[Project Title]

Use Case Description

[Insert a comprehensive and accurate description of the project's purpose, goals, and users' needs here. Be sure to address who will be using the project, why they need it, and what specific tasks or goals they will accomplish with it.]

Class List

[Insert a comprehensive and well-defined list of potential classes here that accurately reflect the project's requirements and demonstrate an understanding of object-oriented design principles. Be sure to include each class's responsibilities and how it will contribute to the overall project.]

Data and Function Members

[Insert a comprehensive and well-defined list of potential data and function members for each class here that aligns with the class's responsibilities and demonstrates encapsulation. Be sure to include how each data and function member contributes to the class's responsibilities and how it will interact with other classes.]

Relationships between Classes

[Insert a well-defined, accurate, and comprehensive description of the relationships between the identified classes here that demonstrates an understanding of the various types of relationships (e.g., inheritance, composition, association). Be sure to include how each relationship contributes to the project's goals and how it affects the classes involved.]

Project Task List and Timeline

[Insert a well-defined and structured list of project tasks here in a logical sequence and a realistic and achievable timeline that demonstrates an understanding of the project's requirements and dependencies. Be sure to include any specific deadlines or milestones that must be met.]

User Interaction Description

[Insert a clear, concise, and accurate description of how the user will interact with the program here that demonstrates an understanding of the user's needs. Be sure to include any input or output methods, user interfaces, and error handling.]

Unit Testing and Debugging Plan

[Insert a well-defined and comprehensive plan for unit testing and debugging here. Be sure to include specific testing methods, criteria for success, and how debugging will be handled in the event of errors or issues.]

Project Title: Video Game Inventory Management System

Use Case Description
The Video Game Inventory Management System is a program designed to help video game store owners manage their inventory efficiently. The system will enable store owners to add, delete, and update physical and digital video games, manage customer information, and track transactions. The project's goal is to provide an intuitive and functional user interface that allows store owners to manage their inventory with ease.

Class List
1. Game: Represents a video game and has attributes like title, developer, publisher, release date, genre, and rating.
2. PhysicalGame: Represents a physical copy of a video game and inherits from the Game class. It has additional attributes like the condition of the disc and its availability status.
3. DigitalGame: Represents a digital copy of a video game and inherits from the Game class. It has additional attributes like the platform it can be played on and the download link.
4. Inventory: Represents the inventory of the video game store and contains all the physical and digital games.
5. Customer: Represents a customer of the video game store and has attributes like name, email, and a unique customer ID.

Data and Function Members (UML can be used here as well)
- Game:
  - Attributes:
    - title: string
    - developer: string
    - publisher: string
    - release_date: date
    - genre: string
    - rating: float
  - Functions:
    - play(): a virtual function that can be overridden by the derived classes to provide different implementations of the function.
- PhysicalGame:
  - Attributes:
    - condition: string
    - availability_status: bool
  - Functions:
    - get_condition(): returns the condition of the disc
    - is_available(): returns the availability status of the game
- DigitalGame:
  - Attributes:
    - platform: string
    - download_link: string
  - Functions:
    - get_platform(): returns the platform the game can be played on
    - get_download_link(): returns the download link of the game
- Inventory:
  - Attributes:
    - physical_games: vector of PhysicalGame
    - digital_games: vector of DigitalGame
  - Functions:
    - add_game(game): adds a game to the inventory
    - remove_game(game): removes a game from the inventory
    - update_game(game): updates the details of a game in the inventory
    - search_game(title): searches for a game by its title in the inventory
    - display_inventory(): displays the inventory of physical and digital games
- Customer:

- Attributes:
    - name: string
    - email: string
    - customer_id: int
- Functions:
    - get_name(): returns the name of the customer
    - get_email(): returns the email of the customer
    - get_customer_id(): returns the customer ID of the customer

Relationships between Classes (UML can be used here as well)
- Game is the parent class of PhysicalGame and DigitalGame.
- PhysicalGame and DigitalGame inherit the attributes and functions of the Game class.
- The Inventory class contains both physical and digital games.
- The Customer class is not related to any other class in the system.

Project Task List and Timeline
1. Design the class structure and relationships (x days) (group member x)
2. Implement the Game, PhysicalGame, and DigitalGame classes (x days) (group member x)
3. Implement the Inventory and Customer classes (x days) (group member x)
4. Implement the user menu and user interface (x days) (group member x)
5. Implement the file input/output functions (x days) (group member z,y)
6. Write unit tests for all classes and functions (x days) (group member x,y)
7. Debug and refine the code (x days) (group member x,y,z)
8. Write user documentation (x days) (group member x,y,z)

User Interaction Description
The user will interact with the program through a user-friendly menu that provides options for adding, deleting, and updating games in the inventory, managing customer information, and viewing transaction history. The menu will be implemented using prompts that will allow the user to select an option by entering a corresponding number or character.

The program will also use error messages and confirmation prompts to guide the user through the program and ensure that the input is valid. For example, if the user tries to add a game that already exists in the inventory, the program will display an error message and prompt the user to enter a different game.

The program will allow the user to view the inventory of physical and digital games by selecting the corresponding option from the menu. The user can search for a game by its title and view its details, including its availability status, condition, platform, and download link.

The user can also add and manage customer information by selecting the customer management option from the menu. The program will allow the user to add a new customer, update customer information, and view a list of all customers in the system.

Unit Testing and Debugging Plan

The program will be tested using a combination of unit testing and organized input/output testing. The unit tests will cover each function of the classes and will ensure that the program performs as expected. The input/output tests will verify that the program reads and writes data correctly from the file and that the program handles user input and output correctly.
The program will also use exception handling to handle unexpected inputs or situations and prevent crashes or errors. The program will log any errors or issues encountered during runtime and will provide error messages or prompts to guide the user through the program.
Makefile and README files will be included in the project, which will provide clear instructions on how to compile, run and test the program. The Makefile will handle dependencies, and the debugging and release builds will be separated. The README file will explain how to use the program and will list any known issues or limitations of the program.