FABMLA: Framework for Agent-Based Modeling With Unity's ML-Agents

Arvick Chandnani[1†], Emma Brown[2]
[1]*12th, Eastlake High School, 400 228th Ave NE, Sammamish, WA 98074 USA*
[2]*Media Art & Technology Program (MAT), University of California, Santa Barbara, CA 93106 USA*
[†]*corresponding author:* arvick@outlook.com

Unity3D, a free Game Engine (GE), is commonly used in academic simulations. However, Unity3D brings a limitation: there is a lack of Agent-Based Modeling (ABM) frameworks that use the built-in ML-Agent package. In this paper, we investigate ABM and how it can be used to model complex social systems. We developed an ABM framework that supports custom ML-Agents implementations and is generalizable to various scenarios. We aim to show the value of our proposed framework in its feasibility and accessibility as a simulation resource in Unity3D. Our proposed framework creates a real-time 3D environment where ML-Agents can interact. To assess FABMLA we recorded data including consistency, applicability, and accessibility. Additionally, we aim to display the benefits of visual simulations in Unity3D environments and discuss how these tools can be used in educational environments to teach younger generations and support academic research.

Keywords: Agent-based modeling (ABM), visual simulations, Navigation, Ecosystem simulations, Framework, Unity3D

I. INTRODUCTION

The Agent-Based Modeling (ABM) paradigm simulates complex social situations in the real world and models crowd behavior and interactions through individual, autonomous members within the system (Cheliotis, 2022). It has been proven to be an effective method for simulating complex systems (Yu, 2022; Bonabeau, 2002; Manson, Sun, and Bonsal, 2011). For instance, Possik et al (2023) used ABM to model an intensive care unit during the COVID-19 pandemic, Reynolds (1987) used ABM to model generic flock crowds, and Mls et al (2023) used ABM to simulate responses to natural disasters.

Many modern ABM development platforms largely do not support three dimensions. The limited platforms that do are confined to 3D visualizations of 2D models (Cheliotis, 2021). The lack of three-dimensional capability creates limitations when using ABM in fields requiring three-dimensional systems. However, a development platform that seems to be a worthwhile alternative is Game Engines (GEs) (Cheliotis, 2021).

Modern GEs support up-to-date 3D graphics and other features that can be used in an academic setting (Sobota and Pietriková, 2023). A popular choice for similar studies—for instance, in Virtual Reality (VR) (Iparraguirre-Villanueva, 2024)—is Unity3D, a free GE. Unity3D offers in-built features such as a C# Application Processing Interface (API), pathfinding, and physics engines that provide sophisticated control over the underlying sub-processes of a three-dimensional simulation (Cheliotis, 2022; Sobota and Pietriková, 2023).

In 2020, an Agent-Based Modelling Framework for Unity3D (ABMU) was released (Cheliotis, 2021). ABMU is an open-source project providing Unity3D with tools to conduct ABM simulations. It has been used successfully in various complex social simulations (Yu, 2022; Cheliotis, 2022).

However, ABMU has a crucial limitation—it is incompatible with Unity3D's in-built Machine Learning Agents Toolkit (ML-Agents). ML-Agents is a package developed by Unity3D that enables the training of intelligent agents (Unity Technologies, 2020), creating a central platform for evaluating Artificial Intelligence (AI) advancements within custom Unity3D environments.

Social system models often require agents to react to the environment, which is especially important for human decision-making simulations (Ghaffarzadegan et al, 2024). Machine Learning (ML) models are ideal candidates because of their ability to make informed decisions based on their observations. Filling this gap and combining these methods—ABM and ML—could open the door to more realistic, complex, and generalizable simulations. Furthermore, there is a surprising lack of academic research in Unity3D's ML-Agents in particular.

To support the recent rise of GEs, this paper proposes a novel Framework for Agent-Based Modeling with ML-Agents (FABMLA). FABMLA creates a real-time, complex, visual, 3D environment where ML-powered agents can interact. We recorded data to assess the quality of FABMLA and ML-Agents including consistency, applicability, and accessibility. We also discuss the benefits of visual simulations in Unity3D and how these tools can be used in educational and academic scenarios.

