System Verification and Validation Plan for SFWRENG 4G06: Dice Duels: Duel of the Eights

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

March 26, 2025

Revision History

Table 1: Revision History

Date	Developer(s)	Change
2024-10-27	John Popovici	Preparing file and added 1.1, 1.2, 1.3
2024-10-28	John Popovici	Removed table of symbols; Added 1.4
2024-10-30	Hemraj Bhatt	Added content to section 2
2024-10-31	John Popovici	Finalized section 1 formatting and content
2024-11-01	Isaac Giles	Filled in content for section 3
2024-11-02	Naishan Guo	Added new content for section 3
2024-11-04	Hemraj Bhatt	Added content to reflection
2024-11-04	Hemraj Bhatt	Updated section 2 content
2024-11-04	John Popovici	Added reflections
2024-11-04	John Popovici	Added symbols table new section 1
2025-03-07	John Popovici	Updated extras and added usability testing questions
2025-03-08	Isaac Giles	Added section 5 unit test section
2025-03-10	John Popovici	Updated and converted tables to matrices
2025-03-08	Isaac Giles	Corrected module names in section 5
2025-03-26	John Popovici	Removed references to C# as per TA feedback

Contents

1 Symbols, Abbreviations, and Acronyms **iv**

2 General Information **1**
- 2.1 Summary . . . 1
- 2.2 Objectives . . . 1
- 2.3 Challenge Level and Extras . . . 2
- 2.4 Relevant Documentation . . . 2

3 Plan **3**
- 3.1 Verification and Validation Team . . . 4
- 3.2 SRS Verification Plan . . . 5
- 3.3 Design Verification Plan . . . 5
- 3.4 Verification and Validation Plan Verification Plan . . . 6
- 3.5 Implementation Verification Plan . . . 6
- 3.6 Automated Testing and Verification Tools . . . 7
- 3.7 Software Validation Plan . . . 7

4 System Tests **8**
- 4.1 Tests for Functional Requirements . . . 8
  - 4.1.1 Gameplay and Rules . . . 8
  - 4.1.2 User Interface and Settings . . . 11
- 4.2 Tests for Nonfunctional Requirements . . . 12
  - 4.2.1 Performance and Usability . . . 13
  - 4.2.2 System Compatibility and Reliability . . . 13
- 4.3 Traceability Between Test Cases and Requirements . . . 15

5 Unit Test Description **18**
- 5.1 Unit Testing Scope . . . 18
- 5.2 Tests for Functional Requirements . . . 18
  - 5.2.1 DynamicDiceContainer Module . . . 18
  - 5.2.2 ScoreCalculator Module . . . 20
- 5.3 Traceability Between Test Cases and Modules . . . 22

6 Appendix **24**
- 6.1 Symbolic Parameters . . . 24
- 6.2 Usability Survey Questions? . . . 24

List of Tables

1 Revision History i
2 Symbols, Abbreviations, and Acronyms iv
3 Verification and Validation Team Roles 4
4 Key for Test Cases 15
5 Key for Requirements 16
6 Traceability Matrix for Test Cases and Requirements 17
7 Table of Traceability Between Test Cases and Modules 22
8 Matrix of Traceability Between Test Cases and Modules . . . 22

1 Symbols, Abbreviations, and Acronyms

Table 2: Symbols, Abbreviations, and Acronyms

symbol	**description**
SRS	Software Requirements Specification
HA	Hazard Analysis
MIS	Module Interface Specification
MG	Module Guide
VnV or V&V	Verification and Validation
CI	Continuous Integration
CD	Continuous Development
Test-X	Test number X
RX	Functional Requirement number X
NFRX	Non Functional Requirement number X

2 General Information

2.1 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.

2.2 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.

2.3 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 develop our project to be both a fun game and a capstone project. The challenge difficulty approved is that of general.

We will include extra elements to aid in better developing our game and allow us to design for a fun gaming experience. Following industry practices in game development, we will do the following extras:

- Usability Testing built off of different elements of focus groups, surveys, and interviews. Questions and elements of focus are highlighted in section 6.2
- User manual

2.4 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.

