Comprehensive Exercise Report

Team OMARK of Section 123

Omar Fourati, 240ADM004

Magnus Christian Schneider, 240ADM006

[**Requirements/Analysis**](#_uwgqwd5ezv2w) **2**

[Journal](#_lsityg2iq9m6) 2

[Software Requirements](#_2h0vru1u2mla) 3

[**Black-Box Testing**](#_prhaxdxmf8n8) **4**

[Journal](#_18f11w613jft) 4

[Black-box Test Cases](#_2xn4jzot820y) 5

[**Design**](#_24fdizefyocn) **6**

[Journal](#_esp2ocs9j6bk) 6

[Software Design](#_aifbl1x6rddt) 7

[**Implementation**](#_hya8f3jqkba6) **8**

[Journal](#_acupzfhai7gz) 8

[Implementation Details](#_ojhtwkms2z3b) 9

[**Testing**](#_3qvya3vi836q) **10**

[Journal](#_ckfs4xbl5pyr) 10

[Testing Details](#_bzt1547yxzxi) 11

[**Presentation**](#_hdjvrbf45b1p) **12**

[Preparation](#_xbiquwtmf36n) 12

[**Grading Rubric**](#_u0hfnmdgusmf) **13**

# Requirements/Analysis

Week 2

## Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* After reading the client’s brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
  + The goal is to create a game software that is as close to the existing Bunco game as possible.
  + From the client, the only available information is about the game rules which will help us deduct the requirements of the game. These requirement will be described within the next bullet points and sections.
* After reading the client’s brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.
  + None, all the requirements can be deducted from the game rules.
* Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
  + None
* Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
  + Broad audience:
    - Casual Gamers who enjoy simple and entertaining games without complex rules:
    - Party goers and social gathering enjoyers.
    - All age groups.
    - Limited gaming experience people
* Describe how each user would interact with the software.
  + Casual gamers
    - Casual gamers interact with the software by launching the game and entering a desired username.
    - They select a difficulty level (Easy, Medium, Difficult) based on their preferences.
    - The option to choose a tutorial or skip it caters to users with varying levels of familiarity with the game.
    - Users engage in the main game interface, where they roll dice, accumulate points, and progress through rounds.
  + Party goers and social gathering enjoyers.
    - Participants at social events open the software and join the game, entering a username.
    - The software sets up the gaming environment, creating tables and assigning random names to players.
    - Users take turns rolling dice during each round, fostering a social and interactive atmosphere.
    - As rounds progress, players move between tables, enhancing social interaction and engagement.
  + All age groups.
    - Individuals of all age groups launch the game and input a username.
    - The software accommodates different age preferences by offering various difficulty levels.
    - Children and adults alike engage in the excitement of rolling dice, scoring points, and enjoying the social aspect of the game.
  + Limited gaming experience people
    - Users with limited gaming experience find the software accessible, with clear instructions and a potential tutorial.
    - They follow on-screen prompts to roll dice and accumulate points, with the game's simplicity promoting ease of play.
    - The software's user-friendly interface ensures that even those new to gaming can navigate and enjoy the experience.
* What features must the software have? What should the users be able to do?
  + - User Registration
    - Difficulty Selection
    - Game Rule Visualization
    - Game Initialization
    - Round Display
    - Dice Rolling
    - Score Tracking
    - Round Progression
    - Scoring Rules
    - Scoring Display(Table)
    - Game Termination
    - Result Calculation
    - User-Friendly Interface
    - Option to Quit or Restart

## Software Requirements

***Functional Requirements:***

1. **User Registration:**
   * Users should be able to enter a username to participate in the game.
2. **Difficulty Selection:**
   * Users should be able to choose the game difficulty (Easy, Medium, Hard), affecting your scoring probabilities
3. **Game Rule Visualization:**
   * Users can access a extra view to read the game rules during the game experience.
4. **Game Initialization:**
   * Based on the selected difficulty, the system should create the specified number of tables with randomly assigned names for each player.
5. **Round Display:**
   * The system should display and announce the current round number (1 to 6).
6. **Dice Rolling:**
   * Players should be prompted to roll the dice during each round.
7. **Score Tracking:**
   * The system should keep track of the points earned by each player and display them in a table.
8. **Round Progression:**
   * The game should progress to the next round when the first table reaches 21 points.
9. **Scoring Rules:**
   * Points should be awarded based on the rules, including one point for each dice matching the round number, five points for three of a kind with no matching round number, and 21 points for three of a kind matching the round number.