II. METHODS

A. *Software Architecture*

The FABMLA structure is inspired by the architecture of ABMU; we introduce the concept of Steps, an approach that enables behavior to be repeated throughout an episode, a single iteration of a simulation. For example, an agent could move in a certain direction each *Step* based on a custom algorithm. Our framework contains two core classes: *Agent* and *Controller*. The Agent class is responsible for managing a singular agent within a simulation; it largely controls steps and observations (See Section II.B). The Controller class is a singleton module responsible for managing the overall simulation. It conducts high-level tasks such as (re)setting the episode, synchronizing agent steps, and managing agent rewards (See Section II.C). An overall schematic of the FABMLA architecture is found in FIG. 1.

B. *FABMLA Agents*

The Agent class manages a singular agent in a simulation. With FABMLA, custom agents can be designed to fit the needs of a simulation without a deep understanding of simulation management. For more details about FABMLA, visit the open-source GitHub page[1].

Inheriting ML-Agents — The Agent class extends the ML-Agents, exposing key features of ML-Agents in FABMLA. Additionally, we expose key behavior parameters; an agent could override methods and properties without needing extensive information on internal agent logic, allowing for generalizability. For instance, in an ecosystem simulation, agents could modify deer-agent behaviors, such as moving in a certain direction based on calculated rewards and goals, while FABMLA abstracts away implementation details and manages the internals.

Step — The virtual Step() method allows for a variety of applications, such as simple movement actions or complex, interactive puzzles, (See Section III), and contains information regarding repeatable action(s) the agent completes throughout the course of the episode, in other words, the *step* an agent must take. The abstract nature of this function allows for our controller to handle the execution of agent *Steps*, without exposing agent behavior. The non-dependent agent behavior supports generalization allowing FABMLA to be compatible with *any* agent action.

C. *FABMLA Controller*

The Controller class is a standalone module that manages the system and environment. The Controller class executes high-level tasks such as initialization, instantiation, and synchronization. Similar to the Agent class, custom controllers are developed with FABMLA's template. This allows FABMLA to handle the underlying subprocesses, while the controller handles behaviors fitted to the simulation. For example, in a spatial sciences simulation, the controller can declare certain qualities of the solar system, such as the amount of planets, initial distance of planets, gravity strength, and environmental influences and FABMLA manages agent registration, agent-stepping, reward distribution, and episode management.

[1] https://github.com/arvickc/FABMLA

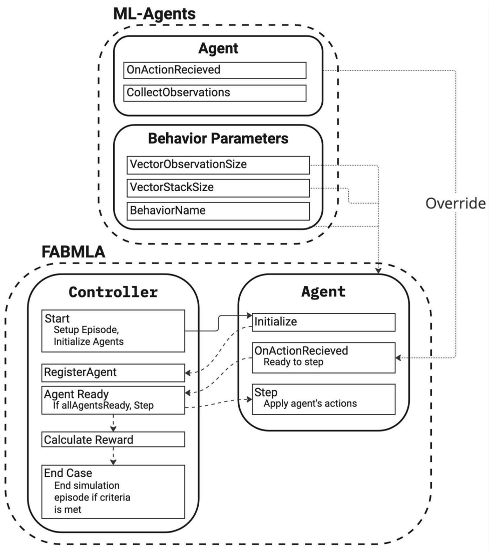

FIG. 1. FABMLA architecture schematic. Dashed rounded boxes indicate packages, solid rounded boxes indicate classes, and solid boxes indicate methods. Solid arrows indicate instantiations, dashed arrows indicate function calls, and dotted arrows indicate overrides or exposures.

Initialization — On initialization, the Controller summons a number of agents depicted by the user. These agents are registered to the Controller and implemented with ML details. This allows support for multiple simulation types including Reinforcement Learning (RL), a training algorithm where agents are rewarded, for training singular agents with an individual goal; Self-play, a training algorithm where agents play against previous snapshots of itself, for training competitive agents with a goal of winning a game/sport; and Cooperation, a training algorithm where agents are rewarded as a *group* instead of individually, for training agents to work towards a common goal (See Section II.D).

Agent Stepping — The Controller class is responsible for synchronizing the steps of all agents. Once all agents have indicated their readiness to execute their actions, the Controller class calls the Step() method in all agents and calculates their reward.

