# CS 271 – Progress Report

Date: 11/10/2024

## Project Overview:

This phase focused on setting up the foundational elements for our application. We implemented core features as outlined in the design document, including basic structure and algorithms, while ensuring the code was clean and modular. The initial features have been developed, and we began testing them to verify their functionality.

## Features Implemented Thus Far:

1. Hero Classes

* Created a base Hero class that includes the core attributes of Health, Attack damage, Defense points, current Level, and Experience amount. This class is the foundation of all 4 Hero subclasses.

1. Enemy Classes

* Created a generic Enemy class to serve as an organized shell for Goblins and Boss Types during dungeon exploration.

1. Game Initialization Shell:

* Established the Game class with a set of methods to initialize the game; methods like InitializeGame(), which will call upon other methods to run the game modularly.
* A placeholder StartGameLoop() method has been implemented, setting the foundation for future integration with the game’s primary loop. A restructuring of the program loops may need to be operated.
* The shell also includes a TerminateProgram() method to cleanly exit the application when the game ends or the user quits. Saving will need to be implemented.

1. Placeholder Game Menu

* A placeholder menu has been implemented for the main menu and the turn menu for when it will get replaced by GUI elements.

1. Combat System Placeholder

* The combat system has not been fully implemented yet, thus far proper combat cannot be engaged with. However, the Classes with which it will rely on are almost finished.

1. Level Management Shell

* A shell has been implemented that will allow for the game to keep track of what floor players are on and how to tweak the enemies accordingly.

1. User Interface Module

* A UserInterface class has been implemented for clean integration with the Unity GUI.

1. Base Stats for Players and Enemies

* Assigned the player attributes of Health, Attack, Damage, Defense with base stats that can be improved throughout the game
* Assigned the enemy’s attributes of Health, Attack, Level with base values that will increase throughout the duration of the game

1. Player Heal

* Created a player heal ability used by the Cleric subclass that heals a player for an amount within a range

1. Button Selection

* Created a UI to select options within the game using buttons instead of reading the players keystrokes

1. Turn Asynchronous Functions

* Developed the game loop to where the player’s turn happens asynchronously to the enemy’s turn, AKA true turn-based combat

1. General Text Output

* Created text outputs describing what is happening on screen, as well as the calculations such as damage dealt, healed, items acquired, etc.

1. Damage Multiplier

* a

## Testing:

* Performed some light testing on existing features, but full functionality has yet to be implemented.

## Future Plans:

1. We plan to finish implementing the Cleric, Sorcerer, Troubadour, and Cellist subclasses with their individual skills.
2. ~~Finalizing the CombatManager to handle individual turns within the gameloop~~ (Completed) and to allow for enemy population checking to handle rewards.
3. Expanding on the DungeonRoom class to allow for different room descriptions and for random loot generation
4. Finalizing Items class to create a limited availability of usable items in player inventories.
5. Developing an effective and safe file management system.
6. Continue to test and modularize individual components of the program to make a holistic approach to playstyles.
7. Connecting the UI with the game logic to get it to run efficiently.

## Codebase:

using System;

using System.Reflection.Metadata.Ecma335;

// Start Program

public class Program

{

static void Main(string[] args)

{

Game.InitializeGame();

GameMenu.DisplayMainMenu();

Console.WriteLine("Hello, World!");

}

}

// End Program

// Start Hero Classes

public class Hero

{

public string Name { get; set; }

public int Health { get; set; }

public int MaxHealth { get; set; }

public int Attack { get; set; }

public int Defense { get; set; }

public int SkillCoolDown { get; set; }

public int Level { get; set; }

public int Experience { get; set; }

public List<Item> Inventory { get; set; }

public Hero(string name, int health, int maxhealth, int attack, int defense, int skillcooldown, int level, int experience)

{

Name = name;

Health = health;

Attack = attack;

Defense = defense;

SkillCoolDown = skillcooldown;

Level = level;

Experience = experience;

Inventory = new List<Item>();

}

public virtual void UseSkill()

{

Console.WriteLine($"{Name} uses a skill.");

}

public void LevelUp()

{

// Random Numbers to increase replayability

Random random = new Random();

Level++;

Health += random.Next(2, 10);

Attack += random.Next(1, 5);

Defense += random.Next(1, 3);

