System Verification and Validation Plan for ProgName

Team #, Team Name
Student 1 name
Student 2 name
Student 3 name
Student 4 name

October 25, 2025

Revision History

Date	Version	Notes
Oct 24	Alvin	System test functional requirements

[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 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Author, 2019) tables, if appropriate —SS]

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

This document ... [provide an introductory blurb and roadmap of the Verification and Validation plan —SS]

2 General Information

2.1 Summary

[Say what software is being tested. Give its name and a brief overview of its general functions. —SS]

2.2 Objectives

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Author (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

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

[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

[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

[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

[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 walkthroughs, code inspection, static analyzers, etc. —SS]

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

3.6 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]

3.7 Software Validation

[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]

[You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS]

[For those capstone teams with an external supervisor, the Rev 0 demo should be used as an opportunity to validate the requirements. You should plan on demonstrating your project to your supervisor shortly after the scheduled Rev 0 demo. The feedback from your supervisor will be very useful for improving your project. —SS]

[For teams without an external supervisor, user testing can serve the same purpose as a Rev 0 demo for the supervisor. —SS]

[This section might reference back to the SRS verification section. —SS]

4 System Tests

This section defines end to end system tests that validate the software against the SRS Functional Requirements (FR-1..FR-9) for the MVP implementation state. Tests are grouped by related functionality to improve traceability and reuse of fixtures. Unless stated otherwise, tests assume a deterministic deck seed, two authenticated test users, and a clean database. All expected results are observable via the UI and server state exposed to the test harness.

4.1 Tests for Functional Requirements

The subsections below cover: (1) game flow, rule validation, special cards, and end conditions (SRS Game Manager: FR-1..FR-5) and (2) scoring and

education features including dual-base display, hints, and valid move highlighting (SRS Score Manager and Education Support: FR-6..FR-9). Where tests rely on input constraints, they use the card value and move legality definitions in the SRS.

4.1.1 Game Manager: Start, Turn, Rules, Specials, End (FR-1..FR-5)

This area verifies new game initialization, turn sequencing, rule validation for same suit or same value or sum equals 12 (base-12), handling of wild eights, and detecting the end of a game. It directly traces to FR-1 Start new game, FR-2 Turn management, FR-3 Rule validation, FR-4 Special cards, FR-5 End of game.

Start new game initializes state (ST-GM-01)

1. ST-GM-01

   Control: Automatic

   Initial State: No active sessions. Two users are authenticated and in the lobby. Deterministic deck seed S is configured.

   Input: User A clicks New Game and invites User B. User B accepts.

   Output: A new session is created with a shuffled deck using seed S, a discard starter card is placed, each player receives the configured hand size H, and the UI shows turn = User A.

   Test Case Derivation: From FR-1 and SRS deck initialization semantics. With a fixed seed S, the initial state is uniquely determined.

   How test will be performed: Run a Node console harness that calls initGame('S'), then logs and checks via simple equality that hands have size H, discard.length == 1, and turn == UserA. No browser, no sockets.

Legal move: same suit (ST-GM-02)

1. ST-GM-02

Control: Automatic

Initial State: Active session. Discard top = 5♡. User A hand contains K♡ and others. Turn = User A.

Input: User A plays K♡.

Output: Move is accepted. Discard top becomes K♡. Turn advances to User B.

Test Case Derivation: FR-3 allows a move when suits match.

How test will be performed: In the console harness set discard = [suit:'H',v:5] and handA contains K♡. Call canPlay() then playCard(). Log PASS if top becomes K♡ and turn flips to UserB.

Legal move: sum equals 12 (base-12) (ST-GM-03)

1. ST-GM-03

   Control: Automatic

   Initial State: Discard top = 5♢ (value 5). User A hand contains 7♠ (value 7). Turn = User A.

   Input: User A plays 7♠.

   Output: Move is accepted. Discard top becomes 7♠. Turn advances to User B.

   Test Case Derivation: In base-12, 5 + 7 = 10_{12} which equals 12 in decimal. FR-3 permits play when values sum to 12 base-12.

   How test will be performed: Set top to value 5 then attempt to play value 7. Use canPlay() to assert true and log PASS.

Special card: wild eight suit selection (ST-GM-04)

1. ST-GM-04

   Control: Automatic

   Initial State: Discard top = K♣. User A hand contains 8♡. Turn = User A.

Input: User A plays 8♡ and selects Spades as the new suit.

Output: Move is accepted. Discard shows 8 with chosen suit indicator = Spades. Only Spades or otherwise legal responses are allowed for the next player.

