

Robot Sumo Simulation:
Milestone 2 Document

Team Members

Adebare Kofo-Abayomi: akofoaabyomi2022@my.fit.edu

Avlynn Wade: awade2024@my.fit.edu

Faysal Zaki: fzaki2023@my.fit.edu

Joshua Oziel Fernández Alvarado: jfernandezal2023@my.fit.edu

Client & Faculty Advisor

Department of Electrical Engineering and Computer Science

Dr. Hanks, thanks@fit.edu

Section 1.0 Progress of Current Milestone

Figure 1. Progress Matrix

Task	Completion %	Adebare	Avlynn	Faysal	Joshua	To do
Implement robot models with corresponding hitboxes.	50%	0%	0%	50%	0%	Create adaptive hitboxes for future scalability
Give robots the ability to interact with the environment and set start and end conditions for a sumo match.	50%	0%	0%	0%	50%	Alternate environments and robot types.
Give exposable inputs for the robot controller.	100%	0%	0%	0%	100%	None
Provide readable full state feedback for RL training.	100%	50%	0%	0%	50%	None
Implement a basic interface for the simulation and RL API.	80%	0%	80%	0%	0%	Finalize Godot-Python connection (TCP).
Research specific RL algorithms that are best suited to the project.	100%	0%	100%	0%	0%	None

Section 2.0 Discussion of Milestone 2 Tasks

Section 2.1 Implement robot models with corresponding hitboxes.

The current robots use hardcoded hitboxes. The team explored and tested various approaches for connection hitboxes, including using skeleton and join nodes in the Godot environment. These approaches allow robots to be composed of different combinations of hitboxes, provided they form one continuous body. However, the team did not make significant progress and other tasks were prioritized. The groundwork for implementing adaptive hitboxes for more complex models has been planned for future milestones.

Section 2.2 Give robots the ability to interact with the environment and set start and end conditions for a sumo match.

The win detection supports any number of robots and correctly selects and displays a winner when the Robot Sumo match ends. Additionally, any ring type or size can be defined by expanding or shrinking the ring area node in the main scene. Implementation of alternate environments and robot types was deferred to the next semester as they are not necessary to reach the project's MVP and are currently out of scope.

Section 2.3 Give exposable inputs for the robot controller.

An imported vehicle controller class was repurposed for the Robot Sumo simulation. However, references to vehicles were specifically tailored for the use of simulation robots. Moreover, the team made the controllers specific for a single robot and added a controller per robot. An exposed field in the editor was added to facilitate toggling between a user-controlled robot and an agent-controlled robot.

For user-controlled robots, inputs were added to the project so both robots can be controlled and inputs were made changeable in the Godot editor. On the other hand, for agent-controlled robots, a method in the controller was added, through which input values can be passed by the agent for every physics tick. The given input was then interpreted and adjusted to fit the controller's convention, then applied as steering and throttle on the vehicle.

Section 2.4 Provide readable full state feedback for RL training.

The team's initial approach for state feedback used Godot scripts to collect robot state and match data and included a Python class that would organize the feedback for RL training. The feedback covered robot positions, rotations, velocities, control values, and the match status (win/loss or draw). This initial implementation provided a good starting point for future development.

After some consideration, the team pivoted to a singular StateFeedback class in Godot that works as an autoload. Meaning that all scripts can feed information into it, which it then compiles into an array to deliver feedback. Like the first implementation, the array contains information on the match status (ongoing, who won, etc.) and a transform state for each robot

(position, rotation, velocities, etc.). The data can then be obtained and sent through a connection script yet to be defined (task for Milestone 3).

Section 2.5 Implement a basic interface between the simulation and RL environment.

To start, a custom Gymnasium environment, “RobotSumoEnv” was created utilizing tutorials from the Gymnasium documentation. This environment implemented core methods required for RL training such as the `step()` and `reset()` functions, as well as the agent’s reward function. Additionally, the Robot Sumo environment was connected to the full-state feedback framework to facilitate the transfer of state information from the simulation to the training interface.

Potential approaches for establishing communication/connection between Godot and Python, including TCP sockets, HTTP, and extension packages, were researched. The actual implementation of this connection is planned for early within the next milestone due to time constraints. In the meantime, RL training demos were created using placeholder data, which was modeled after the information the Godot simulation returns through its state feedback components. These demos are direct reflections for how the actual, fully connected, simulation to RL interface will function.

