Project Plan - Group 12

Group members

1. Alanna Anna Shibu
2. Summer
3. Bo-Hong Chen

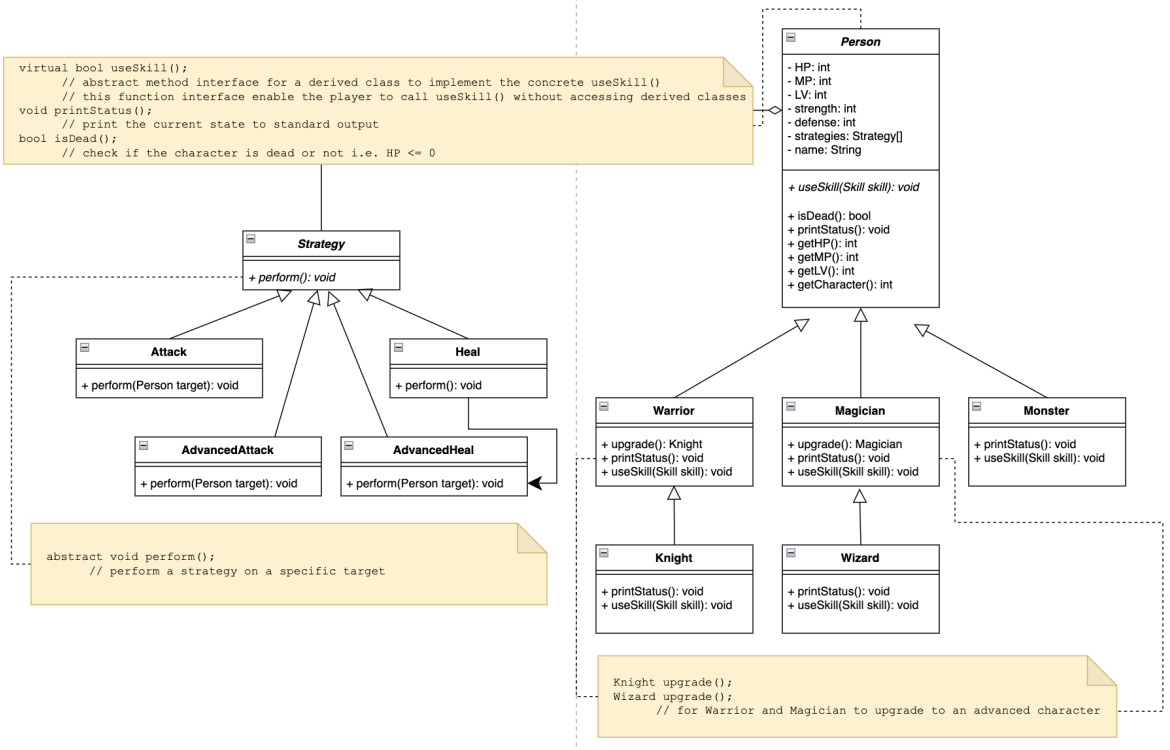
Use Case Description

The project aims to create an RPG (role-playing game) wherein the user chooses a character and fights monsters (NPCs) in three separate rounds with the player being able to win if they defeat all the monsters in every round. Once the player chooses a character i.e., either a “Warrior” or a “Magician”, they acquire the set amount of HP (healing power) and MP magical power) for that round. Once the round begins, the user is allowed to make the first move where they are allowed to choose a “strategy” i.e., attack or healing. Attack would allow them to utilize their particular skill set of the character to attempt to maim the opponent (i.e. lower their HP) and healing would allow them to restore their own HP by a few points. Each character has their own specific strengths with the Magician being able to heal themselves, and the Warrior being able to exact greater damage with the attack strategy.

Each round is played “unto the death” that is until one player, either the user or all of the monster(s) are dead i.e., their HP lowered to 0 where their state would be defined as dead. Each round has an increasing level of difficulty, with the number of monsters increasing each round by a certain number. The game continues for three rounds, during each of which the player makes their move by supplying it in the form of input into the terminal line. For each monster that the player kills in a round, their “level” is increased by one and upon attaining a status of level 10, the player’s character would be automatically upgraded. The upgraded version of the Warrior, called the Knight exacts more damage whereas the upgraded version of the Magician called the Wizard, can both heal to a greater extent as well as exact more damage.

If the player chooses to abandon the game and wishes to return to it later for whatever reason, the game will allow the user to restore their last saved game and allow them to continue playing from that saved checkpoint.