***Non-Functional Requirements:***

1. **Usability:**
   * The user interface should be intuitive and easy to navigate.
2. **Performance:**
   * The system should respond promptly to user actions, especially during dice rolls.
3. **Reliability:**
   * The game should reliably progress through rounds and accurately track scores.
4. **Security:**
   * Protect against potential exploits or cheating.

***Constraints:***

1. **Platform Compatibility:**
   * The game should be developed for a specific operating system or be platform-independent.
2. **Programming Language:**
   * Specify the programming language to be used for system development.
3. **Responsive Design**
   * The game must be playable on both PC and mobile devices while maintaining a fitting resolution at all times.

***End-of-Game Requirements:***

1. **Result Calculation:**
   * Calculate and display the best table and best player based on the highest scores after six rounds.
2. **Game Termination:**
   * End the game after six rounds and declare the winner.

# Black-Box Testing

Instructions: Week 4

## Journal

***Remember:*** Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

* What does input for the software look like (e.g., what type of data, how many pieces of data)?
  + Username: A string representing the user's chosen username.
  + Difficulty Level: An integer or string indicating the selected difficulty level (Easy, Medium, Hard).
  + Game Initialization: No direct user input; system-generated tables with random player names.
  + Dice Rolling: User-triggered action; no specific input required.
  + Round Progression: Triggered by system based on points reaching 21; no direct user input.
  + Option to Quit or Restart: User input for selecting to exit the game or restart for additional rounds.
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
  + Round Display: An integer indicating the current round number (1 to 6).
  + Score Tracking: Tables displaying points for each player and the cumulative points earned.
  + Result Calculation: Displaying the best table and best player based on the highest scores.
* What equivalence classes can the input be broken into?
  + Username:
    - Valid usernames (alphanumeric characters, special characters, length constraints).
    - Invalid usernames (empty, excessively long).
  + Difficulty Level:
    - Valid difficulty levels (Easy, Medium, Hard).
    - Invalid difficulty levels (non-existent levels, numeric values).
  + Tutorial Option:
    - True or False (Boolean).
  + Option to Quit or Restart:
    - User selects to quit or restart (valid input).
    - User selects other options or provides unexpected input (invalid input).
* What boundary values exist for the input?
  + Username Length: Test for minimum and maximum length.
  + Difficulty Level: Test for minimum and maximum values.
  + Dice Rolling: Simulate dice rolls with minimum and maximum values.
* Are there other cases that must be tested to test all requirements?
  + Game Termination: Testing the end of the game after six rounds.
* Other notes:
  + Test the system's response to unexpected or erroneous user inputs.
  + Evaluate the impact of simultaneous actions from multiple users in a multi-player setting.

## Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

|  |  |  |  |
| --- | --- | --- | --- |
| Test ID | Description | Expected Results | Actual Results |
| 1 | Valid Username | User enters a valid alphanumeric username. |  |
| 2 | Invalid Username (Empty) | User leaves the username field empty. |  |
| 3 | Invalid Username (Excessively Long) | User enters a username beyond the character limit. |  |
| 4 | Valid Difficulty Level | User selects a valid difficulty level (Easy, Medium, Difficult). |  |
| 5 | Invalid Difficulty Level (Numeric) | User enters a numeric value as the difficulty level. |  |
| 6 | Valid Tutorial Option (True) | User opts for the tutorial. |  |
| 7 | Valid Tutorial Option (False) | User skips the tutorial. |  |
| 8 | Invalid Tutorial Option (Non-Boolean) | User provides an unexpected value for the tutorial option. |  |
| 9 | Option to Quit Game | User selects to quit the game. |  |
| 10 | Option to Restart Game | User selects to restart the game. |  |
| 11 | Username Length (Minimum) | User enters a username with the minimum length. |  |
| 12 | Username Length (Maximum) | User enters a username with the maximum length. |  |
| 13 | Dice Rolling (Simulation) | Simulate dice rolls with minimum and maximum values. |  |
| 14 | Social Interaction (Move Between Tables) | Test the system's response to moving between tables during rounds. |  |
| 15 | Game Termination (End of 6 Rounds) | Test the end of the game after completing six rounds. |  |
| 16 | Input Validation (Unexpected Input) | Enter unexpected or erroneous input for various user actions. |  |
| 17 | Multi-Player Interaction | Simulate simultaneous actions from multiple users in a multi-player setting. |  |