- Hazard Analysis (HA): This document is key to identifying and managing risks associated with system failures, particularly in dice physics, scoring, and multiplayer stability.
- Module Interface Specification (MIS): The MIS details the interfaces between components, which is essential for ensuring seamless interaction across modules and verifying that dependencies are managed correctly.
- Module Guide (MG): The MG provides a structural breakdown of the system's components, allowing verification that each module aligns with the SRS specifications and functions as intended.

3 Plan

This section covers the project's verification and validation (V&V) strategy, including the methodologies and strategies used to ensure software dependability, correctness, and usability. Each component of the plan will establish verification and validation procedures appropriate for each stage of the project lifecycle, from reviewing the Software Requirements Specification (SRS) to testing the developed system.

3.1 Verification and Validation Team

Team Member	Role	Responsibilities
Isaac Giles	SRS Verification Lead	Oversees SRS reviews, organizes feedback, coordinates checklist compilation, and ensures any new or modified requirements are accurate.
John Popovici	Design Verification Lead	Oversees design reviews, develops checklists, and organizes peer review meetings regarding design. They also ensure that any design modifications that either are outcomes of meetings or decisions made within the team are properly formulated and accurate.
Nai Shan Guo	Implementation Tester	Performs unit tests, isolates tests for core capabilities, and ensures that individual code components are in line with the implementation plan outlined in this document. In charge of testing any new modifications and green-lighting them.
Nigel Moses	Software Validation Lead	In charge of ensuring that the software matches user requirements through using feedback provided by supervisor and peer reviews. Additionally, in charge of performing validation strategies outlined in the document.
Hemraj Bhatt	Tool and Automation Lead	Oversees the installation and usage of automated testing tools, unit testing frameworks, and test coverage metric tracking tools. In charge of sourcing the correct tools needed for the specific system as well.
Dr. Paul Rapoport	Project Supervisor	Provides oversight, participates in structured review sessions, and provides feedback on requirements, design, and implementation reviews. Due to the supervisor's area of expertise, feedback will focus on requirements and design as opposed to programming implementation advice.

Table 3: Verification and Validation Team Roles

3.2 SRS Verification Plan

The SRS verification will take an organized approach, including supervisor and peer review sessions, to ensure that the stated requirements are clear, accurate, and thorough.

- **Structured Review Meeting**: Conducted with the supervisor in person to present the essential SRS components. The team will highlight the essential functional and non-functional requirements and solicit feedback on the clarity and feasibility of each requirement. The feedback will then be tracked as issues and be used to make any modifications deemed necessary.
- **Checklist Creation**: A detailed checklist will be prepared to analyze each SRS item, ensuring the requirements are accurate and feasible for execution. The checklist will be utilized at each review session. If the checklist is filled out completely, the team will know that there are no necessary changes and if it is not then necessary modifications will be made to the SRS.
- **Peer Review Sessions**: The team will have our peers, such as classmates, complete task-based inspections of the SRS to identify any ambiguities or missing components. Tasks may include examining the document for completeness, by ensuring that all functional and non-functional requirements are properly stated, checking for terminology consistency, and ensuring that any assumptions or restrictions are well-defined and reasonably made. This will ensure that the SRS is complete and viable.

3.3 Design Verification Plan

To ensure that the design complies with the SRS standards, the team will conduct a structured review process including the entire team, external peers and the teams supervisor:

- **Design Checklists**: A checklist will be developed to guide reviewers through each design component and confirm whether or not they correspond to both the functional and non-functional requirements outlined in the SRS.

- **Peer Review**: Classmates and primary reviewers will use the checklist to determine if the design is consistent with the SRS. This includes the code structure, front-end design, and game mechanics.
- **Structured Review Meeting**: The team will schedule a review session with our supervisor where each design element will be presented. The goal being to receive feedback and make any necessary modifications.

3.4 Verification and Validation Plan Verification Plan

The V&V plan itself will be verified to make sure it adequately addresses every facet of the project and sufficiently plans appropriate methods of verification and validation.

