seek, using standard reinforcement learning algorithms at scale. The result is the emergence of a self-supervised autocurriculum, leading to sophisticated tool use and coordination among the agents involved. The system identifies six distinct phases of agent strategy, each posing new challenges for opposing teams. Notably, agents learn to construct shelters and overcome obstacles using movable boxes and ramps. It demonstrates scalability with increasing environmental complexity, highlighting behavior aligned with human-relevant skills compared to other reinforcement learning methods.

Skill Reinforcement Learning and Planning for Open-World Long-Horizon Tasks:

Skill Reinforcement Learning and Planning for Open-World Long-Horizon Tasks represents an advancement in the realm of MARL. This system addresses the challenge of building multi-task agents in open-world environments without human demonstrations. By breaking down the multi-task learning problem into acquiring basic skills and planning over these skills, it achieves significant improvements in sample efficiency. Utilizing Minecraft as a testbed, the system successfully accomplishes diverse tasks, outperforming baselines by a large margin. It proves to be the most sample-efficient demonstration-free RL method for solving Minecraft Tech Tree tasks.

CERN for AGI: A Framework for Autonomous Simulation-Based Artificial Intelligence Testing:

CERN for AGI: A Framework for Autonomous Simulation-Based Artificial Intelligence Testing and Alignment presents a novel approach to testing and aligning artificial general intelligence (AGI) and Large Language Models (LLMs). It introduces a simulation-based multi-agent system within a virtual reality framework to replicate real-world environments for examining and optimizing AGI. The framework integrates theories from various disciplines, aiming for a more human-aligned and socially responsible AGI. Despite its promising potential, challenges such as unpredictable real-world social dynamics remain.

PADL: Language-Directed Physics-Based Character Control:

PADL: Language-Directed Physics-Based Character Control introduces a system that leverages natural language processing to enable language-directed controllers for physics-based character animation. This system enhances accessibility and versatility by allowing users to specify tasks and skills for characters using natural language commands. It effectively directs simulated humanoid characters to perform diverse motor skills.

Generative Agents: Interactive Simulacra of Human Behavior:

Generative Agents: Interactive Simulacra of Human Behavior presents a system that simulates believable human behavior, empowering interactive applications across various domains. Its architecture extends large language models to enable agents to store and dynamically retrieve experiences, planning behavior based on reflections. The system exhibits believable individual and emergent social behaviors in interactive sandbox environments.

GROOT: Learning to Follow Instructions by Watching Gameplay Videos:

GROOT: Learning to Follow Instructions by Watching Gameplay Videos introduces a system capable of following open-ended instructions in open-world environments by utilizing reference videos as instructions. This approach eliminates the need for text-gameplay annotations. GROOT demonstrates significant success in bridging the human-machine gap and exhibits a high winning rate over baseline agents on a Minecraft SkillForge benchmark.

Solving Rubik's Cube with a Robot Hand:

In a groundbreaking demonstration, OpenAI and collaborators showcase the ability to solve a manipulation problem of unprecedented complexity using models trained solely in simulation and deployed on a real robot. This achievement is enabled by two key components: a novel algorithm called automatic domain randomization (ADR) and a specialized robot platform designed for machine learning applications. ADR automatically generates a distribution over randomized environments of increasing difficulty, facilitating enhanced sim-to-real transfer for control policies and vision state estimators. Notably, control policies trained on an ADR-generated distribution exhibit signs of emergent meta-learning during test time, particularly memory-augmented models. The integration of ADR with the custom robot platform enables the successful solving of a Rubik's Cube using a humanoid robot hand, addressing both control and state estimation challenges inherent in the task.

The Unity Machine Learning Agents Toolkit (ML-Agents Toolkit):

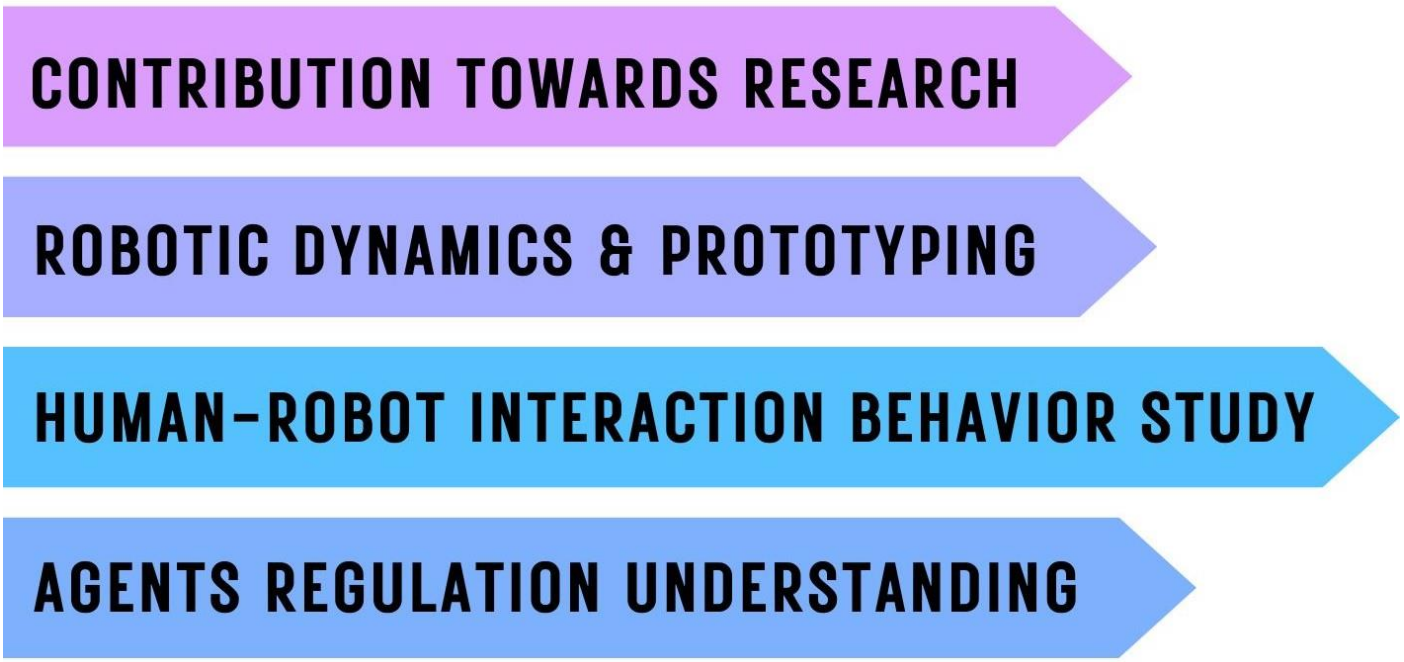
An open-source project enabling games and simulations to train intelligent agents. With a straightforward Python API, developers can use reinforcement learning, imitation learning, and other methods to train agents. Implemented with PyTorch, it offers cutting-edge algorithms for training agents in 2D, 3D, and VR/AR games. These agents can control NPC behavior, automate game testing, and evaluate game design decisions pre-release. The toolkit serves as a central platform for AI evaluation in Unity environments, benefiting both AI researchers and game developers.

BigDog, the Rough-Terrain Quadruped Robot:

BigDog represents a leap forward in robotics, aiming to expand the range of motions and functionalities available to machines by incorporating characteristics found in animals, particularly those with skeletons. Unlike conventional machines, most mobile robots are constructed from hard materials, either by equipping them with treads or wheels for increased mobility or by modeling them after animals, as exemplified by BigDog and similar projects. Despite significant progress in robotics over the past five decades, hard robots still face several limitations. These include mechanical challenges such as instability on difficult terrain, limited ranges of motion due to rigid structures and actuators, and complexities in control, particularly when interacting with soft, delicate, or complex-shaped materials. Moreover, hard robots made from metals tend to be heavy and expensive, limiting their applicability in certain scenarios.

NEED FOR THE MARL

Multi-Agent Reinforcement Learning (MARL) is imperative for various domains due to its contributions across several dimensions:



[Figure.2: why we need marl?]

Research Contribution:
Multi-Agent Reinforcement Learning (MARL) is pivotal for advancing research across domains, modeling decentralized decision-making and emergent behaviors. It informs cooperative/competitive scenarios, illuminating distributed optimization and game-theoretic strategies, bolstering AI's theoretical foundations and understanding of real-world complexities.

Robotic Dynamics & Prototyping:
MARL shapes robotic team dynamics, fostering coordination and communication among agents for effective collaboration on varied tasks. This drives the creation of adaptable robotic systems, supporting autonomous operation or human interaction. MARL expedites prototyping, accelerating design iterations and spurring innovation in robotics R&D.

Human-Robot Interaction Behavior Study:
MARL serves as a framework for studying Human-Robot Interaction (HRI), modeling cooperative/competitive behaviors. Insights gleaned aid in designing socially-aware robots by understanding human perceptions and responses, enhancing interaction protocols and interface designs for better user experience and acceptance.

Agents Regulation Understanding:
MARL provides a platform to explore regulatory mechanisms within multi-agent systems, analyzing policy and incentive impacts on agent decision-making. This informs ethical guidelines and safety protocols for responsible AI deployment, ensuring alignment with societal values and legal standards.

PROJECT DESCRIPTION

Multi-Agent Reinforcement Learning (MARL) is a specialized domain within reinforcement learning that delves into the dynamics and interactions among multiple learning agents operating within a shared environment. Each agent, pursuing its own set of reward objectives, takes actions aimed at advancing its individual interests. As these agents engage with one another, intricate group dynamics emerge, bridging MARL with disciplines such as game theory and multi-agent systems.

The primary objective of developing Multi-Agent Reinforcement Learning (MARL) simulations is to construct, analyze, and cultivate intelligent systems capable of adaptation and interaction within intricate, dynamic environments featuring multiple agents, thereby facilitating the simulation of real-world scenarios.

For this endeavor, I employ the Unity ML-Agents Toolkit, an open-source project integrated with PyTorch, to train intelligent agents for 2D, 3D, and VR/AR games. This toolkit furnishes a Python API tailored for reinforcement learning, serving as a valuable resource for game developers and AI researchers alike by providing a centralized platform for evaluating and disseminating AI advancements within Unity environments.

Project Includes List of Simulation:

- Ants Simulation
- Sheepherding
- Decision Making NPC
- Imitation Learning
- Pathfinding Cars
- MotoGP-Offroad
- Crossy Road Simulation
- Car Parking Simulation
- Football Training
- Volleyball Training
- Fighting AI Simulation
- Aircraft Simulation
- Surveillance Drone Simulation
- SpaceX Rocket Simulation
- Village Simulation
- Military Combat Simulation

The ML-Agents toolkit employs an external Python process to train agents within Unity, communicating with the Unity scene to generate experiences and optimize a neural network for the agent's policy. The resultant output is a TensorFlow model file utilized within Unity as an NN Model (.onnx) to guide agent actions based on learned behaviors, complemented by TensorBoard for visualizing training progress.

Project contributes advancements in addressing a myriad of challenges inherent in multi-agent scenarios, while also positing intriguing avenues for future research. These avenues encompass inquiries into how agents can adapt to the learning behaviors of other agents when their rewards or observations remain unknown, strategies for learning communication protocols within contexts of partial common interest, and considerations for accommodating human agency within the environment.

FRONTEND TOOLS

Unity

Flask
web development,
one drop at a time

HTML CSS JS

BACKEND TOOLS

C#

TensorFlow

PyTorch

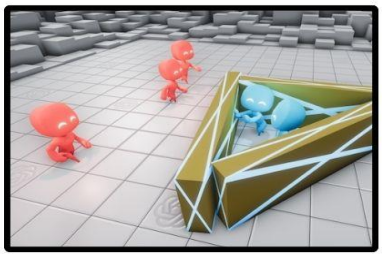
DESIGN TOOLS

Unity
ML-Agents
Toolkit

Unity Asset Store

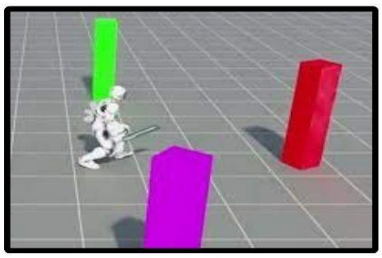
OTHER TOOLS

pixabay

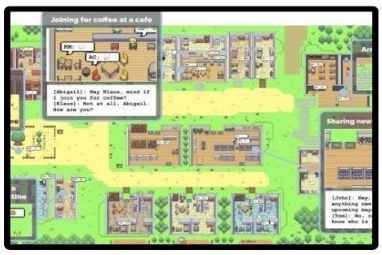


Emergent Tool Use From Multi-Agent Autocurricula

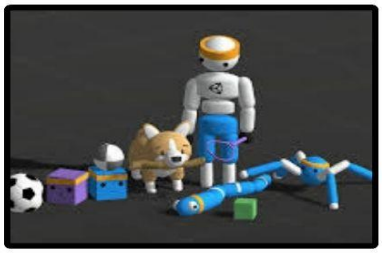
Skill Reinforcement Learning and Planning for Open-World Long-Horizon Tasks



PADL: Language-Directed Physics-Based Character Control

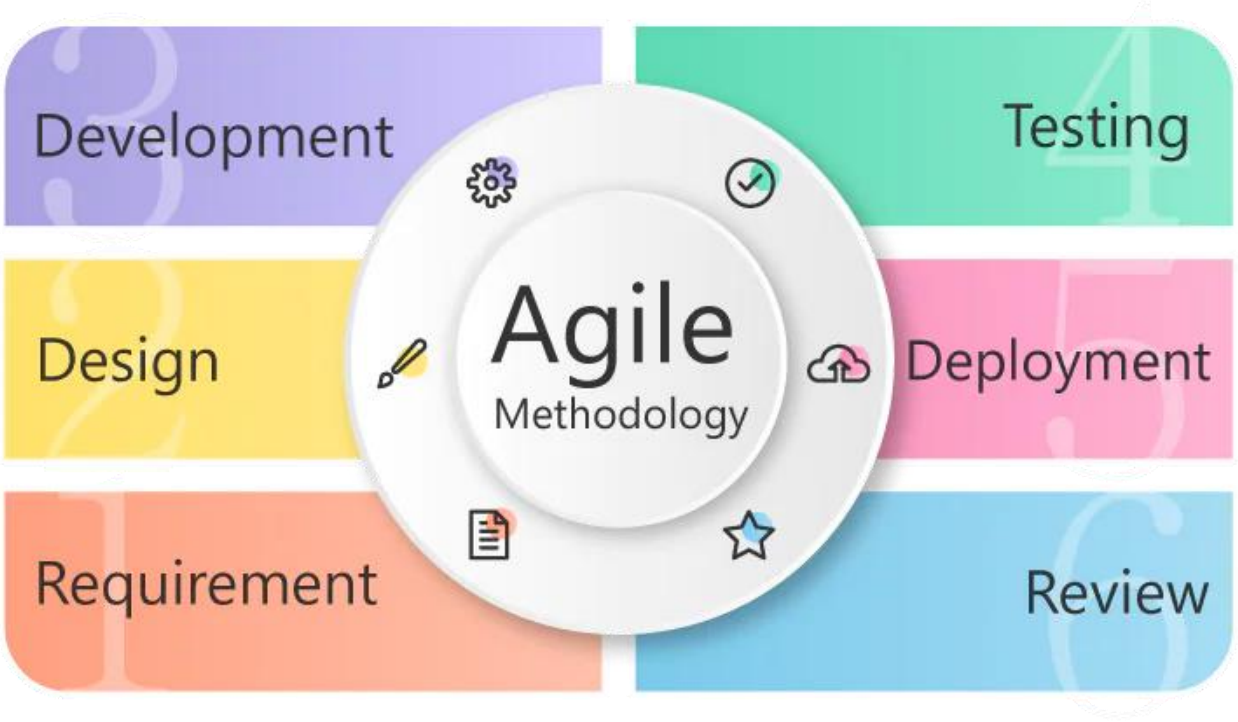


Generative Agents: Interactive Simulacra of Human Behavior



Unity Technologies: mlagents Unity Machine Learning Agents

Note: All literature surveys are subject to existing research conducted on agents, and all outcomes and evaluations are based on personal observations. However, further in-depth discussions are



[Figure.3: agile methodology]

Agile is a software development methodology that emphasizes iterative, incremental, and flexible approaches. It promotes close collaboration between cross-functional teams, rapid delivery of working software, and continuous improvement of the development process. In the context of your project on Multi-Agent Reinforcement Learning simulations, the Agile methodology would be the best fit due to its ability to adapt to changes and deliver high-quality results.

