Software Requirements Specification for TPG: Tangled Program Graphs

Team 3, Tangle
Calvyn Siong
Cyruss Allen Amante
Edward Gao
Richard Li
Mark Angelo Cruz

March 30, 2025

Contents

Revision History

Date	Revision	Notes
10/9/2024	1.0	Intial draft of SRS document.
3/11/2025	1.1	Modified functional requirements based on feedback
3/30/2025	1.2	Revised non-functional requirements

1 Purpose of the Project

1.1 User Business

The Tangled Program Graphs (TPG) framework is an alternative approach to reinforcement learning (RL) that leverages evolution-based techniques instead of the widely used deep neural networks (DNN). In traditional RL, agents learn through trial and error by generating actions and receiving rewards. The DNN approach requires extensive computational resources, often involving thousands of GPUs, which can be expensive and inefficient. In contrast, TPG aims to provide a more cost-effective and resource-efficient solution, with the long-term goal of embedding this evolution-based learning directly into hardware, reducing dependency on large-scale computational infrastructure.

Currently, TPG has been tested primarily in fully observable, stationary mini-game environments, which are not representative of the dynamic and partially observable nature of real-world scenarios. This limitation presents a challenge, as the real-world problems TPG is meant to address are much more complex and constantly changing. To safely and effectively evolve TPG's capabilities, it is crucial to test the framework in advanced simulation environments, like MuJoCo, which can better mimic real-life dynamics in a controlled and risk-free manner.

Additionally, TPG's codebase has been developed by graduate students primarily focused on research output, often neglecting the software engineering practices necessary for creating a robust and maintainable open-source framework. Without standardized practices such as unit testing, continuous integration/continuous deployment (CI/CD), and architectural guidelines, TPG's long-term goal of becoming a widely adopted, open-source framework could be hindered. This project seeks to address these gaps, ensuring that TPG can scale, evolve, and attract contributions from other researchers and reinforcement learning enthusiasts in a standardized and efficient manner.

1.2 Goals of the Project

1. **Enabling Software Engineering Standards**: We aim to create a seamless and standardized process for all contributors to the Tangled Program Graphs (TPG) framework. This includes simplifying the onboarding process, establishing clear contribution guidelines, and build-

ing a robust, scalable architecture that is open for extension but closed for modification. By doing so, we ensure that future development is both collaborative and sustainable, allowing for a consistent quality of contributions while maintaining the integrity of the core framework.

2. **Physics Engine Integration**: We also seek to enhance TPG's ability to handle more complex, real-world-like scenarios. This will be achieved by integrating TPG with the MuJoCo physics engine, which allows for experimentation in dynamic and partially observable environments. By expanding the testing environments, we can evolve TPG's capabilities beyond its current limits, ensuring it is better suited for real-world applications.

2 Stakeholders

2.1 Client

- **Description:** Dr. Stephen Kelly and his research team. They are responsible for the development and overseeing changes made to the TPG framework.
  - **Role:** Main decision-makers on the scope and direction of the project. They will provide feedback on the project and approve the final implementation.

2.2 Customer

N/A

- **Reasoning:** This project is not a commercial product made for an intended customer audience in mind, but an extension to an existing research project.

2.3 Other Stakeholders

- **Description:** External French research team (GEGELATI). They are an organization that is adopting the TPG algorithm for research purposes differing from Dr Kelly's project.

  - **Role:** Access to a more robust and maintainable framework for testing RL algorithms in high-fidelity simulators like MuJoCo. They may also share findings that may be beneficial to Dr Kelly's research as well.

- **Description:**The broader reinforcement learning research community. This includes research organizations and teams working with evolutionary algorithms and genetic programming who may be interested in Dr. Kelly's research.

  - **Role:** End-users who will benefit from the enhancements made to the TPG framework, especially in the form of improved documentation, testing, and real-world applicability. The framework may be utilized or referenced in other research and contribute to further development of the field.

2.4 Hands-On Users of the Project

- **Description:** Researchers and developers working directly on the TPG framework. This includes Ph.D. students and collaborators involved in testing and developing within the codebase.

  - **Role:** Active users who will interact with the code, run experiments, and test integrations (e.g., with MuJoCo). They are responsible for ensuring that the system works as intended and fits the project's research goals.

2.5 Personas

- **Researcher Persona:** Dr Stephen Kelly is a postdoctoral researcher focusing on genetic programming in predictive control environments. He is interested in how emergent forms of memory and hierarchy allow digital evolution to build programs in complex, multi-task environments, which he works on through his project - the TPG framework. His goal is to evaluate and test TPG in complex environments, such as MuJoCo, to further his research in the field of RL. Dr Kelly is driven by the prospect of publishing impactful research and contributing to the RL community. However, he finds the lack of user-friendly documentation in TPG frustrating, as it makes setting up experiments

and testing difficult. Despite this, Alex is committed to using TPG to demonstrate how genetic programming can outperform or complement traditional RL methods. He regularly uses technologies such as C++ for his work, and utilizes Gitlab for version control. He also aims to keep his project as an open source framework to allow others to benefit from his research.

- **Developer Persona:** Oliver is a software developer with a background in C++ and knowledge in the fundamentals of software engineering best practices. Currently, Jamie is working with Dr. Kelly's research group to improve the TPG framework. His primary focus is to introduce modern software engineering principles, such as continuous integration, automated testing, and comprehensive documentation, to enhance the maintainability and scalability of TPG. Jamie is highly motivated to refactor the TPG codebase to make it more user-friendly for other researchers and contributors. However, he will have to balance the challenge of modernizing the codebase without disrupting the existing functionality or performance of TPG, especially without the full context of the system when he starts to work on it. Additionally, he will also be contributing to work on extending TPG to be integrated with Mujoco, an advanced physics simulator. He will be doing research on the best ways to create an integration between the two systems.

2.6 Priorities Assigned to Users

N/A

2.7 User Participation

- Hands-on users (researchers and developers) will be actively involved throughout the project lifecycle, providing feedback on codebase refactoring, testing, CI development and the MuJoCo integration.
- Frequent discussions with the Dr. Kelly and the research team during the development process, with periodic reviews at project milestones.
- The reinforcement learning community or other research organizations may provide indirect feedback post-development through research papers, informal discussion, and open-source contributions

2.8 Maintenance Users and Service Technicians

The future maintainers of the TPG framework will likely be within Dr. Kelly's research group or external contributors from the open-source community, will handle ongoing updates, bug fixes, and enhancements. User documentation will be provided to help improve the maintainability of the project.

3 Mandated Constraints

3.1 Solution Constraints

The solution must strictly adhere to modern software engineering best practices, including comprehensive documentation, robust testing, and the implementation of continuous integration and continuous deployment (CI/CD) pipelines. The implementation must be in C++ to maintain compatibility with the existing Tangled Program Graphs (TPG) codebase. The system must integrate seamlessly with MuJoCo, a physics engine developed by Google DeepMind, to enable testing in dynamic and partially observable environments. All code contributions must comply with open-source licensing agreements and standards to facilitate community collaboration. Additionally, the solution should minimize computational resource requirements to support deployment on embedded systems with limited processing capabilities.

3.2 Implementation Environment of the Current System

The current TPG framework is implemented in C++. MuJoCo is originally written in C, and its core API is in C, so interacting with C++ is a good option. Development will occur in a UNIX-like environment with as much standardization as possible. Virtualization environments such as Docker, VSCode Dev Containers, WSL, and VMs will be considered based on the ease of environment standardization across the team. OpenGL will be used to display the RL simulation, so display compatibility with the virtualization environments will need to be considered.

3.3 Partner or Collaborative Applications

The solution should maintain or improve compatibility with applications or tools used by secondary stakeholders, such as the GEGELATI team, who utilize TPG for their research purposes. This includes ensuring interoperability with their existing frameworks and facilitating data exchange where necessary.

3.4 Off-the-Shelf Software

The project may utilize off-the-shelf software components to enhance functionality and reduce development time. Potential components include physics engines like MuJoCo and unit testing libraries such as Catch or Google Test. All off-the-shelf software used must be compatible with the project's open-source licensing and should not introduce dependencies that conflict with the solution constraints. See section 19 for further analysis of potential off-the-shelf software integrations.

3.5 Anticipated Workplace Environment

Development will utilize standard C++ development tools, compilers, debuggers, and possibly integrated development environments (IDEs). The team will work in a collaborative environment, employing version control systems like GitHub to manage code contributions and facilitate communication among team members. Meetings with our supervisor, Dr. Kelly, and amongst the team will be default in-person. Modes of communication have been discussed in further detail in the Development Plan document.

3.6 Schedule Constraints

The project must be completed according to the deadlines listed in the capstone course outline, aligning with team availability and academic schedules. Key milestones include the early implementation of the CI/CD pipeline to facilitate subsequent testing and development. Achieving the target code coverage is critical and should be scheduled with sufficient time for iterative testing. Integrating MuJoCo should be prioritized to allow ample time for experimentation and evaluation. Time must be allocated for conducting experiments and analyzing results before project completion.