Test Case Derivation: FR-4 specifies 8s are wild and allow the player to declare a suit.

How test will be performed: Give A an 8, play with playCard(state,'A',eight,chooseSuit:'S'), then check chosenSuit == 'S' and turn == 'B' via console logs.

Illegal move is rejected with guidance (ST-GM-05)

1. ST-GM-05

   Control: Automatic

   Initial State: Discard top = 9♢. User A hand contains Q♣ and no other legal card.

   Input: User A attempts to play Q♣.

   Output: Move is rejected. UI shows polite message explaining why the move is invalid and highlights available actions: draw or play a legal card if any.

   Test Case Derivation: FR-3 rejects moves that are not same suit, not same value, and do not sum to 12 base-12.

   How test will be performed: Call canPlay() on an invalid card and expect false. For now we just assert the engine rejects the move; UI message will be verified later in UI tests.

End of game detection (ST-GM-06)

1. ST-GM-06

   Control: Automatic

   Initial State: User A has 1 card, which is legal to play. User B has multiple cards.

   Input: User A plays the last card.

Output: Game ends. Winner = User A. System transitions to score calculation and displays results.

Test Case Derivation: FR-5 requires ending the game when a player has no cards left.

How test will be performed: Have a script a state where A has one legal card, call playCard(), then log PASS if state.status == 'Completed' and winner == 'A'.

4.1.2 Score and Education Features: Dual-base, Hints, Highlights (FR-6..FR-9)

This area validates round scoring in decimal and Dozenal, correct conversion and display, context explanations for Dozenal arithmetic, and visual highlighting of valid moves. It traces to FR-6 Calculate score, FR-7 Display score, FR-8 Hints, FR-9 Highlight valid moves.

Score calculated and shown in decimal and Dozenal (ST-SC-01)

1. ST-SC-01

   Control: Automatic

   Initial State: Game just ended. Opponent remaining cards have numeric values 2, 4, 6 (per SRS value function).

   Input: Trigger score calculation screen.

   Output: Decimal total = 12. Dozenal total = 10_{12}. Both are displayed side by side.

   Test Case Derivation: 2 + 4 + 6 = 12 decimal which equals 10 in base-12.

   How test will be performed: Call calculateScore(B:[2,4,6]) and expect decimal:12, dozenal:'10' . Log PASS/FAIL.

Dozenal explanation for sum equals 12 move (ST-SC-02)

1. ST-SC-02

Control: Automatic

Initial State: Discard top = 5. Hover help is enabled. Input: Player hovers a 7 in hand and then selects it.

Output: Hint appears: explanation that $5 + 7 = 10_{12}$ which equals 12 decimal, therefore the move is valid.

Test Case Derivation: FR-8 requires on demand guidance for Dozenal arithmetic.

How test will be performed: For MVP engine tests, only assert the explanation function returns a string containing $5 + 7 = 10_{12}$. UI tooltip will be covered later.

Valid moves highlighted exactly (ST-SC-03)

1. ST-SC-03

   Control: Automatic

   Initial State: Discard top = 4♠. Hand contains 4♣, 8♢, 7♡, Q♠, 6♣.

   Input: None beyond rendering the hand.

   Output: Highlighted cards are exactly {4♣, Q♠, 8♢}. 7♡ and 6♣ are not highlighted.

   Test Case Derivation: Same value (4), same suit (Spades), and wild eight are valid. Others are invalid because $7 + 4 = 11$ decimal which is not 10_{12} and $6 + 4 = 10$ decimal which is not 10_{12}.

   How test will be performed: Call an exported listValidMoves(state, hand) and assert it returns exactly the IDs for same value, same suit, and wild eight. Log PASS/FAIL.

Primary base toggle affects labeling only (ST-SC-04)

1. ST-SC-04

   Control: Automatic

   Initial State: Score view shows decimal on the left and Dozenal on the right with decimal as primary.

Input: Toggle primary base to Dozenal.

Output: Labels and prominence swap, but numeric values remain identical pair $\{12, 10_{12}\}$. No recomputation changes total values.

Test Case Derivation: FR-7 specifies dual display; the toggle is a presentation choice.

How test will be performed: For MVP engine tests, verify the conversion outputs are stable. The UI toggle is deferred; assert that conversion of 12 dec remains 10_{12} before and after calling a setPrimaryBase() no-op stub.

...

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

Author Author. System requirements specification. https://github.com/..., 2019.

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions?

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

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.

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?