Agile has become a widely adopted approach in the software development industry due to its ability to adapt to change and deliver high-quality products. These characteristics make Agile particularly well-suited for my project on Multi-Agent Reinforcement Learning Simulations, where the need for flexibility, rapid iterations, and continuous improvement are crucial.

In my project, I implemented the Agile methodology with a comprehensive approach. This included defining clear user stories to capture requirements, organizing work into iterative sprints, setting milestones to track progress, utilizing burndown charts to monitor sprint performance, and conducting regular retrospectives for continuous improvement. By integrating these elements, I ensured a structured and collaborative development process, facilitating flexibility, transparency, and adaptability to meet project goals effectively.

SYSTEM REQUIREMENTS

Software Requirements:

- Unity Editor 2021.3.17f ^
- mlagents 2.0.1
- Python 3.9.13(Specific - as on April 2024)
- torch==2.2.0
- torchvision==0.17.0
- torchaudio==2.2.0
- protobuf==3.20.3
- flask==2.0.2
- jinja2==3.0.2
- gunicorn==20.1.0
- python-dotenv==0.19.2
- Flask-Minify==0.37
- Visual Studio 2022^
- Other Unity Package Manager Libraries.

Hardware Requirements:

Minimum Requirements:

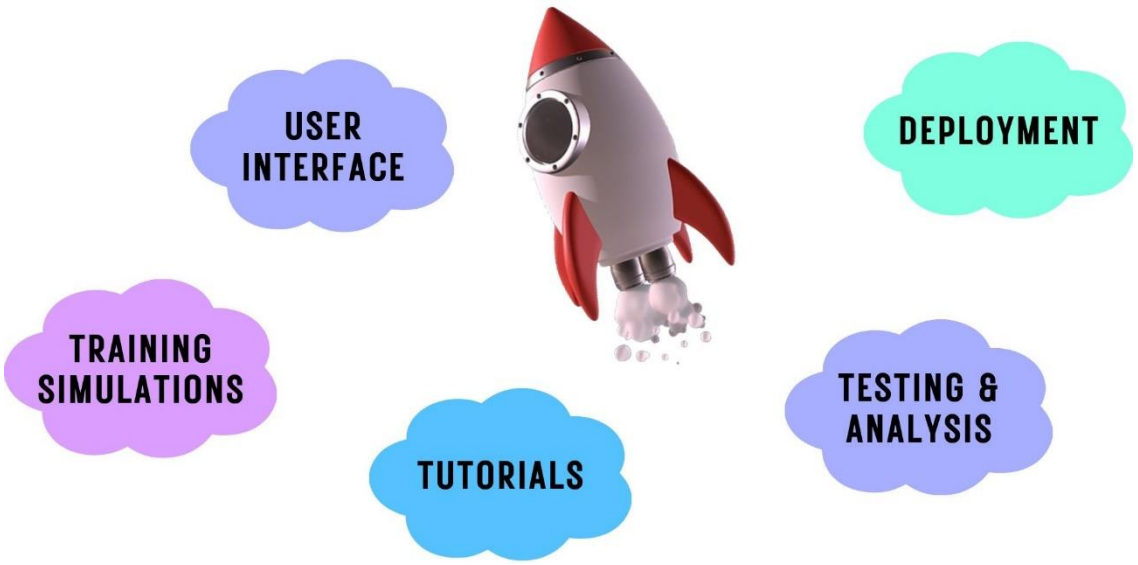
Processor: Intel Core i5 or AMD Ryzen 5 equivalent
RAM: 8GB
Graphics Card: NVIDIA GeForce GTX 1060 or AMD Radeon RX 580
Storage: 256GB SSD
Operating System: Windows 10, Linux, macOS
WebGL: Any Browser (Chrome, Safari, Microsoft Edge,…)

Recommended Requirements:

Processor: Intel Core i7 or AMD Ryzen 7 equivalent
RAM: 16GB or higher
Graphics Card: NVIDIA GeForce RTX 2070 or higher
Storage: 512GB SSD or higher
Operating System: Windows 10, Linux, macOS
WebGL: Any Browser (Chrome, Safari, Microsoft Edge,…)

Note: If running MARL simulations on the web, it requires a faster internet connection for a better experience.

MODULES IN PROJECT



[Figure.4: modules in marl]

Training:

This module facilitates the training of multiple agents through reinforcement learning algorithms, enabling them to learn optimal strategies by interacting with their environment.

Simulations:

Within this module, simulated environments are created to replicate real-world scenarios, providing a platform for agents to interact and learn from their experiences through various multi-agent reinforcement learning techniques.

Tutorials:

Designed to guide users through the intricacies of multi-agent reinforcement learning, this module offers step-by-step instructions and educational resources to assist users in understanding concepts and implementing algorithms effectively.

Testing & Analysis:

This module is dedicated to evaluating the performance of trained agents in various scenarios, conducting analysis to assess their effectiveness and robustness in different environments.

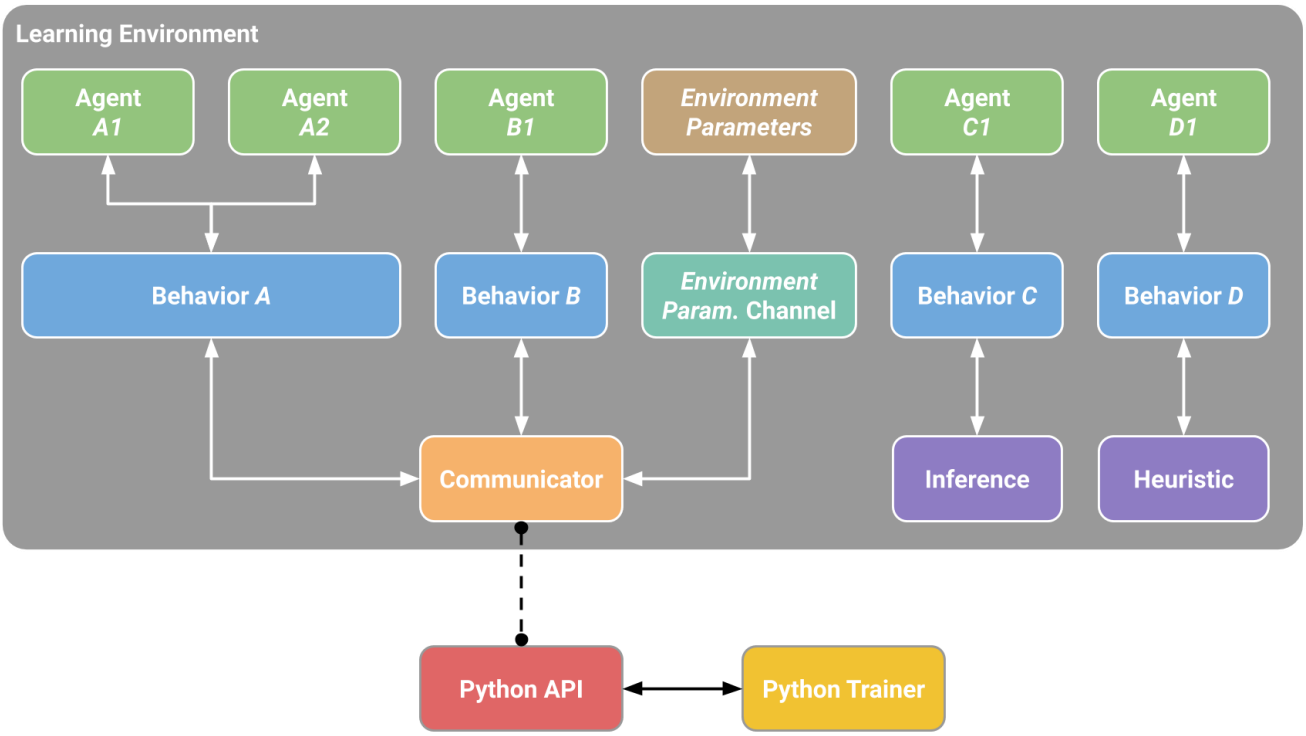
Deployment:

Here, mechanisms are provided to deploy trained models and integrate them into real-world systems or applications, ensuring seamless transition from simulation to practical implementation.

User Interface:

This module offers an intuitive interface for users to interact with the system, facilitating easy configuration, monitoring, and control of simulations and trained agents.

BLOCK DIAGRAM - MARL



[Diagram.1: block diagram for marl-simulations]

Agents: Represent the different entities within the environment, each exhibiting distinct behaviors.

Behaviors: The actions and responses exhibited by the agents, shaped by the environment.

Communicator: Facilitates communication and information exchange between the agents and their behaviors.

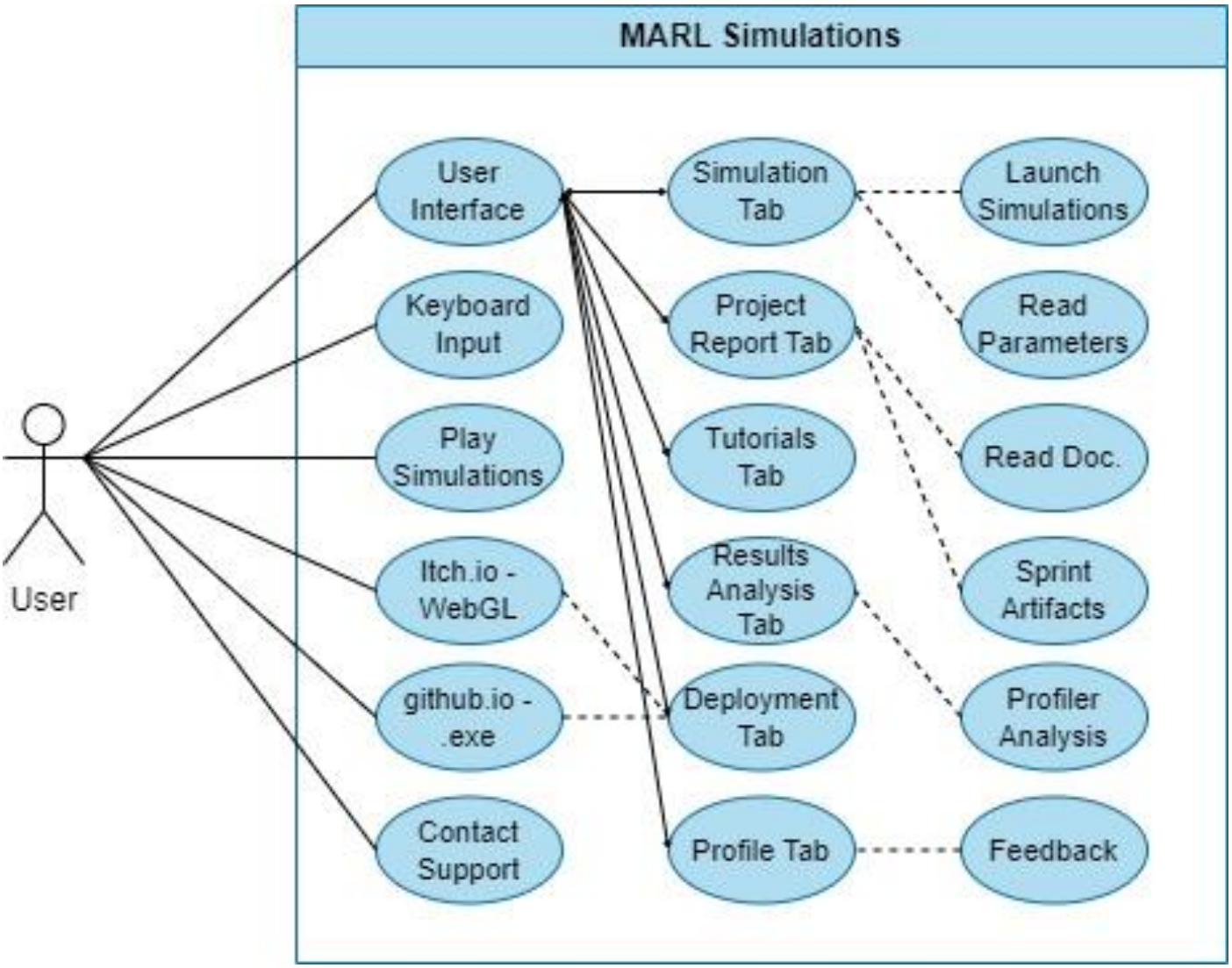
Environment Parameters: Define the characteristics and variables that influence the agents and their behaviors.

Environment Parameter Channel: Pathway through which the environment parameters affect the agents and behaviors.

Inference and Heuristic: Higher-level cognitive processes used by the agents to interpret the environment and respond accordingly.

