StrongHold Silver: Java RPG

By David Cruz

# Premise

Java RPG is a simple game where the user controls a band of characters as they fight enemies to find more characters. The user can manage these characters by changing the party composition, removing characters from the collection, and controlling them when they are in fights. To get more characters, the user must send a party out on expeditions that get harder as you get to higher level expeditions. In expeditions the player has a chance of finding more characters or fighting a random party of enemies. When an expedition is completed by beating the boss, the user can unlock a higher-level expedition to explore if the expedition was the hardest available. Characters are what the user controls, they have a few statics such as name, icon, health, attack, level, and experience. Characters icons are important in this game to show that the character has gained bonus statistics. Characters can upgrade to a better icon by fighting a certain number of enemies. When the player is done playing, they can save their game to a SQL database and reload it when they want to go back to playing.

# More details

This little game will need to use a couple of classes to function. The java RPG class is the main program which controls the rest of the classes via user text input. The character class is the main object that both the user and enemy controls. The Party Management class is an abstract class that controls party composition, collection, and character generation. Both the Player and Enemy classes are inherited from the Party Management class. The Player class will allow the user to control their party and create their first character. The Enemy class will generate a party of enemies and generate a boss party. The expedition class is the main entertainment loop of the game which allows the user to go and find more characters via entering a number which indicates the level that enemies will be generated as and contain the controls to fight in combat. As a terminal game, the inputs given from the player will be checked for valid inputs using try and catch.

# Needed classes.

* JavaRPG
  + Player – player object.
  + Expedition – expedition object.
* Character Base
  + Rank – int
  + Name – string
  + Level – int
  + currentExperience – int
  + defeatedEnemies – int
  + healthPercent – int
  + attackPercent – int
  + criticalPercent – int
  + rankMaxHealth - int
  + rankAttack - int
  + rankCriticalChance – int
  + leveledMaxHealth – int
  + leveledAttack – int
  + leveledCriticalChance – int
  + leveldExperienceThreshold – int
  + rankedEnemiesThreshold – int
  + currentHealth - int
* Party Management - abstract
  + The Player and Enemy class inherit this.
  + Party index list – int Array.
  + Character collection – Array list.
* Explore
  + Enemy – Enemy object
  + Player – instance of the Player object
  + Current floor - int
  + Max Difficulty – int.
  + Explore Difficulty - int
  + Current Characters Found - int

# Inheritance hierarchy

1. Java RPG class.
2. Character class.
3. Party Management class – Abstract.
   1. Player class.
      1. Save Manager class
   2. Enemy class.
4. Expedition class.

# Control hierarchy

* Java RPG class.
  + SceneController
  + Player class.
    - Save Manger class
  + Expedition class.
    - Enemy class.
* Party Management class
  + Character class.

CharacterBase Class

The character class is what stores individual information.

# Variables

* - Int icon
* - String name
* - Int level
* - Int currentExperience
* - Int defeatedEnemies
* - Int baseMaxhealth
* - Int baseAttack
* - Int baseCriticalChance
* - Int currentHealth
* - Int leveledMaxHealth
* - Int leveledAttack
* - Int leveledCriticalChance
* - Int leveldExperienceThreshold
* - Int rankedEnemiesThreshold
* + Int MAX\_LEVEL - 50
* + Char array ICONS – {☺, ☻, ♥, ♦, ♣, ♠}
* + String array NAMES – {"Ron", "Bob", "John", "Jane", "Margret", "Daniel", "Mimkro", "Viper", "Chumly", "PineApple", "Apple Pie", "Rosy Custard", "Spiffo", "Sabeth", "Kekht", "Leviathan", "Q Master", "Lain", "George", "Woods", "Kenny", "Ragged Johns", "Danny", "Test", "Text", "The Alchemist", "Trikoa", "Mismier", "Toro", "Megan", "Dream C", "Miami"}

