# Agile Iterative SDLC for of PF2E Auto Battler (Turn-Based RPG Game)

**Agile Iterative Software Development Life Cycle (SDLC)** focuses on flexibility, incremental delivery, and continuous feedback. By breaking down the game development into smaller, manageable iterations (sprints), the developers can refine game mechanics, improve user experience, and adapt to changes quickly.

## 1. Agile Overview and Application

Agile SDLC is designed to address evolving requirements by promoting iterative progress, and prioritize features, build them incrementally, and test them early.

In our game, Agile is reflected in how individual components are modular and independently functional. Each sprint could focus on one key element—like building the `Character` base class in one iteration, adding specific subclasses like `Fighter` or `Wizard` in the next, and integrating effects, grid movement, or combat logic in following cycles.

## 2. Iteration-Based Feature Development

The game started with a **basic playable prototype**: a window rendered with Pygame, a simple grid system (`GridPosition`), and the ability to move a single character. This MVP (Minimum Viable Product) allowed us (the developers) to test core mechanics early. From there, iterations might include:

- **Sprint 1**: Grid and movement logic (`GridPosition`, simple drawing)
- **Sprint 2**: Core character class with basic combat (`Character`)
- **Sprint 3**: Enemy AI and enemy class (`Enemy`)
- **Sprint 4**: Player character classes (`Fighter`, `Rogue`, `Wizard`)
- **Sprint 5**: Action system and visual effects (`Effect`)
- **Sprint 6**: Turn management and UI (`Game`, action buttons, combat log)
- **Sprint 7**: Upgrade system and wave progression
- **Sprint 8**: Polish, visual improvements, balancing, and bug fixing

Each sprint delivers a working build that can be tested, shown to users or peers, and refined based on feedback with continues update to all the related documentations.

## 3. User Stories and Backlog Management

Our development is driven by **user stories**—short descriptions of features from the user's perspective. Example user stories for this game might include:

- *As a player, I want to move my character on a grid so that I can position strategically.*
- *As a wizard, I want to cast magic missile to deal damage from a distance.*
- *As a player, I want to see visual effects when an attack hits or misses.*
- *As a rogue, I want to flank enemies for sneak attack bonuses.*

These stories form the **product backlog**, which is prioritized and implemented incrementally. Each sprint focuses on a subset of these stories, aiming for functional delivery.

## 4. Continuous Testing and Feedback

Testing is integral in Agile, and our PF2E Auto Battler (RPG game) supports that with clear visual and logical outcomes. The combat results, grid movements, and effects provide instant feedback. During each iteration, the developers can test if new features work and how they interact with existing systems. Playtesting with others (or self-testing) helps uncover balance issues or bugs early, reducing long-term maintenance.

## 5. Flexibility and Refactoring

As new mechanics or user expectations arise, our Agile supports pivoting without scrapping the whole project. For example, if a new ability is added to the Rogue (like Twin Feint), the `Character` base class can be updated to accommodate it without rewriting the whole combat system. The modular structure—e.g., separate `Effect` types for different animations—makes refactoring safer and faster.

## 6. Deployment and Continuous Improvement

Our Agile Iterative method emphasizes **early and frequent delivery**, which aligns well with modern game development practices. Our team of developers can release an alpha version with basic mechanics, then continuously update it with new characters, enemies, effects, and UI enhancements. This iterative release model keeps users engaged and helps guide development with real-world feedback.

## Conclusion

Using an **Agile Iterative SDLC** to develop our PF2E Auto Battler (turn-based RPG) in Python ensures that the game is built incrementally, tested continuously, and improved iteratively. This approach enables us (the developers) to respond to change, deliver working software quickly, and refine gameplay mechanics based on feedback. It promotes modular design, which is clearly reflected in the code structure, and supports scalability and maintainability for future enhancements.