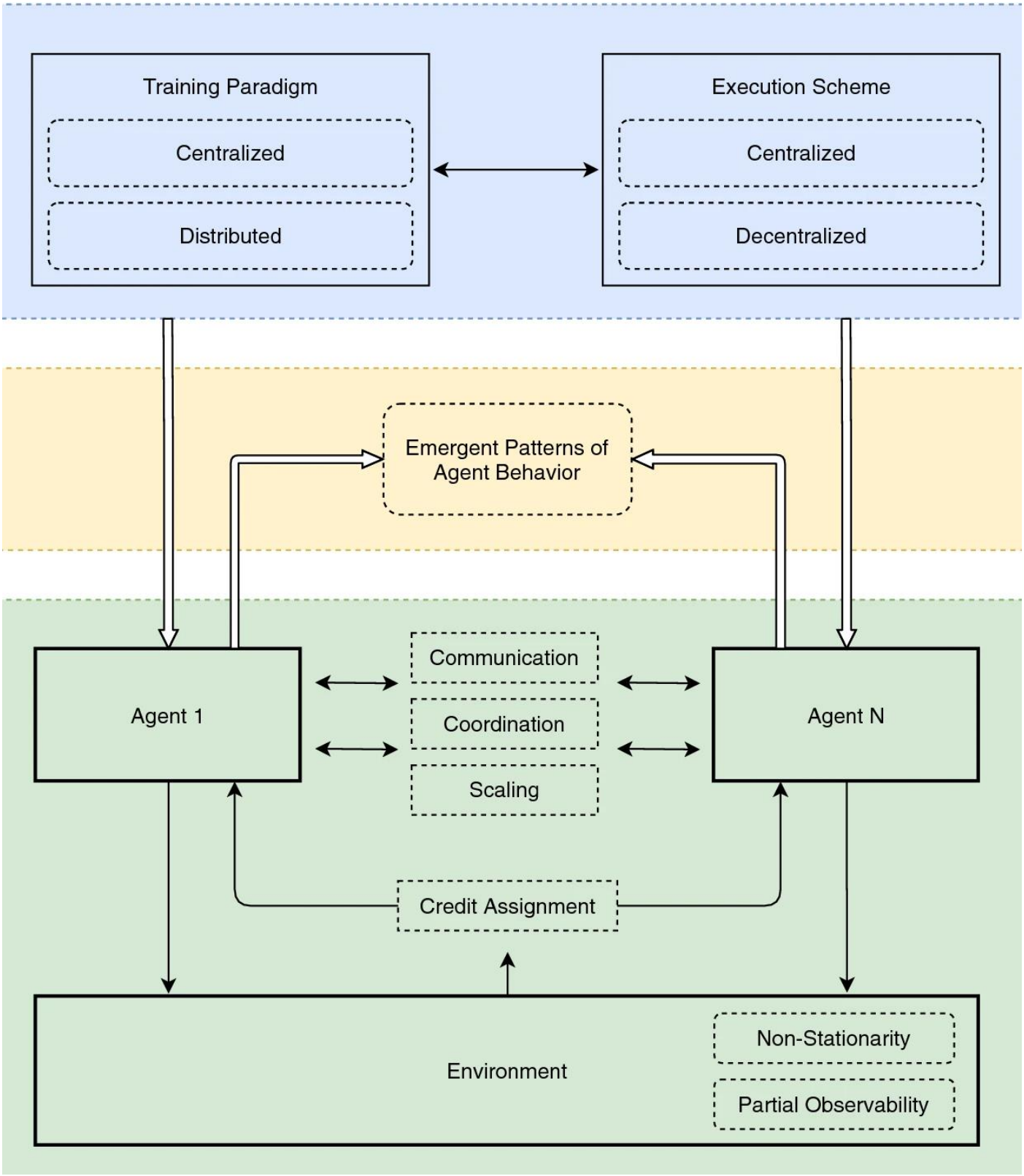
Python API and Python Trainer: External components for interacting with and developing the learning environment.

The diagram illustrates the structured relationships and interdependencies between these core components, providing a comprehensive view of the learning environment system.

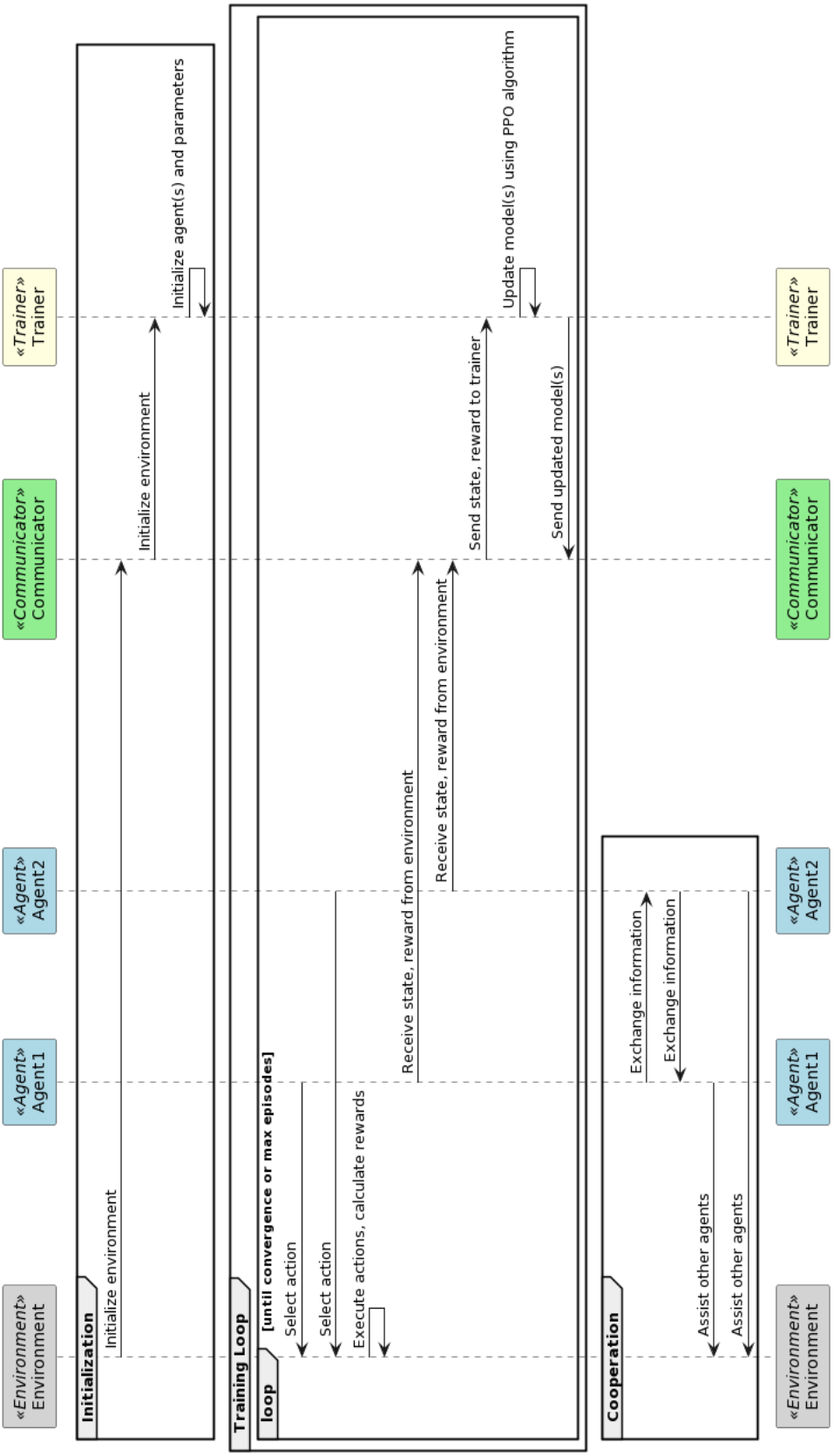


[Diagram.2: use-case for marl-simulations]

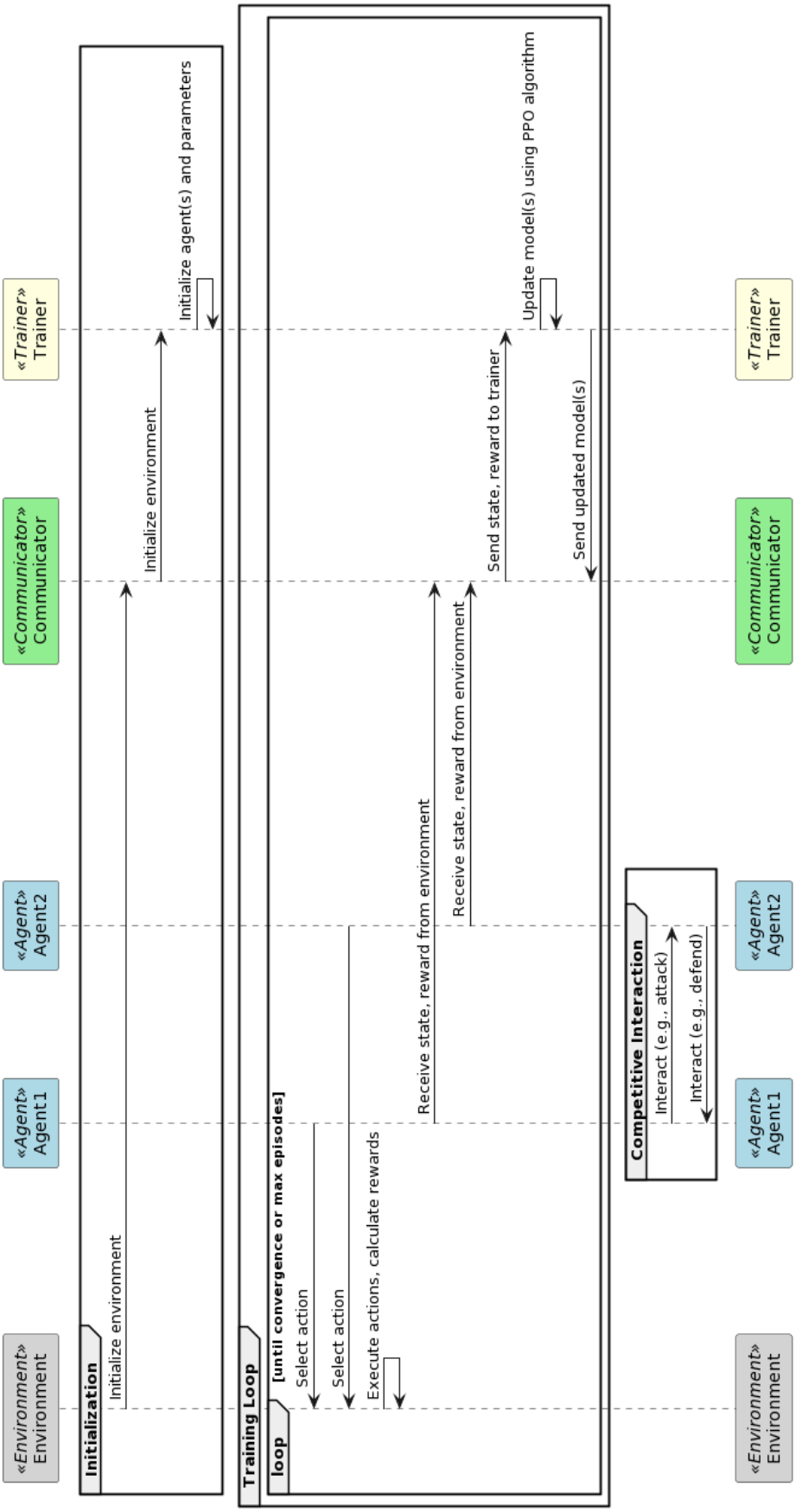
WORKFLOW DIAGRAM - MARL



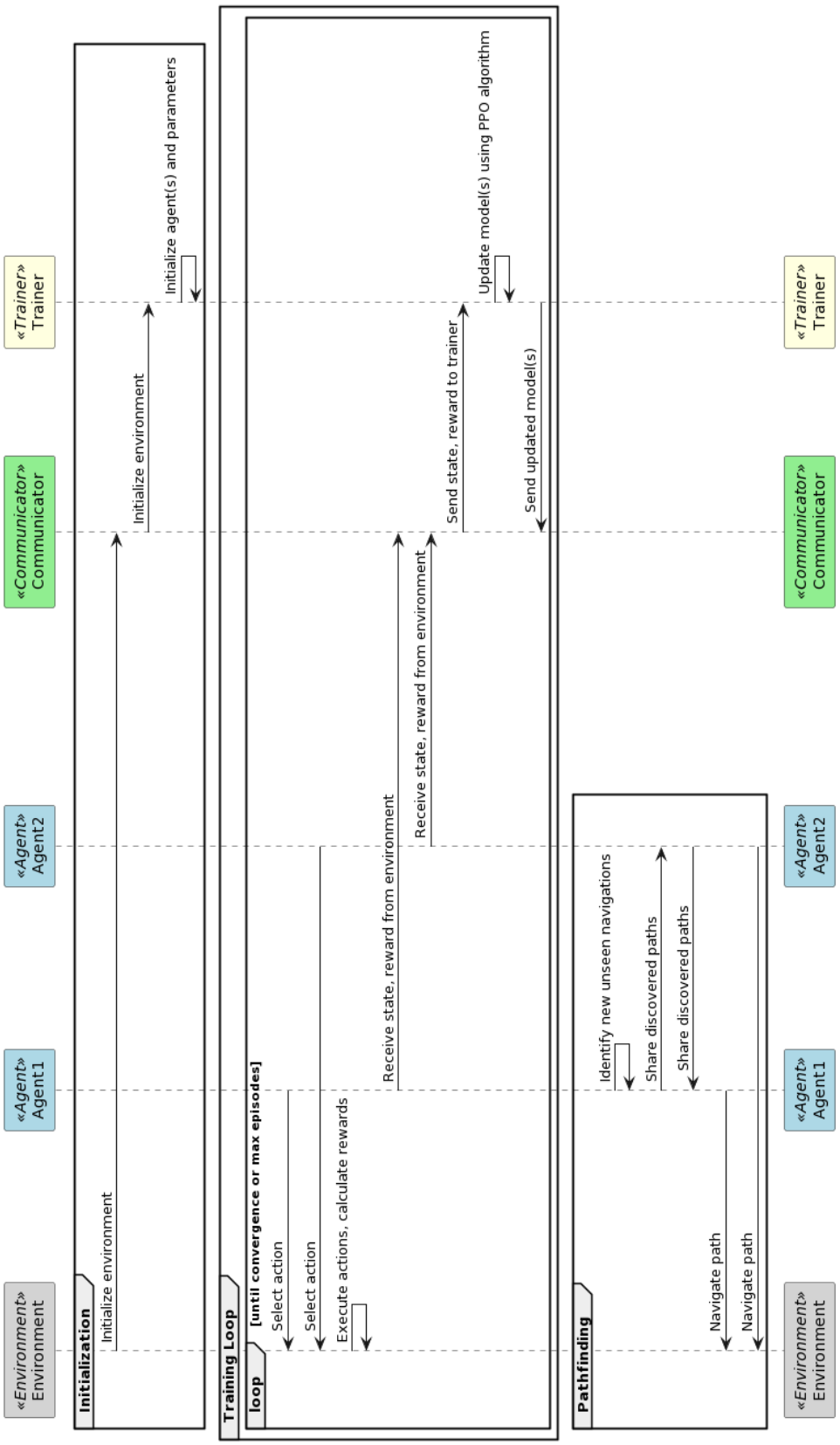
[Diagram.3: workflow for marl-simulations]



