Comprehensive Exercise Report

Team <<Chubaruk>> of Section <<000>>

Oleksii Lychov 201ADB084

Firad Aslanov 211ADB020

Shamil Mammadrzayev 211ADB022

Murad Rajabov 201ADB075

NOTE: You will replace all placeholders that are given in <<>>

[**Requirements/Analysis**](#_heading=h.3znysh7) **[2](#_heading=h.3znysh7)**

[Journal](#_heading=h.2et92p0) [2](#_heading=h.2et92p0)

[Software Requirements](#_heading=h.tyjcwt) [3](#_heading=h.tyjcwt)

[**Black-Box Testing**](#_heading=h.3dy6vkm) **[4](#_heading=h.3dy6vkm)**

[Journal](#_heading=h.1t3h5sf) [4](#_heading=h.1t3h5sf)

[Black-box Test Cases](#_heading=h.4d34og8) [5](#_heading=h.4d34og8)

[**Design**](#_heading=h.2s8eyo1) **[6](#_heading=h.2s8eyo1)**

[Journal](#_heading=h.17dp8vu) [6](#_heading=h.17dp8vu)

[Software Design](#_heading=h.3rdcrjn) [7](#_heading=h.3rdcrjn)

[**Implementation**](#_heading=h.26in1rg) **[8](#_heading=h.26in1rg)**

[Journal](#_heading=h.lnxbz9) [8](#_heading=h.lnxbz9)

[Implementation Details](#_heading=h.35nkun2) [9](#_heading=h.35nkun2)

[**Testing**](#_heading=h.1ksv4uv) **[10](#_heading=h.1ksv4uv)**

[Journal](#_heading=h.44sinio) [10](#_heading=h.44sinio)

[Testing Details](#_heading=h.3j2qqm3) [11](#_heading=h.3j2qqm3)

[**Presentation**](#_heading=h.1y810tw) **[12](#_heading=h.1y810tw)**

[Preparation](#_heading=h.4i7ojhp) [12](#_heading=h.4i7ojhp)

[**Grading Rubric**](#_heading=h.2xcytpi) **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 aim of this project is to create a software program that enables players to participate in Bunco, which is a straightforward dice game
    - the capacity to roll three dice
    - the capacity to keep track of scores for each round
    - to facilitate multiple players
    - to indicate the current player's turn.
* 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.
  + What kind of design we are going to use ?
  + What are the terms and when are the deadlines?
  + Will we use advertisement inside the product?
  + What are the rules of Privacy Policy we are going to use?
* Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
  + Game design
  + UX design
  + GUI design
* Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
  + Junior player category (13-18)
  + Adult player category (18-65)
  + Seniour player category (65~)
* Describe how each user would interact with the software
  + User would be able to use basic menu functions: Start game, and Exit.
  + User can roll a dice by pressing a button and see the progress of the game
* What features must the software have? What should the users be able to do?
  + Sound effects: Sound effects can add to the fun and excitement of playing Bunco, so the software should have a feature that plays different sound effects when certain events occur during the game.
  + Dice rolling: Since Bunco is a game that revolves around rolling dice, the software should have a feature that allows players to roll the dice virtually.
  + Easy to use: The software should be intuitive and easy to use,
* Other notes:
  + <<Insert notes>>

## Software Requirements

<<Use your notes from above to complete this section of the formal documentation by writing a detailed description of the project, including a paragraph overview of the project followed by a list of requirements (see lecture for format of requirements). You may also choose to include user stories.>>

Software Requirements:

Overview: The aim of this project is to create a software program that enables players to participate in Bunco, which is a straightforward dice game. The software will be designed to accommodate multiple players, allowing each player to roll three dice and keep track of scores for each round. The software will also indicate the current player's turn. The target users of this software are junior players (ages 13-18), adult players (ages 18-65), and senior players (age 65 and above). The software should be intuitive and easy to use, allowing users to interact with the program by using basic menu functions to start the game and exit, as well as the ability to roll a dice by pressing a button and seeing the progress of the game. The software should have sound effects that add to the fun and excitement of playing Bunco.

Requirements: Functional:

1. The software shall allow multiple players to participate in a game of Bunco.
2. The software shall allow each player to roll three dice.
3. The software shall keep track of scores for each round.
4. The software shall indicate the current player's turn.
5. The software shall have a feature that allows players to roll the dice virtually.
6. The software shall have a feature that plays different sound effects when certain events occur during the game.
7. The software shall have basic menu functions, including the ability to start the game and exit.

Non-functional:

1. Scalability: the software should be able to accommodate a varying number of players without performance issues
2. Reliability: the software should be stable and not crash or produce errors during use
3. Performance: the software should not be laggy . It should work properly
4. User-friendly : It should be clean and neat so users can use easly .

Constraints:

1. Graphics : It may be limited on terms of graphics compare to other bunco games on market .
2. Time: Limited amount of time is given to finish
3. Experience: Lack of experience on game development
4. Resources: Not the best resources may team have.

