System Verification and Validation Plan for SFWRENG 4G06

Team 9, dice_devs
John Popovici
Nigel Moses
Naishan Guo
Hemraj Bhatt
Isaac Giles

October 28, 2024

Revision History

Table 1: Revision History

Date	Developer(s)	Change
2024-10-27	John Popovici	Preparing file and added to section 1
2024-10-28	John Popovici	Removed table of symbols; Added to section 1
...	...	...

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

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 General Information

1.1 Summary

This project creates an online multiplayer game platform that allows for the creation of custom Yahtzee-like games and variants. This family of games will come with some presets such as classic Yahtzee, an octahedron version, and more, but will allow for the users to set their own variables such as number of dice, what kind of dice, and some elements of scoring, and then play that game. Kinds of dice would include cubes and octahedrons, among other multi-sided dice, and scoring could be calculated at the end of the game, as in classic Yahtzee, or on a per-round basis, where hands go in a head-to-head matchup.

1.2 Objectives

The main objective of this project is to ensure robust and flexible functionality that supports customizability in game creation, with confidence in their interactability. Specifically, the project will focus on:

- Demonstrating Adequate Usability: User interactions should be intuitive, enabling smooth navigation and game setup without requiring extensive technical knowledge.

- Supporting User-Driven Customizability: Emphasis will be placed on making the platform adaptable, allowing users to easily create and modify games with various dice types and scoring rules. These elements must be modular such that a custom game can make use of these components in a flexible fashion.

- Ensuring Software Correctness: Testing will confirm accurate dice roll usage, score calculations, and game outcomes to build confidence in the correctness of the core mechanics.

Out of Scope Objectives: Due to time constraints, certain areas are out of scope for this project. These include:

- Probability Testing: While a statistical analysis could be done on the randomness of dice rolls, it would be out of scope for our current plans

and we will proceed with a general assumption that functions that rely on random values are as accurately pseudo-random as needed, without set seeding.

- Performance Optimization: While performance will be monitored, extensive optimization for faulty connections or intricate rule configurations will be left out, with plans to address this in potential future releases.

This prioritization allows us to allocate resources efficiently, focusing on core functionality and foundational elements.

1.3 Challenge Level and Extras

Through the implementation of not just a single Yahtzee game variant, but by implementing a system where the user can create a custom game variant and connect to another online player to play it, we are looking to achieve the advanced challenge level. We would also provide some pre-set game variants that have been tested and were found to be more fun than others.

We will include extra elements to aid in better developing our game and allow us to design for a fun gaming experience, as we wish this game to not be just a capstone project, but a game we can be proud of and put out into the world. Following industry practices in game development, we wish to potentially do the following extras:

- Focus groups, Surveys, and Interviews

- Usability testing

- User personas

- User documentation

[TODO update problem statement and gitlab once we know difficulty level and what extras —SS]

1.4 Relevant Documentation

The following documents are crucial for the Verification and Validation process, ensuring the project meets its objectives and operates reliably:

- Software Requirements Specification (SRS): This document outlines the core requirements for the platform, defining features and limitations, as well as expected behaviors. The SRS serves as a primary reference to validate that the developed software meets user and system requirements.

- [TODO add more —SS]

[TODO add BibTeX and hyperlinks —SS]

2 Plan

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

2.1 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]

2.2 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]

2.3 Design Verification Plan

[Plans for design verification —SS]
[The review will include reviews by your classmates —SS]
[Create a checklists? —SS]

2.4 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]

2.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS]

[In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walk-throughs, code inspection, static analyzers, etc. —SS]

[The final class presentation in CAS 741 could be used as a code walk-through. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

2.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS]

[If you have already done this in the development plan, you can point to that document. —SS]

[The details of this section will likely evolve as you get closer to the implementation. —SS]

2.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS]

3 System Tests

3.1 Tests for Functional Requirements

3.1.1 Area of Testing1

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:

3.1.2 Area of Testing2

...

3.2 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]

3.2.1 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:

3.2.2 Area of Testing2

...

3.3 Traceability Between Test Cases and Requirements

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

4 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]

4.1 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]

4.2 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]

4.2.1 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. ...

4.2.2 Module 2

...

4.3 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]

4.3.1 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:

4.3.2 Module ?

...

4.4 Traceability Between Test Cases and Modules

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

5 Appendix

This is where you can place additional information.

5.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

5.2 Usability Survey Questions?

[This is a section that would be appropriate for some projects. —SS]

Appendix — Reflection

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.

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?