# Design

Instructions: Week 6

## Journal

***Remember:*** You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* List the nouns from your requirements/analysis documentation.
  + GameSettings(difficulty)
  + Tutorial
  + Game
  + Round
  + Dice
  + Scoreboard
  + Scoring
  + UI
  + Security
  + GameResult
* Which nouns potentially may represent a class in your design?
  + GameSettings
  + Tutorial
  + Game
  + Round
  + Dice
  + Scoreboard
  + Scoring
  + UI
  + Security
  + GameResult
* Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
  + GameSettings
    - difficultyLevel (GameSettings)
  + Round
    - roundNumber (Round)
  + Scoreboard
    - scores (Scoreboard)
    - bestTable (Scoreboard)
    - bestPlayer (Scoreboard)
  + Dice
    - value (Dice)
  + Scoring
    - roundScore (Scoring)
    - totalScore (Scoring)
  + UI
    - displayMessage (UI)
  + Security
    - validatedInput (Security)
  + GameResult
    - winner (GameResult)
    - endStatus (GameResult)
* Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.
  + Design Option 1: Monolithic Design
    - Classes and Attributes:
      * User (username)
      * GameSettings (difficultyLevel)
      * Tutorial
      * Game
      * Round (roundNumber, currentPlayers)
      * Dice (value)
      * Scoreboard (scores, bestTable, bestPlayer)
      * Scoring (roundScore, totalScore)
      * UI (displayMessage)
      * Security (validatedInput)
      * GameResult (winner, endStatus)
    - Pros:
      * Simple and straightforward
      * All functionalities in one place
    - Cons:
      * Less modular
      * Difficult to maintain and scale
  + Design Option 2: Modular Design
    - Classes and Attributes:
      * Settings (difficultyLevel)
        + GameSettings
        + Tutorial
      * Game
        + User (userName)
        + Round (roundNumber, currentPlayers)
        + Dice (value)
        + Scoreboard (scores, bestTable, bestPlayer)
        + Scoring (roundScore, totalScore)
      * UI (displayMessage)
        + Security (validatedInput)
        + GameResult (winner, endStatus)
    - Pros:
      * Modular and easier to maintain
      * Encapsulation of functionalities
    - Cons:
      * More classes may lead to complexity
      * Inter-class communication might be needed
* Which design do you plan to use? Explain why you have chosen this design.

We plan to use Modular Design (Design Option 2) because it offers better modularity, easier maintenance, and encapsulation of functionalities. This design allows for scalability and makes it easier to manage each component of the game separately, leading to a more organized and maintainable codebase.

* List the verbs from your requirements/analysis documentation.
  + enter, choose, indicate, create, display, announce, prompt, keep, progress, award, match, navigate, respond, track, protect, calculate, terminate, declare
* Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
  + After taking those exact verbs and adding the context of the requirement to them we end up with:
  + Settings: setDifficulty(), displayTutorial()
  + Game: initializeGame(), setPlayerUserName(), displayRound(), nextRound(), rollDice(), calculateRoundScore(), calculateTotalScore(), endGame(), calculateWinner(), determineEndStatus()
  + Scoreboard: updateScoreBoard(), caculateBestTable(), calculateBestPlayer()

## Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Your design should include UML class diagrams along with method headers. ***Prior to starting the formal documentation, you should show your answers to the above prompts to your instructor.****>>*