User Stories: As a junior player, I want to be able to easily navigate the software and understand how to play Bunco so that I can have fun and enjoy the game with my friends. As an adult player, I want the software to be reliable and bug-free, so that I can play the game without any interruptions or issues. As a senior player, I want the software to have a clear and readable interface, so that I can see my scores and understand the game easily.

# 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)?
  + The input for the software includes various types of data, including player names, dice rolls, and game settings. The number of pieces of data will depend on the number of players, number of rounds, and other settings selected.
* What does output for the software look like (e.g., what type of data, how many pieces of data)?
  + The output for the software includes game scores, current player turn, and other game-related information. The number of pieces of data will depend on the number of players, number of rounds, and other settings selected.
* What equivalence classes can the input be broken into?
  + The input can be broken into equivalence classes based on the number of players, number of rounds, type of dice rolls, and other game settings selected.
* What boundary values exist for the input?
  + Boundary values for the input include minimum and maximum number of players, minimum and maximum number of rounds, minimum and maximum number of dice rolls, and other game settings
* Are there other cases that must be tested to test all requirements?
  + Other cases that must be tested include error-handling cases such as incorrect input values, invalid user actions, and unexpected software behavior.
* Other notes:
  + The testing should also include testing for sound effects, GUI design, and user experience. Additionally, the software should be tested for reliability and performance, ensuring that it can handle multiple players and maintain stability without crashing or producing errors during use.

## 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** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# 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.
  + Software, program, players, Bunco, dice, scores, round, turn, menu, button, sound effects, events, performance, stability, scalability, graphics, time, experience, resources.
* Which nouns potentially may represent a class in your design?
  + Game: This class would represent the game itself, and would have attributes such as the number of players, the current round, and the scores for each player.
  + Dice: This class would represent the dice, and would have attributes such as the number of sides, and the value of the dice.
* Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
  + Number of players: This attribute would be part of the Game class.
  + Current round: This attribute would be part of the Game class.
  + Scores: This attribute would be part of the Game class.
  + Number of sides: This attribute would be part of the Dice class.
  + Value of the dice: This attribute would be part of the Dice class.
* 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:

Classes:

Game

Dice Attributes/Fields:

Number of players

Current round

Scores

Number of sides

Value of the dice

Pros: This design is simple and straightforward, with only two classes needed to represent the core functionality of the game.

Cons: This design may not be flexible enough to accommodate additional features or changes in requirements.

* + Design Option 2: Classes:

Game

Dice

Player Attributes/Fields:

Number of players

Current round

Scores

Number of sides

Value of the dice

Player name

Player score

Pros:

This design is more flexible, with a separate class for players that could potentially allow for additional features such as player profiles or statistics.

Cons: This design may be more complex and may require additional development time to implement.

* Which design do you plan to use? Explain why you have chosen this design.

First one, as it is simpler to implement

* List the verbs from your requirements/analysis documentation.
  + Allow
  + Roll
  + Indicate
  + Play
  + Accommodate
  + Have
  + Start
  + Exit
  + Scale
  + Be
  + Crash
  + Produce
  + Work
  + Use
* Which verbs potentially may represent a method in your design? Also list the class each method would be part of.
  + Allow: This could potentially be a method in a class that handles the game logic and manages the players.
  + Roll: This could potentially be a method in a class that handles the dice rolling functionality.
  + Keep track: This could potentially be a method in a class that manages the scorekeeping for each round.
  + Indicate: This could potentially be a method in a class that manages the current player's turn and updates the interface accordingly.
  + Play: This could potentially be a method in a class that handles the overall game flow and progression.
  + Start: This could potentially be a method in a class that handles the initial setup and configuration of the game.
  + Exit: This could potentially be a method in a class that handles the termination of the game and cleanup tasks.
* Other notes:
  + <<Insert notes>>

## 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.****>>*

A picture containing table

Description automatically generated

* The Dice class represents a single dice object. It has two fields, sides and value, and a method roll() that shows a random number between 1 and the number of sides.
* The Game class represents the game. It has fields for the number of players, the current round, the scores of each player, and a list of three Dice objects. It has \_\_init\_\_ method that takes the number of players as input, and initializes the fields. It also has methods play\_round() that plays a single round, and play\_game() that plays the game. play\_round() rolls the dice, calculates the score, and displays the dice images and the score. play\_game() calls play\_round() for each round, and displays the final scores at the end. It also has a method get\_scores() that returns the scores.
* The start\_game() function is called when the "Start" button is clicked on the start screen. It creates a Game object with the number of players, and calls its play\_game() method.
* The main\_menu() function is called when the program starts. It displays a main menu with two options, "Start Game" and "Exit". If "Start Game" is clicked, it calls start\_game().

class Dice:

def \_\_init\_\_(self, sides=6):

Initializes a new Dice object with the given number of sides.

param sides: the number of sides on the dice (default 6)

class Game:

def \_\_init\_\_(self, players):

Initializes a new Game object with the given number of players.

param players: the number of players

def play\_round(self):

Plays a single round of the game.

def play\_game(self):

Plays the entire game.

def get\_scores(self):

Returns a list of the scores for each player.

def start\_game():

Starts a new game.

def main\_menu():

Displays the main menu.

