**C S 271 - Group Project - Requirements and Design Documentation**

**Project Title:** Binary Bards: Unleashed  
**Team Members:**

* Shabhan George (Project Lead & UI/UX Designer)
* Jason Edgington (Documentation Specialist)
* Michael Gremse (Lead QA Tester)
* Zachary Carmichael (Lead Developer

**Date:** 10/31/2024

## 1. Requirements (Michael & Jason)

### 1.1 Functional Requirements

1. The application will allow the user to experience a 2D turn-based RPG.
2. The user will have different fighting options, will gain experience, and gain items to help them in battle.
3. The game will get increasingly more difficult, and the user will be allowed to leave after every boss fight if they chose to.

### 1.2 Non-Functional Requirements

1. The UI for the game should be smooth, and the application should run smoothly
2. The graphics of the game will be two-dimensional, but there will be changing background and enemy types.
3. The calculations for the game will be concise and accurate throughout the gameplay experience

### 1.3 User Stories or Use Cases

* **User Story 1:** As a player, I want to have the freedom to select different abilities, items, and decide how long I want to play
* **User Story 2:** On additional playthroughs, I want a completely different selection of items, and a variance of different enemies, and variable difficulty.

### 1.4 Constraints and Assumptions

* **Assumptions:** Users will launch the application to play the game, and have a general understanding of how a turn based RPG works prior.
* **Constraints:** The application will use a UI and be 2D with changing scenery and different enemy types. It will not have any 3D aspects or animations to it.

## 2. Design

### 2.1 Application Architecture

1. **Frontend (UI/UX):**
   * **Gameplay Interface:** The user interface (UI) will display all necessary gameplay elements, including health bars, combat options (Attack, Skill, Item, and Guard), character stats, and relevant item information.
   * **Interaction and Feedback:** Users interact with the game through clickable menu options. Each button they press will execute a function correlating to the desired option.
2. **Backend (Game Logic):**
   * **Turn-based Combat System:** The backend will handle turn-based sequencing, allowing both player and enemy actions to alternate in a queue. This includes all core combat calculations such as damage dealt, experience points gained, and status effects.
   * **Character Progression:** Handles experience tracking, leveling up, and attribute improvements for characters as they progress through the game. A scaling mechanism adjusts difficulty based on player levels.
   * **Game State Management:** This component will manage in-game events, saving/loading progress, and initiating boss encounters as players advance.
3. **Data Management (Save System):**
   * File-Based Storage: The save system will store game progress in a file format that includes character levels, inventory, and unlocked features. Players can load, delete, or create new save files from the main menu.

### 2.2 Data Structures and Algorithms

#### Data Structures

1. **Classes:**
   * **Character:** Stores attributes such as health, attack, defense, skills, and inventory. Includes methods for updating stats and applying experience points.
   * **Enemy:** Similar to the Character class, but with properties tailored for enemies, such as aggression level and unique attack patterns.
   * **Items:** Defines the type (e.g., potion, weapon), effect (healing, damage, etc.), and duration for each item.
   * **DungeonRoom:** Represents rooms within the dungeon, including enemies, environmental features, and potential loot drops with a correlating room sprite.
2. **Lists and Dictionaries:**
   * **PartyList:** Holds active characters in the player’s party for turn-based sequencing.
   * **EnemyList:** Contains enemies for dungeon rooms, enabling sequential actions and random targeting. This will be reinitialized in every room.
   * **InventoryDictionary:** Maps items to their quantities for fast access, allowing the game to check, add, or remove items as needed.
   * **DungeonRoomDescriptionList:** instead of using unique sprites for every room, we can include an unlimited number of rooms by including a random assortment of room descriptions to be read out before an encounter is started.

#### Algorithms

1. **Turn-Based Combat Logic:**
   * **Turn Queue:** Alternates turns between player and enemy parties with foreach loops.
   * **Damage Calculation:** The game calculates damage by determining whether or not a character is guarding or using a skill.
   * **Health Check:** After each turn, health is checked to determine if any characters or enemies have been defeated, subtracting from the number of enemies in the room. If it reaches zero, then the room is cleared.
2. **Character Leveling and Experience:**
   * **Experience Gain:** Players earn experience points (XP) after combat, tracked per character. On reaching certain XP thresholds, characters level up, improving their attack damage and total HP.
   * **Difficulty Scaling:** Enemy stats increase according to the player levels. Enemies can be 3-5 levels higher or lower than the party.
3. **Inventory and Item Usage:**
   * **Item Effect Application:** Items, when used, apply their effects immediately, such as healing or stat boosts, and are then deducted from the inventory.
   * **Loot Drops:** After defeating enemies, a randomized loot function calculates potential drops using a random number generator.