3.7 Budget Constraints

The project operates under tight or nonexistent budget constraints, typical of academic projects. Any costs associated with software licenses must be minimized or covered through academic licensing options. Expenditures on additional hardware or paid tools are constrained and must be justified or avoided if possible.

3.8 Enterprise Constraints

The solution must comply with university policies, academic integrity standards, and any guidelines provided by the supervising faculty. All development must adhere to the policies regarding intellectual property and open-source contributions.

4 Naming Conventions and Terminology

4.1 Glossary of All Terms, Including Acronyms, Used by Stakeholders involved in the Project

Agent An autonomous intelligence system performed to do specific tasks without human assistance.

Environment This is referred to the external system that the agent interacts with. The environment can provide information such as the current state and reward, and the agent can provide the environment with its action.

Tangled Program Graphs (TPG) A framework currently being developed under Dr. Stephen Kelly that will help modular programs apply genetic programming principles to embedded systems.

Deep Neural Networks (DNNs) A machine learning technique that trains an agent to complete difficult tasks that would be difficult to do using conventional programming.

Genetic Programming A technique used to evolve programs, that first start a population of "unfit" agents. Through RL, agents are destroyed,

kept, and mutated to evolve into a more suitable population. This is continued until the population reaches its desired fit.

Reinforcement Learning (RL) A machine learning technique that utilizes a reward-and-punishment system towards agents, providing a reward for a correctly done task and a punishment for incorrectly done tasks.

Multi-Task Reinforcement Learning (Multi-Task RL) A type of reinforcement learning in which agents are learning multiple tasks at the same time.

Policy A strategy that a particular agent uses to complete a specific task. This is also known as agent behaviour.

MuJoCo A free and open-source physics engine created by Google DeepMind that assists in facilitating research and development in areas such as robotics (https://mujoco.org/).

Phylogenetic Learning Also known as Policy Search, this is a class of reinforcement learning in which agent-environment interactions are episodic. The policy becomes updated as a whole following the final episode outcome.

Temporal Credit Assignment Problem Actions with neutral or negative rewards may still contribute to a successful outcome.

Stationary Environments The transition function does not change over time.

Non-Stationary Environments Transition function changes over time. e.g. video games get harder the longer you play (physics of the world change)

Fully-Observable Environment The state contains all information required to make action decisions, e.g. Chess.

Partially-Observable Environment The state provides partial world-view, e.g. first-person perspective video games.

5 Relevant Facts And Assumptions

5.1 Relevant Facts

The Tangled Program Graphs (TPG) framework is currently implemented in C++ and is utilized for genetic programming in reinforcement learning (RL) environments. The existing codebase lacks comprehensive documentation, testing suites, and continuous integration/continuous deployment (CI/CD) pipelines, which hinders its usability and maintainability. Additionally, the integration of MuJoCo, a physics engine developed by Google DeepMind, is intended to enhance the TPG framework by enabling testing in dynamic and partially observable environments.

5.2 Business Rules

All code contributions must comply with open-source licensing agreements, such as the MIT license, to ensure that the framework remains accessible to researchers and developers. Secondly, any modifications to the codebase must be accompanied by comprehensive documentation and testing to maintain high standards of quality and usability, meeting the requirements of rigorous scientific research. Additionally, the project must implement a continuous integration/continuous deployment (CI/CD) pipeline to automate testing and deployment processes, ensuring that all changes are validated before integration into the main codebase.

5.3 Assumptions

It is assumed that all team members possess a working knowledge of C++ and are familiar with software engineering best practices. Furthermore, it is expected that the necessary computing resources and development tools will be available and accessible to the team throughout the project duration. The integration of MuJoCo into the TPG framework is presumed to be feasible without encountering insurmountable licensing or technical impediments. Additionally, it is anticipated that external contributors from Dr. Kelly's research group may become involved after the development phase, necessitating clear guidelines and documentation for collaboration.

6 The Scope of the Work

6.1 The Current Situation

For the scope of this capstone, there are two main areas of focus: Software Engineering practices and physics engine integration. Both of these have current states which will be improved upon over the course of this project.

6.1.1 Software Engineering Practices

The TPG (Tangled Program Graph) project is currently managed with basic software engineering practices. The codebase is hosted on GitLab, and Dr. Kelly's research group uses Git branches to separate and manage their work. This current setup does allow for parallel work and multiple contributors. However, several key practices are missing:

- Unit Testing: There are no unit tests in place. This absence means that code changes are not systematically validated, increasing the risk of introducing bugs and regressions.
- Continuous Integration/Continuous Deployment (CI/CD): The project lacks automated pipelines for building, testing, and deploying code. Without CI/CD, integrating changes can be time-consuming and error-prone.
- Pull Request Templates and Standards: There are no standardized templates or guidelines for pull requests, leading to inconsistencies in code reviews and collaboration.
- Open Source License: The project has not yet adopted an open-source license, which can deter external contributions and limit the software's usage.
- Issue Management: There is no formal system for tracking bugs, feature requests, or tasks, making project management less efficient.

The current structure of the TPG codebase may not be optimized for use as an open-source library. Current researchers need to run shell scripts as the entry point, which can be a barrier to entry for those unfamiliar with the system. Additionally, MacOS with ARM based chips may have extra

difficulty in onboarding to this project due to the many dependency conflicts in this current state. This approach limits accessibility and may discourage potential users and contributors. More research into code structure and analysis of how other open source libraries allow for their frameworks to be integrated into open source contributor workflows should be studied.

6.1.2 Physics Engine Integration

The TPG framework has been validated using OpenAI Gym's classic control problems such as CartPole, Acrobot, and Pendulum. These environments are stationary, meaning their transition functions (the rules determining the next state given a current state and action) do not change over time. In contrast, real-world environments are typically non-stationary. Their transition functions can evolve due to external factors, requiring agents to adapt continuously. Currently, the TPG framework needs to evolve and be adapted to more dynamic environments.

6.2 The Context of the Work

6.2.1 Software Engineering Practices

- Unit Testing: Focused on improving code quality, unit tests ensure that individual pieces of the codebase work as expected.
- CI/CD Pipelines: The integration and automation of software processes, such as building and testing, will allow for continuous integration and deployment of changes, ensuring the project remains robust as it scales.
- Software Architecture: This examines how users interact with TPG, the use of shell scripting, and how the framework is structured using API development and object-oriented programming (OOP) principles. It emphasizes improving design principles for a better developer and user experience.
- Documentation: This is critical for onboarding new developers, ensuring explainability, and providing clear, thorough project documentation.

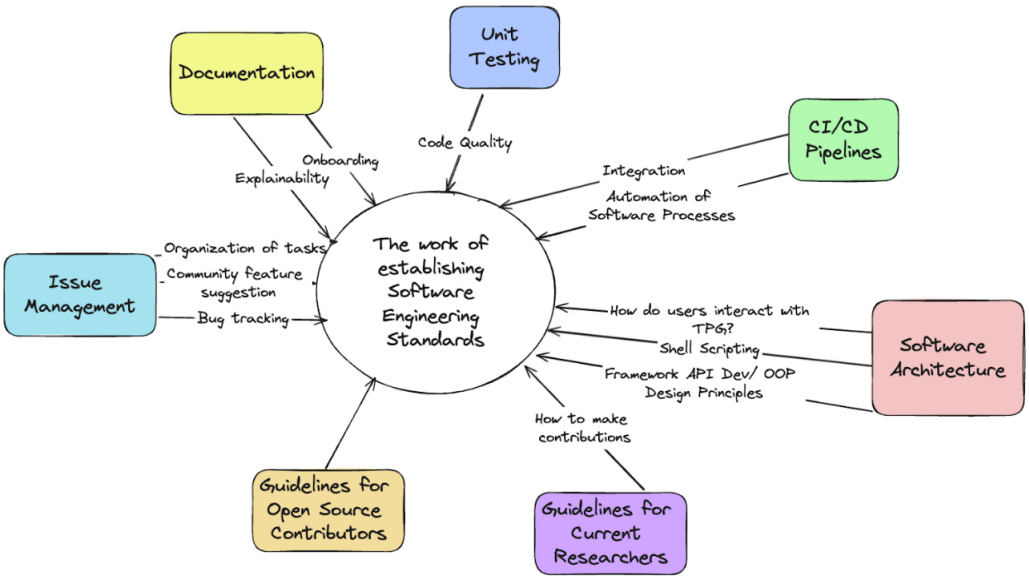

Figure 1: System Context

- Issue Management: Introducing a formal system to track bugs, feature requests, and community suggestions will help organize tasks and streamline project development.
- Guidelines for Open Source Contributors: Establishing clear guidelines will provide a roadmap for external contributors to participate in the project, increasing collaboration and contributions.
- Guidelines for Current Researchers: This component covers how researchers and developers within the project can contribute effectively, ensuring consistency and alignment with the project's goals.