- **Mutation Testing**: To guarantee robustness of our software, we'll use mutation testing to assess how well the test cases work and to make sure that the methods used for verification can accurately detect software faults.
- **Review Sessions**: To ensure that all necessary verification and validation methods are covered, the V&V plan will be reviewed by the teams peers and supervisor utilizing a checklist. The team will also independently review the methods to ensure all bases are covered.

3.5 Implementation Verification Plan

The implementation verification will primarily focus on unit testing and manual testing, emphasizing the isolation of modules and testing them with constant inputs mimicking user inputs. Given that project is a game, testing poses unique challenges and nuances that require careful consideration.

- **Unit Testing**: The team will employ isolated unit tests, setting fixed inputs to verify core functionalities like game logic, scoring systems, and user input handling. Simulations of dice rolls and the scores that go along with them are one example. Additionally, to ensure that tests only concentrate on the module's functionality and are not influenced by outside factors, mocking will be used to mimic external dependencies.

- **Manual Testing**: Manual testing will be used to evaluate the game from a player's point of view and make sure that gameplay is fluid, intuitive, and stimulating, manual testing will be used. In order to find any possible errors or discrepancies in the game mechanics, test cases will cover a range of scenarios, including edge cases and user interactions.

- **Code Walkthroughs**: In order to collect feedback on code correctness, standard compliance adherence, and alignment with design specifications outlined in the SRS, team-based code walkthroughs will be carried out. These walkthroughs will be held with our peers and external individuals with expertise in the field.

3.6 Automated Testing and Verification Tools

To maintain code quality and reliability, teh team will implement a variety of automated testing tools. Any mentioned tools are not finalized and may be substituted based on team preferences and needs:

- **Unit Testing Framework**: The team will employ Godot's 'GUT Test' unit testing framework, to test all in-game functionality that can be reasonably tested via unit tests. This will help ensure that each game feature functions consistently and accurately.

- **Continuous Integration (CI)**: The team will implement continuous integration through GitHub Actions to automate pdf generation when latex files are committed, and also to run unit tests on pull requests whenever code changes are made. Unit tests will be written within the GUT(Godot Unit Testing) testing framework and we will create a github action to automatically download Godot and GUT, to run the full test suite and block code merges when test cases fail.

3.7 Software Validation Plan

Our software validation will involve confirming that the system meets user expectations and the requirements outlined in the SRS.

- **Stakeholder Review Sessions**: In order to make sure that requirements match user expectations and needs, we will arrange semi-regular

review meetings with our supervisor and potential users, who are stakeholders. These meetings will be used to modify requirements as needed and will provide validation for newly developed components of the system.

- **Task-Based Inspection**: In order to demonstrate that the system performs as intended, the team will develop use-case scenarios and conduct task-based inspections to validate whether or not the system operates correctly. For example, one use-case scenario could be a player seeking to accomplish a "Full House" by rolling three dice of one number and two of another. A task-based inspection would include ensuring that the system correctly recognizes and scores this combination as a Full House, awarding the required points.

- **Rev 0 Demo Feedback**: For the purpose of further refining the system and optimizing it prior to the final release, the team will collect feedback from our supervisor both before and after the Rev 0 demo.

These strategies are additions to the ones outlined in section 2.2.

4 System Tests

This section covers tests for functional and non-functional requirements to verify the game's adherence to its design specifications. The functional requirements focus on verifying the gameplay mechanics, score calculations, and user interface, while the non-functional requirements test aspects like performance, usability, and system compatibility.

4.1 Tests for Functional Requirements

Functional tests are organized into different areas based on gameplay features, score handling, and user interface components, as defined in the software requirements specification (SRS).

4.1.1 Gameplay and Rules

Testing here covers the core game mechanics, such as player vs player mode, score calculations, and dice roll simulation.

Test: PvP Support

- **Test ID:** Test-1
- **Control:** Manual
- **Initial State:** Game loaded to main menu.
- **Input:** Two players connect to a match.
- **Output:** Game begins with both players connected, and each takes turns as expected.
- **Test Case Derivation:** Confirms successful setup and turn-taking in online mode, satisfying R1.
- **How test will be performed:** Two players join a match in PvP mode and verify that the game can be initiated and turns can be taken as expected.