# 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.
* ***Importing Modules:*** Software code starts with importing of specific modules. These are random for random number generation, PIL for manipulating images and PySimpleGUI for creating GUI interfaces.
* ***Concepts of Object-Oriented Programming:*** Software represents the implementation of the OOP principles by defining two classes: "Dice" and "Game". The first one corresponds to a single die and allows the method-usage of rolling the die with retrieving its value. The second class represents the game itself by handling the gameplay with its rounds, players, scores, and dice rolls.
* ***Image Processing:*** The code utilizes the PIL module to handle the loading and manipulation of dice images. It reads the dice images, compose their size using the ImageOps.fit() method to a predefined size, and converts them to the PNG format. These processed images are then stored in a dictionary for convenient retrieving during the game.
* ***GUI Development:*** To create the graphical user interface for the game, the code uses the PySimpleGUI module. It defines two separate screens: the main menu screen and the game screen. These screens are responsible for displaying various elements such as text, input fields, buttons, and images. They enable user interaction and provide a visual interface for the game, allowing players to input information and receive visual feedback during gameplay.
* ***Random Number Generation:*** The code uses the random module to generate random values for the dice rolls. By calling the random.randint() function, the roll() method in the Dice class generates a random integer between 1 and the number of sides on the die. This random number generation ensures that each dice roll produces a fair and unpredictable outcome.
* ***Error Handling:*** The code implements error handling to ensure valid input from the user. In the start\_game() function, it attempts to convert the user-provided number of players to an integer. If the conversion fails, an exception is caught, and an error message is displayed using sg.popup().
* ***Looping and Iteration:*** The game logic is implemented using loops and iterations. The play\_game() method in the Game class runs a loop that iterates over the rounds until the sixth round is completed. Within each round, another loop iterates over the players, allowing each player to take their turn.
* ***List and Dictionary Manipulation:*** Lists and dictionaries are used to store and manipulate game data. The scores are stored in a list, with each player's score at a corresponding index. The dice images are stored in a dictionary, with the dice values as keys and the image data as values.
* ***String Manipulation and Formatting:*** Strings are manipulated and formatted to display messages and scores to the user. The code constructs strings using concatenation and the str() function to convert numerical values to strings. It also utilizes string formatting to display round numbers and player scores dynamically.
* ***User Input and Interaction:*** The GUI allows the user to input the number of players and interact with the game by clicking buttons and closing windows. The code uses event handling to capture user actions and perform corresponding operations, such as starting the game or exiting the program.
* ***Modularity:*** The code is modular, with different functionalities separated into functions and classes. This modular design improves code organization, readability, and maintainability.
* Other notes:
  + - In summary, the implementation of the Bunco game showcases various programming concepts, including OOP, GUI development, image processing, random number generation, error handling, looping and iteration, data manipulation, string formatting, and user input handling. These concepts work together to create an interactive and visually appealing game experience.

## Implementation Details

# Bunco Game

Welcome to the Bunco game! This game allows you to enjoy the classic dice game in a graphical user interface. In Bunco, players roll three dice and earn points based on the outcome of their rolls. Let's explore how to interact with the system and play the game.

## System Requirements

- Python 3.x

- Required Python packages: random, PIL, PySimpleGUI

## Installation

1. Make sure you have Python 3.x installed on your system.

2. Install the required Python packages by running the following command in your terminal or command prompt:

```

pip install Pillow PySimpleGUI

```

3. Download the Bunco game files from the repository or copy the provided code.

## Game Instructions

1. Run the program by executing the following command in your terminal or command prompt:

```

python bunco\_game.py

```

2. The main menu screen will appear, showing the options to start the game or exit.

![Main Menu](main\_menu.png)

3. Click the "Start Game" button to begin the game.

4. In the game screen, enter the number of players in the input field and click the "Start" button.

![Start Game](start\_game.png)

5. The game will progress through six rounds, with each player taking turns.

6. A popup will display the current round number and the player's turn.

![Popup](round\_popup.png)

![Popup](player\_popup.png)

7. Click the "OK" button to roll the dice.

8. The dice values and their corresponding images will be displayed.

![Dice Roll](dice\_roll.png)

9. The player's score for the round will be shown in a popup.

10. Repeat steps 6 to 9 for each player in the game.

11. After six rounds, the game will end, and a popup will show the final scores for each player.

![Game Over](game\_over.png)

12. Close the popup to exit the game.

Enjoy playing Bunco! If you have any questions or feedback, please contact us at SMFLrtu@gmail.com  
  
  
*Note: Image resources are available on GitHub*

# 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.
  + <<Insert answer>>
* 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
  + <<Insert answer>>
* 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.>>