6.2.2 Physics Engine Integration

- TPG (Tangled Program Graphs): TPG is a reinforcement learning framework. In this context, its role is to act as the core system that will be tested and integrated with dynamic environments. Establishing

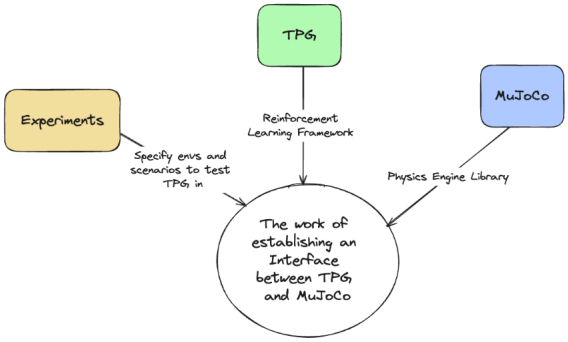

Figure 2: System Context

the interface between TPG and MuJoCo will allow the TPG framework to be tested in physically realistic simulations.

- MuJoCo (Multi-Joint dynamics with Contact): MuJoCo is a high-performance physics engine used for modeling and simulating dynamic systems. In this diagram, it represents the external library that provides the physics-based environments required for testing TPG. The interface with TPG will enable the reinforcement learning agents created using TPG to interact with complex, real-world-like physics simulations provided by MuJoCo.

- Experiments: Experiments define the environments and scenarios where TPG will be tested. This component represents the experimental setups that specify the parameters for evaluating TPG's performance in various MuJoCo-based scenarios. These experiments are critical for determining how well TPG adapts to different physics-based tasks, environments, and scenarios.

6.3 Work Partitioning

Table 1: Work Partitioning Table

Event Name	**Inputs**	**Outputs**	**Summary**
Contributor wants to merge code changes they made	New changes (code that was modified in a PR)	New code is successfully integrated with the main code	CI - Continuous Integration practices into the repo, ensuring all devs can make changes in a seamless manner and be up to date while concurrent work is occuring
Contributor wants to evaluate the code they wrote	New code blocks that are written by a contributor	Test functions are created to evaluate new code	Automated tests are generated for new code blocks that are written ensuring code robustness
Contributor wants to onboard and use TPG	N/A	Contributor is able to run the framework	Seamless onboarding experience that allows a new contributor/user to get up and running

Continued on next page

Table 1 – continued from previous page

Event Name	Inputs	Outputs	Summary
Contributor wants to perform an experiment to test the TPG framework in MuJoCo	TPG, MuJoCo	Functioning simulation of an experiment integrated with a physics engine	New experiments to be conducted in more realistic scenarios evolving the development of this reinforcement learning framework
Contributor wants to get visual results of from test data	TPG, MuJoCo	Graphs of the recent experiment	Evaluation of the experiment is crucial to improving the framework and allowing it to become good at multi task reinforcement learning tasks

6.4 Specifying a Business Use Case

Here are process flow diagrams for both Software Engineering Practices and Physics Engine Integration.

6.4.1 Software Engineering Practices

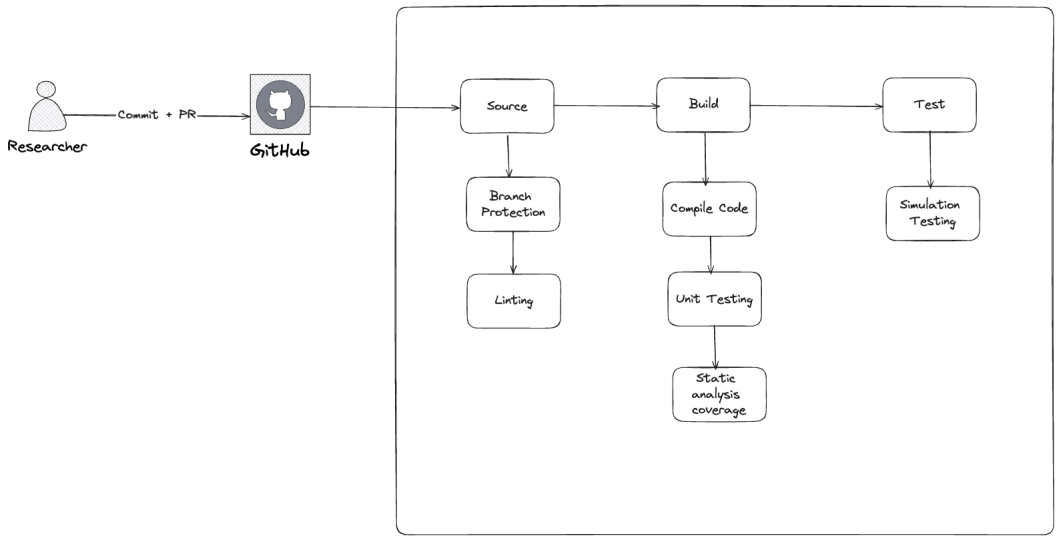

Figure 3: GitHub Actions CI/CD Pipeline Process Flow Diagram

6.4.2 Physics Engine Integration

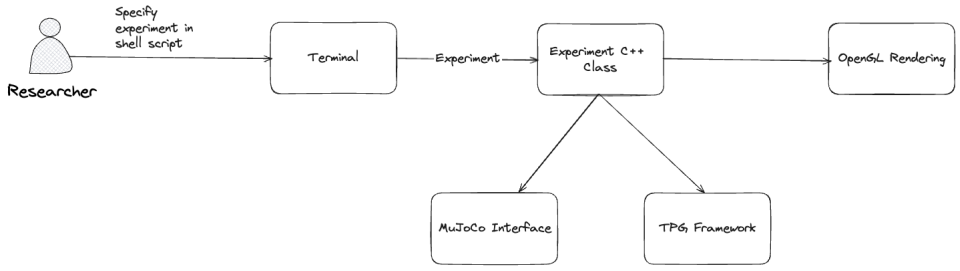

Figure 4: MuJoCo Integration Process Flow Diagram

7 Business Data Model and Data Dictionary

7.1 Business Data Model

7.1.1 Software Engineering Practices

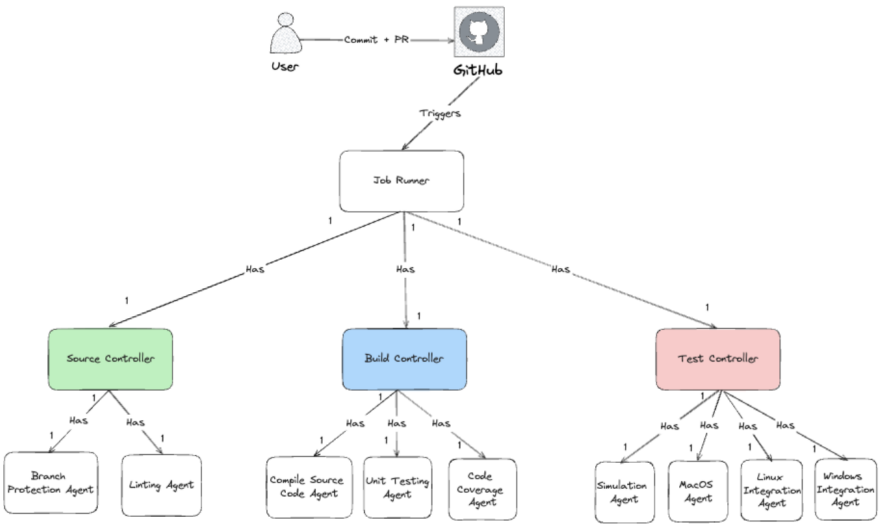

Figure 5: Data Model

Figure 3 illustrates a UML data model of a traditional CI/CD pipeline implemented using GitHub Actions. The architecture follows a master-slave design pattern, where the pipeline steps are defined in a YAML configuration file. For each step in the pipeline, a Controller (master) class orchestrates the process by delegating tasks to Agents (slaves) that handle specific actions. This design is commonly used in CI/CD pipelines because it reflects the sequential nature of the CI/CD process—each step must wait for the previous one to complete before proceeding.

7.1.2 Physics Engine Integration

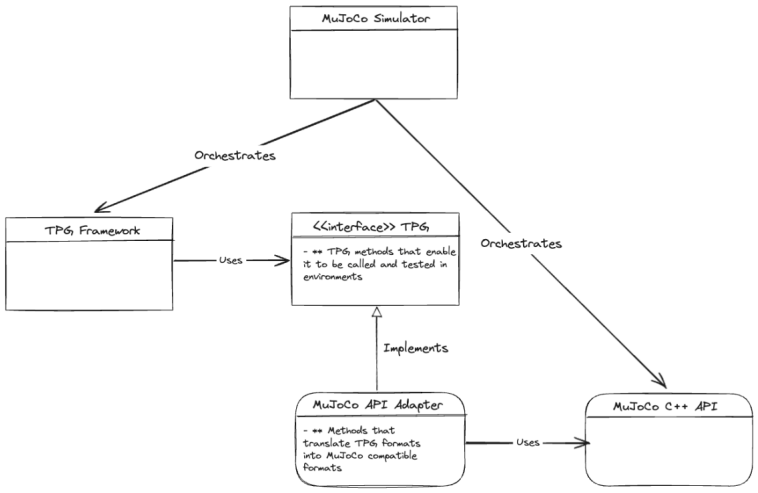

Figure 6: Physics Engine Class Diagram

Figure 4 presents a simplified UML data model of the interface being developed to integrate the TPG framework with MuJoCo. The primary objective is to adapt how the TPG framework handles state changes into a format that MuJoCo can interpret. To achieve this, we are utilizing the Adapter design pattern to translate incompatible formats and ensure compatibility between the two systems.