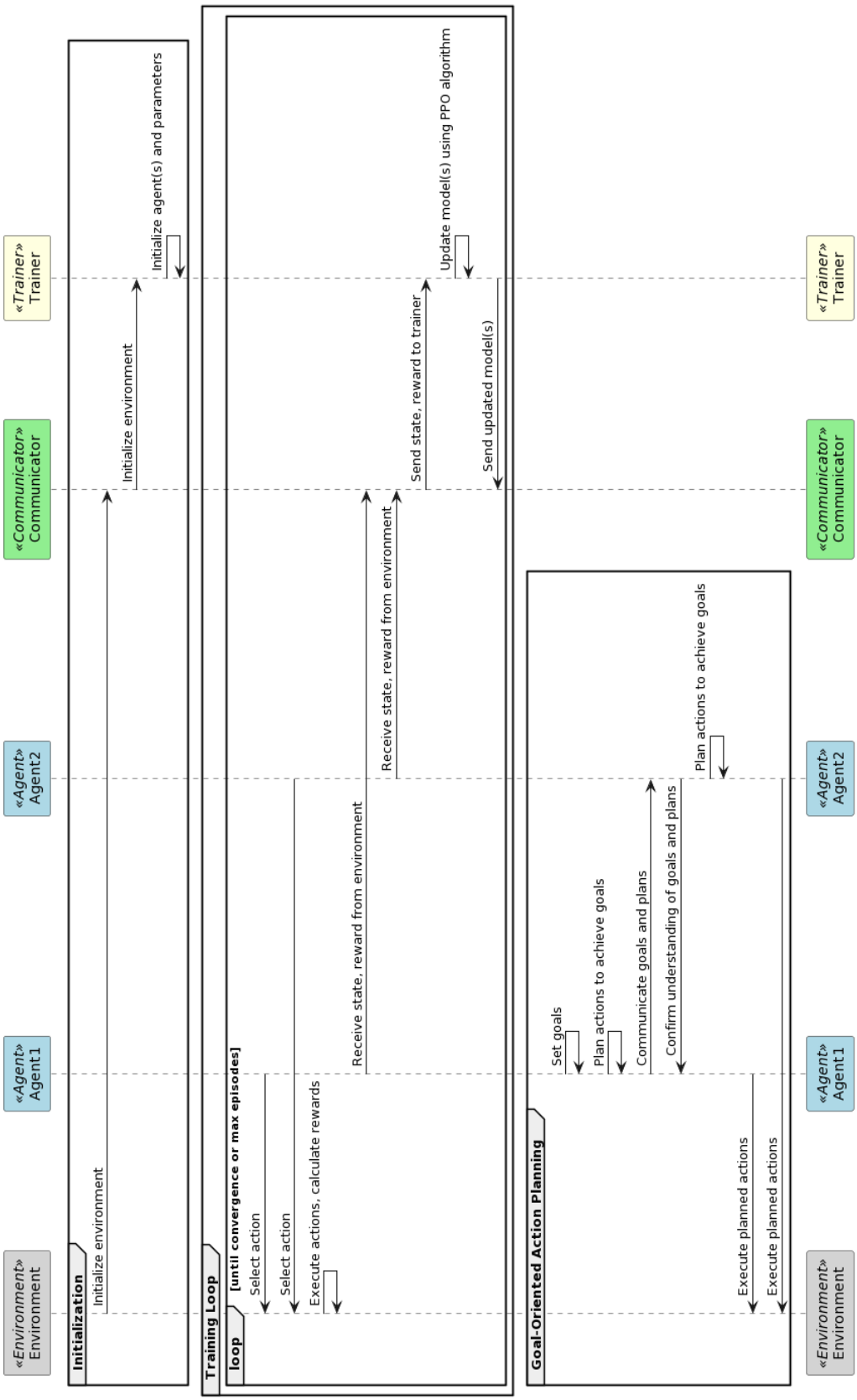
[Diagram.4: cooperative agent's sequence-to-sequence for marl-simulations]



[Diagram.5: competitive agent's sequence-to-sequence for marl-simulations]



[Diagram.6: pathfinding agent's sequence-to-sequence for marl-simulations]



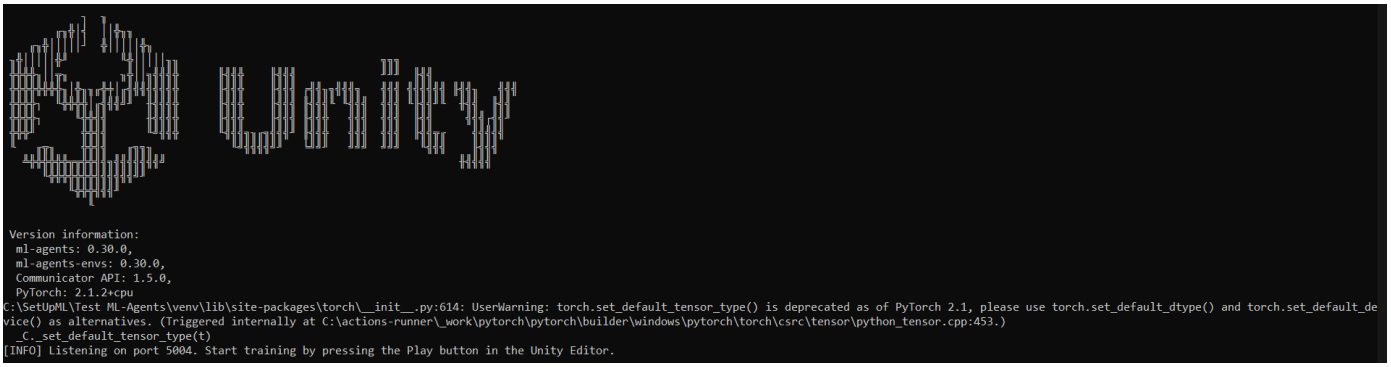
[Diagram.7: goal oriented action planning agent's sequence-to-sequence for marl-simulations]

Summarized Steps:

1. Install Unity from the official website.
2. Create a new Unity project, selecting the appropriate template (3D/3D URP).
3. Install the latest version of MLAgents from the Unity Package Manager.
4. Install Python 3.9.13 (or the required version) from the official Python website.
5. Add Python to the system's environment PATH during installation.
6. Navigate to the directory where your Unity project is located.
7. Check the installed Python version using the command prompt.
8. Create a virtual environment using Anaconda, Miniconda, or traditional Python methods.
9. Upgrade pip if required.
10. Activate the virtual environment.
11. Install required Python packages using pip (mlagents, torch, torchvision, torchaudio, packaging, protobuf).
12. Verify MLAgents installation by checking its availability.
13. Set up your Unity environment, agents, and parameters as needed.
14. Initiate training using the appropriate MLAgents command.
15. Start training in Unity by hitting the "Play" button in the Unity Editor.
16. Monitor the training process within Unity and analyze the results.
17. Evaluate training using the provided MLAgents command.
18. Launch TensorBoard to visualize training metrics by running the command with the appropriate log directory parameter.

List of Python Libraries to Install:

- mlagents
- tensorflow
- torch
- torchvision
- torchaudio
- packaging
- protobuf
- tensorboard
- gym
- tf-agents



[Figure.5: mlagents-learn for unity in python]

HOW TO RUN?

1. **Install Unity:**

Download and install Unity from the official website.
Follow the installation instructions provided during the installation process.

2. **Create New Project 3D/3D URP:**

Open Unity Hub.
Click on "New" to create a new project.
Choose the 3D or 3D Universal Render Pipeline (URP) template based on your preference.

3. **Install mlagents Latest Version:**

Open Unity.
Navigate to Windows > Package Manager.
Select the Unity Registry tab.
Search for "mlagents" and install the latest version available.

4. **Install Python Specific Version** (3.9.13 in my case): *[different version in future]*

Download and install Python 3.9.13 from the official Python website.
Add Python to the environment PATH during installation.

5. **Check Python Version and Create Virtual Environment:**

Open a command prompt.
Type python and hit Enter to check the Python version. Exit Python using exit().
Navigate to your Unity project folder directory.
Create a virtual environment using ***python -m venv venv***.
Upgrade pip if required using ***python -m pip install --upgrade pip***.

6. **Activate Virtual Environment and Install Packages:**

Activate the virtual environment using venv\Scripts\activate.
Install required packages:

pip install mlagents
pip install torch
pip install torchvision
pip install torchaudio
pip install packaging
pip install protobuf==3.20.3 *# Note: version may change in the future*

7. **Set Up Unity Environment and Agents:**

Set up your Unity environment, agents, and parameters as required.

8. Training and Evaluation:

Return to the command prompt.
Start training with the specified run ID:
mlagents-learn --run-id=test1

9. Start training with the specified run ID:

Hit the Play button from the Unity editor to begin the training process.
Monitor and analyze the training process.
If training stops, resume with:
mlagents-learn --run-id=test1 --resume

10. Evaluate the trained model:

Evaluate the trained model:
mlagents-learn eval --run-id=YourRunID

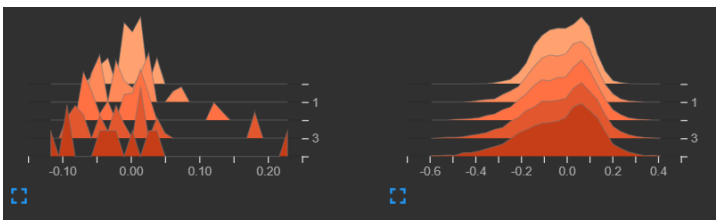
11. TensorBoard Visualization:

Launch TensorBoard to visualize training progress:
tensorboard --logdir <dir_name>

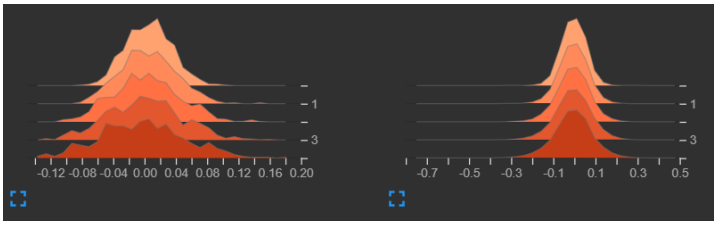
```
[INFO] Hyperparameters for behavior name Searcher:
        trainer_type:   ppo
        hyperparameters:
          batch_size:   1024
          buffer_size:  10240
          learning_rate:        0.0003
          beta: 0.005
          epsilon:      0.2
          lambd:        0.95
          num_epoch:    3
          shared_critic:        False
          learning_rate_schedule:       linear
          beta_schedule:        linear
          epsilon_schedule:     linear
        network_settings:
          normalize:    False
          hidden_units: 128
          num_layers:   2
          vis_encode_type:      simple
          memory:       None
          goal_conditioning_type:       hyper
          deterministic:        False
        reward_signals:
          extrinsic:
            gamma:      0.99
            strength:   1.0
            network_settings:
              normalize:        False
              hidden_units:     128
              num_layers:       2
              vis_encode_type:  simple
              memory:   None
              goal_conditioning_type:   hyper
              deterministic:    False
        init_path:      None
        keep_checkpoints:       5
        checkpoint_interval:    500000
        max_steps:      500000
        time_horizon:   64
        summary_freq:   50000
        threaded:       False
        self_play:      None
        behavioral_cloning:     None
```

[Figure.6: hyperparameters]

[Figure.7: tensorboard dashboard]



[Figure.8: training vizualization]



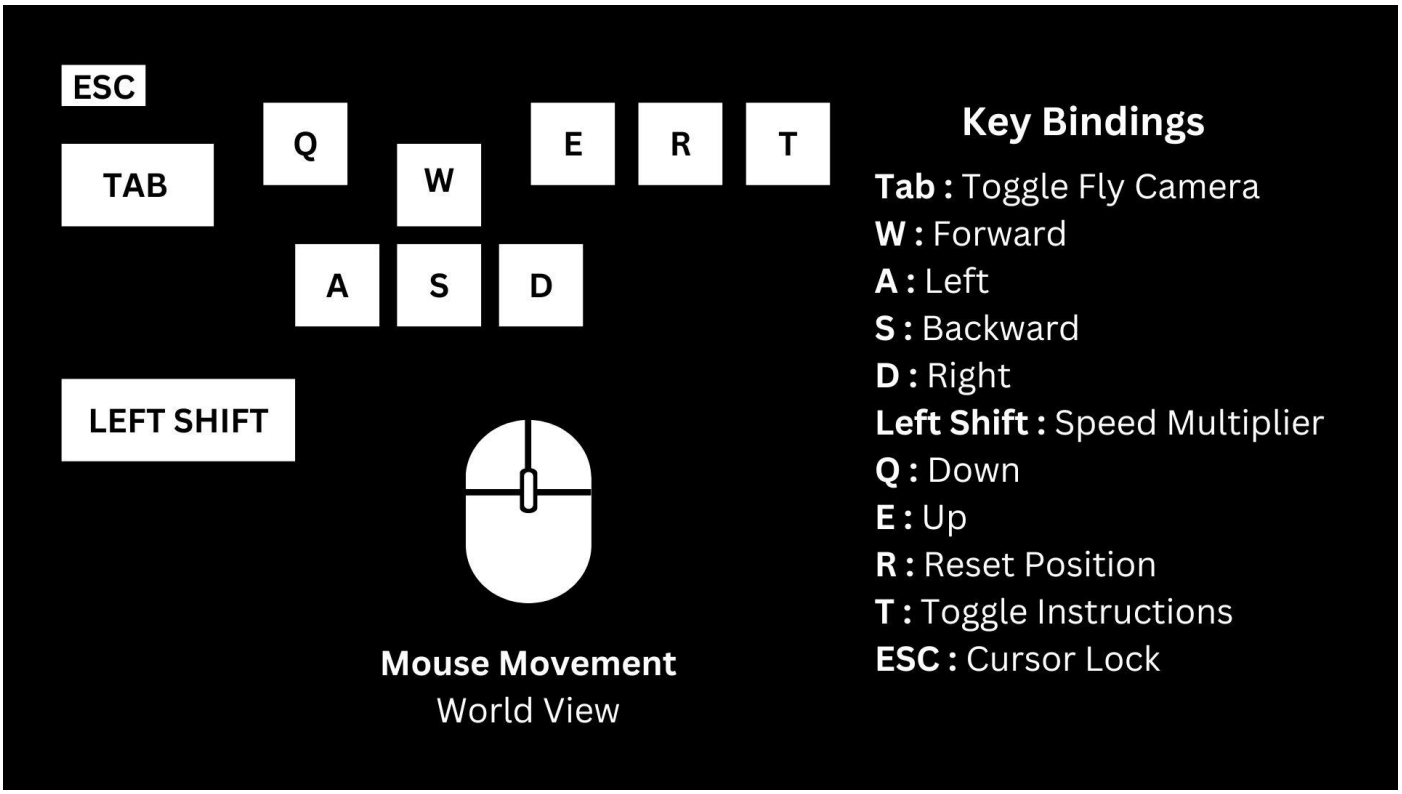
[Figure.9: model analysis]

CLI COMMANDS

One-Shot Command Line Interface Commands:

- cd path\to\your\Unity\project
- python --version
- python -m venv venv
- python -m pip install --upgrade pip
- venv\Scripts\activate
- pip install tensorflow
- pip install mlagents
- pip install mlagents-envs
- pip install torch
- pip install torchvision
- pip install torchaudio
- pip install packaging
- pip install protobuf
- pip install tensorboard
- pip install tensorboard-data-server
- pip install setuptools
- pip install pywin32
- pip install grpcio
- pip install onnx
- mlagents-learn --run-id= YourRunID i.e..test1
- mlagents-learn --run-id=test1 --resume
- mlagents-learn eval --run-id=YourRunID i.e..test1
- tensorboard --logdir YourRunID i.e..test1
- deactivate

Note: I have listed every possible command-line interface command as of my knowledge in April 2024. There may be changes to these commands in the future, especially with updates to ML-Agents or Python versions.



