System Verification and Validation Plan for SE 4G06

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

November 1, 2024

Revision History

Date	Version	Notes
11/01/2024	0.0	Initial draft of VnV plan.

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
TPG	Tangled Program Graphs
DNNs	Deep Neural Networks
RL	Reinforcement Learning
Multi-Task RL	Multi-Task Reinforcement Learning
SRS	Software Requirement Specification
FR	Functional Requirement
NFR	Non-Functional Requirement
MIS	Module Interface Specification

This document provides the Software Validation and verification plan of the TPG Capstone Project. It consists of general information such as objectives and relevant documentation, plans for the system's verification and validation, as well as test descriptions for both system and unit tests. This information is subject to change at any point throughout the project and will be updated accordingly with changes mentioned in the revision history table.

2 General Information

2.1 Summary

The software being tested is the TPG framework. More specifically, the overall workflow of the open-source repository as well as the interface between the TPG framework and the physics engine environment, MuJoCo. TPG is a research framework currently being developed by a team led by Dr. Stephen Kelly. The software's general functions include training agents within the OpenAI Gymnasium environment using reinforcement learning techniques to achieve a certain task. With TPG, the data is outputted through a variety of methods. You may choose to visualize the data within multiple plots or view an OpenGL simulation of the agent with the best performance at the time of running. The software to be implemented within this repository is an interface that allows for experiments to be run with MuJoCo, a physics-engine simulator. This is in addition to creating a software development pipeline within the repository that allows for an easier development experience as changes are implemented.

2.2 Objectives

Within the duration of the Capstone project, it is intended for this software to become fully integrated within the MuJoCo environment to allow for reinforcement learning to be performed. This is a crucial objective, as TPG is currently being used for research purposes. Allowing MuJoCo will assist in the research of Tangled Program Graphs and assist with solving potential research questions within this space of research.

Another objective for the software is to adhere to software engineering stan-

dards and practices. As this code is constantly being developed, good practices are required for the sustainability of the repository code. Building CI/CD pipelines that will automatically perform test cases, lint code that ensures code syntax follows Google C++ guidelines and the compilation of code will ensure a smooth development and prevent unnecessary development halts due to major code structure difficulties.

Furthermore, another objective is to ensure that the TPG framework is an easy-to-install and seamless experience for developers who are on the major operating systems (MacOS, Linux, and Windows). This is important, as new researchers who aim to utilize TPG should be able to install and start reinforcement learning without hassle. As of right now, it is very difficult for users within MacOS to install the program without having to use any virtual machines. Additionally, if they do manage to start the environment, users are unable to view the OpenGL simulation. While installation and execution within Windows is much easier than MacOS, workarounds are required, which severely impact the usability of the software.

A future objective that won't be covered within the scope of the capstone project is the ability to easily implement new environments other than MuJoCo further to enhance the usefulness of research projects for reinforcement learning. This will not be covered due to the time constraints within the capstone project and the overall complexity. However, if the opportunity to continue this project arises, this will be an objective that will be considered.

The above objectives describe the main ones for the duration of the capstone project. Any complex and complicated aspects that may be required to complete an objective will be assisted by utilizing open-source external libraries.

2.3 Challenge Level and Extras

The challenge level of the project is **general** as agreed upon by the course instructor since this project is an extension of the current Tangled Program Graphs repository created by Dr. Stephen Kelly. The extras that will be tackled by this project include user documentation. This extra will allow new users to be guided through the installation and execution, as well as assist with any troubleshooting issues that may occur. Code documentation will

also be completed to assist new research developers in navigating the new functionalities and integrations within the system. In addition, a DevOps pipeline integration will be integrated for developers and a research paper aims to be completed.