7.2 Data Dictionary

Table 2: Data Dictionary Table

Name	Content	Type
Job Runner	Orchestrates all the jobs for the GitHub Actions CI/CD Pipeline	Class
Source Controller	Responsible for orchestrating Branch Protection rules and Linting during the Source stage of the CI/CD Pipeline	Class
Branch Protection Agent	Responsible for enabling branch protection on the repo to prevent accidental integrations on the main branch	Class
Linting Agent	Static analysis on the code and cleans it up	Class
Build Controller	Responsible for orchestrating the Build stage of the CI/CD Pipeline	Class
Compile Source Code Agent	Compiles the code to prepare it for validation	Class
Unit Testing Agent	Runs through all the unit tests in an automated fashion	Class

Continued on next page

Table 2 – continued from previous page

Name	Content	Type
Code Coverage Agent	Static analyzes the coverage from the unit tests, usually needs to be above 85%	Class
Test Controller	Responsible for orchestrating the Test stage of the CI/CD Pipeline	Class
MacOS, Linux, Windows Agents	Containers for each OS to test the integration of the framework in each environment (industry standard)	Class
Simulation Agent	Responsible for integration testing to ensure the new changes are robust and don't break	Class

8 The Scope of the Product

8.1 Product Boundary

Table 3: Product Boundary

In Scope	Out of Scope
CI/CD Pipeline: Software Engineering practices need to be integrated into the code base	Ongoing research experiments with TPG: Since Dr. Kelly and his grad students are still developing TPG, the scope of our capstone won't encompass current efforts

Continued on next page

Table 3 – continued from previous page

In Scope	Out of Scope
Interface between MuJoCo + TPG: Development of the interface that enables testing and evaluation of TPG in a physics engine must be established	
Code refactoring: Reorganizing code structure to make it suitable as a C API to allow open source contributions/seamless integrations into other projects	

8.2 Product Use Case Table

Table 4: Product Boundary

User	Use Case
Researchers in Dr. Kelly's research group	- When researcher make changes to the code base, they are able to validate their logic through unit testing, integration testing in the CI/CD pipeline - Perform experimentations using TPG in a physics engine, to further improve the reinforcement learning framework - Access the code not by only using shell scripts, but through the C API - Share how the integration is done in an Open Source manner to allow other community members to access the code - Issue management to organize who is working on what currently
Continued on next page	

Table 4 – continued from previous page

User	Use Case
Open Source Contributors	- Access and embed the TPG framework into their own applications - Make changes to the TPG framework in an automated manner - Suggest changes to the framework in an standardized way

8.3 System Boundaries

8.3.1 GitHub Actions Pipeline

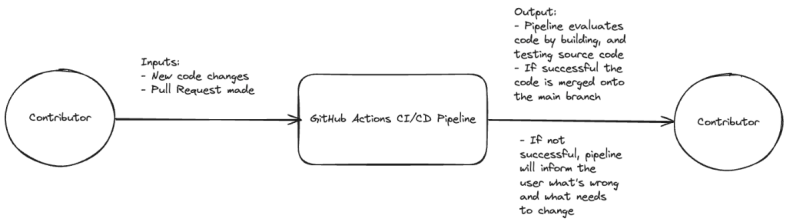

Figure 7: GitHub Actions Pipeline System Boundary

8.3.2 Physics Engine Integration

Figure 8: MuJoCo Physics Integration

8.4 Formalized Math and Application Flow

8.4.1 Software Engineering Practices

Definitions: Let C be the set of all code changes, T be the set of all tests, S be the set of all test successes, E be the set of all test failures

Functions:

1. $Success : C \cap T \rightarrow S$
2. $Failure : C \cap T \rightarrow E$

8.4.2 Physics Engine Integration

Definitions: Let T represent the TPG framework, M represent the MuJoCo framework, E represent the experiment

Relation:

1. $R \subseteq E \cup (T \cap M)$

9 Functional Requirements

9.1 Functional Requirements

- **FR-1:** The system shall provide an interface for configuring, executing, and monitoring reinforcement learning experiments within MuJoCo environments.
  - **Rationale:** This is a requirement brought on by the stakeholders to have an interface that integrates TPG with the MuJoCo environment.
- **FR-2:** The system shall be able to visualize an experiment running from test data.
  - **Rationale:** Visualizing the experiment from test data will allow for verification of the simulator and interface compatibility.
- **FR-3:** The system shall have an integrated CI/CD pipeline.

  - **Rationale:** To have a good workflow environment for the development of the project, a CI/CD pipeline is a beneficial way to do so.

- **FR-4:** The system shall automatically verify code changes to ensure they meet quality standards before integration.
  - **Rationale:** Maintaining code quality is essential for a research framework that will be used and expanded upon in the future.

- **FR-5:** The system shall automatically run test cases once a new code change has been detected in the main branch.
  - **Rationale:** This is to ensure that all test cases pass when a new code update has been implemented and breakage of code or functionality is not all introduced.

- **FR-6:** The system shall adhere to SOLID code principles and the Google C++ Style Guide by enforcing:
  a) Code reviews to ensure compliance with SOLID design principles.
  b) Automated formatting checks using `clang-format` to enforce Google C++ Style Guide rules.
  c) Static analysis tools (e.g., `clang-tidy`) to detect style and structural violations.
  d) Unit test coverage of at least *80%* to maintain code reliability.
  - **Rationale:** This is to allow for easy maintainability of the code. As this is a research project, the people working on the project will be constantly changing. Adhering to these practices minimizes development hassle.

- **FR-7:** The system shall provide a well-defined interface for integrating new experiments from the simulator, ensuring that new experiments can be implemented and deployed without requiring modifications to the core system architecture.
  - **Rationale:** This is important as the system should not be dependent on the experiments being simulated, rather the interface shall be able to adapt easily to newly introduced experiments.

10 Look and Feel Requirements

10.1 Appearance Requirements

- **LFAR1:** The codebase should follow clear and consistent formatting guidelines. This includes proper indentation, descriptive variable names, and potentially inline documentation to enhance readability for both new and experienced project developers.
  - **Rationale:** A consistent code style and formatting increases the maintainability of the project and allows developers to easily understand and contribute to the project.
  - **Phase In Plan:** This requirement will become phased in once most of the project's features are completed, which begins after February 3, 2025.

10.2 Style Requirements

- **LFSR1:** The style of the code, including naming conventions, commenting, and structure, should remain consistent across all modules of the TPG framework.
  - **Rationale:** This consistency will ensure a cohesive development experience and make the project easier to maintain and expand in the future.
  - **Phase In Plan:** This requirement will become phased in once most of the project's features are completed, which begins after February 3, 2025.

11 Usability and Humanity Requirements

11.1 Ease of Use Requirements

- **UH-E1:** The system shall have a clear and comprehensive documentation.
  - **Rationale:** A consistent and comprehensive documentation on the CI/CD pipeline and agent-environment integration ensures ease of usage for all future users regardless of their technical level.

  - **Phase In Plan:** This requirement will become phased in once most of the project's features are completed, which begins after February 3, 2025.

- **UH-E2:** The system shall provide real-time, accurate logging of messages for every action.
  - **Rationale:** Real-time, accurate logging of messages allows users to understand the outcome of their inputs instantaneously, reducing confusion, habilitating more opportunities for debugging and empowers users of their expectations.
  - **Phase In Plan:** This requirement will become phased in midway through the project development process, which should be approximaly around December 2024.

11.2 Personalization and Internationalization Requirements

- **UH-PI1:** The system shall have the flexibility to adjust certain parameters when configuring MuJoCo.
  - **Rationale:** Having the flexibility to change parameters provides users the opportunity to make the system fit to their specific needs and requirements.
  - **Phase In Plan:** This requirement will become phased in once the MuJoCo interface integration is completed, which begins after February 3, 2025.

11.3 Learning Requirements

- **UH-L1:** The system shall have tutorials available to the users.
  - **Rationale:** Having tutorials, simulation examples and guides for the TPG framework and MuJoCo provides guidance to understand user interaction between the entire system and agent-environments.
  - **Phase In Plan:** This requirement will become phased in once the CI/CD and MuJoCo integration are completed, which begins after February 3, 2025.

- **UH-L2:** The system shall ensure that relevant technical concepts and resources shall be accessible by users with ease.
  - **Rationale:** Acknowledging that TPG builds up on several machine learning concepts, listing resources that will complement learning the framework can accelerate user onboarding.
  - **Phase In Plan:** This requirement will become phased in once the CI/CD and MuJoCo integration are completed, which begins after February 3, 2025.