Console.WriteLine($"{Name} leveled up! Health: {Health}, Attack: {Attack}, and Defense: {Defense}");

}

public void GainExperience(int experience)

{

Experience += experience; // Sets new experience

Console.WriteLine($"{Name} earned {experience} xp!");

if (Experience >= Level \* 100)

{

LevelUp();

Experience = 0; // Resets XP after leveling up

}

}

public void UseItem(Item item)

{

Console.WriteLine($"{Name} used {item.Name}");

/\*

if (Inventory.Contains(item))

{

// FINISH ME:

// item.ApplyEffect();

Console.WriteLine($"{Name} used {item.Name}");

}\*/

}

}

// FINISH ME:

/\*

public class Cleric : Hero

{

public Cleric(string name, int health, int maxhealth, int attack, int defense, int skillcooldown, int level, int experience)

: base(name, health, maxhealth, attack, defense, skillcooldown, level, experience) { }

public override void UseSkill()

{

// FINISH ME: Heals another player for a certain random amount within a range

SkillCoolDown = 2;

}

}

public class Sorcerer : Hero

{

public Sorcerer(string name, int health, int maxhealth, int attack, int defense, int skillcooldown, int level, int experience)

: base(name, health, maxhealth, attack, defense, skillcooldown, level, experience) { }

public override void UseSkill()

{

// FINISH ME: Fire spell - AOE attacks all enemies

SkillCoolDown = 5;

}

}

public class Troubadour : Hero

{

public Troubadour(string name, int health, int maxhealth, int attack, int defense, int skillcooldown, int level, int experience)

: base(name, health, maxhealth, attack, defense, skillcooldown, level, experience) { }

public override void UseSkill()

{

// FINISH ME: Add Smoke Screen // Let's him use two turns but not skill again

SkillCoolDown = 3;

}

}

public class Cellist : Hero

{

public Cellist(string name, int health, int maxhealth, int attack, int defense, int skillcooldown, int level, int experience)

: base(name, health, maxhealth, attack, defense, skillcooldown, level, experience) { }

public override void UseSkill()

{

// FINISH ME: Add Absolute Defense Skill

SkillCoolDown = 3;

}

}

// End Hero Classes

\*/

// Start Enemy Classes

public class Enemy

{

public string Name { get; set; }

public int Health { get; set; }

public int Attack { get; set; }

public int Level { get; set; }

public Enemy(string name, int health, int attack, int level)

{

Name = name;

Health = health;

Attack = attack;

Level = level;

}

public virtual void AttackPlayer(Hero target)

{

// FINISH ME:

Console.WriteLine($"{Name} attacks player for {Attack} points of damage!");

target.Health -= Attack;

}

}

// FINISH ME:

/\*

public class Boss : Enemy

{

public Boss(string name, int health, int attack, int level)

: base(name, health, attack, level) { }

public override void AttackPlayer(Hero target)

{

int bossDamage = Attack \* 2;

}

public void UseSkill(Hero target)

{

// FINISH ME:

Console.WriteLine("");

}

}

// End Enemy Classes

\*/

// Game Classes

public class Game

{

public static void InitializeGame()

{

Console.WriteLine("Initializing Binary Bards: Unleashed...");

// FINISH ME:

// Likely to be Unity garbo, please come back at a later point in time.

}

public static void StartGameLoop()

{

LevelManager.LoadLevel(0); // FIX ME: Check what level previous games had saved

CombatManager.StartCombat();

}

public static void TerminateProgram()

{

// FINISH ME:

// save game

Console.WriteLine("Exiting game. Thank you for playing!");

Environment.Exit(0);

}

}

public class GameMenu : Game // Main Menu

{

public static void DisplayMainMenu()

{

bool continueProgram = true;

while (continueProgram)

{

Console.Clear();

Console.WriteLine("----- Binary Bards: Unleashed -----");

Console.WriteLine("1. New Game");

Console.WriteLine("2. Load Game");

Console.WriteLine("3. Exit");

Console.Write("Choose an option: ");

string choice = Console.ReadLine();

switch (choice)

{

case "1":

StartNewGame();

Game.StartGameLoop();

break;

case "2":

LoadGame();

break;

case "3":

TerminateProgram();

continueProgram = false;

break;

default:

Console.WriteLine("Invalid option. Please try again.");

Console.ReadKey();

break;

}

}

}

private static void StartNewGame()

{

Console.Clear();

InitializeGame();

Console.WriteLine("Starting a new game...");

StartGameLoop();

}

private static void LoadGame()

{

Console.Clear();

Console.WriteLine("Loading game... (Feature not yet implemented)");

Console.ReadKey();

}

}

public class CombatManager : Game

{

public static void StartCombat()

{

// While hero health total >0 && Goblin > 0

// If goblins are dead -> you win

// if you're dead -> defeat screen

string choice = "";

Console.WriteLine("Chaos insues...");

Console.WriteLine("Hero 1's turn");

Console.ReadLine();

Console.WriteLine("Hero 2's turn");

Console.ReadLine();

Console.WriteLine("Hero 3's turn");

Console.ReadLine();

Console.WriteLine("Hero 4's turn");

Console.ReadLine();

Console.WriteLine("----Enemies Turn----");

Console.WriteLine("Goblin 1's turn");

Console.WriteLine("Goblin 2's turn");

Console.WriteLine("Goblin 3's turn");

}

}

// FINISH ME:

public class LevelManager : Game

{

public static void LoadLevel(int levelNumber)

{

Console.WriteLine($"You are on level: {levelNumber}...");

// FINISH ME: Level loading logic

}

}

// FINISH ME:

public class InventoryManager : Game

{

public static void OpenInventory(Hero hero)

{

Console.WriteLine($"{hero.Name}'s Inventory:");

foreach (var item in hero.Inventory)

{

Console.WriteLine($" - {item.Name}");

}

}

}

// FINISH ME:

public class UserInterface : Game

{

public static void DisplayHeroStats(Hero hero)

{

Console.WriteLine($"{hero.Name} Stats - Health: {hero.Health}, Attack: {hero.Attack}, Defense: {hero.Defense}");

}

public static void DisplayEnemyStats(Enemy enemy)

{

Console.WriteLine($"{enemy.Name} Stats - Health: {enemy.Health}, Attack: {enemy.Attack}");

}

public static void DisplayVictoryScreen()

{

Console.WriteLine("Victory! You defeated the enemy!");

}

public static void DisplayDefeatScreen()

{

Console.WriteLine("Defeat! You have been defeated by the enemy.");

}

}

// End Game Classes

// DungeonRoom Class

public class DungeonRoom

{

public List<Enemy> Enemies { get; set; }

public List<Item> Loot { get; set; }

public string RoomSprite { get; set; }

public string Description { get; set; }

public DungeonRoom(string roomsprite, string description)

{

Enemies = new List<Enemy>();

Loot = new List<Item>();

RoomSprite = roomsprite;

Description = description;

}

// Populates the Dungeon

public void EnterDungeon()

{

// FINISH ME:

Random random = new Random();

int numberOfEnemies = random.Next(1, 4); // Add between 1 and 3 enemies

for (int i = 0; i < numberOfEnemies; i++)

{

// Simple enemy generation example

Enemies.Add(new Enemy($"Enemy {i + 1}", 100, 10, 1));

}

}

public void DescribeCurrentRoom()

{

// FINISH ME:

Console.WriteLine($"You are in a room: {Description}");

}

// Gives Loot

// public List<Item> GiveLoot() { }

// Visuals

public void DisplayRoomSprite()

{

// FINISH ME:

Console.WriteLine($"Room Sprite: {RoomSprite}");

}

public void DisplayRoomEnemies()

{

// FINISH ME:

Console.WriteLine("Enemies in the room:");

foreach (var enemy in Enemies)

{

Console.WriteLine($"- {enemy.Name}");

}

}

public void DisplayHeroes()

{

// FINISH ME:

Console.WriteLine("heroes:");

}

}

// End DungeonRoom

// Start Item Classes

public class Item

{

public string Name { get; set; }

public int EffectAmount { get; set; }

public int Duration { get; set; }

public bool IsActive { get; set; }

public Item(string name, int effectAmount, int duration)

{

Name = name;

EffectAmount = effectAmount;

Duration = duration; // in rounds

IsActive = false;

}

public void UseItem(Hero target) { }

private void ApplyEffect(Hero target) { }

}

// End Item Classes