**Project Proposal: Turn-Based Battle System**

**Git Link: https://github.com/louis-christian/CISC191Project\_Battle\_System.git**

**Project Pitch:**

We will be developing a turn-based battle system inspired by games like *Final Fantasy* or *Pokémon*, using Java and the javax.swing package to create a graphical user interface (LO6). This system will utilize object-oriented programming (OOP) principles (LO1), featuring multiple classes and objects representing player characters, enemies, actions, and game logic (LO3).

Players will control one to three heroes and face off against a randomly generated number of enemies, leveraging arrays and multidimensional arrays to manage the game grid, player/enemy stats, and combat logs (LO2). Each character will have hit points and a set of actions: attack, defend, and rest. Actions and characters will be implemented using inheritance and polymorphism, including abstract classes and interfaces for defining shared behavior (LO4).

We will also implement generic collections such as ArrayList or HashMap to manage dynamic lists of actions, combatants, and turn order (LO5). The program will feature exception handling to manage invalid user inputs and runtime errors gracefully (LO7). Additionally, we may implement players’ progress and battle logs to be stored and retrieved through text file input/output, allowing for saved battles and logs of previous sessions (LO8).

By integrating these components, we aim to build a modular, object-oriented, and user-friendly battle system that demonstrates comprehensive understanding of Java and software development best practices.

**GUI Protype:**

A green and blue rectangular object

AI-generated content may be incorrect.

**Learning Objectives and Project Application**

|  |  |
| --- | --- |
| **Learning Objective** | **How We Use It** |
| LO1: OOP - Inheritance, Polymorphism | Abstract Character class; PlayerCharacter/EnemyCharacter inherit; Action uses interface |
| LO2: Arrays and Multi-dimensional Arrays | ArrayList for character lists; optional 2D array for battle grid or combat log |
| LO3: Classes and Objects in Java | Program is class-based with Character, BattleManager, etc.; Strong object interaction |
| LO4: Inheritance and Polymorphism | Character subclasses override methods; Action types show polymorphism; potential for more specialized classes |
| LO5: Generic Classes and Collections | Use of ArrayList<Character>, HashMap<String, Action>; potential for custom generic utilities |
| LO6: GUI | Swing components like JFrame, JPanel, JButton, JTextArea to display game GUI |
| LO7: Exception Handling | try-catch blocks for input, file I/O, and invalid actions; improves program stability |
| LO8: File Input/Output | Dunno yet. |

**Team Time Plan** **(Per Week)**

* **Weekly check-ins (1 hrs):** We’ll meet to plan what we’re doing, go over progress, and divide up tasks.
* **Working together on code (2 hrs):** Pair Coding through Discord or in person.
* **Code review & merging (2 hrs):** We’ll go over each other’s work, fix bugs, and make sure everything runs smoothly together.
* **Design & planning stuff (1 hrs):** This includes working on our UML diagrams, GUI sketches, and making sure we hit the learning objectives.
* **Testing & fixing bugs (1 hrs):** We’ll playtest the game, check for glitches, and fix anything that breaks.

**Suggested Timeline:**

|  |  |
| --- | --- |
| **Week** | **Task** |
| Week 1 | Write the project pitch. |
| Create CRC cards for major classes. |
| Design initial UML class diagram. |
| Set up GitHub repo and project workspace. |
| Begin the project page |
| Week 2 | Start coding model classes (Character, PlayerCharacter, etc.). |
| Write unit tests for actions and turns. |
| Apply OOP principles (LO1, LO3, LO4) |
| Update project page with CRC/UML/code. |
| Submit code progress and tests. |
| Week 3 | Complete core class implementation. |
| Identify exception handling needs (LO7). |
| Sketch GUI layout on paper or tool. |
| Update project page with GUI plan. |
| Submit current code snapshot. |
| Week 4 | Test combat logic extensively. |
| Fix bugs in core game loop. |
| Update test logs and bug reports. |
| Submit refined backend logic. |
| Week 5 | Build basic Swing GUI layout (LO6). |
| Implement JPanels, JButtons, JTextArea. |
| Focus on visual layout, not function. |
| Update project page with GUI images. |
| Submit visual GUI prototype. |
| Week 6 | Connect GUI events to game logic. |
| Add File I/O |
| Demonstrate full action flow via UI. |
| Update project with working demo. |
| Submit functional prototype. |
| Week 7 | Test full game loop via GUI. |
| Polish UI, fix bugs, balance gameplay. |
| Ensure all LOs are visible in project. |
| Submit nearly-final version. |
| Week 8 | Final debugging and cleanup. |
| Record demo video showing LOs. |
| Write summary of LO usage. |
| Submit final project on Canvas with video. |