11.4 Understandability and Politeness Requirements

- **UH-UP1:** The system shall accommodate symbols and words that are naturally understandable by all potential users.
  - **Rationale:** In addition to having clear and concise message outputs, avoiding the use of slang, jargons or ambiguous terms minimizes confusion and learning curve for users. This includes using traditional symbols and words that the majority of the users are familiar with in the English language. Usage of resources such as dictionaries are used to verify that something is naturally understandable.
- **UH-UP2:** The system shall not include offensive language in its logging.
  - **Rationale:** Explicit, biased or unnaturally used language may harm any users. It is best for the framework to avoid using any words to ensure a healthy environment. The use of offensive languiage can create an unsafe relationship between the user and the system, compromising user trust. This can be verified by using Content-Moderation APIs available from Google and OpenAI.

11.5 Accessibility Requirements

- **UH-A1:** The system shall accommodate users with different kinds of disabilities.

  - **Rationale:** Supporting features such as text-to-speech, and color blindness assistance systems can allow users with different kinds of disabilities to leverage TPG, fostering an inclusive environment.
  - **Phase In Plan:** This requirement will become phased in after the Rev 0 is completed, which begins after March 18, 2025.

12 Performance Requirements

- **PR-SL:** The system shall execute without introducing significant computational overhead. The integration with MuJoCo must not degrade performance, and any additional computational costs should be minimized through optimization.
  - **Rationale:** Ensures that processing times remain acceptable for real-time experimentation and testing.
  - **Phase In Plan:** During development, in conjunction with the CI/CD pipeline.
- **PR-PA:** The system shall produce accurate and reliable results in simulations and experiments. All calculations must maintain high numerical precision to ensure the validity of results, especially when dealing with complex reinforcement learning tasks and physics simulations.
  - **Rationale:** Ensures the validity of the reinforcement models being developed. Otherwise, inaccurate results can lead to incorrect conclusions and hinder the development of effective learning agents.
  - **Phase In Plan:** During development, starting Oct 14, 2024.
- **PR-RFT:** The system shall be robust against invalid inputs and unexpected environmental conditions. It must handle exceptions gracefully, recover from errors without data loss, and maintain operational stability under varying workloads and stress conditions.
  - **Rationale:** In cases where the agent returns an action that is not defined by the environment, or the agent receives an out-of-bounds state, the system should be able to manage this gracefully.
  - **Phase In Plan:** During development, starting Oct 14, 2024.

- **PR-CR:** The system shall support scaling up to handle multiple simultaneous experiments or large-scale simulations without significant degradation in performance. Efficient resource management is essential to accommodate the demands of complex reinforcement learning tasks.
  - **Rationale:** Efficient resource management is necessary to ensure that the system can accommodate the needs of diverse experiments without performance degradation.
  - **Phase In Plan:** After phase 1 of the project, which starts in February 2025. The scale of the computational requirements will be better understood after this phase.
- **PR-SE:** The system shall be designed for scalability, allowing for easy addition of new features, integration with other environments, and support for future research needs. The codebase should be modular and extensible, facilitating contributions from other researchers and developers.
  - **Rationale:** The system should be designed for scalability to allow for future enhancements to the TPG algorithm and the addition of new environments within MuJoCo. A modular and extensible design will facilitate these updates and ensure that the framework remains relevant.
  - **Phase In Plan:** During development, starting Oct 14, 2024.
- **PR-RR:** The system shall maintain high reliability, with minimal downtime or failures during operation. Continuous integration and automated testing shall be employed to detect and address issues promptly, ensuring consistent performance over time.
  - **Rationale:** High reliability is crucial for maintaining user trust and ensuring that experiments yield consistent results. Continuous integration and automated testing will help identify and address issues promptly, minimizing downtime and ensuring that the system operates smoothly over time.
  - **Phase In Plan:** During development, starting Oct 14, 2024.

13 Operational and Environmental Requirements

- **OE-EPE:** The system will operate in typical computing laboratory environments, which may include personal computers or servers running UNIX-like operating systems. It should function effectively on standard hardware without the need for specialized equipment, supporting researchers and developers in academic settings.
  - **Rationale:** Ensuring compatibility across different hardware allows for broader accessibility and usability among researchers and developers.
  - **Phase In Plan:** Before development. Virtual environments are being considered at this stage.
- **OE-RIA:** The system must interface seamlessly with existing environments, MuJoCo, and potentially other simulation environments. Compatibility requires adherence to their API specifications and proper handling of data exchange protocols. Any dependencies or libraries required for integration must be managed effectively.
  - **Rationale:** Proper handling of data exchange protocols is essential for maintaining the integrity of experiments and ensuring that the TPG framework can be evaluated in diverse settings.
  - **Phase In Plan:** During development, starting Oct 14, 2024.
- **OE-RR:** The system shall be released under an appropriate open-source license (e.g., MIT license) to enable community collaboration. All releases must include comprehensive documentation, installation instructions, and user guides to assist researchers and developers in utilizing and contributing to the framework.
  - **Rationale:** Releasing the system under an open-source license encourages collaboration and ensures accessibility for researchers and developers. Clear documentation and guides help users utilize and contribute effectively, fostering a collaborative environment for framework improvement.
  - **Phase In Plan:** Before development, starting Oct 14, 2024.

- **OE-OSR:** Internal resources and mechanisms should be established to assist current and future members of the research group in utilizing and contributing to the TPG framework. This includes the GitHub repository and associated documentation such as this SRS report, troubleshooting resources, and history of issue tracking.
  - **Rationale:** Improves efficiency of knowledge sharing and ensures continuity of the project as team members change over time.
  - **Phase In Plan:** Throughout lifecycle of the project.

14 Maintainability and Support Requirements

14.1 Maintenance Requirements

- **MS-M1:** If a change is made to the software after the capstone project has concluded, any major changes to the software shall take no more than 20% of the original time to compile and build, assuming the same resources are available.
  - **Rationale:** One of the goals of the project is to ensure a more efficient and smooth software structure. As such 15% is an appropriate reduction in development time to verify the project's efficiency.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.
- **MS-M2:** The software shall have code documentation covering all major functions, classes, and modules. The information shall include descriptions of the component's purpose, inputs, outputs, and any exceptions that may occur.
  - **Rationale:** Detailed and organized code documentation makes it easier for developers when it comes to traceability and verifiability.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.

- **MS-M3:** The software shall maintain regular updates, at least once a month, assuming a change regarding dependent frameworks or libraries has occurred.
  - **Rationale:** Maintaining regular updates is critical to ensure that the project is protected against any security or software bugs caused by the dependent frameworks. Ensuring updates at least once a month minimizes the risk of errors occurring.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.

14.2 Supportability Requirements

- **MS-S1:** The software shall maintain an online repository with resources, documentation and FAQs that can address common user issues or questions.
  - **Rationale:** As new developers are introduced into the project, a repository with all information regarding common issues will help prevent delayed development changes.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.
- **MS-S2:** The software shall continue to support the latest stable version of Linux and other dependent code libraries throughout the duration of the Capstone project.
  - **Rationale:** Keeping the versions of dependent libraries up-to-date allows developers and researchers to continue work on the project without hassle.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.

14.3 Adaptability Requirements

- **MS-A1:** The development team shall utilize a CI/CD pipeline to deliver new software additions.
  - **Rationale:** CI/CD pipelines allow for an automated process to ensure code changes have met standards.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.
- **MS-A2:** The software shall take up to 15 steps to install and execute on major operating systems (Windows, Mac, Linux).
  - **Rationale:** As many users of the system are on different platforms, ensuring easy installation and execution of the project is crucial to avoid unnecessary workarounds.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.
- **MS-A3:** The software shall be able to adapt to newly implemented experiments provided by the training environment.
  - **Rationale:** Adaptability of new experiments is a focal point when it comes to seeing the effectiveness of TPG's RL technique.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.

15 Security Requirements

15.1 Access Requirements

- **SR-A1:** The system shall allow a minimal and necessary amount of contributors to the framework.
  - **Rationale:** Limiting the number of contributors to only Tangle team members and supervisors makes it easy to monitor all code changes integrated into the main codebase.

  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.

- **SR-A2:** A form of secondary authentication shall be enabled for contributors.

  - **Rationale:** Having Multi-factor Authentication (MFA) and Role-Based Access Control (RBAC) enabled restricts the roles of each contributor in the repository, minimizing the chance of unauthorized changes into the codebase.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.

15.2 Integrity Requirements

- **SR-I1:** The system shall have a protection mechanism for the main branch.

  - **Rationale:** Protecting the main branch on Github avoids unauthorized, corrupted code to be merged. With this, pull requests will require at least 1 review from other contributors and resolved comments before merging their changes.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.

15.3 Privacy Requirements

- **SR-P1:** Any data that may include sensitive, personal information shall be obfuscated or anonymized where necessary.

  - **Rationale:** Obfuscating and anonymizing sensitive data ensures that the system strictly follows the law and regulation, decreasing the chance of data breach. This includes any data that must remain private and should not be included in the public Github repository.

- **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.