[Figure.10: fly camera controls]

Note: Controls different from those listed above may not be mentioned here. These key mappings are common for most simulations.

Introduction to MARL Simulations

Multi-Agent Reinforcement Learning (MARL) is a rapidly advancing field that explores the complex dynamics of multiple autonomous agents operating within a shared environment. MARL simulations provide a powerful framework for studying these intricate systems, where agents must learn to adapt their behaviors and interact with one another to achieve individual or collective goals.

At the core of MARL simulations are groups of agents, each equipped with their own sensory inputs, actions, and reward functions. These agents learn through a trial-and-error process, gradually refining their decision-making capabilities as they navigate the environment and respond to the behaviors of their counterparts. The emergent patterns that arise from these agent interactions can be remarkably complex and often surprising.

The applications of MARL simulations span a wide range of domains, including robotics, game AI, traffic management, and economic modeling. By modeling the intricate interplay between agents, researchers and developers can gain invaluable insights into the design of effective multi-agent systems that can tackle real-world challenges.

[Figure.11: types of agents simulations]

CHAPTER 1: COOPERATIVE AGENTS

COOPERATIVE AGENTS

Cooperative agents are artificial intelligence systems designed to work together towards a common goal. They are programmed to share information, coordinate their actions, and assist each other in completing tasks. Cooperative agents are often used in scenarios where multiple autonomous systems need to collaborate, such as in robotic swarms, distributed sensor networks, or multi-agent simulations. The key characteristics of cooperative agents are their ability to communicate, negotiate, and make decisions as a group to achieve their collective objectives.

Implementation of MARL Simulations for Cooperative agents:

Ants Simulation –
Ant Simulation is an interactive project that simulates the behavior of ants as they search for food, create pheromone trails, and navigate their environment.

Sheepherding Simulation –
Sheepherding simulation where trained virtual dogs employ strategic herding and guarding techniques to protect a vulnerable flock of sheep from relentless virtual wolf threats.

Decision Making NPC –
Decision-making NPC simulation where autonomous agents are trained for self-learning, making choices on activities like food, sleep, watching TV, using the toilet, and other tasks.

Imitation Learning –
Imitation learning simulation in which two autonomous agents evolve and strategically use nearby objects to collaboratively learn climbing walls, assisting each other in achieving a common goal.

Note: To play the simulations, always refer to the deployment page and documentation for instructions on how to play the simulations on WebGL or executable versions.

CHAPTER 2: COMPETITIVE AGENTS

COMPETITIVE AGENTS

Competitive agents are AI systems that are designed to operate in an adversarial or competitive environment. They are programmed to pursue their own individual goals, which may be in conflict with the goals of other agents. Competitive agents can engage in a variety of competitive behaviors, such as resource allocation, task scheduling, or strategic decision-making. The goal of competitive agents is to outperform or outmaneuver their rivals, often through the use of sophisticated planning, negotiation, or learning algorithms. Competitive agents are commonly used in game-theoretic scenarios, economic simulations, or security applications.

Implementation of MARL Simulations for Cooperative agents:

Football Training Simulation–
Football training simulation where player agents, trained on diverse models, exhibit varied performances in different situations, aiming to score goals in a two-team setup with distinct player and goalie strategies.

Volleyball Training Simulation–
Volleyball training simulation where player agents, trained on diverse models, learn to pass the ball, aiming to hit it into the opponents' area and score points in a two-team setup with distinct player strategies.

Fighting AI Simulation –
Fighting AI simulation with 1-on-1 agents engaging in punching and defense, utilizing randomly selected moves and aiming to defeat each other through a reward-based system.

MotoGP-Offroad Simulation–
MotoGP-offRoad simulation where autonomous bike agents learn offroad driving using grid sensors, equipped with comprehensive bike parameters, while adeptly following a designated path.

Note: To play the simulations, always refer to the deployment page and documentation for instructions on how to play the simulations on WebGL or executable versions.

CHAPTER 3: PATH FINDING AGENTS

PATH FINDING AGENTS

Path finding agents are AI systems that are designed to navigate through complex environments and find optimal or near-optimal paths between two or more locations. They use a variety of algorithms and techniques, such as Dijkstra's algorithm, A* search, or reinforcement learning, to analyze the environment, identify obstacles, and plan the most efficient route. Path finding agents are often used in applications such as robotics, video games, transportation systems, or logistics planning, where the ability to navigate through complex environments is critical.

Implementation of MARL Simulations for Cooperative agents:

Pathfinding Car Simulation –
Pathfinding car simulation featuring autonomous vehicle equipped with sensors, adept at avoiding various obstacles and self-identifying a new path if distracted during navigation.

Crossy Road Simulation –
Crossy Road simulation where an agent learns to safely cross roads, avoiding collisions with moving cars, by exploring and discovering all possible paths to achieve the goal.

Aircraft Simulation –
Aircraft simulation where planes learn to fly by collecting checkpoints, navigating with varied brain parameters, and avoiding collisions with mountains.

Surveillance Drone Simulation - Surveillance drone simulation where the drone is trained to navigate through a narrow tunnel, detecting angles, adjusting stability, and learning the optimal path through a brute-force approach.

Note: To play the simulations, always refer to the deployment page and documentation for instructions on how to play the simulations on WebGL or executable version.

CHAPTER 4: GOAL ORIENTED ACTION PLANNING AGENTS

GOAL ORIENTED ACTION PLANNING

Goal-oriented action planning is a technique used in AI systems to enable them to achieve specific objectives or goals. It involves the agent breaking down a high-level goal into a sequence of lower-level actions or sub-goals, and then using planning algorithms to determine the most efficient way to execute those actions. This allows the agent to reason about its environment, anticipate potential obstacles or challenges, and develop a strategic plan to reach its desired outcome. Goal-oriented action planning is widely used in a variety of AI applications, including task automation, decision support systems, and intelligent control systems.

Implementation of MARL Simulations for Cooperative agents:

Car Parking Simulation –
Car parking simulation where a vehicle learns precise parking in a designated area through a reward system, navigating with a trail of randomly spawning challenges.

SpaceX Rocket Simulation –
SpaceX's landing simulation employs a reinforcement learning algorithm where a rocket agent optimizes axis control and stability with primary and secondary thrusters to achieve precise landings on a specified pad.

Village Simulation –
Village simulation where agents simulate a comprehensive workday environment, navigating between work buildings, restaurants, homes, hospitals, gardens, and more, as they learn to simulate the daily dynamics of life in a functioning community.

Military Combat Simulation –
Military combat simulation in a dynamic No Man's Land environment involves self-evolving agents, including terrorists, counter-terrorist assulters, turrets, snipers, aircrafts, and missiles & adaptive warfare.

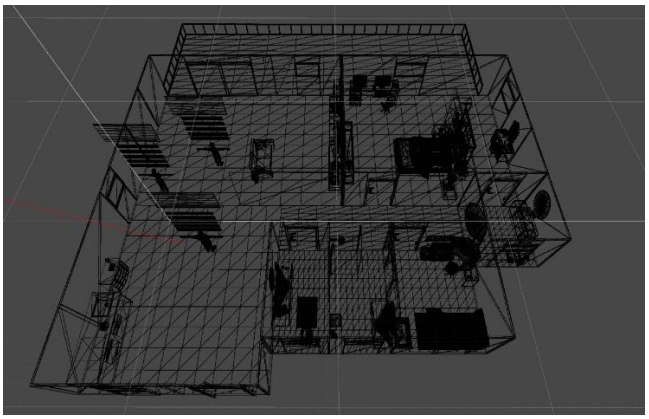
Note: To play the simulations, always refer to the deployment page and documentation for instructions on how to play the simulations on WebGL or executable versions.

CHAPTER 5: PROJECT F11 – HYBRID AGENTS

PROJECT F11 – HYBRID AGENTS

Project F11 is a research initiative focused on the development of hybrid agents. These agents are designed to navigate complex, dynamic environments and interact with cooperative, competitive, pathfinding, goal-oriented action planning agents. Leveraging complex Formula 1 mechanics, self-coded and rigorously tested on the Unity environment, these hybrid agents may utilize a range of techniques including multi-agent learning, negotiation, and conflict resolution to accomplish their objectives. The primary goal of Project F11 is to push the boundaries of AI-based decision-making and problem-solving, potentially benefiting areas such as smart city management, logistics, and national security.

In a simulated environment, users can manually toggle drive using keyboard input. However, when the mode switches to autopilot, our autonomous F1 agent, trained on an NVIDIA GeForce RTX 4090 GPU, simulates a racing circuit and can navigate any unknown track. It observes parameters such as speed (longitude, latitude, absolute), gear (R, N, D), steering, throttle, brake, RPM (front, rear), power efficiency, power balance, wheel torque balance, MGU rotor torque, pedal shifters, stator, drive shafts, 4-wheel drive, engine, clutch transmission, fuel consumption, steering, suspension, load, torque, force, slip, ground material, driver damping, and more. All these parameters are processed simultaneously in real-time, ensuring perfect accuracy for the F1 mechanics. The environment includes a detailed racing track, scenic side views, a lap system, tracking sensors, and other elements. With variations in camera angles, the simulation delivers an impressive visual experience. Moreover, it required 319 hours to train on the specified GPU.



[Figure.12: wireframe]

[Figure.13: shaded wireframe]

[Figure.14: shaded]

[Figure.15: greybox prototyping]

Note: The figures shown are just for example purposes to illustrate concepts.
[For Definition Purpose]

Wireframes:

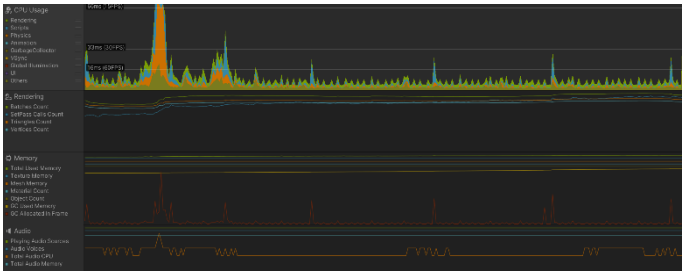
Wireframes are low-fidelity visual representations of the structure and layout of a user interface, focusing on the placement of elements on the page without visual design details.

Greybox Prototyping:

Greybox prototyping is a type of low-fidelity prototyping that uses simple, neutral-colored shapes and placeholders to represent user interface elements, focusing on the interactive functionality and user experience rather than visual design.

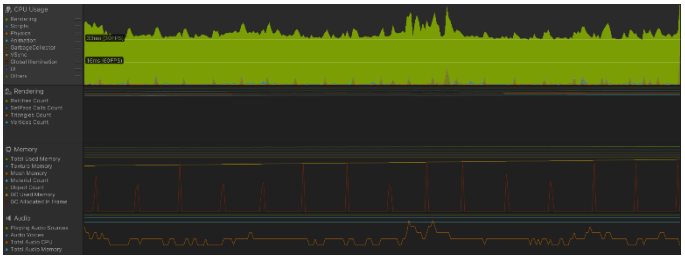
PROFILER ANALYSIS FOR EACH SIMULATIONS

The Profiler gathers and displays data on the performance of your application in areas such as the CPU, memory, renderer, and audio.