Rewards — Rewards are an essential part of ML, training the agents by rewarding good actions and punishing bad actions. The Controller class includes a virtual method designed for a custom controller to conduct calculations and determine rewards for each agent. This enables complete control over agent rewarding. This method is compatible with multiple simulation types and called for *each* agent on *each* step.

Ending Episodes — The final responsibility for the Controller class is to conclude each episode. The Controller class includes a method designed for a custom controller to declare cases in which the episode ends. For example, an agent could go out of bounds or an agent meets the max step count. This method is called after each step. The

Controller class also includes an EndAll() method, which can be called to end and reset the episode for each agent.

D. *Example Development*

To demonstrate the capabilities of FABMLA we designed five unique example simulations. Each scene showcases a different aspect of FABMLA. In the following paragraphs, we describe how each simulation was developed and its intended purpose.

Simple Movement — This example was developed to test the functionalities of ML-Agents and FABMLA compatibility. The Controller instantiates an amount of agents. The agents cannot collide and are rewarded for staying close to the center of their bounds (See FIG. 2A). The agent can only observe its world position.

Simple Navigation — This example assesses the core functionality of ML-Agents and FABMLA. The Controller, again, instantiates an amount of agents. The agents cannot collide and are rewarded for reaching a target cube and punished for leaving their bounds (See FIG. 2B). The agent can observe its and the target's world position. This example will be used to assess the reliability of FABMLA.

Sugarscape — Epstein and Axtell's Sugarscape model is a classic ABM simulation. Our version is simplified and utilizes the following rules:

1. Agents spawn with a random amount of sugar between 5-7
2. Agents have a random metabolism of 1-3 (subtracted from total sugar each step)
3. The agents must move each step and gain the amount of sugar of the square they moved to
4. If two agents collide, the agent with less sugar is removed from the simulation.

This simulation was developed to demonstrate the interactive and parameterized abilities of FABMLA. The Controller instantiates 32 agents on an 8 by 8 randomized grid (See FIG. 2C). The agent can observe its direct neighbor sugar squares, its world position, and the amount of sugar it has. This example will also be used to assess the consistency of FABMLA.

2D Pong — This example was developed to showcase the self-play feature of ML-Agents and its compatibility with FABMLA. Self-play is an ML training technique used for competitive scenarios. The Agent plays against past snapshots of itself, ideally getting better over time—in this case, the agent is playing pong. The Controller instantiates two agents (on two different teams) and launches the ball in a random direction (See FIG. 2D). The agents can observe its and the ball's world position.

Cooperation Puzzle — This example was developed to display Agent Groups. This is a ML training technique in which agents are *not* rewarded individually, rather, the agents are rewarded as a *group*. This process is used for agents working towards a common goal, such as escape rooms, coordinating routes, or patrolling/security.

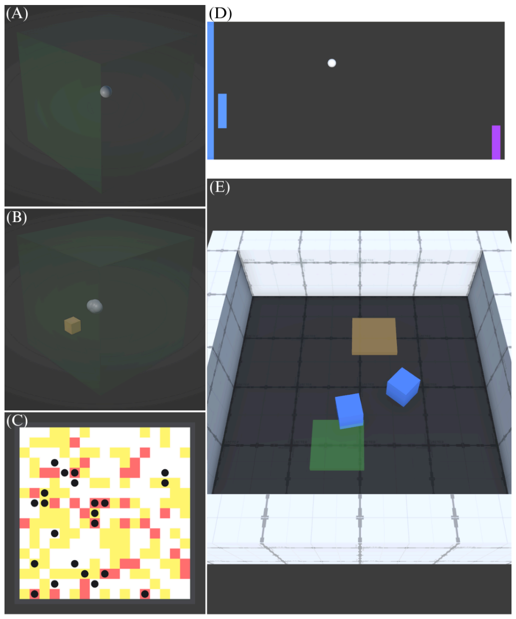

FIG. 2. Screenshots of FABMLA example simulations provided in our GitHub repo.

Additionally, this technique can be used alongside self-play for team sports, such as soccer or volleyball. The controller instantiates 2 agents and randomizes the location of two pressure plates (See FIG. 2E). The agents can observe their location and rotation and see the objects in their field of view (through the use of Unity3D's RayCasts).