15.4 Audit Requirements

- **SR-AU1:** All detailed message logging and simulation results shall be available for audit purposes.
  - **Rationale:** Storing all message logs and simulation results allows users to have access to them at a later time for auditing. This ensures traceability for compliance and debugging.
  - **Phase In Plan:** This requirement will become phased in once CI/CD component of the project is completed, which begins after November 13, 2024.

15.5 Immunity Requirements

- **SR-IM1:** The system shall check and search for unauthorized and undesirable code.
  - **Rationale:** Having regular code checks through the CI/CD pipeline guarantees that TPG's code is well-preserved and protected from undesirable code such as viruses, malware, and spyware, minimizing the probability of breaches, or data theft.
  - **Phase In Plan:** This requirement will become phased in once the CI/CD pipeline is completed, which begins after November 13, 2024.
- **SR-IM2:** The system shall have a robust mechanism for handling errors and malfunctions.
  - **Rationale:** A robust mechanism for handling errors and malfunctions ensures the system recovers gracefully from system malfunctions, avoiding downtime for users.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.

16 Cultural Requirements

16.1 Cultural Requirements

N/A - There are no relevant cultural requirements related to this project.

17 Compliance Requirements

17.1 Legal Requirements

- **CRL1:** The TPG framework must comply with the intellectual property policies of the contributing institutions and researchers. It must comply with proper licensing (MIT license) to avoid conflicts with proprietary or open-source software practices.
  - **Rationale:** Ensuring compliance for intellectual property protects the rights of developers and contributors.Licensing allows the TPG framework to be shared and built upon by other researchers and open source developers while maintaining legal clarity regarding the use and distribution of the software.
  - **Phase in Plan:** This requirement has already been phased in the first week after the start of the project, which begun after September 16, 2025.

17.2 Standards Compliance Requirements

- **CRL2:** TPG framework should follow the Google C++ style, a widely adopted C++ coding standard.
  - **Rationale:** Adhering to a well established coding standard helps reduce errors throughout development, increases accessibility, readability, and long-term maintainability of the code.
  - **Phase in Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.

18 Open Issues

The integration of TPG (Tangled Program Graphs) with MuJoCo presents significant challenges, particularly regarding the complexity of the integration process. The current TPG implementation is primarily designed for simpler environments, and adapting it to MuJoCo's physics-based simulations may require substantial modifications to the TPG class and its components. Additionally, the state representation in MuJoCo environments is often high-dimensional and continuous, necessitating adaptations in TPG to efficiently handle these more complex state representations. Furthermore, the action space compatibility poses another challenge, as TPG currently supports discrete action spaces, while MuJoCo typically requires continuous action spaces, which may necessitate changes to the action selection and execution mechanisms within TPG.

Performance optimization is crucial, as the computational demands of MuJoCo simulations combined with TPG's evolutionary approach could lead to significant runtime issues. This necessitates exploring optimization strategies to ensure reasonable training and execution times. Moreover, the memory management strategies employed in TPG, which utilize various memory structures such as working_memory_ and const_memory_, may need to be reassessed to efficiently store and manipulate the potentially large state spaces encountered in MuJoCo environments. The scalability of genetic operations within TPG also requires attention, as the existing mutation and crossover mechanisms may need adaptation to effectively work with the increased complexity of MuJoCo tasks, impacting convergence and learning speed.

Handling partial observability is another critical aspect, as TPG has mechanisms for this, but their effectiveness in the more complex, physics-based scenarios of MuJoCo remains uncertain. Additionally, the integration may complicate debugging and visualization efforts, making it challenging to analyze the TPG's decision-making process. New tools or approaches may be necessary to facilitate effective development and analysis. Finally, the development of comprehensive test suites for the integrated TPG-MuJoCo system will be challenging due to the increased complexity and stochastic nature of the environments, necessitating careful consideration to ensure robust performance across a wide range of scenarios.

19 Off-the-Shelf Solutions

19.1 Ready-Made Products

All-in-one reinforcement learning frameworks that are integrated with software simulations already exist in today's market; however, many are very costly. Nevertheless, the project can use some of these products as a benchmark and make comparisons between them. Here are some current ready-made products.

- NVIDIA Issac Sim
- Amazon SageMaker RL
- Google DeepMind Control Suite
- Unity ML-Agents

19.2 Reusable Components

As an open-source project, the use of reusable components is beneficial to keep the project modular and maintainable. Some libraries that could be utilized as components include:

- Physics Engines such as MuJoCo, and NVIDIA Omniverse
- Unit Testing libraries such as Catch, and Google Test
- Static Analysis and Code Coverage libraries such as Cppcheck, and Gcov
- CI/CD tools such as GitHub actions, GitLab CI/CD

19.3 Products That Can Be Copied

There exists some already open-source reinforcement learning frameworks that have interfaces that can be incorporated with robotic simulators. These products have suitable licenses (such as MIT or BSD) that allow the project to make modifications or take them as a reference.

- MC_MuJoCo

- Blue MuJoCo - Copyrighted by Berkeley Open Arms
- MuJoCo-Sim - Copyrighted by Hoang Giang Nguyen - Institute for Artificial Intelligence, University Bremen

20 New Problems

20.1 Effects on the Current Environment

20.1.1 Software Engineering Practices

1. CI/CD Pipeline Implementation: The introduction of a continuous integration and continuous deployment (CI/CD) pipeline will automate the process of building, testing, and deploying changes. This will significantly streamline development workflows, ensuring that code is consistently tested and integrated into the main branch.

2. Unit Testing: Every new feature or bug fix must be accompanied by unit tests to ensure code quality and prevent regressions.

3. Documentation: Developers will need to document their changes, explaining new features or updates to existing functionality to ensure clarity for current and future contributors.

4. Pull Requests: All pull requests must include detailed comments outlining the changes made, the purpose behind them, and any potential impacts. This will ensure transparency and improve code review processes.

5. Issue Management: Proper issue tracking and management will be enforced, allowing for clear prioritization of tasks, bug reporting, and feature requests. This ensures that project development is organized and scalable.

These new practices will improve collaboration, reduce errors, and increase the maintainability of the codebase. However, they will also introduce some initial overhead, as contributors will need to adapt to these new processes and maintain a higher standard of code quality.

20.1.2 Physics Engine Integration

By integrating the MuJoCo physics engine into the current environment, researchers will have access to more advanced and realistic simulations, allowing them to experiment beyond the OpenAI Gym's Classic Control environments (which are typically limited to simpler OpenGL-based environments). This will enable the testing of reinforcement learning algorithms in more complex, non-stationary environments that better mimic real-world scenarios.

The goal of introducing these changes is to foresee and address potential challenges early in the development cycle. By setting clear software engineering standards and expanding the types of environments available for testing, we minimize the risk of conflicts during implementation. Additionally, these improvements will future-proof the project, making it more maintainable and adaptable to new research directions. However, there is also the need to manage potential conflicts that could arise from transitioning to more complex workflows and environments, ensuring that the adoption of new tools and practices does not create friction in the existing development culture.

20.2 Effects on the Installed Systems

20.2.1 Software Engineering Practices

1. CI/CD Pipeline Development: The new CI/CD pipeline is being developed using GitHub Actions, which poses a challenge as the current codebase is hosted on GitLab. GitLab does not directly support GitHub Actions, necessitating a workaround to integrate the two platforms.

2. Synchronization between GitLab and GitHub: Dr. Kelly's codebase, which is regularly updated on GitLab, must be synced with the capstone project repository hosted on GitHub, where the CI/CD pipeline is being developed. To facilitate this, we have utilized a Git subtree mechanism. However, this migration and synchronization process is not straightforward and may require continuous maintenance to ensure both systems remain aligned.

3. Adjustment for Current Researchers: Researchers familiar with the current workflow on GitLab will need to adapt to the new development

guidelines and processes, including contributing to the GitHub repository and adhering to the CI/CD requirements. This transition will require some initial adjustment in their development practices.

20.2.2 Physics Engine Integration

1. OpenGL Compatibility Issues: In its current state, the OpenGL animations work seamlessly on Ubuntu environments, particularly for less resource-intensive 2D animations. However, on other platforms such as macOS and Windows, OpenGL dependencies conflict with native system libraries, resulting in difficulties in running the animations smoothly.

2. MuJoCo Integration for Cross-Platform Support: The introduction of MuJoCo aims to provide more powerful physics-based simulations compared to the current OpenGL animations. However, solving the issue of platform dependency remains a priority. Making the framework platform-independent, or providing a streamlined onboarding process for different platforms, will be critical in ensuring that researchers and developers can easily evaluate the TPG framework on their preferred systems.

These changes highlight the need to carefully manage the interfaces between the new system (GitHub CI/CD and MuJoCo integration) and the existing systems (GitLab repository and OpenGL-based animations). Synchronizing code between GitHub and GitLab introduces complexity and potential for conflict, especially with regular updates on both platforms. Likewise, transitioning from OpenGL to MuJoCo must account for platform-specific challenges, as cross-platform compatibility is essential for making the framework accessible to a broader range of users. Identifying and addressing these conflicts early will be crucial for ensuring a smooth integration of the new development with the existing systems.

