Ryan Sheffler

Software Development I

Milestone Writeup

Abstract

My goal was to create a working RPG-style game using only Java. It includes a full-fledged battle system, complete with unique enemies for the player to fight and different abilities for the player to learn as they progress through the game. In addition, the game is built to be modifiable, allowing players to add new enemies and abilities to the game.

Motivation

My dream job is working as a game developer, so when I'm assigned a freeform project, I try to work in game development somehow. RPGs in particular are a major favorite of mine and it seemed like a simple turn-based RPG wouldn't be an overly complex project to code in Java. It was fun seeing everything come together and form a working game.

Detailed System Description

The game operates around a turn-based system. Both the player and the enemy have set statistics. For the enemy, these are simply taken from a list of possible enemies and their attacks, located in data/data/enemy.txt. The player's level determines which enemy is chosen from the file. For the player, these statistics are randomly rolled at the start of the game and slightly increase with each won battle. Each turn the game takes in a command from the user and generates a command for the enemy. It uses these commands to calculate the amount of damage the two deal to each other. Different values are used for different abilities. For example, a normal attack will calculate its damage from the user's attack stat and reduce the damage based on the target's defense stat, whereas an offensive magic spell, such as Ember, would use the user and target's mental stats. In addition, the game increases and reduces damage based on elements. Each enemy and spell belong to an element, either physical, fire, water, electric, or almighty. If a water spell is used on a fire enemy, it will deal significantly more damage than normal. If that same spell was used on an electric enemy, it would deal notably less damage than usual. The game repeats this process until either the player's or the enemy's health stat drops below zero. If the player dips below zero, it's game over, but, if the enemy dies, the player levels up, increasing

their stats and learning new skills based on their new stats, and moves on to the next enemy on the list.

Attached at the end of this document are my UML diagrams.

Literature Survey

No one game can be pointed to as the model for my project. However, I drew inspiration from a few different games. The biggest influences as follows:

Dragon Quest (Dragon Warrior): As one of the first turn-based RPGs, and certainly the most popular of them, most of *Dragon Ques*t's features are still shown in RPGs today. For example, the common overworld roaming followed by random encounters in a separate UI was one of *Dragon Quest*'s major features. I emulated the simplicity of the battle system as well as the standard fantasy aesthetic when creating my enemies.

Final Fantasy: Probably the single most popular RPG franchise of all time. The original took what was done in *Dragon Quest* and added more ideas into the mix. I didn't want to replicate the feel of *Final Fantasy* as much as I did *Dragon Quest*, but the higher complexity of *Final Fantasy* definitely was on my mind every step of the way.

Persona: The *Persona* series, while not well-known, is . It's a more modern style of game, so the battle system isn't as bare. The primary element from the *Persona* series that I implemented into my project was elemental damage. Enemies and spells have elements attached to them, allowing for certain enemies to be stronger or weaker to certain abilities. For example, a fire-based enemy would be weaker to water skills.

User Manual

Eclipse Setup Instructions: To set up the game, open Eclipse, open the File menu, and choose "Open Projects from File System". Click "Directory..." and navigate to the prj2 file. Click on the code folder, select OK, and select Finish. Then, run MainRunner.java to start the game.

The game is played using the console. The game prompts users, giving the situation and their choices and asking what they want to do. Upon starting the game, the player will be asked whether they want to start a new game, or load a previous game through a file. Either will ask the player their name. The new game option will randomly roll new stats for the player and start

them off against the first enemy, while the load game will search for the file with the given name. From this point forward, the game will inform the user of their abilities, their status, and the results of the previous turn. The player will type in a command (for example, “attack” to do a normal attack or “ember” to use the spell Ember), the game will record that command and carry out a turn with the AI enemy. The game will provide feedback in the form of messages describing what took place and how much damage was done (e.g. “You swung at [enemy name].| [Enemy name] took [damage] damage!”). In case the user forgets their current possible actions, the game will also accept a “help” and return all of the user’s commands. If the player needs more explanation, the game also accepts “describe abilities” and returns all possible moves the player can perform and a short description of them. After winning a battle, the game automatically saves the player’s information to a file in the “saves” folder, allowing them to restart from roughly where they left off at a later point. Every necessary input has a prompt before it to let the player know their options.

The game is also modifiable. In the “data” folder, there are four files: “enemy.txt”, “enemysample.txt”, “spells.txt”, and “spellsample.txt”. By editing “enemy.txt” or “spells.txt”, the player can add new enemies or learnable spells to the game. Both sample files show a format and explain how to make a new enemy or spell. Modifying these files should be easy and is encouraged. The only thing limiting the length of the game is how many enemies are in “enemy.txt”.