E. *Assessing FABMLA*

To assess the quality of FABMLA, we recorded its (1) *consistency*, (2) *applicability*, and (3) *accessibility*. Consistency is crucial for a ML simulation—an inconsistent simulation is not reliable and can not be applied to the real world. To measure consistency, we recorded 5000 episodes of three out of our five example simulations (B, D, E) and their result. Comparing data between these examples will yield insightful conclusions on the consistency of FABMLA and ML-Agents. Applicability is a major requirement of FABMLA—it enables our framework to be generalized to different fields and expands its use cases. To assess applicability, we discuss the implications of the design of our framework (Section III.B). As ML simulations advance, these features must be accessible—this opens the door to using ML in education, work, or products for free. To assess the accessibility of FABMLA we analyze the process taken to develop and *use* our framework.

III. RESULTS & DISCUSSION

A. *Consistency*

Examples (B) and (E) revealed a near, if not, 100% Success Rate (SR). These tasks involved a *puzzle* (i.e. reaching a target or pressing two plates), thus, the SR

Ex.	MEAN STEP	MODE STEP	STD. DEV.	SUCCESS RATE	TRIALS
(B)	60.995	70	17.815	99.98%	5000
(D)	9110.968	14840	9931.82	74.3%	5000
(E)	501.37	343	342.91	100%	5000

TABLE. 1. Results from 5000 episodes of examples (B), (D), and (E).

demonstrates the agent's ability to complete the task every time. Partnered with a relatively low Standard Deviation (SD) we can conclude that the agents had sophisticated behavior and *extreme* consistency, enabling support for complex simulations. Example (D) was a 2D Pong simulation (Section II.D); to declare FABMLA consistent, we expect a high step count and ~50% success rate. The data from this example follows this trend, supporting the consistent nature of FABMLA. Refer to TABLE. 1. for complete data. It is important to note: that this consistency comes from multiple hours of training, it is expected that an agent is inconsistent with insufficient training steps. While we only trained our agents for <1M steps, the consistency reached expectations for complex models, demonstrating ML-Agents' capability as an academic resource.

B. Applicability

The generalizable nature of FABMLA allows it to be used in various subjects. In academia, our framework can model complex social, spatial, competitive, and interactive systems to advance research. FABMLA features, such as ML and visuals, can be utilized to create unique models and methods to analyze results. In education, FABMLA can be used to demonstrate a variety of complex systems in multiple areas of science. The ability of visualization makes FABMLA an easy tool for students to understand complex situations. This is especially useful when parameters are exposed, allowing for a variety of settings, for example, altering initial parameters or rules of a sugarscape model (Example C) and observing results or a physics simulation in which students can strengthen or weaken forces (i.e. gravity or drag) and visualize the changes in the system.

C. Accessibility

FABMLA provides an effective process for creating ABM simulations and automatically managing internal simulation mechanics in a Unity3D environment. Thus, those familiar with the Unity3D GE will find FABMLA straightforward. Those new can leverage the extensive Unity3D online videos and documentation to familiarize themselves with the platform. Additionally, FABMLA's GitHub contains detailed documentation on our framework and its source code, allowing exploration of the inner workings of FABMLA.

Parameters of FABMLA and ML-Agents are exposed in the framework, allowing internal control through Unity3D's in-built Graphical User Interface (GUI) or custom agent(s) and controller. Thus, FABMLA is flexible, allowing the creation of complex models. Furthermore, FABMLA is open-source, encouraging modification of its content to fit modeling needs.

Finally, Unity3D, ML-Agents, and FABMLA are free and low-intensive programs. This encourages those unable to purchase or run demanding platforms to utilize FABMLA, thus, increasing valuable research in ML and creative projects powered by ML.

IV. CONCLUSION

This paper investigated the limitations of current ABM development platforms and the useful features of GE. We proposed a Framework for Agent-Based modeling with Unity's ML-Agents (FABMLA) to counter these drawbacks. FABMLA creates a 3D, ABM training environment and offers three key advantages: (1), its ML-Agents implementation provides deep learning capabilities. Additionally, the properties of FABMLA and ML-Agents are exposed, encouraging the creation of consistent, complex, environment-reactive, and interactive agents. (2), FABMLA supports generalization—its abstract nature obfuscates internal behavior, enabling compatibility with any action. (3), its integration with Unity3D provides unique features, allowing utilization of in-built Unity features (3D rendering, AI pathfinding, and physics engines).

