**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**