Test: Score Calculation

- **Test ID:** Test-2
- **Control:** Automated
- **Initial State:** Active game session with dice rolled.
- **Input:** Player achieves four of a kind and selects the four of a kind option.
- **Output:** Correct score added from the four of a kind option.
- **Test Case Derivation:** Ensures that score calculations adhere to specified Yahtzee-like rules, satisfying R2.
- **How test will be performed:** Test case evaluates various combinations to ensure scoring accuracy.

Test: Dice Roll Physics

- **Test ID:** Test-3
- **Control:** Manual
- **Initial State:** Game session loaded.
- **Input:** Player rolls the dice.
- **Output:** 3D dice roll with randomized, realistic outcome.
- **Test Case Derivation:** Validates realistic physics for dice roll, ensuring that each outcome resembles real world physics, satisfying R3.
- **How test will be performed:** Tester initiates rolls and observes animation and outcomes.

Test: Simultaneous turn implementation

- **Test ID:** Test-4
- **Control:** Manual
- **Initial State:** Online game session loaded, Both players rolled once, one player confirmed choice of dice to reroll and is in waiting screen
- **Input:** Second player confirms choice of dice to reroll
- **Output:** Game rerolls chosen dice for both players, updates both players with the results of both rerolls, and allows both players to select new dice to reroll at the same time
- **Test Case Derivation:** Verifies that one player can't move to their next turn until both players are ready, thus enforcing the rules of Simultaneous turn implementation and satisfying requirement R6
- **How test will be performed:** Two testers in the same online session will end their turns in different orders to confirm the functionality of the implementation of Simultaneous turns.

4.1.2 User Interface and Settings

This section validates the interface, display, and customization options to ensure they meet user needs and game standards.

Test: User interface initiation

- **Test ID:** Test-5
- **Control:** Manual
- **Initial State:** Game loaded to main menu
- **Input:** Player connects to a match
- **Output:** Game begins and a UI displaying important information, such as Player names, scores, number of rolls and time limits, is displayed.
- **Test Case Derivation:** confirms the Display of UI elements conveying important information to the User, thus satisfying requirement R8
- **How test will be performed:** Upon the beginning of a match, testers will verify that expected UI is visible for display, and updates as the game goes on.

Test: Dice Selection

- **Test ID:** Test-6
- **Control:** Automated
- **Initial State:** Mid-game, with player allowed to roll again.
- **Input:** Player selects specific dice to roll and keeps those remaining.
- **Output:** Only selected dice are re-rolled, while unselected dice remain.
- **Test Case Derivation:** Validates dice selection mechanics as per game rules, satisfying R7.
- **How test will be performed:** Automated test case selects some dice and performs a roll. The test then checks that the dice that were selected are rolling, and the dice that were not selected are not rolling.

Test: Game Presets

- **Test ID:** Test-7
- **Control:** Manual
- **Initial State:** Game loaded to game select menu
- **Input:** Player selects a preset game option
- **Output:** A new game session begins with the pre-made settings applied.
- **Test Case Derivation:** Confirms that each preset produces a unique variant of the game, Satisfying R10
- **How test will be performed:** Tester selects the pre-made options and runs the games to confirm functionality.

Test: Custom Game Settings Modification

- **Test ID:** Test-8
- **Control:** Manual
- **Initial State:** Custom Game settings menu open.
- **Input:** Player modifies settings (e.g., changes dice sides, scoring rules).
- **Output:** Changes are successfully applied to new game sessions.
- **Test Case Derivation:** Verifies that customizable game settings reflect unique variants, satisfying R9 and R15.
- **How test will be performed:** Tester modifies various settings, saving them, and initiates games to confirm functionality.

4.2 Tests for Nonfunctional Requirements

Nonfunctional tests focus on measuring performance, responsiveness, reliability, and usability. These tests ensure the game performs well under different conditions and on various systems.

4.2.1 Performance and Usability

Test: Frame Rate Consistency

- **Test ID:** Test-9
- **Type:** Dynamic, Manual (from usability testing)
- **Initial State:** Game in progress.
- **Input/Condition:** Standard gameplay, including dice rolls, UI interactions, and animations.
- **Output/Result:** Game maintains a frame rate of at least 30 FPS.
- **How test will be performed:** Frame rate monitored by testers during gameplay on low and high-end systems to ensure stability.