Class Diagram and Details



Name	Description
`Person`	Abstract base class for different characters. Class Person has one or more Strategy objects
`Warrior`	Derived from class Person. A warrior can attack an enemy and cause more damage than a magician does.
`Magician`	Derived from class Person. A magician can attack an enemy and heal themself but causes less damage than a warrior does.
`Knight`	Derived from class Warrior. After obtaining a certain level, a warrior can upgrade to a knight, which can cause more damage.
`Wizard`	Derived from class Magician. After obtaining a certain level, a magician can upgrade to a wizard, which can cause more damage as well as heal themselves by a greater HP.
`Monster`	NPC that are designed to attack the player.
`Strategy`	Abstract base class for different types of strategies. A Person object will use this class to attack another Person object or heal themself.
`Attack`	Attack an enemy and deduct the enemy's HP.
`Heal`	Increase the HP of the player.

`AdvancedAttack`	Can kill an enemy (state=dead) immediately but costs more MP.
`AdvancedHeal`	Takes a certain amount of the HP of the enemy and adds it to the HP of the player.

Timeline

Task	**Deadline**
Finish class Person and Strategy	06 Oct - *Week 9*
Finish derived classes from class Person	08 Oct - *Week 9*
Finish derived classes from class Strategy	10 Oct - *Week 10*
Finish the main function	12 Oct - *Week 10*
Finish testing the project and debug	14 Oct - *Week 10*

User interface description

The project plans on implementing an interactive interface through the terminal for the execution of the game. The user then goes on to select a character from the provided menu. The player's input from the terminal determines their character for the game and they are informed that the first round of the game is about to commence. Once the round begins, the player is allowed to make their move first where they are allowed to choose a "strategy" i.e., attack or healing by entering their move as a written command. Attack would allow them to utilize their particular skill set of the character to attempt to maim the opponent (i.e., lower their HP) and heal would allow them to restore their own HP by a few points.

Each round is played until one player, either the computer (NPC monsters) or the end user player has their HP lowered to 0, their state would be defined as dead. The game continues for three rounds, during each of which the player makes their move by supplying it in the form of input into the terminal line. The player can check the "status" of any given character i.e., their own or the monsters by entering the check status command. This would output the particular character's HP and MP and if the player chooses to view their own status their character's level would be displayed as well.

If the player chooses to abandon the game and wishes to return to it later for whatever reason, the game stores the data and game statistics in a text file which can be used to restore the game to the last saved state by typing in the command “restore” into the terminal. In order to do this the player should have used the “save game” command before exiting the game.

Unit testing and debugging plan

Rough outline of variables:

- n-previous HP
- m-previous MP
- r-Damage caused by Magician
- s-Damage caused by Warrior
- t=(r+5)-Damage caused by Wizard
- u=(s+10)-Damage caused by Knight
- v-Amount healed by Magician

- w=(v+5)-Amount healed by Wizard

Class	Method	Test case	Expected output
`Person`	`bool isDead()`	`Person.HP <= 0`	`True`
		`Person.HP > 0`	`False`
	`void printStatus()`	Call this method after creating a Person object	A string contains all the attributes of Person
`Warrior`	`Knight *upgrade()`	`(Person.getCharacter()== Knight)`	`True`
`Magician`	`Wizard *upgrade()`	`(Person.getCharacter()== Wizard)`	`True`

`Attack`	`void perform (Person* target)`	For a Warrior object:
		`Monster.getHP()==(n-s)`
		For a Magician object:
		`Monster.getHP()==(n-r)`
			`True` for either case

`Heal`	`void perform()`	For a Magician object:
		`Player.getHP==(n+v)`
		For a Wizard object:
		`Player.getHP==(n+w)`
			`True` for either case

`Advanced Attack`	`void perform (Person* target)`	`(Monster.getHP==0) && (Player.getMP==(m-80))`	`True`
`Advanced Heal`	`void perform (Person* target)`	`Player.getHP==(n+w) && Monster.getHP()==(n-t)`	`True`

Essential Features (10 points each)	Check
Has more than 5 classes	V
Has inheritance with at least three levels A-->B-->C	V
Has demonstratable use of polymorphism	V
Has abstract classes	V

Functional Features (5 points each)	**Check**
Another object can print or display the state of an object on the screen.	V
An end-user can change the state of an object.	V
Can store and recall multiple entries of data/objects, such as in a database or game stats.	V
Can Write/Read data from a file. Save a game checkpoint or backup data to a database or file.	V
Has an intuitive and functional user menu.	V
It is interactive by giving feedback to the user and receiving input from the user	V
Non-Functional Feature (5 points each)	**Check**
well-commented code with function comments in the headers	V
Consistently well-named, well organised, correctly indented code	V
Test a good portion of your program in a structured way with evidence - unit testing, organised input/output testing	V
Makefile builds the program.	V
It runs without crashing or going into infinite loops even when the tutor attempts to break it (reasonably)	V
Validates user input correctly - accepts only correct inputs.	V
The end product matches your plan document accurately without many significant design changes.	?
Uses dynamic/heap memory to store a variable amount of data correctly.	V
Usefully allocates memory from the heap using `new` and correctly free them with `delete`	V