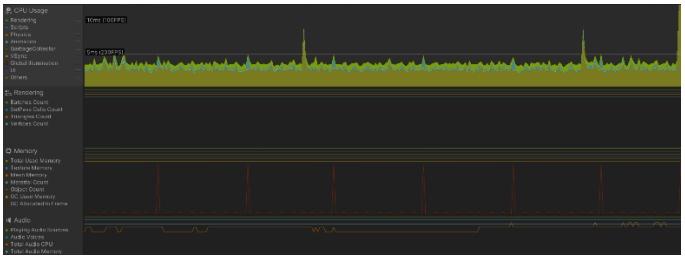


Ant Simulation Profiler

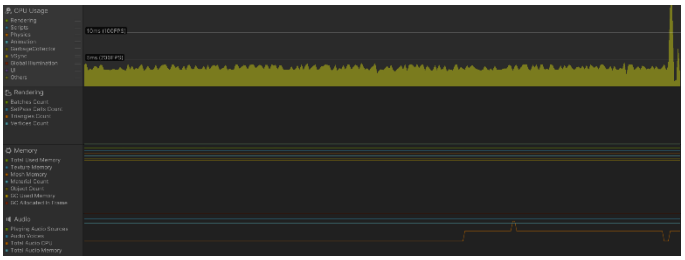
Sheepherding Simulation Profiler



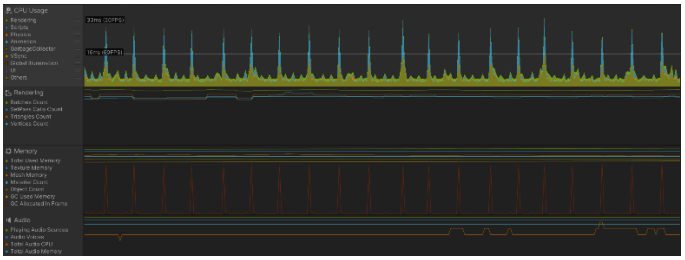
Decision Making NPC Simulation Profiler



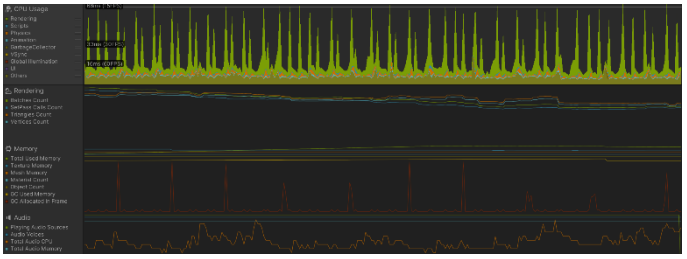
Imitation Learning Simulation Profiler



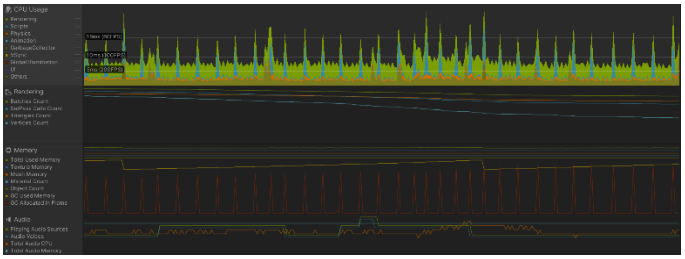
Football Training Simulation Profiler



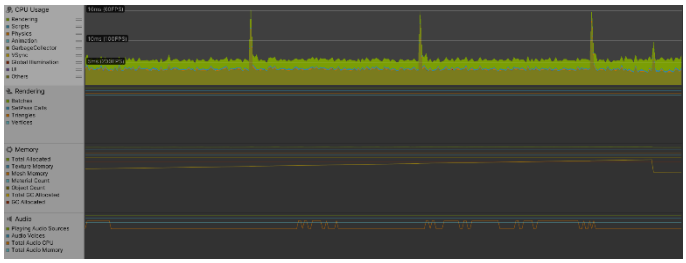
Volleyball Training Simulation Profiler



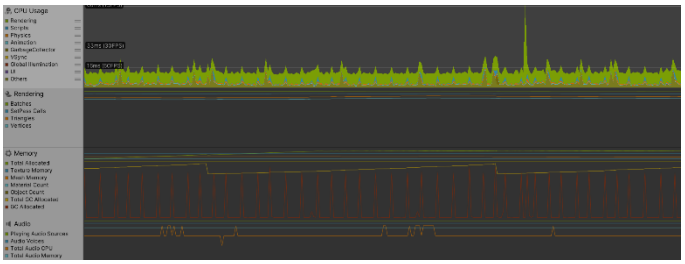
Fighting AI Simulation Profiler



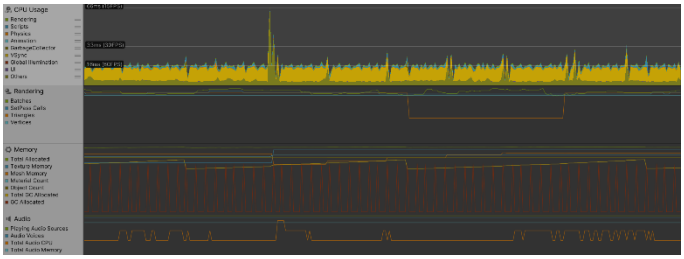
MotoGP-Offroad Simulation Profiler



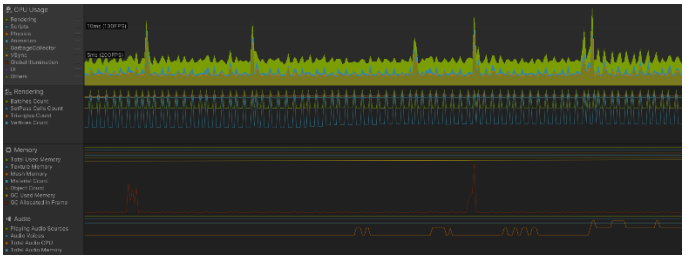
Pathfinding Car Simulation Profiler



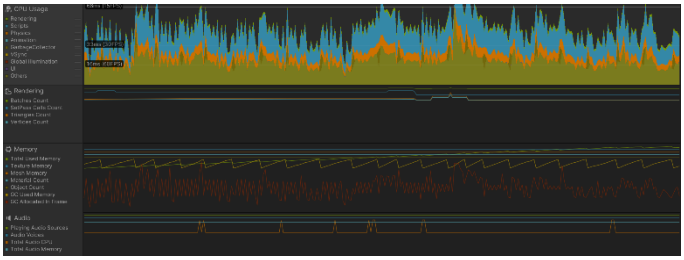
Crossy Road Simulation Profiler



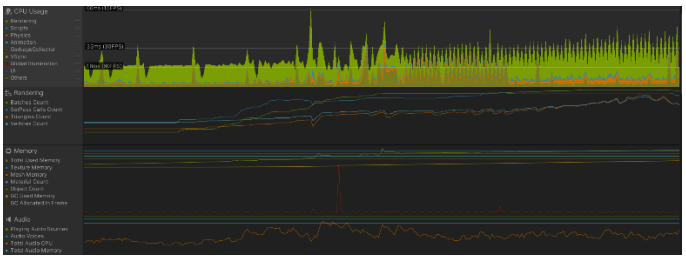
Aircraft Simulation Profiler



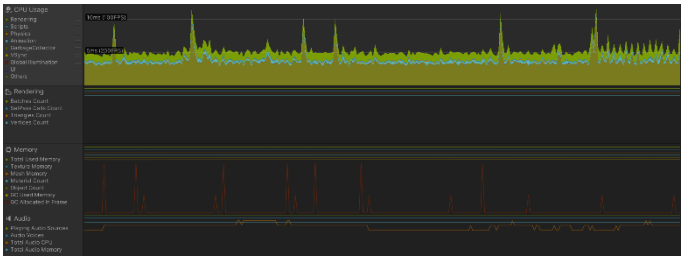
Surveillance Drone Simulation Profiler



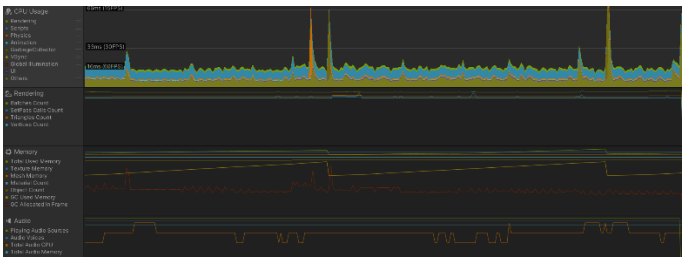
Car Parking Simulation Profiler



SpaceX Rocket Simulation Profiler



Village Simulation Profiler



Military Combat Simulation Profiler

Note: All the above profilers are subjective to the test environment for the specified time duration during testing phase may differ on different benchmarks with GPU if executable, or Browser if on WebGL.

PROJECT LIMITATIONS

- MARL Simulations require specific versions of plugins and Python 3.9.13(Specific), April 2024.
- The ml-agents Unity Toolkit for MARL performs well in stable or Long Term Support Versions (beta, experimental Unity versions are not recommended for the ML-Agents toolkit).
- ML-Agent simulations take a significant amount of time to train on local machines with GPUs.
- Learning game mechanics and implement algorithms for reinforcement learning on agents can be challenging.(PPO)
- Issues such as RAM memory crashes, GPU overclocking, and system failures may occur during the training process.
- More codebase is required to handle the optimization of every single movement in the simulation.
- Low frame rates on low-end devices may cause executable crashes.
- Deadlocks frequently occur due to algorithm overlapping in the same environment with different agents.
- Some training processes are not predictable due to slow learning processes.
- Fine-tuning parameters can be one of the most challenging tasks in MARL simulations.

Note: Rapid advancements in AI agents and increasing computing power are steadily overcoming limitations in Multi-Agent Reinforcement Learning (MARL). With improved software compatibility, optimized training speeds, enhanced algorithm robustness, and hardware reliability, MARL simulations are becoming more efficient, robust, and accessible. These advancements promise to unlock new possibilities for MARL across various applications and industries.

FUTURE WORK

- **Urban Planning and Design:** MARL can aid in urban planning by simulating and optimizing the interactions between various urban elements, such as transportation systems, infrastructure, and public spaces, to enhance livability, sustainability, and resilience in cities.

- **Environmental Conservation:** MARL can be utilized to coordinate the actions of autonomous drones, sensors, and wildlife robots in monitoring and protecting endangered species, conserving habitats, and mitigating the impact of natural disasters on ecosystems.

- **Space Exploration:** MARL techniques can assist in coordinating the actions of fleets of autonomous spacecraft, rovers, and satellites in space exploration missions, including planetary exploration, asteroid mining, and interstellar travel, optimizing resource utilization and mission success.

- **Disaster Resilience in Smart Cities:** MARL can play a role in enhancing disaster resilience in smart cities by coordinating the actions of emergency response teams, infrastructure systems, and community resources to prepare for, respond to, and recover from natural and man-made disasters.

- **Bioinformatics and Drug Discovery:** MARL techniques can aid in drug discovery and development by coordinating the actions of virtual agents in simulating molecular interactions, predicting drug efficacy and toxicity, and optimizing drug design strategies to accelerate the discovery of novel therapeutics.

- **Traffic Management:** MARL can be applied to optimize traffic flow in urban areas by coordinating the actions of autonomous vehicles, traffic signals, and other infrastructure elements to minimize congestion and improve overall efficiency.

- **Supply Chain Management:** MARL techniques can help optimize inventory management, transportation logistics, and resource allocation in complex supply chain networks by coordinating the actions of multiple agents such as suppliers, manufacturers, distributors, and retailers.

- **Peer-to-Peer Energy Trading:** MARL techniques can facilitate peer-to-peer energy trading in decentralized energy systems, where households with renewable energy sources (e.g., solar panels) can trade excess energy with each other, optimizing energy distribution and reducing reliance on centralized power grids.

- **Collaborative Robotics:** MARL can facilitate collaboration among teams of robots in industrial settings, where they can work together on assembly lines, warehouse operations, and other tasks requiring coordination and cooperation.

- **Marine Conservation and Exploration**: MARL can be utilized in marine conservation efforts by coordinating the actions of underwater drones, autonomous vessels, and marine sensors to monitor and protect marine ecosystems, detect illegal fishing activities, and explore uncharted ocean depths.

[Figure.16: stakeholders]

- **Researchers:** Academics, scientists, and experts in fields like artificial intelligence, machine learning, and robotics are deeply invested in MARL simulations. They aim to explore new algorithms, methodologies, and applications of MARL in solving complex problems.
- **Military technologies:** Government agencies and defense contractors are leveraging MARL simulations to enhance autonomous systems, strategic decision-making, and mission planning. Applications span unmanned aerial vehicles (UAVs), autonomous vehicles, battlefield coordination, and cybersecurity to bolster national security and defense capabilities.
- **Developer communities** are vital stakeholders in MARL simulations, contributing to the open-source ecosystem, developing tools, libraries, and frameworks for MARL research and applications. These communities foster collaboration, knowledge sharing, and innovation in the field by providing resources and platforms for experimentation and learning.
- **Tech organizations** and **startups** are at the forefront of applying MARL simulations to real-world problems. They develop and deploy MARL-based solutions in diverse domains such as transportation, logistics, finance, and healthcare. These stakeholders are focused on leveraging MARL to optimize processes, enhance efficiency, and create competitive advantages in their respective industries.
- **Investors** and **entrepreneurs** recognize the potential of MARL simulations as a disruptive technology with wide-ranging applications. They provide funding, resources, and support to startups, research labs, and tech companies working on MARL projects. Their primary interest lies in identifying promising opportunities for investment and fostering innovation in MARL-based products and services.

Deployment on Itch.io (WebGL):

The Multi-Agent Reinforcement Learning (MARL) simulations are now accessible on Itch.io as WebGL applications. This deployment offers users a seamless experience, enabling direct interaction with the MARL experiments through their web browsers. Leveraging WebGL technology ensures high-quality graphics and smooth performance, enhancing the overall user experience. Providing a user-friendly interface, the simulations on Itch.io facilitate effortless exploration and experimentation with various parameters and scenarios. Whether you're a researcher, developer, or enthusiast, accessing the MARL simulations on Itch.io guarantees a hassle-free experience, fostering engagement and exploration in the realm of reinforcement learning.

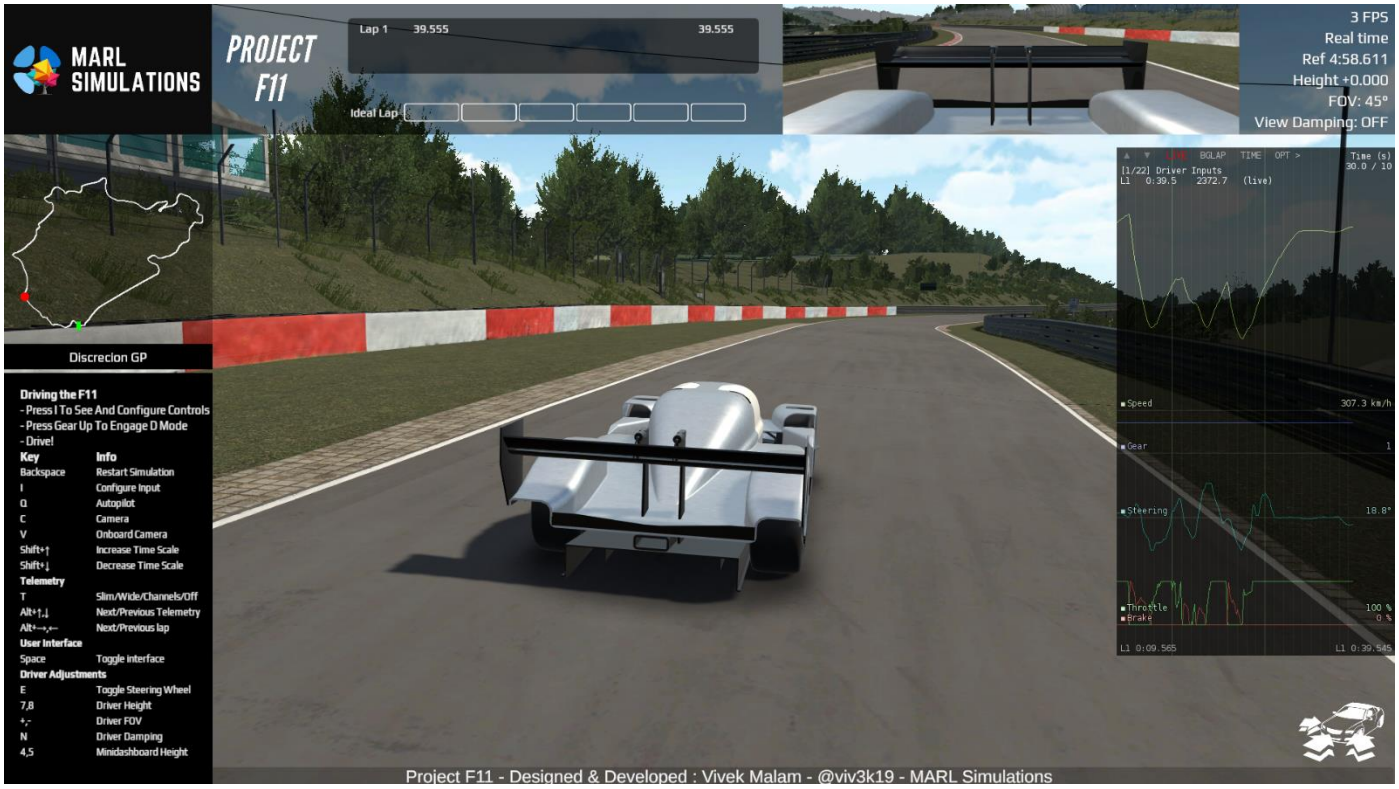
Follow the link: https://viv3k19.itch.io/