Test: Input Responsiveness

- **Test ID:** Test-10
- **Type:** Dynamic, Manual (from usability testing)
- **Initial State:** Game loaded.
- **Input:** Player performs multiple actions (e.g., roll dice, end turn).
- **Output:** System responds within 500 ms of input.
- **How test will be performed:** Poor response times will be noted by users during usability testing.

4.2.2 System Compatibility and Reliability

Test: System Compatibility

- **Test ID:** Test-11
- **Type:** Manual
- **Initial State:** Game installed on a Windows 10 system.
- **Input/Condition:** Launch and operate the game as per standard.

- **Output/Result:** Game loads and operates without compatibility issues.
- **How test will be performed:** Run game on Windows 10 (and MacOS - stretch goal) to confirm compatibility, satisfying NFR3 (and NFR10).

Test: Crash Resilience in Multiplayer

- **Test ID:** Test-12
- **Type:** Dynamic, Manual
- **Initial State:** Multiplayer game session in progress.
- **Input/Condition:** Two players connected and actively playing.
- **Output/Result:** Game remains stable, with crashes occurring less than 1% of the time.
- **How test will be performed:** Simulated load testing over numerous sessions, logging any crashes to validate NFR4.

4.3 Traceability Between Test Cases and Requirements

The following tables provides traceability between the test cases and the requirements. Each test case is mapped to one or more requirements that it validates, ensuring coverage of our functional and non-functional requirements. Tables for keys are provided to help clarify test and requirements.

Table 4: Key for Test Cases

Test Case ID	**Test Case Name**
Test-1	PvP Support
Test-2	Score Calculation
Test-3	Dice Roll Physics
Test-4	Simultaneous turn implementation
Test-5	User interface initiation
Test-6	Dice Selection
Test-7	Game Presets
Test-8	Custom Game Settings Modification
Test-9	Frame Rate Consistency
Test-10	Input Responsiveness
Test-11	System Compatibility
Test-12	Crash Resilience in Multiplayer

Table 5: Key for Requirements

Requirement ID	**Requirement Name**
R1	Online player vs player mode
R2	Score calculation using Yahtzee rules
R3	Realistic physics for 3D dice rolls
R4	Dice values determined by actual roll outcome
R5	Support for different-sided dice
R6	Simultaneous turn-based mechanism
R7	Player selection of dice to roll
R8	User interface displaying game information
R9	Customizable game settings
R10	Preset game modes
R11*	Local player vs player mode
R12*	Player vs computer mode
R13*	Algorithmic computer opponent
R14*	Online matchmaking
R15*	Saving custom game settings
R16*	Round statistics display
R17	Accurate display of current game state
NFR1	Minimum 30 FPS performance
NFR2	Clear and easy-to-use user interface
NFR3	Support for Windows 10 and later
NFR4	Multiplayer crash rate below 1%
NFR5	Maximum input response time of 500ms
NFR6	Modular and maintainable codebase
NFR7	Optimization for low-end systems
NFR8	Enjoyability rating of at least 75%
NFR9	Consistent UI and 3D visual style
NFR10*	MacOS support

** indicates stretch goal - out of scope*

Table 6: Traceability Matrix for Test Cases and Requirements

	Tests											
Req.	1	2	3	4	5	6	7	8	9	10	11	12
R1	X											
R2		X										
R3			X									
R4			X									
R5			X									
R6				X								
R7			X			X						
R8					X							
R9								X				
R10							X					
R11												
R12												
R13												
R14	X											
R15								X				
R16												
R17												
NFR1									X			
NFR2	X											
NFR3											X	
NFR4												X
NFR5										X		
NFR6												
NFR7												
NFR8												
NFR9												
NFR10											X	

5 Unit Test Description

5.1 Unit Testing Scope

The unit testing scope includes the Dice Container Module and the Scoring Module. Other modules related to UI and networking are not the focus of unit testing at this stage. The testing priority is given to modules that directly impact gameplay mechanics and fairness.