Technicalities of Battle - Player abilities: Players have two main choices in battle: attack or use a spell. Simply attacking doesn’t deal as much damage as using a spell (most of the time, at least), but attacks do not cost any of your limited MP to use and they can critically strike, increasing their damage massively. Spells normally do higher damage by default, but cost MP and don’t have the chance to critically strike. Spells have an element attached to them, as well. Enemies also have the same elements attached and may be weak or strong against certain spells. Hit them with something they’re weak to to enhance your damage greatly. Elemental weaknesses and strengths are as follows:

Nothing is weak/strong against physical.

Fire resists electric, but is weak to water.

Water resists fire, but is weak to electric.

Electric resists water, but is weak to fire.

Almighty is weak to itself but nothing else.

Win the battle to level up, restoring your health and MP, and continue on to the next enemy. The game also saves at this point, so you can close it and continue your game by loading it later.

Technicalities of Battle - Enemy abilities: Enemies have five abilities, one of which they can perform each turn. While random number generation is used to determine which move they use, they are weighted differently. Two of their moves are very common and often have lesser effects. Two are slightly less common and normally are more damaging or heal the enemy. The final move is a Desperation Move. It is very uncommon normally (happening only around 5% of the time), but it is almost guaranteed to be used when the enemy is low on health (happening around 90% of the time). This move is often more powerful than their other moves, so watch out.

Conclusion

I'm proud of my project. Though I didn't complete my "stretch goals", I feel like the finished project is of good quality. I accomplished what I wanted to accomplish and created a working game. In addition, I feel that adapting the project to work in its own window, rather than the console, is not far off and could easily be added from this point. I feel stronger in my programming ability and positive about what I've created with it.

References/Bibliography

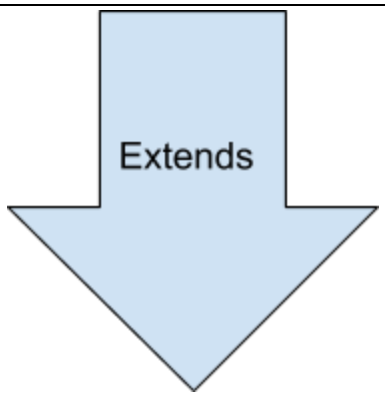
No outside sources were directly used.

All inspirations were listed above.

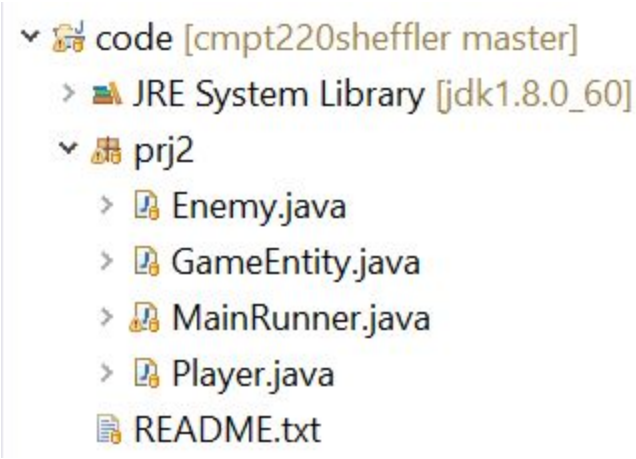
UML Diagrams and Additional Resources

MainRunner
+player: Player
+enemy: Enemy
+startGame()
+mainTurn()
+deathCheck()
+playerStatus()
+playerDecision(): String
+playerTurn(String: act)
+enemyTurn()
+save()
+load()

GameEntity
+name: String
+maxhealth: int
+health: int
+attack: int
+defense: int
+mental: int
+speed: int
+precision: int
+getName(): String
+getMaxHP(): int
+getHP(): int
+getAttack(): int
+getDefense(): int
+getMental(): int
+getSpeed(): int
+getPrecision(): int
+calcDamage(boolean: isMagic, boolean isCrit): int



Player	Enemy
+maxmp: int	+element: String
+mp: int	+description: String
+level: int	+Enemy()
+spells: Object[][]	+Enemy(int: enemynum)
+allspells: Object[][]	+init(int: enemynum)
+Player(name: String)	+describe()
+Player(String: name, int: level, int: maxhealth, int: attack, int: defense, int: mental, int: maxmp, int: speed, int: precision)	+takeDamage(int: damage, String: element)
+getSpells()	+generateAction(): int
+getMaxMP(): int	+getActType(int: actnum): String
+getMP(): int	+act(int: actnum): int
+listSpells(): String	+isWeak(String: element): boolean
+checkSpellValid(String: name): boolean	+isStrong(String: element): boolean
+describeSpells()	
+getSpellNumber(String: spellname): int	
+getSpellName(int: spellnum): String	
+getSpellElement(int: spellnum): String	
+takeDamage(int: damage)	
+calcSpellDamage(int: spellnum): int	
+levelUp()	



What the Eclipse filesystem should look like before running.