2.4 Relevant Documentation

The following documentation is considered to be relevant and may provide more context about the project outside of this document:

- **Problem Statement and Goals** (Tangle, 2024c)**:** This document specifies a more detailed outline of the project's goals and purpose. This involves important stakeholder information, inputs and outputs of the project as well as information in regards to the project's environment.
- **Development Plan:** (Tangle, 2024a)**:** This document provides a guideline regarding all of the tools and technologies that will be utilized throughout this plan for verification and validation.
- **Hazard Analysis:** (Tangle, 2024b)**:** This document describes detailed hazards that may occur throughout the development of the project. It specifies the system boundaries and components, as well as mitigation strategies in the form of safety and security requirements that may need verification and validation.
- **Software Requirements Specification** (Tangle, 2024d)**:** This document specifies all the non-functional and functional requirements that the project should satisfy by the end of the capstone period. This is useful for creating test cases that will verify and validate that the requirements have been met.
- **Module Interface Specification:** This document describes how different components within the system will interact with each other. This is beneficial to determine the modules that will be present within the system, and help determine the scope of testing.

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

[Your teammates. Maybe your supervisor. You should do more than list names. You should say what each person's role is for the project's verification. A table is a good way to summarize this information. —SS]

3.2 SRS Verification Plan

[List any approaches you intend to use for SRS verification. This may include ad hoc feedback from reviewers, like your classmates (like your primary reviewer), or you may plan for something more rigorous/systematic. —SS]

[If you have a supervisor for the project, you shouldn't just say they will read over the SRS. You should explain your structured approach to the review. Will you have a meeting? What will you present? What questions will you ask? Will you give them instructions for a task-based inspection? Will you use your issue tracker? —SS]

[Maybe create an SRS checklist? —SS]

3.3 Design Verification Plan

[Plans for design verification —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

[The review will include reviews by your classmates —SS]

[Create a checklists? —SS]

3.5 Implementation Verification Plan

The implementation verification process will focus on ensuring that our modifications to TPG and the MuJoCo integration meet the project's core requirements. This will be accomplished through several complementary approaches.

Code review will serve as the primary verification mechanism. All pull requests will require review by at least one other team member before merging. During these reviews, we will verify adherence to the Google C++ Style Guide, completeness of documentation, potential impacts on existing TPG functionality, and the correctness of MuJoCo integration points.

Version control validation will be maintained through strict repository management practices. Branch protection rules will prevent direct pushes to the main branch, ensuring all changes go through the required review process. We will maintain regular synchronization between the GitHub and GitLab repositories to ensure consistency across development platforms.

Manual testing and inspection will verify that core TPG functionality remains intact. This includes verifying program generation and mutation, team/graph evolution, and action selection mechanisms. For the MuJoCo integration specifically, we will verify basic environment setup, proper handling of state and action spaces, and compare performance against the existing implementations such as CartPole as a baseline.

3.6 Automated Testing and Verification Tools

Our primary automated verification will be implemented through a GitHub Actions CI/CD pipeline. This pipeline will perform automated build verification across multiple platforms including Linux, Mac, and Windows to ensure cross-platform compatibility. It will also run basic integration tests and enforce code style requirements.

For manual verification, we will leverage TPG's existing suite of plotting and statistics tools to analyze performance and behavior. The OpenGL visualization capabilities built into TPG will allow us to inspect agent behaviors directly. Similarly, MuJoCo's built-in visualization tools will provide another avenue for verifying correct environment integration and agent behavior.

3.7 Software Validation Plan

The software validation plan begins with establishing CartPole as a baseline for comparison. We will compare TPG's performance between the existing environment implementation and the new MuJoCo implementation. This

comparison will include validating expected agent behaviors through visual inspection using both OpenGL and MuJoCo visualizations, as well as verifying that fitness scores achieve similar thresholds in both implementations.

Core integration validation will focus on verifying TPG's ability to properly interact with MuJoCo environments. We will verify that TPG can successfully initialize MuJoCo environments, receive valid state information, send valid actions, and receive appropriate rewards. Throughout the training process, we will monitor the environment's response to agent actions and verify training progression through both fitness scores and visual behavior inspection.