**A diagram of a company

AI-generated content may be incorrect.CRC Cards:**

|  |  |  |
| --- | --- | --- |
| **Class** | **Responsibilities** | **Collaborators** |
| Character (Abstract) | Store name, HP, and status; Execute actions (attack, defend, rest); Take/receive damage; Update state each turn | PlayerCharacter, EnemyCharacter, Action, BattleManager |
| PlayerCharacter | Allow user-controlled action selection; Display GUI elements; Inherit from Character | Character, Action, GameUI, BattleManager |
| EnemyCharacter | Choose actions using basic AI; Inherit from Character | Character, BattleManager |
| Action (Interface) | Execute a game action; Define energy/cooldown; Affect targets | Character, BattleManager |
| BattleManager | Control game flow; Handle turns; Manage character list; Trigger GUI updates; Log actions | Character, Action, GameUI, FileManager |
| GameUI | Display battlefield; Show character info; Handle input; Update view | PlayerCharacter, BattleManager, Action |
| FileManager | Save/load battle logs; Manage file I/O; Handle exceptions | BattleManager, Character, Action |
| GameLogger | Record game events; Store logs; Print summaries | BattleManager, FileManager |

**UML Diagram**

**Week 2:**

Over the past week, our team has successfully completed the first major milestone of our turn-based battle system project. We have designed and implemented some of the core model classes that form the foundation of our game:

* **Character** (abstract class): Defines the shared attributes and actions of all characters, such as name, hit points (HP), status effects, and basic methods for attacking, defending, resting, and taking damage. This class allows us to easily extend functionality later by creating different types of characters.
* **PlayerCharacter**: Inherits from Character and represents the user-controlled character. It implements customized attack, defend, and rest behaviors, giving players consistent but flexible options during their turn.
* **EnemyCharacter**: Also inherits from Character and adds a simple AI system. On the enemy’s turn, it randomly decides whether to attack, defend, or rest, simulating autonomous enemy behavior without needing player input.
* **BattleManager**: Manages the overall game flow, including turn order, user input, enemy decisions, health tracking, and win/loss conditions. It also handles displaying the battle status after each action.
* **TestBattle**: A basic driver class that initializes a player and an enemy, then launches a battle. This helped us validate and debug our core battle loop before moving onto building the full application.

At this stage, the battle is playable through the console. Players input action choices through the terminal, and battle progress is displayed after each move. This helped us focus first on making sure the game's logic was working correctly before introducing a more complex interface.

Currently, our team is transitioning from console testing to building the **graphical user interface (GUI)**. We are designing the GUI **exclusively using Java Swing components**—such as JFrame, JPanel, JButton, and JTextArea—without using any external libraries, images, or imported assets. Our goal is to create a clean, professional, and functional visual representation of the battle while staying within the tools provided by standard Java.

**Next Steps**

Our next priority is to finish setting up the basic GUI layout and connect the player action buttons to the battle logic. After that, we will focus on updating the game status display dynamically, refining user interactions, and adding exception handling to ensure the program remains stable during gameplay. We plan to complete an initial playable version of the GUI within the next two weeks.

**Sources:**

**https://itsokaytocode.medium.com/object-oriented-programming-explained-in-gaming-76310792ce60**