20.3 Potential User Problems

1. Refactoring to a C API: As we transition the TPG framework to a C API, existing users, particularly researchers who are accustomed to accessing TPG via shell scripts, may experience difficulties adapting to the new interface. This change could disrupt their current workflow

and require additional effort to modify their methods of interaction with the system.

2. Onboarding Challenges with CI/CD: The introduction of a CI/CD pipeline could pose onboarding challenges for developers, especially those who may not have prior experience with continuous integration systems. Debugging issues that arise from the pipeline could divert their time and focus away from core development tasks, slowing down overall productivity.

3. Increased Burden of Testing and Pipelines: Researchers who are used to more informal development practices may find the requirement to write unit tests and follow a full pipeline process (testing, building, and deploying) burdensome and time-consuming. This could lead to frustration, as they may feel that more time is being spent on meeting software engineering standards than on actual research and development.

Introducing these new systems and refactoring efforts can have unintended consequences on existing users, particularly researchers. If not properly managed, these changes may lead to a decline in productivity or reluctance to adopt the new processes. It is important to identify these potential adverse reactions early and determine whether they can be mitigated through training, better documentation, or phased rollouts. Ensuring that these changes do not overly burden the existing user base is critical for the smooth adoption of the new features and practices.

20.4 Limitations in the Anticipated Implementation Environment That May Inhibit the New Product

1. Adherence to New Code Structure: As we refactor the TPG framework into a more modular and structured codebase, there is a risk that current developers may not fully adopt or follow the new coding standards and structure. This could result in inconsistent contributions and hinder the maintainability of the project.

2. Reality Gap in MuJoCo Experiments: While integrating MuJoCo provides a more powerful environment for testing, there is a concern that

experiments conducted in simulation may not accurately reflect real-world outcomes. This "reality gap" problem could limit the practical application of the results obtained from the MuJoCo simulations.

3. Inadequate Testing Coverage: The current testing framework may not be comprehensive enough to catch all potential code errors or edge cases. Without thorough testing, new bugs and regressions could emerge, potentially leading to unstable releases and unforeseen issues during deployment.

4. CI/CD Incompatibility between GitHub and GitLab: The planned translation of the CI/CD pipeline from GitHub to GitLab poses compatibility challenges. Since GitLab does not natively support GitHub Actions, integrating the pipeline on GitLab could lead to technical incompatibilities or require significant re-engineering of the CI/CD process.

Identifying these potential problems early allows us to address them proactively before they manifest during implementation. By recognizing the risks posed by developer adoption, the reality gap, insufficient testing, and platform incompatibility, we can take steps to mitigate conflicts, ensure smoother integration, and improve the likelihood of success with the new technologies and processes being introduced.

20.5 Follow-Up Problems

N/A

21 Tasks

21.1 Project Planning

- **Develop SRS document:** Initial draft of the project's requirements document to outline the core functionality and goals.

- **Conducting Hazard Analysis:** Analyze potential risks and hazards to the project's success.

- **Developing V&V Plan:** Create a test plan to outline testing and validation procedures.
- **Proof of Concept Demonstration:** Present a basic demonstration of the core functionalities and integration.
- **Design Document Revision:** Formulate first revision of the design document, detailing system architecture and design choices.
- **Revision 0 Project Demonstration:** Showcase the initial system with key features implemented.
- **Create user guide:** Develop user documentation for the core features of the project.
- **V&V Report Revision:** Create a test report to highlight progress and updates in testing and validation procedures.
- **Final Demonstration:** Complete demonstration with finalized features and documentation at the expo.
- **Final Documentation:** Revise and complete final project documentation.

21.2 Planning of the Development Phases

- **Initial Codebase Evaluation and Testing Integration:** Conduct thorough evaluation of the current TPG codebase while getting a better understanding of relevant reinforcement learning concepts to integrate a testing suite (unit tests) to ensure code quality and coverage.
- **Refactor TPG for Continuous Integration:** Implement continuous integration pipelines (e.g., using GitHub Actions) to automate testing and deployment.
- **Design and Develop Interface with MuJoCo:** Design and implement the interface between the TPG framework and the MuJoCo simulator to test agents in a more complex and dynamic environment.

- **Experimentation and Validation:** Conduct tests or experiment with TPG agents in the MuJoCo environment to evaluate performance and behavior, potentially being evaluated against previously established simple models.
- **Documentation and Knowledge Transfer:** Develop comprehensive user documentation for the new features and integration, including usage guides and development decisions.
- **Final Testing and Refinement:** Conduct final testing process to ensure that the TPG framework is stable, reliable, and meets all the initial requirements.

22 Migration to the New Product

The main code of the TPG framework is stored in Gitlab. However, due to the nature of the Capstone project, the team will be migrating the repository as a subtree of the Github repository to implement the two main goals of this project: support software engineering best practices and the integration of the machine learning framework into another agent-environment.

22.1 Requirements for Migration to the New Product

- **MR-R1:** The project's Github and GitLab repositories shall be continuously synchronized, ensuring changes on one version are reflected on the other.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.
- **MR-R2:** Github Actions shall be functional, establishing a seamless support for standard software engineering principles.
  - **Phase In Plan:** This requirement will become phased in once the CI/CD integration is completed, which begins after November 13, 2024.

- **MR-R3:** The agent-environment interface between TPG and MuJoCo shall be functional, ensuring no conflicts with dependencies or existing code.
  - **Phase In Plan:** This requirement will become phased in once Rev 0 is completed, which begins after February 3, 2025.

22.2 Data That Has to be Modified or Translated for the New System

At this point, there is no data that requires modification or translation.

23 Costs

The project's goal for TPG will not accumulate any costs. Throughout the project's duration, the primary tools will be open source frameworks, Github Actions and MuJoCo, all of which are free of charge. This may be subject to change during the project's development if unforeseen challenges arise or the scope is modified.

24 User Documentation and Training

24.1 User Documentation Requirements

- **USD-UD1:** All user documentation shall be on the code repository, with versions provided in PDF format, so that they may be suitable for offline use.
  - **Rationale:** It is important to have documentation available for both online and offline use in the event that online access is unavailable.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.
- **USD-UD2:** Written user guides shall include platform-specific instructions (Windows, macOS, Linux, etc.), system requirements, and troubleshooting for common issues.

  - **Rationale:** These detailed user guides allow for developers to access the project with minimal disruptions.
  - **Phase In Plan:** This requirement will become phased in once Revision 0 of the project is completed, which begins after February 3, 2025.

- **USD-UD3:** Release notes shall accompany each software update and provide a clear summary of new features, enhancements, and resolved issues.
  - **Rationale:** Release notes are crucial to improving traceability and transparency in addition to keeping all code modifications organized.
  - **Phase In Plan:** This requirement will become phased in once the initial development of the project has started, which begins after October 14, 2024.

24.2 Training Requirements

This section is not applicable to the project as no training is required for the users of this software. All necessary information required to run the software will be provided in the corresponding documentation.

25 Waiting Room

At this point, there are no additional tasks or requirements needed for the project's completion. This may be subject to change during the development if unforeseen challenges arise or the scope is modified.

26 Ideas for Solution

At this point, there are no additional tasks or requirements needed for the project's completion. This may be subject to change during the development if unforeseen challenges arise or the scope is modified.

27 Traceability Matrix

Table 5: Traceability Matrix

Requirement ID	System Objective / Justification	Verification Method	Dependencies
FR-1	Integration with MuJoCo for reinforcement learning	System test: Verify data exchange with MuJoCo	Depends on OE-RIA
FR-2	Visualizing experiments for validation	Test: Verify graphical representation of experiments	Depends on FR-1
FR-3	Implementing CI/CD pipeline	Test: Validate automatic pipeline execution on commits	Supports MS-A1
FR-4	Automatic code change verification before integration	Test: Check system builds successfully and code changes meet predetermined standard	Depends on FR-5
FR-5	Automated testing of new code changes	Test: Ensure all test cases run on code update	Depends on FR-3
FR-6	Maintainability through SOLID principles and style guides	Code review, static analysis (clang-tidy), test coverage	Supports MS-M2, MS-M3
FR-7	Adaptability for new experiments	Test: Add a new experiment and validate system integration	Depends on FR-1, OE-RIA
LFAR1	Consistent code formatting and documentation	Code review and static analysis	Supports FR-6, MS-M2

Table 5 – Continued

Requirement ID	System Objective / Justification	Verification Method	Dependencies
LFSR1	Consistent coding style across all modules	Code review and automated formatting checks	Supports FR-6, MS-M2
UH-E1	Comprehensive documentation	Documentation review, user feedback	Supports MS-M2
UH-E2	Real-time logging for debugging	System test: Check logs for accurate real-time messages	Depends on PR-RR
UH-PI1	Parameter customization for MuJoCo	Configuration test: Verify parameter adjustments	Depends on FR-1
UH-L1	Provide tutorials and guides	User test: Validate documentation with new users	Supports UH-E1
UH-L2	Ensure technical concepts are accessible	User feedback and onboarding review	Supports UH-E1
UH-UP1	Use universally understandable symbols and words	UI review and user feedback	Supports UH-E1
UH-UP2	Avoid offensive language in system logs	Log review and automated keyword scan	Depends on UH-E2
UH-A1	Ensure accessibility features (text-to-speech, color blindness support)	Accessibility compliance test	Supports UH-E1
PR-SL	Minimize computational overhead	Performance test: Measure execution time and resource usage	Depends on FR-1