Deployment on GitHub Repository (Standalone Executable):

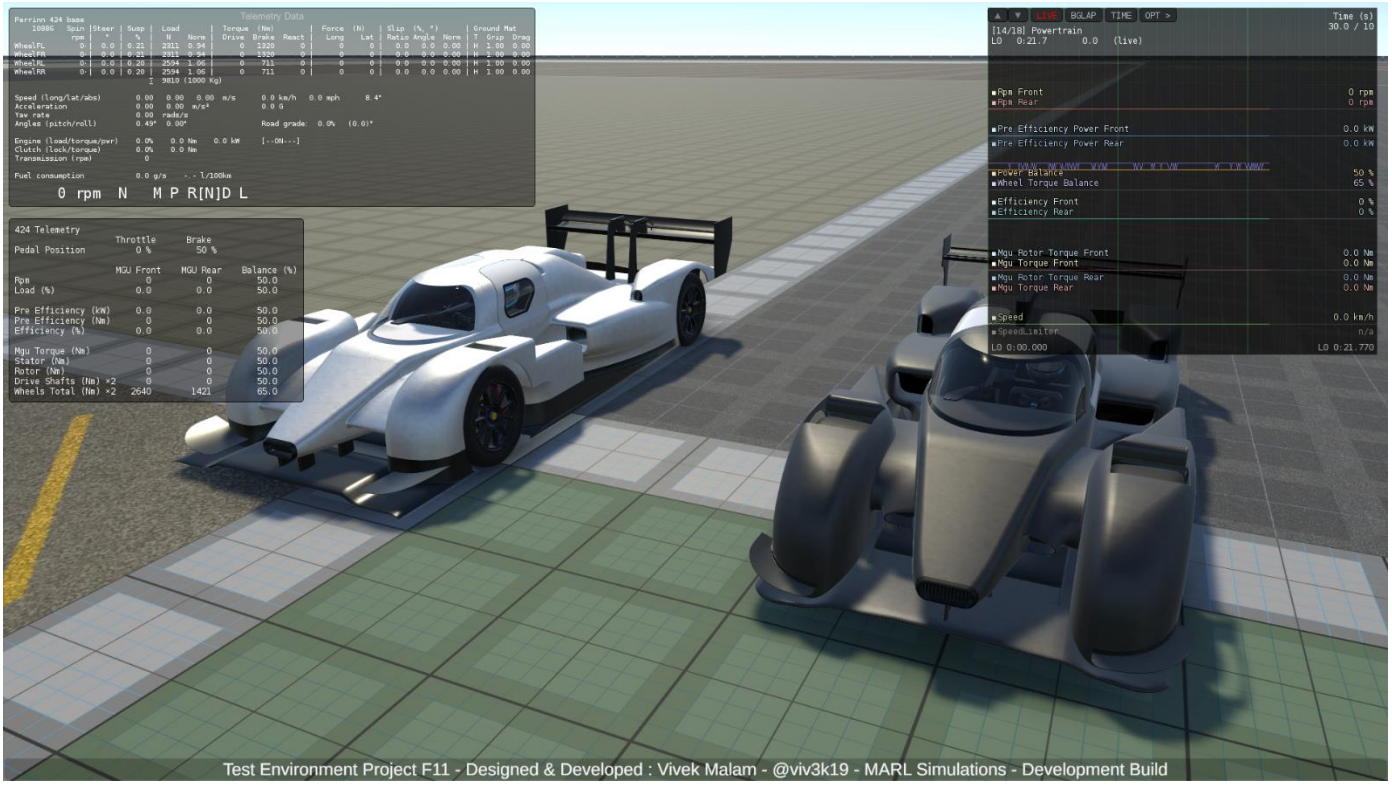
The Multi-Agent Reinforcement Learning (MARL) simulations are also available as executable files, conveniently hosted on the GitHub repository. This deployment approach grants users the flexibility to download and execute the simulations locally on their machines. Offering accessibility across different platforms and environments, users can explore the MARL simulations at their convenience. With the simulations readily accessible on GitHub, users can clone the repository, compile the code, and run the simulations effortlessly. Whether for research, education, or experimentation purposes, accessing the executable files on the GitHub repository provides a straightforward and customizable experience, empowering users to delve into the intricacies of multi-agent reinforcement learning.

Follow the link: https://github.com/viv3k19

Project F11

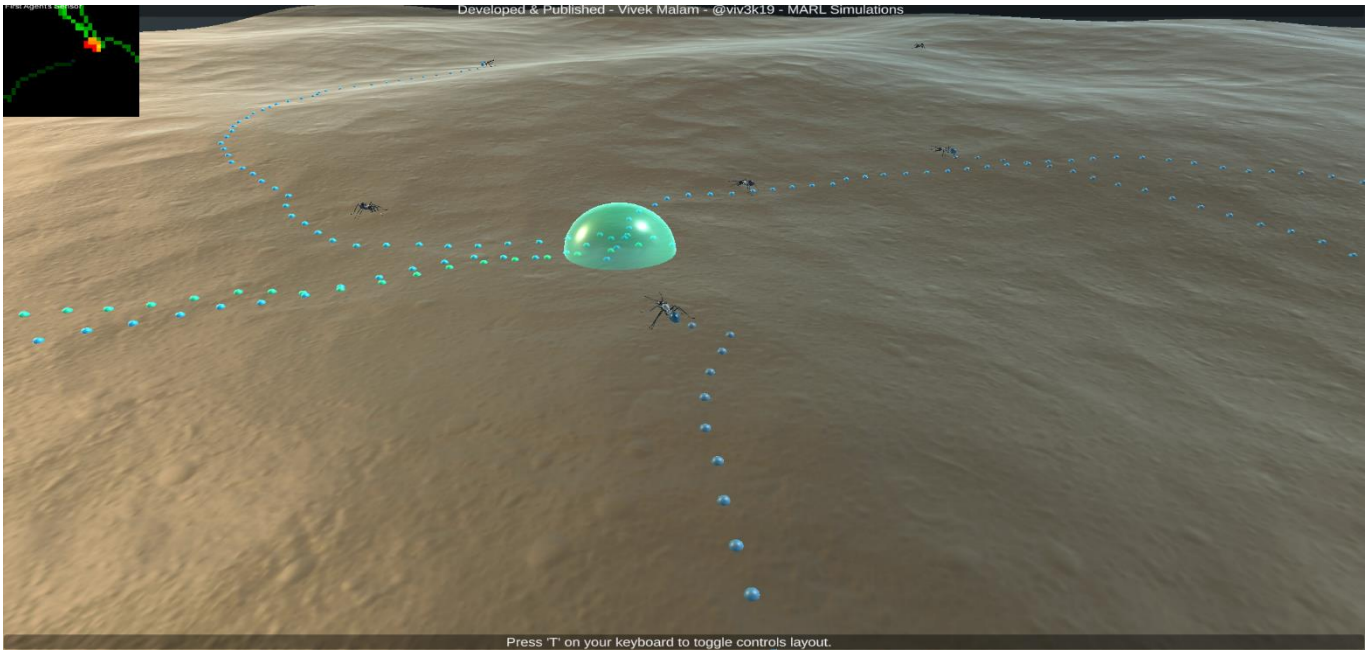


Project F11 Gameplay View



Prototype Project F11 Test Environment

Ant Simulation



In a simulated environment, ants are searching for food, and their trails are helping other ants navigate towards the food source.

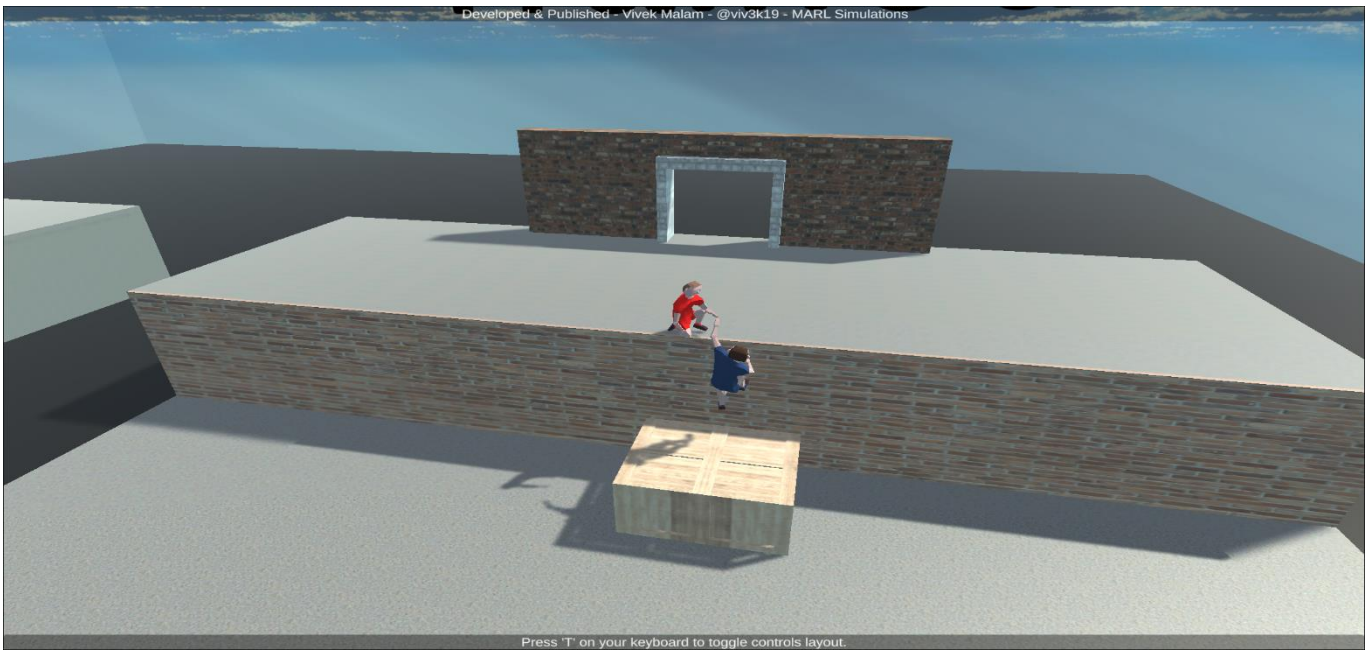
Sheepherding Simulation

In a simulated environment, dogs are guarding a flock of sheep from wolf attacks, wolves are attempting to kill the sheep, and the sheep are fleeing from the wolves.

Decision Making NPC

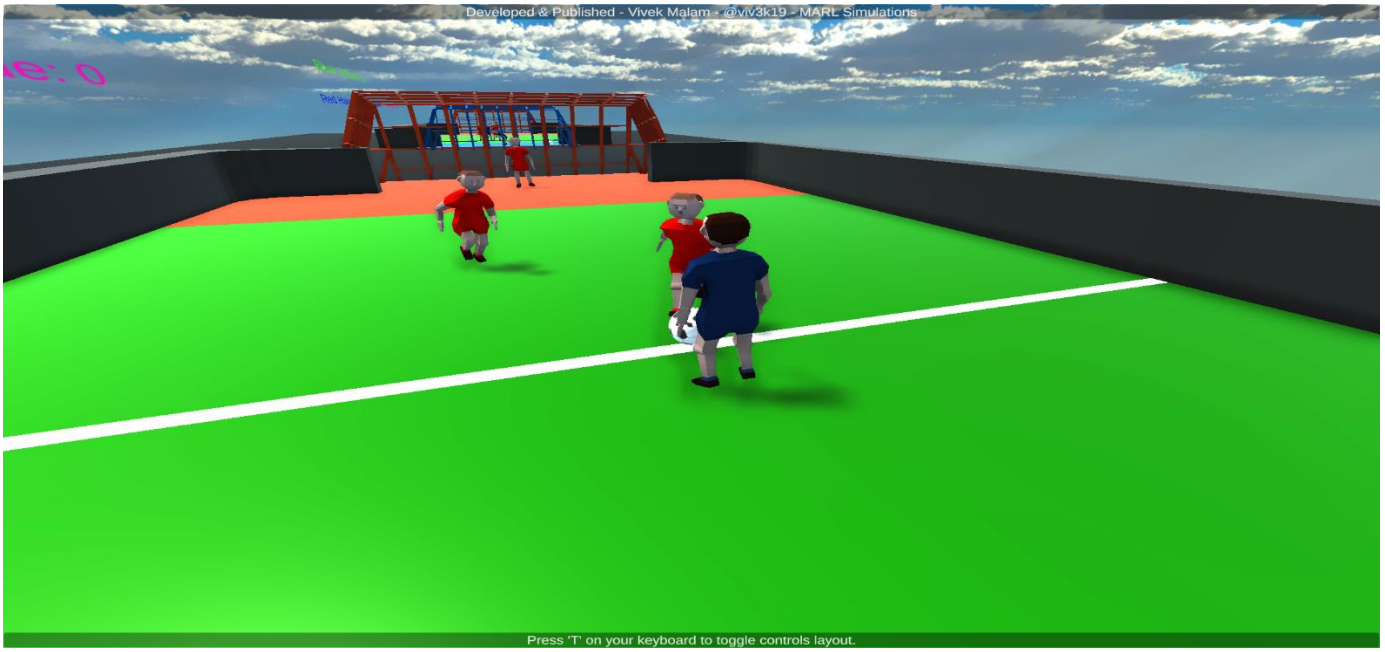
In a simulated environment, three Non-Playable Characters (NPCs) are learning from their home environment through autonomous decision-making based on triggered parameters. For instance, when hungry, they head to the food area, and when low on energy, they go to the sleep area.