Section 2.6 Research specific RL algorithms that are best suited to the project.

To begin, the team researched several on-policy and off-policy RL algorithms. The research focused exclusively on model-free methods, which learn a policy from trial and error (interactions with the environment), rather than model-based methods, which learn a model of the environment's dynamics to support decisions. This design choice was made to reduce implementation complexity, and given the available low-cost training runs through AI Panther.

The algorithms investigated included Proximal Policy Optimization (PPO), Soft Actor-Critic (SAC), Twin Delayed DDPG, (TD3), and Deep Q-Network (DQN). Following this research, the team selected PPO, an on-policy method, as the baseline algorithm due to its widespread use and suitability for the project. Additionally, the team selected SAC as a potential off-policy comparison algorithm.

Next, the team explored potential RL libraries that provided implementations of PPO and SAC. The first library researched was Stable Baselines3 (SB3), which supported PPO, SAC, TD3, and DQN. Additionally, its documentation provided information on how to incorporate Gymnasium, making it a suitable choice for the team’s custom environment. Next, the team looked into CleanRL, which provided relatively simple code with shorter algorithms (single-file implementation). However, its documentation was less structured, which the team marked as a major drawback. Following this, the team evaluated RLlib (Ray) which did support the selected algorithms. However, this library is specifically designed for large-scale datasets/multiple machines. Therefore, the team decided to table this library for now but reconsider it for the MARL reach goal. Finally, the team briefly researched PyTorch, as it can be integrated with

Gymnasium. But, using it would require the creation of additional components such as policy networks and training procedures, rather than already implemented algorithms. Therefore, this library would not be feasible given the project's timeline. Overall, the team decided to select SB3 as the final RL library, due to its support for the needed algorithms and easy connection to Gymnasium.

Section 3.0 Discussion of Team Contributions

Section 3.1 Adebare Kofo-Abayomi

Adebare developed the initial full-state feedback implementation, including Godot scripts for collecting robot and match information and a Python class for organizing the data for RL training. My Python class was later used as a reference for the updated feedback class. Adebare also reviewed Joshua's Godot implementation and suggested including linear and angular velocity to describe robot movement.

Section 3.2 Avlynn Wade

Avlynn Wade researched and selected the initial RL algorithms PPO and SAC. Additionally, she investigated suitable RL libraries and selected SB3 for the baseline training runs and RLlib for potential MARL tasks in future milestones. The results of this research were communicated to the team through formal write-ups. Avlynn was also responsible for creating the custom Gymnasium environment, as well as integrating it with the state feedback, provided by the Godot simulation and the Python full-state feedback class. Finally, with assistance from the client/advisor, Avlynn researched potential Godot-Python connections and will implement said connection within the next milestone. Due to this constraint, Avlynn was also responsible for creating the RL training demos using placeholder data.

Section 3.3 Faysal Zaki

Faysal Zaki created development tools which allowed for the easy creation of Sumo bodies. Godot has a suite of tools that already allow developers to mix and match components that enable physics, collision, and other in-engine features from inside the editor. Even then, manually having to recreate each different new physics body for a given sumo is tedious and required many keystrokes, especially for newer members unfamiliar with Godot's editor. So, Faysal created an editor tool which simply hooked up the needed components onto a selected entity with a model component.

Section 3.4 Joshua Oziel Fernández Alvarado

Joshua fully restructured the robot controller to make it adaptable to both human and agent input, generalizing the controller to work in both ways. While working on the controllers, Joshua did research into making Godot step correctly in headless mode, which led to the implementation of

the improved SimulationManager with headless mode compatibility and made it so that it can speed up the simulation for training. Joshua made adaptable win conditions for the robots and defined the Godot StateFeedback class for obtaining information as an autoload and returning the information in an array.

