Plan of Attack

# Project Breakdown & Schedules:

|  |  |  |  |
| --- | --- | --- | --- |
| **Task & Milestone** | **Details** | **Assignee** | **Completion Date** |
| Brainstorm, discussion & general planning | Interpret project requirements, communicate mutual expectations, and consolidate understanding of object-oriented design | Simon & George | Monday, Nov. 17 |
| Specific planning & UML prototype | Plan hierarchical relations between game components and experiment with various design patterns  Produce rough paper draft of UML diagram | Simon & George | Tuesday, Nov. 18 |
| Plan of attack draft | Note down tentative answers for design questions and vague schedule | George | Wednesday, Nov. 19 |
| Final UML design diagram | Finalize details of object hierarchy and game logic  Produce organized UML diagram | George | Thursday, Nov. 20 |
| Final plan of attack | Proofread and edit plan of attack  Revise schedule to reflect changes | Simon | Thursday, Nov. 20 |
| Header files & Skeleton code | Translate UML design to header files and make relational linkages | Simon | Friday, Nov. 21 |
| **Milestone 1:** Design phase completed | | | Friday, Nov. 21 |

|  |  |  |  |
| --- | --- | --- | --- |
| Main driver & Game | Implement driver function, game loop, command interpreter and command-line interface | George | Saturday, Nov. 22 |
| Character & Entity | Implement overall framework for spawnable entities | Simon | Saturday, Nov. 22 |
| Player & Enemy | Implement concrete character subclasses with specific overrides for each race | George | Sunday, Nov. 23 |
| Floor, Cell & Tile | Implement game map components and file loading/parsing logic | Simon | Sunday, Nov. 23 |
| Potion, Treasure & Stair | Implement status effect items, and make relational linkages to game map and characters | George | Tuesday, Nov. 25 |
| Unit testing | Write unit tests for complex, error-prone, or intricately overridden methods | Simon | Tuesday, Nov. 25 |
| Integration testing | A comprehensive testing of the entire game system, with particular focus on:  - Spawning mechanism  - Combat system (special effects)  - Potion effects  - Level advancement & end game | George & Simon | Wednesday, Nov. 26 |
| **Milestone 2**: Core implementation phase completed | | | Wednesday, Nov. 26 |

|  |  |  |  |
| --- | --- | --- | --- |
| DLC: Inventory | Implement inventory space to store items | Simon | Thursday, Nov. 27 |
| DLC: WASD with curses library | Implement a more natural command input system | George | Thursday, Nov. 27 |
| DLC: Equipment system | Implement equipment slots for permanent status effect items | Simon | Friday, Nov. 28 |
| DLC: Random floor generation | Implement system to generate floors with random layouts | George | Friday, Nov. 28 |
| Integration testing of enhancement features | Ensure additional features does not break any core game functionalities, | George & Simon | Friday, Nov. 28 |
| **Milestone 3**: Enhancement Implementation phase completed | | | Friday, Nov. 28 |

|  |  |  |  |
| --- | --- | --- | --- |
| Revised UML design diagram | Revise UML to reflect actual structure of the project | Simon | Saturday, Nov 29 |
| Revised answers to design questions | Revise answers to reflect changes made during the implementation phase | George | Saturday, Nov 29 |
| Design document draft | Provide overview of all aspects of the project, including high level implementation and design patterns | Simon & George | Sunday, Nov 30 |
| Final design document | Proof read and edit final design document | Simon & George | Sunday, Nov 30 |
| **Milestone 4**: Final design & documentation phase completed | | | Sunday, Nov 30 |

# Preliminary Design of Core Components

# Design Questions:

1. **[Question]** How could your design your system so that each race could be easily generated? Additionally, how difficult does such a solution make adding additional classes?

**[Answer]** Our system will have classes representing each race that extend off a superclass. The superclass will contain fields common to all the races, as well as a method that returns a certain race based off the user's input. This will make race generation easy as the mapping between 's' to Shade for example, will be encapsulated in one method. It will also be easy to add additional classes by simply inheriting off the superclass.

1. **[Question]** How does your system handle generating different enemies? Is it different from how you generate the player character? Why or why not?

**[Answer]** Our system will use the Factory Design Pattern to generate enemies as there can be multiple instances of them. The player character will be generated differently with the Singleton Design Pattern as there is only one instance of the PC.

1. **[Question]** How could you implement the various abilities for the enemy characters? Do you use the same techniques as for the player character races? Explain.

**[Answer]** We will use the Visitor Design Pattern implement the abilities for both the enemies and player races, basing effects on the attacker and defender. Other effects such as Drow's potion multiplier or Human's gold drop will be implemented through overring method in their respective classes. Thus the techniques used for both the enemies and the player will be similar in nature.

1. **[Question]** What design pattern could you use to model the effects of temporary potions (Wound/Boost Atk/Def) so that you do not need to explicitly track which potions the player character has consumed on any particular floor?

**[Answer]** We will use the Decorator Design pattern so that the potion itself will wrap around the original player. Thus, multiple effects can be implicitly tracked.

1. **[Question]** How could you generate items so that the generation of Treasure and Potions reuses as much code as possible? That is, how would you structure your system so that the generation of a potion and then generation of treasure does not duplicate code?

**[Answer]** The generation of items will be done by the individual cells themselves. Each of these cells will have methods to spawn treasures and potions. Code will be reused as finding the random cell to spawn an item will be the same for any item.