Ein Bild, das Screenshot, Text, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

# Implementation

Instructions: Week 8

## Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation:

* State Management: To keep track of the game's state, including player scores, current round, dice rolls, and game progress.
* Event Handling: For capturing user interactions such as rolling dice, starting a new game, or navigating through the game's interface.
* Modularization: Breaking down the application into smaller, reusable components for better code organization and maintainability.
* Version Control (Git): Tracking changes, collaborating with other developers, and managing different versions of the application.
* Testing: Ensuring the correctness and reliability of the application through unit tests, integration tests, and automated testing.
* Documentation: Providing clarity and guidance to developers and users through comments, README files, and user guides.

Other notes:

* Code Reviews: Conducting peer code reviews to identify potential issues, share knowledge, and improve code quality.
* Agile Development: Adopting agile methodologies for managing project scope, prioritizing tasks, and delivering incremental value.
* User Experience (UX) Design: Considering the overall user experience and interface design of the game to ensure it is intuitive and engaging for players.

## Implementation Details

<<Use your notes from above to write code and complete this section of the formal documentation with a README for the user that explains how he/she will interact with the system.>>

**Simplified Bunco**

Welcome to the Bunco Game System! This system allows users to play a simplified version of the Bunco dice game. Follow the instructions below to interact with the system.

**Getting Started**

To use the Bunco Game System, you'll need to have Node.js and npm (Node Package Manager) installed on your machine. If you haven't installed them yet, you can download and install them from [nodejs.org](https://nodejs.org/).

**Installation**

1. Clone this repository to your local machine:

bash

Copy code

git clone <repository-url>

1. Navigate to the project directory:

bash

Copy code

cd bunco-game-system

1. Install project dependencies:

bash

Copy code

npm install

**Running the Application**

To run the Bunco Game System, follow these steps:

Web-Application:

1. Open <https://simplified-bunco.netlify.app/> in the web browser.

Alternative NPM:

1. Start the application:

bash

Copy code

npm start

1. The application will open in your default web browser. If not, navigate to http://localhost:3000 in your web browser.

**Interacting with the System**

**New Game**

1. Click on the "New Game" button on the header to start a new game.
2. Enter your username and select a difficulty level (Easy, Medium, or Hard) in the modal window that appears.
3. Click "Start Game" to begin playing.

**Game Rules**

1. Click on the "Game Rules" button on the header to view the rules of the game.
2. Read through the game rules and close the modal window by clicking "Close".

**Gameplay**

1. Once the game starts, you will see the dice roll area and game board components.
2. Click the "Roll Dice" button in the dice roll area to roll the dice.
3. Try to match the target number for each round and score points according to the game rules.
4. The game consists of multiple rounds, and the winner(s) will be displayed at the end of the game.

**Final Results**

1. After completing all rounds, the final results modal will appear automatically.
2. The winners of the game (both players and tables) will be displayed in the final results modal.

**Additional Information**

* The footer contains a disclaimer about the experimental nature of this game.

# Testing

Instructions: Week 10

## Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

* Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
  + <<We changed a tutorial option towards a modal showing the game rules for simplicity>>
* List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
  + <<Insert class>>
    - <<Insert needed tests>>
  + <<Insert class and tests for each class>>
* Other notes:
  + <<Insert notes>>

## 

## 

## Testing Details

<<Use your notes from above to write your test programs and complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing. You will also complete the black-box test plan by running the program and filling in the Actual Results column.>>

# Presentation

Instructions:Week 12

## Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

* Give a brief description of your final project
  + <<A simplified Bunco game>>
* Describe your requirement assumptions/additions.
  + <<Insert answer>>
* Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
  + <<Insert answer>>
* How did the extension affect your design?
  + <<Insert answer>>
* Describe your tests (e.g., what you tested, equivalence classes).
  + <<Insert answer>>
* What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
  + <<Insert answer>>
* What functionalities are you going to demo?
  + <<Insert answer>>
* Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)
  + <<Insert answer>>
* Other notes:
  + <<Insert notes>>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>