Section 4.0 Task Matrix for Milestone 3

Task	Avlynn	Joshua	Adebare	Faysal
Setup scripted opponent for agent to train with	22.5%	32.5%	22.5%	22.5%
Extend RL training interface to loop through Godot rounds	32.5%	22.5%	22.5%	22.5%
Train an agent until it is competent in a round of Sumo	22.5%	22.5%	32.5%	22.5%
Test different algorithms for RL training to gauge performance	22.5%	22.5%	22.5%	32.5%

Section 5.0 Discussion for Planned Milestone 3 Tasks

Section 5.1 Setup scripted opponent for agent to train with

Write a script for the hardcoded opponent that the agent will train with. The scripted opponent should simply face the agent and drive towards it in an attempt to push it off the ring. The opponent must have some random variance, chance, and alternate behavior modes so that the agent does not memorize a pattern. In addition, the developers must ensure the scripted opponent is easy to defeat by a human but is still capable of beating an untrained agent.

Section 5.2 Extend RL training interface to loop through Godot rounds

Fully connect the RL training interface, specifically finalize the transfer of state information between the RL environment and the Godot simulation. This will ensure that multiple rounds of robot sumo can be run in headless mode, and the simulation can reflect real training inputs. The training interface needs to interact with the simulation to start, end, and switch between rounds while interacting with the selected RL library/algorithm so that training can be conducted. The

results of this training pipeline will be saved in log format to facilitate training on both local devices and AI Panther.

Section 5.3 Train an agent until it is competent in a round of Sumo

Upload necessary files into AI Panther and conduct multiple training runs to train the first agent. The agent's performance will be considered successful if the trained agent is able to consistently defeat the scripted opponent in a round of Robot Sumo. Training progress will be supervised and checked through output to verify consistency and keep track of progress. Feedback, match conditions, and settings will be adjusted accordingly for the goal to be met.

Section 5.4 Test different algorithms for RL training to gauge performance

Retrain the base agent on the scripted opponent multiple times, utilizing different RL algorithms and track performance. The different algorithms will include SAC, and continuous space DQN. The team will compare training runs, time to train, win percentage, and other metrics to compare RL algorithms. This will help the team decide on the final RL algorithm to move forward with in future milestones.

Section 6.0 Dates of Meetings with Faculty Advisor (and Client) During Milestone 2

Meeting Purpose	Date and Time
Discussed the planned Milestone 2 tasks and the breakdown of the work within the team. Created more structured Github push request system, as well as new ways to report progress.	September 30 th , 2026 – 12:00 PM
Discussed the current progress of the Milestone 2 tasks. Outlined what each team member should accomplish before the deadline.	October 15 th , 2026 – 12:00 PM

Section 7.0 Client and Faculty Advisor Feedback for Milestone 2

The team's progress overall on this milestone looks good. However, there is still a lot to be done before the next milestone. I would advise prioritizing that everyone has well defined tasks and that the team follows branch-commit-PR-review-merge GitHub workflows to facilitate productive collaboration.

Section 7.1 Implement robot models with corresponding hitboxes.

This should be prioritized for the next milestone. A generic hitbox computation module would provide extensibility to different physical robot geometries, which will be important for simulating matches with teams of heterogeneous robot types.

Section 7.2 Give robots the ability to interact with the environment and set start and end conditions for a sumo match.

Teams and rings of any size are perfect for an MVP.

Section 7.3 Give exposable inputs for the robot controller.

This is a really good design. I like the decoupling between robot dynamics and controllers, allowing different types of RL agents or humans to easily be swapped into the running simulation.

Section 7.4 Provide readable full state feedback for RL training.

This seems reasonable for an MVP. For the finished project, it would be nice to have a well-defined observation interface that can be implemented by other classes beyond StateFeedback. This would enable easy integration of different observation models such as partial state feedback or various simulated sensing models (e.g., ultrasonic, light, and touch sensors, which are standard for physical robots).

Section 7.5 Implement a basic interface between the simulation and RL environment.

Good work implementing the Gymnasium environment interface. Hopefully, the team can figure out a way to interface Godot and Python without needing to use TCP or HTTP (though I think HTTP is a reasonable and fairly standard backup option).

Section 7.6 Research specific RL algorithms that are best suited to the project.

I think PPO is the right choice for an RL baseline algorithm. It should work well for the continuous state and action spaces of our robots, and the fact that it is implemented in SB3 makes things even better! Complex, multi-agent actor-critic setups are best left as reach goals.