### 2.3 Wireframes for Key User Interface Screens

#### Main Menu

A black and white image of a building

Description automatically generated

#### Lore Menu Option

A black and white image of a tower

Description automatically generated

#### Dungeon Cleared Screen

A screen shot of a phone

Description automatically generated

#### Game Over Screen

A black and white screen with red text

Description automatically generated

## 3. Pseudocode

// Main Game Loop

InitializeGame()

do {

DisplayMainMenu()

}

while (not quitOption)

// Menu Option 1 - Start Game

if StartGameOption

if SaveFileExists()

LoadGame()

EnterDungeon()

else

CreateNewSaveFile()

LoadGame()

EnterDungeon()

// Menu Option 2 - Edit Save File

else if EditSaveOption

DisplayEditSaveMenu()

if LoadSaveSelected

LoadGame()

EnterDungeon()

else if DeleteSaveSelected

DisplayDeleteWarning()

if ConfirmDelete()

DeleteSaveFile()

else

LoadGame()

// Menu Option 3 - Lore

else if LoreOption

ClearScreen()

DisplayBestiaryInfo()

DisplayHeroesInfo()

// Menu Option 4 - Quit Game

else if QuitOption

TerminateProgram()

// EnterDungeon

Function EnterDungeon:

DisplayDungeonIntro()

DescribeCurrentRoom()

DisplayRoomSprite()

DisplayRoomEnemies()

DisplayHeroes()

while (inCombat)

// Player Turn

for each character in PlayerParty:

if playerHasCharacters()

DisplayActionMenu()

if AttackSelected

ExecuteBasicAttack()

else if SkillSelected

ExecuteClassSkill()

else if ItemSelected

UseItem()

if NoItemSelected

SkipTurn()

else if GuardSelected

ApplyGuardEffect()

else if StatsSelected

DisplayCharacterStats()

else

DisplayDefeatScreen()

// Enemy Turn

for each enemy in EnemyParty:

if enemyHasCharacters()

if TargetCharacterBelowHealth(50%)

EnemyAttack(TargetCharacter)

else

EnemyAttack(RandomPlayerCharacter)

else

DisplayRoomClear()

RollLoot()

GrantExperience()

PromptContinueOption()

// Character Actions

Function ExecuteBasicAttack(character, target):

damage = character.Attack - target.Defense

ApplyDamage(target, damage)

Function ExecuteClassSkill(character, target):

if character == Cleric

HealPartyMember()

else if character == Sorcerer

CastAreaEffectSpell()

else if character == Troubadour

UseSmokeScreen()

else if character == Chellist

ShieldPartyMember()

Function UseItem(item, target):

ApplyItemEffect(item, target)

RemoveItemFromInventory(item)

Function ApplyGuardEffect(character):

ReduceIncomingDamage(character, 50%)

Function DisplayCharacterStats(character):

ShowCharacterStats(character)

// Enemy Actions

Function EnemyTurn():

for each enemy in EnemyParty

SelectRandomEnemyAction()

TargetRandomPartyMember()

ExecuteEnemyAction()

CheckPlayerPartyDefeatCondition()

// Level Progression and Scaling

Function GrantExperience():

for each character in PlayerParty

AddExperience(character)

if ExperienceThresholdReached(character)

LevelUpCharacter(character)

IncreaseCharacterStats(character)

// Boss Battle

Function StartBossBattle():

while (bossIsAlive)

EnterCombatWithBoss()

if BossDefeated()

PromptContinueOrLeave()

// Inventory Management

Function AddItemToInventory(item):

if ItemIsConsumable(item)

AddToInventory(item)

Function UseConsumableItem(item):

ApplyItemEffect(item)

RemoveFromInventory(item)

// GUI and User Interface

Function DisplayMainMenu():

ShowOptions(StartGame, EditSave, Lore, Quit)

Function DisplayActionMenu():

ShowOptions(Attack, Skill, Item, Guard, Stats)

Function UpdateRoomBackground(level):

ChangeBackgroundForCurrentLevel(level)

// Main Program Entry

Main:

InitializeGameVariables()

LoadGUIElements()

StartGameLoop()