5.2 Tests for Functional Requirements

5.2.1 DynamicDiceContainer Module

1. Test: Roll dice functionality

    - Type: Functional, Automatic
    - Initial State: New dice container instance with new dice added
    - Input: Call the roll dice functionality
    - Output: All dice get rolled
    - Test Case Derivation: Ensures that dice always get rolled when requested
    - How test will be performed: Instantiate new dice container with newly instantiated dice nodes and then verify that all dice are in a rolling state after calling the roll function

2. Test: Rolling selected dice

    - Type: Functional, Automatic
    - Initial State: New dice container with new dice added, some selected and some not
    - Input: Call the roll selected dice functionality
    - Output: Selected dice get rolled and unselected dice do not get rolled
    - Test Case Derivation: Ensures that only selected dice get rolled
    - How test will be performed: Instantiate new dice container with newly instantiated dice nodes and then verify that selected dice

are in a rolling state after calling the roll function and unselected dice are not in a rolling state

3. Test: Dice selection can be inverted

    - Type: Functional, Automatic
    - Initial State: New dice container with new dice added, some selected and some not
    - Input: Call invert selection functionality on dice
    - Output: Selected dice become unselected and unselected dice become selected
    - Test Case Derivation: Ensures that users can choose to roll selected or keep selected dice
    - How test will be performed: Instantiate new dice container with newly instantiated dice nodes where some are selected and some are not, then call the invert selection function and verify that the selected dice are now unselected and vice versa

4. Test: Dice values can be read

    - Type: Functional, Automatic
    - Initial State: New dice container with new dice added, and roll all dice
    - Input: Call the get dice values functionality
    - Output: All dice values get returned
    - Test Case Derivation: Ensures that values can be read after dice get rolled
    - How test will be performed: Instantiate new dice container with newly instantiated dice nodes and roll all dice, then call the get dice values function and verify that all dice return values

5. Test: Dice collisions can be toggled

    - Type: Functional, Automatic
    - Initial State: New dice container with new dice added
    - Input: Call functionality to toggle collisions off, and then back on

- Output: All dice lose collidable property, then regain it
- Test Case Derivation: Ensures that dice collisions can be toggled as needed for dice animations
- How test will be performed: Instantiate new dice container with newly instantiated dice nodes, then call the toggle collisions function and verify that each die has collisions disabled, then call the function again and verify that each die has collisions enabled again

5.2.2 ScoreCalculator Module

1. Test: Calculate score for a full house
    - Type: Functional, Automatic
    - Initial State: Set of dice showing full house
    - Input: Dice values
    - Output: Expected full house score
    - Test Case Derivation: Ensures correct score computation
    - How test will be performed: Pass dice values and check score
2. Test: Calculate score for a kind
    - Type: Functional, Automatic
    - Initial State: Set of dice showing a kind
    - Input: Dice values and the quantity for the kind
    - Output: Expected kind score
    - Test Case Derivation: Ensures scoring logic applies correctly
    - How test will be performed: Pass kind roll and verify score
3. Test: Score calculation for a straight
    - Type: Functional, Automatic
    - Initial State: Set of dice showing sequence
    - Input: Dice values and some expected sequence quantity
    - Output: Expected straight score

- Test Case Derivation: Ensures proper scoring for straights
- How test will be performed: Provide sequence values and check score

4. Test: Score calculation for singles

    - Type: Functional, Automatic
    - Initial State: Set of dice showing singles
    - Input: Dice values and the single to score on
    - Output: Expected singles score
    - Test Case Derivation: Ensures proper scoring for singles
    - How test will be performed: Provide singles values and check score

5. Test: Score calculation for chance

    - Type: Functional, Automatic
    - Initial State: Random set of dice
    - Input: Dice values
    - Output: Expected chance score
    - Test Case Derivation: Ensures proper scoring for chance
    - How test will be performed: Provide dice values and check score

5.3 Traceability Between Test Cases and Modules

Test Case	Modules
test_roll_dice	DynamicDiceContainer Module
test_roll_selected_dice	DynamicDiceContainer Module
test_invert_dice_selection	DynamicDiceContainer Module
test_read_dice_values	DynamicDiceContainer Module
test_toggle_dice_collisions	DynamicDiceContainer Module
test_full_house_scoring	ScoreCalculator Module
test_kind_scoring	ScoreCalculator Module
test_straight_scoring	ScoreCalculator Module
test_singles_scoring	ScoreCalculator Module
test_chance_scoring	ScoreCalculator Module