Imitation Learning



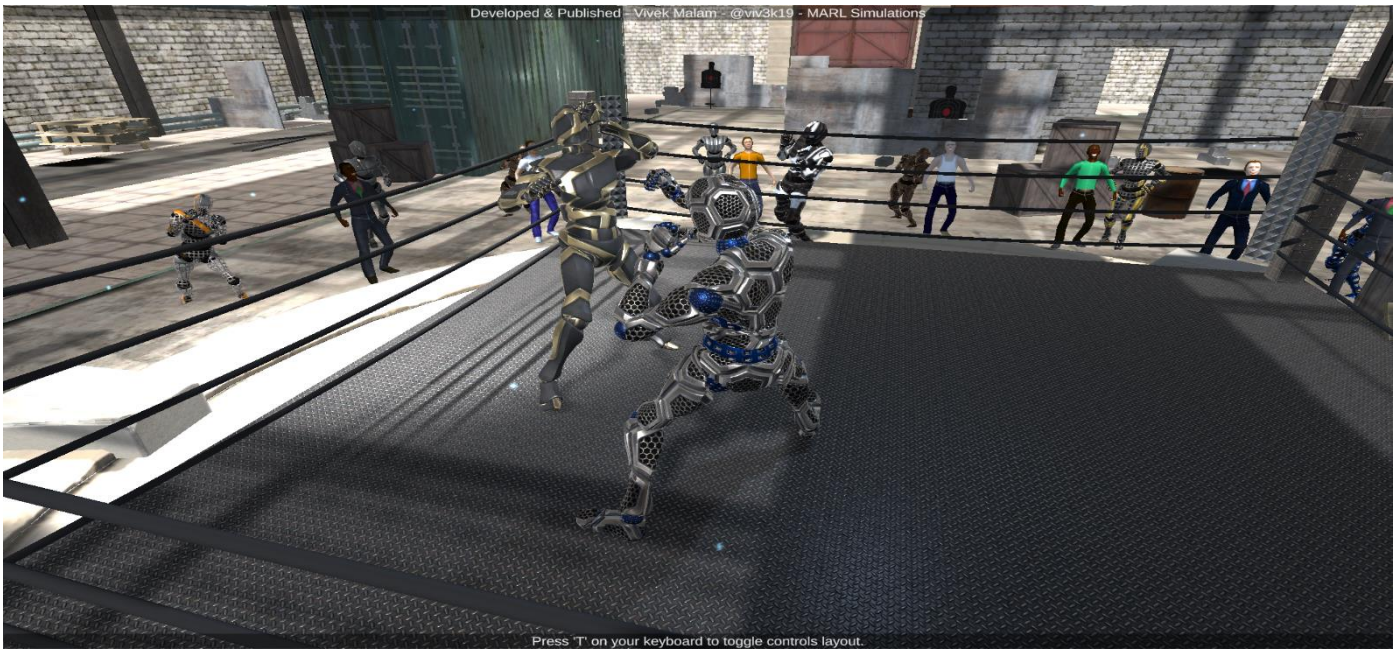
In a simulated environment, two collaborative agents are assisting each other to achieve a shared objective. For instance, they collaborated to push a crate and then climbed atop it together to overcome an obstacle and escape.

Football Training Simulation



In a simulated environment, a 3v3 football training session is underway, with each agent trained on various parameters and given differing amounts of training hours to ensure the overall gameplay appears realistic. These agents autonomously learn skills such as stealing the ball, shooting, defending, goalkeeping, and more.

Fighting AI Simulation



In a simulated environment, there's a one-on-one autonomous fight scene taking place in a boxing ring. Each agent has decision parameters to determine which animation (kick, punch, dodge) to execute in response to various stimuli. These self-learning agents simulate the fight scenario by adapting and improving their strategies over time.

Volleyball Training Simulation

In a simulated 2v2 volleyball training, agents learn and adapt their behaviors (shooting, passing, or a mixed of both randomly) based on varying parameters and training hours. They improve in real-time through a reward system, enhancing performance dynamically.

MotoGP-Offroad Simulation

In a simulated environment, autonomous bike agents compete in a race, equipped with grid sensors to stay on track, avoid obstacles, and scan the environment in real-time. They adjust driving parameters such as speed, turning, leaning angle, overtaking, off-roading and more.

Pathfinding Car Simulation

In a simulated environment, an autonomous car navigates along a designated circuit path, adeptly avoiding all types of obstacles while maintaining its course. Equipped with various decision parameters and sensors, the car detects objects and adjusts its speed, drift, turning, torque, and other parameters accordingly.

Crossy Road Simulation

In a simulated environment, a hen learns to cross the road by stopping and waiting for cars, aiming to choose the shortest path. It evaluates its performance using step count and other reward parameters to gauge accuracy.

Aircraft Simulation

In a simulated environment, aircraft are mastering flight between mountains, skillfully avoiding collisions with rocks. The environment includes checkpoints, lap systems, and time constraints, challenging them to rank 1st, 2nd, or 3rd based on self-learned flying skills, such as boost, longitude, latitude, and flight control.

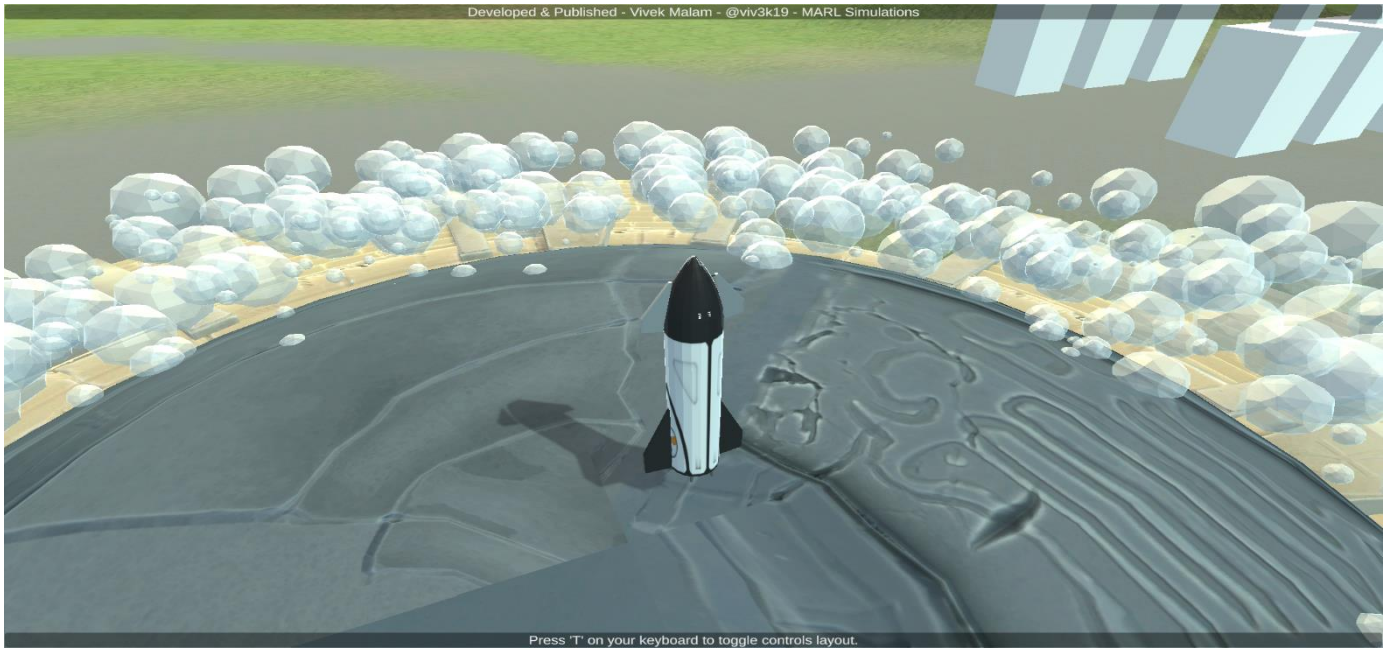
Surveillance Drone Simulation

In a simulated environment, an autonomous surveillance drone is learning to navigate through narrow tunnels using a mechanism of spherical sensors to detect nearby rocks and avoid collisions. It is equipped with numerous sensors to detect the way in real-time.

Car Parking Simulation

In a simulated environment, a car learns to park in randomly spawned parking spaces, dynamically locating the parking spot with the goal of achieving correct orientation. Feedback, shown in red or green, indicates whether the parking is successful or not.

SpaceX Rocket Simulation



In a simulated environment, a SpaceX rocket simulates its landing on a pad upon returning to Earth. It involves numerous calculations based on velocity and thruster force, with 8 thrusters determining landing positions. Learning parameters are utilized to achieve accurate positioning on the landing pad.

Village Simulation

In a simulated environment, villagers are spawned and trained to autonomously decide their daily routines and tasks, such as going to the office, park, home, or bank, based on their decision-making parameters. The unpredictable nature of each agent's next move adds to the realism of the simulation.

Military Combat Simulation

In a simulated combat environment, a base-capturing scenario unfolds between the red and blue teams. Turrets, Snipers, Assaulters, Healers, and Aircrafts simulate a war scenario. Each agent autonomously decides their own parameters and level of participation in the conflict.

CONCLUSION

In conclusion, Multi-Agent Reinforcement Learning (MARL) stands as a promising frontier in artificial intelligence research, offering a robust framework for agents to learn and collaborate in complex environments. Through the interaction and coordination among multiple agents, MARL not only addresses challenges that single-agent approaches encounter but also unlocks possibilities for scalable and efficient solutions in various domains, from robotics and autonomous systems to economic simulations and game theory.

The journey through MARL research has revealed its potential to tackle real-world problems by enabling agents to learn from their interactions, adapt to dynamic environments, and exhibit emergent behaviors that surpass the capabilities of individual agents. However, this journey also highlights the intricacies and challenges inherent in designing effective multi-agent systems, including issues of coordination, communication, and scalability.

This project showcases the effectiveness of Multi-Agent Reinforcement Learning (MARL) in training intelligent agents to navigate complex environments and interact with each other. Leveraging the Unity ML-Agents Toolkit and PyTorch, a diverse range of simulations has been explored, highlighting the adaptability and versatility of MARL approaches.

Through meticulous experimentation and visualization tools like TensorBoard, the project has contributed valuable insights into the dynamics of multi-agent interactions. Moving forward, challenges such as adapting to unknown environments and integrating human agency present intriguing avenues for future research and innovation in MARL.

As researchers continue to push the boundaries of MARL, the ultimate goal remains clear: to develop intelligent and adaptive systems capable of seamlessly interacting with and augmenting human endeavors, ultimately paving the way towards a future where autonomous agents contribute positively to society and our understanding of complex systems.

REFERENCES

[1] *Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., & Mordatch, I. (2019, September 17). Emergent tool use from Multi-Agent Autocurricula.*
https://doi.org/10.48550/arXiv.1909.07528

[2] *Yaoxiang Wang, Zhiyong Wu, Junfeng Yao, Jinsong Su (2024). TDAG: A Multi-Agent Framework based on Dynamic Task Decomposition and Agent Generation.*
https://doi.org/10.48550/arxiv.2402.10178

[3] *Xuefeng Wang, Henglin Pu, Hyung Jun Kim, Husheng Li (2024). DeEPSaFeMPC: Deep Learning-Based Model Predictive Control for Safe Multi-Agent Reinforcement Learning.*
https://doi.org/10.48550/arxiv.2403.06397

[4] *Haoqi Yuan, Chi Zhang, Hongcheng Wang, Feiyang Xie, Penglin Cai, Hao Dong, Zongqing Lu (2023). Skill reinforcement learning and planning for Open-World Long-Horizon tasks.*
https://doi.org/10.48550/arxiv.2303.16563

[5] *Ljubisa Bojic, Matteo Cinelli, Dubravko Culibrk, Boris Delibasic (2023). CERN for AGI: a theoretical framework for Autonomous Simulation-Based Artificial Intelligence testing and alignment.*
https://doi.org/10.48550/arxiv.2312.09402

[6] *Yaodong Yang, Jun Wang (2020). An Overview of Multi-Agent Reinforcement Learning from Game Theoretical Perspective.*
https://doi.org/10.48550/arXiv.2011.00583

[7] *Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, Danny Lange (2018). Unity: a general platform for intelligent agents.*
https://doi.org/10.48550/arXiv.1809.02627

[8] *Ning, Z., & Xie, L. (2024). A survey on multi-agent reinforcement learning and its application. Journal of Automation and Intelligence.*
https://doi.org/10.1016/j.jai.2024.02.003

[9] *Bus¸Oniu, L., Babu˘ska, R., & Bart De Schutter. (2008). Multi-Agent Reinforcement Learning: An Overview. In IEEE Transactions on Systems, Man, and Cybernetics—Part C: Applications and Reviews (Vols. 38–38, Issue 2, pp. 156–172).*
http://dx.doi.org/10.1007/978-3-642-14435-6_7

[10] *Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, Stefano V. Albrecht (2020). Benchmarking Multi-Agent deep reinforcement learning algorithms in cooperative tasks.*
https://doi.org/10.48550/arXiv.2006.07869

[11] LantaoYu. (n.d.). GitHub - LantaoYu/MARL-Papers: Paper list of multi-agent reinforcement learning (MARL). GitHub. https://github.com/LantaoYu/MARL-Papers
https://github.com/LantaoYu/MARL-Papers

[12] Jordan Juravsky, Yunrong Guo, Sanja Fidler, Xue Bin Peng (2022). PADL: Language-Directed Physics-Based Character Control. PADL: Language-Directed Physics-Based Character Control.
https://doi.org/10.1145/3550469.3555391

[13] Joon Sung Park, Joseph C. O'Brien, Carrie J. Cai, Meredith Ringel Morris, Percy Liang, Michael S. Bernstein (2023). Generative Agents: interactive simulacra of Human Behavior.
https://doi.org/10.48550/arXiv.2304.03442

[14] David Shapiro. (n.d.). GitHub - daveshap/ACE_Framework: ACE (Autonomous Cognitive Entities) - 100% local and open source autonomous agents. GitHub.
https://github.com/daveshap/ACE_Framework

[15] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, Jürgen Schmidhuber (2023). MetaGPT: Meta Programming for a Multi-Agent Collaborative Framework.
https://doi.org/10.48550/arXiv.2308.00352

[16] Chen Qian, Yufan Dang, Jiahao Li, Wei Liu, Weize Chen, Cheng Yang, Zhiyuan Liu, Maosong Sun (2023). Experiential Co-Learning of Software-Developing Agents.
https://doi.org/10.48550/arxiv.2312.17025

[17] Guangyao Chen, Siwei Dong, Yu Shu, Ge Zhang, Jaward Sesay, Börje F. Karlsson, Jie Fu, Yemin Shi(2023). AutoAgents: a framework for automatic agent generation.
https://doi.org/10.48550/arxiv.2309.17288

[18] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, Chi Wang. (2023). AutoGen: Enabling Next-Gen LLM applications via Multi-Agent Conversation [Journal-article]. arXiv.
https://doi.org/10.48550/arXiv.2308.08155

[19] Shaofei Cai, Bowei Zhang, Zihao Wang, Xiaojian Ma, Anji Liu, Yitao Liang (2023). GROOT: Learning to follow instructions by watching gameplay videos.
https://doi.org/10.48550/arxiv.2310.08235