Table 5 – Continued

Requirement ID	System Objective / Justification	Verification Method	Dependencies
PR-PA	Ensure numerical precision for simulations	Validation test: Compare outputs with expected values	Depends on FR-1
PR-RFT	Robustness against invalid inputs	System test: Validate error handling and recovery	Supports FR-5
PR-CR	Scalability to support multiple experiments	Stress test: Run multiple experiments in parallel	Depends on FR-1, FR-8
PR-SE	Scalability for future extensions	Code review: Check modularity and extensibility	Supports MS-A1
PR-RR	Ensure reliability with minimal downtime	System uptime monitoring and CI/CD integration	Depends on FR-3, FR-4
OE-EPE	Ensure compatibility with common computing environments	Test on multiple OS and hardware configurations	Supports FR-1
OE-RIA	Interfacing with MuJoCo and other environments	Integration test: Validate data exchange compatibility	Supports FR-1, FR-8
OE-RR	Ensure proper open-source licensing and documentation	License verification and documentation audit	Supports MS-M2
OE-OSR	Establish internal resources for maintainers	Review documentation and repository setup	Supports UH-E1
MS-M1	Ensure future software updates take less time	Performance test: Compare build times before and after changes	Supports PR-SE

Table 5 – Continued

Requirement ID	System Objective / Justification	Verification Method	Dependencies
MS-M2	Code documentation for maintainability	Code review: Ensure all major functions/classes are documented	Depends on UH-E1
MS-M3	Regular updates for maintainability	Version control check: Ensure monthly updates	Depends on FR-3, FR-6
MS-S1	Maintain an online repository with resources	Repository check: Ensure all guides are available	Supports OE-OSR
MS-S2	Support the latest stable Linux version	Dependency check: Verify compatibility with latest OS updates	Supports FR-1
MS-A1	Deliver new features via CI/CD	CI/CD pipeline test: Validate automated builds and tests	Depends on FR-3
MS-A2	Ensure easy installation and execution	Installation test: Verify steps across different OS	Supports OE-EPE
MS-A3	Adapt to new experiments	Experiment validation: Verify system flexibility with new inputs	Depends on FR-8
SR-A1	Restrict contributor access	Access control test: Verify contributor permissions	Depends on MS-S2
SR-A2	Enable multi-factor authentication for contributors	Security test: Validate MFA enforcement	Supports SR-A1

Table 5 – Continued

Requirement ID	System Objective / Justification	Verification Method	Dependencies
SR-I1	Main branch protection	Code repository check: Ensure branch protection rules	Supports FR-3, FR-4
SR-P1	Ensure sensitive data is anonymized	Security audit: Verify no sensitive data is exposed	Supports SR-A1
SR-AU1	Store all message logs and simulation results	Audit check: Ensure traceability of logs	Supports UH-E2
SR-IM1	Scan code for unauthorized modifications	Automated security scan in CI/CD pipeline	Supports SR-I1
SR-IM2	Implement robust error handling mechanisms	System test: Validate error recovery	Supports PR-RFT
CRL1	Ensure legal compliance with open-source licensing	License review: Verify MIT compliance	Supports OE-RR
CRL2	Adhere to Google C++ style guide	Automated linting and static analysis	Supports FR-6
MR-R1	Ensure continuous synchronization between GitHub and GitLab	Version control test: Verify repo sync process	Depends on FR-3
MR-R2	Enable GitHub Actions for CI/CD	CI/CD test: Validate pipeline execution	Depends on FR-3
MR-R3	Ensure seamless TPG-MuJoCo integration	Integration test: Validate system functionality	Depends on FR-1, OE-RIA
USD-UD1	Provide documentation in online and offline formats	Check repository for PDF and online versions	Supports UH-E1

Table 5 – Continued

Requirement ID	System Objective / Justification	Verification Method	Dependencies
USD-UD2	Include platform-specific installation and troubleshooting	Verify documentation covers Windows, macOS, and Linux	Supports MS-A2
USD-UD3	Provide release notes for updates	Check repository for updated release notes	Supports MS-M3

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Lifelong Learning. Please answer the following questions:

1. What knowledge and skills will the team collectively need to acquire to successfully complete this capstone project? Examples of possible knowledge to acquire include domain specific knowledge from the domain of your application, or software engineering knowledge, mechatronics knowledge or computer science knowledge. Skills may be related to technology, or writing, or presentation, or team management, etc. You should look to identify at least one item for each team member.

2. For each of the knowledge areas and skills identified in the previous question, what are at least two approaches to acquiring the knowledge or mastering the skill? Of the identified approaches, which will each team member pursue, and why did they make this choice?

Calvyn

1. I believe that this capstone project will require a solid understanding of C++, and important software development and devops related skills, primarily around developing a CI/CD system from scratch. Project management skills will also be vital to collaborate effectively as a team to meet project milestone deadlines and handle inevitable conflicts.

2. To gain a better understanding of the necessary technical skills required, I believe that referring to existing documentation will be the best option to help guide the learning process. In addition, online resources such as Youtube videos and articles will be very helpful for developing this understanding with a visual/auditory aid.

Cyruss

1. The team will collectively need to acquire knowledge and skills in areas such as C++, reinforcement learning, and DevOps. Team management will also be a critical skill to ensure that the team works smoothly and efficiently, making sure deadlines and goals are met. Another skill that

will be crucial is writing, as the project requires many written reports of documentation and analysis. Being able to convey the intended message in written format will allow for less ambiguity and misinterpretations within the project.

2. Two approaches when it comes to acquiring the knowledge or mastering the skills are the following: Trial and error experimentation of the skill, and online learning resources. Trial and error experimentation allows for exploration of the topic in question, and creates a self-learning environment where practice is crucial. The approach of online learning resources ensures a guided and structured approach where one follows a set of instructions and topics that one may think are beneficial. I will choose to pursue the approach of online learning resources, as time is critical for this project and experimentation is a time-consuming approach. I learn better with a structured format, and it prevents me from getting off-track when it comes to learning.

Richard

1. Knowledge and skills that the team will acquire to successfully complete this capstone project involve understanding the CI/CD process (creating our custom GitHub Actions pipeline), software engineering design as refactoring the codebase to be modular, extensible are key since the end goal of this project is to enable open source development. Additionally, team management is something everyone will experience. In our future careers, we'll all be working in teams so knowing how to do development while being in a team setting is an invaluable experience. Examples of this workflow is task splitting, discussions on what to prioritize, and working on a codebase in a team setting.

2. For understanding CI/CD, I've always been interested in this topic through my co-ops because I always found it cool that the teams I've worked on had this infrastructure set up and integrated in a seamless way. An approach to acquiring knowledge is using what I've previously known and filling in the gaps through research and tutorials. In the course, there has already been a tutorial on CI/CD, however the pipelines we need to build for our project will be a little more complicated. Using that as a base will set us up with the fundamentals. For me, I'm very interested in learning and being a generalist. I want to

contribute to both portions of the project as I want to grow my breadth with how to design a C++ project to make it more modular and usable as a library. Additionally, I've been more interested in DevOps as it is an area I hope to gain more experience in. With development of a framework/library, this is different to the application dev experience I've had before, but ultimately researching other similar open source projects will be invaluable to see what the best practices are.

Mark

1. The team will collectively acquire knowledge in C++, CI/CD and machine learning. For non-technical skills, the team will be leveraging team management. This will ensure that the team will function efficiently, allowing the team to meet critical deadlines and goals. Communication in ways such as writing and speaking can also be acquired throughout the duration of the project. This skill ensures that the team can successfully create documentations and requirements that can provide future users and contributors insights on the product.

2. For learning required skills and knowledge for this project, the team can proactively familiarize themselves into the TPG codebase. This method would allow each member to gain more insights on machine learning, and the C++ programming language. Leveraging online learning resources such as YouTube, Coursera, and Udemy can ensure that the team gain background knowledge on machine learning, CI/CD and C++.

Edward

1. We should acquire a base level of general domain knowledge of existing RL problems and approaches, general overview as well as specific approaches relevant to TPG (genetic programming, program synthesis). Learning relevant terminology will be useful to understanding the TPG codebase and learning to use the MuJoCo API. For an efficient learning approach, sharing the knowledge with team members is important as many of us have little specialized background in RL. Thus, asking Dr. Kelly via teams, or during his presentations is highly encouraged. The alternative of self-study should be used to supplement the aforementioned primary learning approach, as this approach does not facilitate shared learning.

2. Proficiency in C++ is important to be developed as more effort is spent on the integration of TPG and MuJoCo codebases. Proficiency of Python is also beneficial since much of MuJoCo documentation and OpenAI Gym examples use Python. The recommended approach of learning such languages should be based on each individual team members' learning style, whether it be reading documentation, tackling subproblems, etc.