Documentation validation will ensure that all necessary information is available to users and future developers. This includes verifying the completeness and accuracy of installation guides, with particular attention to environment setup steps and dependency management. Usage documentation will cover running experiments, visualizing results, and common troubleshooting procedures.

Research validation will be conducted through regular review meetings with Dr. Kelly to ensure the implementation supports the project's research goals. These reviews will validate that basic research capabilities are preserved and verify that new MuJoCo environments can be added as needed. We will assess TPG's behavior in baseline environments to ensure it provides a solid foundation for future research experiments.

The entire validation process will be iterative, with feedback incorporated throughout development. Our focus is on ensuring the basic integration is sound and providing a foundation that will support future research experiments. Rather than attempting to validate against specific performance criteria, we will validate that the implementation provides the necessary capabilities for exploring research questions about TPG's behavior in MuJoCo environments.

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

Control: Manual versus Automatic

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Area of Testing1

Title for Test

1. test-id1

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here,

you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

Team 3 Tangle. Development plan. https://github.com/TPGEngine/tpg/blob/main/docs/DevelopmentPlan/DevelopmentPlan.pdf, 2024a.

Team 3 Tangle. Hazard analysis. https://github.com/TPGEngine/tpg/blob/main/docs/HazardAnalysis/HazardAnalysis.pdf, 2024b.

Team 3 Tangle. Problem statement and goals. https://github.com/TPGEngine/tpg/blob/main/docs/ProblemStatementAndGoals/ProblemStatement.pdf, 2024c.

Team 3 Tangle. System requirements specification. https://github.com/TPGEngine/tpg/blob/main/docs/SRS/SRS.pdf, 2024d.

6 Appendix

6.1 Symbolic Parameters

This section is not applicable as there are no symbolic parameters used within the project.

6.2 Usability Survey Questions?

Here are some questions that may be asked regarding usability in the form of a survey:

1. What operating system do you use?
   (a) Windows
   (b) Mac OS
   (c) Linux
2. On a scale from 1-10 (higher means better), how would you rate your installation experience?
3. On a scale from 1-10 (higher means easier), how easy would you say it was to execute a simulation environment using MuJoCo?
4. If applicable, on a scale from 1-10 (higher means easier), how easy was it to implement changes to the code?
5. If applicable, did you have any trouble integrating your changes to the remote repository?
6. Do you have any feedback or suggestions when it comes to the usability of the system? Please write them down below.

Appendix — Reflection

[This section is not required for CAS 741 —SS]

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

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. What knowledge and skills will the team collectively need to acquire to successfully complete the verification and validation of your project? Examples of possible knowledge and skills include dynamic testing knowledge, static testing knowledge, specific tool usage, Valgrind etc. You should look to identify at least one item for each team member.

   Edward: One of the key knowledge areas I will need to develop is understanding the various indicators of correct agent behavior across different MuJoCo environments. This includes interpreting fitness scores, analyzing plot outputs, inspecting animations, and recognizing expected behavioral patterns. Since each environment has unique characteristics and learning dynamics, this will require developing a deeper familiarity with TPG's learning progression and how it manifests across different scenarios. Currently, the workflow involving running experiments, checking logs, interpreting plots, and debugging issues is still somewhat

opaque, requiring significant guidance from Dr. Kelly to understand the nuances of what we're observing.

4. 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?

   Edward: There are two main approaches I plan to pursue to develop this knowledge:
   a) Build understanding incrementally through guided exploration. This involves starting with environments where we have clear expectations (like CartPole), consulting with Dr. Kelly to understand what indicators to look for, and documenting these learnings to apply to new environments. Regular communication with the team about observations and issues is crucial since everyone brings different insights to interpreting TPG's behavior.
   b) Take a systematic experimentation approach. This means making controlled modifications to environment parameters or TPG settings on separate GitHub branches, carefully documenting the impacts, and building a mental model of cause-and-effect relationships. By having a safe space to experiment without fear of breaking the main codebase, I can learn through trial and error while maintaining rigorous version control.