Table 7: Table of Traceability Between Test Cases and Modules

Test Case	DynamicDiceContainer	ScoreCalculator
test_roll_dice	X	
test_roll_selected_dice	X	
test_invert_dice_selection	X	
test_read_dice_values	X	
test_toggle_dice_collisions	X	
test_full_house_scoring		X
test_kind_scoring		X
test_straight_scoring		X
test_singles_scoring		X
test_chance_scoring		X

Table 8: Matrix of Traceability Between Test Cases and Modules

References

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?

There will be ongoing discussion and interactions while running the multi-player game, with the observer taking notes during the process. After gameplay, the user will be directed to a form where they can answer the following questions:

- Gameplay Clarity and Usability
  - What is your name?
  - Was the tutorial or onboarding process intuitive?
  - Were the game controls easy to understand?
  - Did you encounter any areas of confusion while playing?
- Engagement and Enjoyment
  - How engaging did you find the game?
  - Would you play this game again?
  - What aspect of the game did you enjoy the most?
- Multi-Player Interaction
  - Did you play the health-scored variant?
  - Did you feel like you were playing against another player?
  - Was the multi-player aspect of the game engaging?
  - What aspect of the multi-player game did you enjoy the most?
  - What aspect of the multi-player game could be improved?

- Difficulty and Challenge
  - Did the game feel fair and balanced?
  - Was there a particular strategy that felt overpowered?
  - Did you feel that your decisions significantly impacted the outcome of the game?
- Suggestions and Improvements
  - If you have played previous versions of the game, what did you notice has changed?
  - What changes would you suggest to improve the game experience?
  - Are there any additional features or modifications you would like to see?

Appendix — Reflection

1. What went well while writing this deliverable?

    - We were able to use past year examples to set up our document and understand what contents to add to each section - Hemraj B
    - Since we had already done the requirements and hazard analysis documentation, we had a better understanding of the project we are designing for. Additionally, we have a better idea of what the project will come to look like given we are developing our proof of concept. - John P.

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

    - Sections 3 and 4, were confusing in terms of us thinking that both essentially meant the same thing. We resolved this by looking at the previous year's examples and were able to figure out the differences between the two sections - Hemraj B
    - We wanted to focus more attention to the proof of concept, but this deadline could come useful for near the end of the project. - John P.
    - When it comes to choosing a challenge level, this is something that needs to be set with a meeting with the professor. We had initially had the project as advanced on the proposal, but the professor said everyone is expected to start with a general level. A meeting with the professor is something that would have helped this section. - John P.

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.

    - We will need to familiarize ourselves with testing methods that were mentioned in our document such as Coverlet and FxCop - Hemraj B

- We are working with GoDot, so we need to do some exploration with what tools best work with that software. Additionally, since this is a game we are developing where most elements are driven by user interaction. This also brings some difficulties and thus something that we need to explore. - John P.
- As of right now there is no CI/CD element to our repository, especially given that 100% of the commits to the main branch as of right now are for documentation. This is a skill we need to develop and brainstorm how we can best make use of this technology. - John P.

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?

    - We will acquire knowledge of Coverlet and FxCop through watching educational videos on YouTube and we will create trial projects to practice the software and familiarise ourselves with the tools available on each software - Hemraj B
    - We need to do research on GoDot and how verification tools can work with it. This would come through looking at online resources that have some experience with GoDot and we can use their expertise, but could also come through locally testing different software methods and bringing them to the team once integrated. Similarly with testing mostly user driven interactions, we would need to brainstorm as a team and test options, but perhaps also bring it up to the professor and TAs who might have useful ideas. - John P.
    - The problem of having CI/CD testing for games has been brought up in lecture, and as such is an issue the professor and TAs should be aware of. Asking for their advice and experience would be useul in this regard, as would searching online forums and videos for what others have done. This can take place at the same time as the above bullet point, since GoDot is primarily used for game development. - John P.