While FABMLA brings advantages, there are some caveats. FABMLA requires Unity3D, therefore knowledge of Unity3D and C# is required. Additionally, due to the limited time of this project, we were unable to pursue certain features of FABMLA (such as Neural Network swapping or group-based self-play)—these features could be explored in future work. FABMLA is still in an early stage and will likely continue to develop. It currently supports core ABM functions, including agents, controllers, episode management, step synchronization, and ML. Current and future development centers around the following goals:

1. Create a real-time, 3D, visual, interactive environment with ML-powered agents
2. Encourage *generalization* and applications to multiple disciplines
3. Demonstrate complex behavior and compatibility

ACKNOWLEDGEMENTS

The first author would like to give thanks to Pratyush Tripathy and Dr. Lina Kim for their support and insightful feedback on his work. He also thanks his peers and family for their encouragement, provision of advice, and constant support.

REFERENCES

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. Proceedings of the National Academy of Sciences, 99(suppl_3), 7280–7287. https://doi.org/10.1073/pnas.082080899

Cheliotis, K. (2021). ABMU: An Agent-Based Modelling Framework for Unity3D. SoftwareX, 15, 100771. https://doi.org/10.1016/j.softx.2021.100771

Cheliotis, K. (2022). Simulating Common Indoor Crowd Behaviours in 3D Environments Using ABMU. 2022 International Conference on Interactive Media, Smart Systems and Emerging Technologies (IMET), 1–4. https://doi.org/10.1109/IMET54801.2022.9929960

Epifanía-Huerta, O. I.-V., Carlos Perez-Benito, Diego Torres-Murga, Andrés. (2024). Application of Virtual Reality in the Treatment of Claustrophobia in Adolescents. International Journal of Engineering Trends and Technology - IJETT, 72(1), 40–47.

Ghaffarzadegan, N., Majumdar, A., Williams, R., & Hosseinichimeh, N. (2024). Generative Agent-Based Modeling: Unveiling Social System Dynamics through Coupling Mechanistic Models with Generative Artificial Intelligence. System Dynamics Review, 40(1), e1761. https://doi.org/10.1002/sdr.1761

Manson, S. M., Sun, S., & Bonsal, D. (2012). Agent-Based Modeling and Complexity. In A. J. Heppenstall, A. T. Crooks, L. M. See, & M. Batty (Eds.), Agent-Based Models of Geographical Systems (pp. 125–139). Springer Netherlands. https://doi.org/10.1007/978-90-481-8927-4_7

ML-Agents. (2024). [C#]. Unity Technologies. https://github.com/Unity-Technologies/ml-agents (Original work published 2017)

Mls, K., Kořínek, M., Štekerová, K., Tučník, P., Bureš, V., Čech, P., Husáková, M., Mikulecký, P., Nacházel, T., Ponce, D., Zanker, M., Babič, F., & Triantafyllou, I. (2023). Agent-based models of human response to natural hazards: Systematic review of tsunami evacuation. Natural Hazards, 115(3), 1887–1908. https://doi.org/10.1007/s11069-022-05643-x

Possik, J., Asgary, A., Solis, A. O., Zacharewicz, G., Shafiee, M. A., Najafabadi, M. M., Nadri, N., Guimaraes, A., Iranfar, H., Ma, P., Lee, C. M., Tofighi, M., Aarabi, M., Gorecki, S., & Wu, J. (2023). An Agent-Based Modeling and Virtual Reality Application Using Distributed Simulation: Case of a COVID-19 Intensive Care Unit. IEEE Transactions on Engineering Management, 70(8), 2931–2943. IEEE Transactions on Engineering Management. https://doi.org/10.1109/TEM.2022.3195813

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. SIGGRAPH Comput. Graph., 21(4), 25–34. https://doi.org/10.1145/37402.37406

Sobota, B., Pietriková, E., Sobota, B., & Pietriková, E. (2023). The Role of Game Engines in Game Development and Teaching. In Computer Science for Game Development and Game Development for Computer Science. IntechOpen. https://doi.org/10.5772/intechopen.1002257

Yu, S. (2022). Agent-based modelling using survey data to simulate occupancy patterns and occupant interactions for workplace design. Building and Environment, 224, 109519. https://doi.org/10.1016/j.buildenv.2022.109519