# Functions

* + CharacterBase() – Create a default test character.
* + CharacterBase(name: String) – Create a random character with a name.
* + CharacterBase(level: int) – Create a random character with a level.
* + CharacterBase(name: String, rank: int, experience: int, defeatedEnemies: int, level: int, healthPercent: int, attackPercent: int, criticalPercent int) – Crate a character with predetermined stats.
* + setName(name String) – Change a character’s name.
* + getExperienceThreshold(): int – get how much experience is needed to level up.
* + getRankedThreshold(): int – get how many enemies need to be defeated to rank up.
* + getName(): String – gets the character’s name.
* + getRank(): int – get the characters rank.
* + getIcon(): char – gets the characters icon.
* + getLevel(): int – get the character’s level.
* + getExperience(): int – get the characters experience.
* + getDeafeatedEnemies(): int – get the characters defeated enemy count.
* + getHealthPercent(): int – gets the characters health percent.
* + getAttackPercent(): int – gets the characters attack percent.
* + getCriticalPercent(): int – gets the characters critical percent.
* + getMaxHealth(): int – gets the combined values of baseHealth and leveledhealth.
* + getCurrentHealth(): int gets the characters current health.
* + getAttack(): int – gets the combined values of baseAttack and leveledAttack. If the character has an attack number of 0 then it defaults to 1.
* + getCriticalChance(): int – gets the combined values of baseCriticalChance and leveldCriticalChance.
* - randomName() – gives the character a random name.
* - randomRank() – gives the character a random rank.
* - randomizeStatisticsPercent() – generates random statistics Percents for a character.
* - randomLevel() – generates a random level from 1 to max level.
* + calculateThreshold(currentNumber: int, threshold: int, level: int, thresholdOffset: int, thresholdMultiplayer, maxLevel int): Integer[] – creates an array of 3 numbers after calculating the next number using the threshold and calculates a new threshold.
* + addExperience(experienceGain: int) – adds experience and levels up a character.
* + addDefeatedEnemies(defeatedEnemiesGain: int) – adds defeated enemies and ranks up a character.
* - levelStatCalculator() – calculate the leveled stats of a character after leveling up.
* - rankStatCalculator() – calculates the ranked statis of a character after ranking up.
* + heal(healthRegain: int) – heal a character a certain number of points.
* + attackDamage(): int – gets the amount of damage a character does. If a character’s critical chance lands less than the critical chance, then the damage is doubled.
* + takeDamge(damage: int) – takes health away from a character.
* + toString(): string – returns the character in string form.
* + toExpansiveString(): string – returns a breakdown of a character.
* + toConsiseString(): String – returns a shorter string of information including icon, name, current heath, max health, attack, and critical chance.
* + toSaveString():String – returns the save data of a character which includes rank, name, currentExperience, defeatedEnemies, healthPercent, and criticalChance.

CharacterManagemer Class

# Variables

* - collection: ArrayList<CharacterBase>
* - party: int[]

# Methods

* + getCharacterFromCollection(index int) – get a character from the collection using an array.
* + getParty(): int[] – get your party’s index values where -1 is an empty spot.
* + getCollection(): ArrayList<CharacterBase> - returns the managers collection.
* + setCollection(newCollection: ArrayList<CharacterBase>) – replace your current collection with a new collection.
* + getPartyAsCollection(): ArrayList<CharacterBase> - converts the party into a list of characters.
* + getLivingPartyMembers(): ArrayList<CharacterBase> - gets the living party members.
* + hasPartyMembers(): boolean – checks if the party has members
* + hasLivingPartyMembers(): Boolean – checks if the party has any living member
* + changePartyIndex(partyIndexL: int, newCharcterIndex: int) – change the party index to a character. -1 means it’s empty.
* + generateCharacter(): CharacterBase – generates a random character.
* + generateCharacter(name String): CharacterBase - generate a random character with a name.
* + generateCharcter(level int): CharacterBase – generate a random character with a level.
* + addCharacter(character CharacterBase) – adds a character to the collection list.
* + removeCharacter(int index) – removes a character from the collection list.
* + clearParty() – empties all the party indexes back to -1.
* + clearCollection() – empties the collection along with the party.
* + collectionSaveList() – return the saved data of all characters in the user collection.
* + fullPartyHeal() – mass heal your party members to max health.

Player Class extends CharacterManager Class

# Variables

* - saveManager: SaveManger

# Methods

* + Player() – creates a default player.
* + Player(filename: String) – creates a player with loaded data.
* + getFileName() – get the current save file name.
* + setFileName() – set the save file name.
* + saveGame() – saves your game.
* + loadGame() – load a different game.
* + saveGameExits() – checks if the save game already exists.

Enemy Class extends CharacterManger Class

# Variables

* + generateTeam(level int) – generate a team to fight
* + generateBossTeam(level int) – generate a boss team to fight
* + generateFinalBossteam(level int) – generates the final boss team to fight

SaveManger Class

# Variables

* - saveName: String
* - SAVEPATH: String

# Methods

* + SaveManger() – create a default save manager.
* + getSaveName(): String – gets the save game name.
* + getSaveGamePath: String – gets the save game path.
* + setSaveName(name: String) – sets the save name.
* + createSavePathIfNotPresent() – creates the dictionary path for saved games if it doesn’t exist.
* + fileExists(): boolean – checks if the save file exists.
* + clearFile() – clears a file to save // slated for deletion.
* + saveToFile(list: ArrayList<CharacterBase>) – saves the game to a save game file.
* + loadSaveGame(): ArrayList<CharacterBase> - loads a game file.

JavaRPG class extends Application Class

# Variables

* - player: Player
* - explore: Explore

# Methods

* + start(primaryStage: Stage) – sets up the game
* + main(args: String[]) – starts the game

Explore Class

The game controller that manages the game’s events

# Variable

* - player: Player
* - enemy: Enemy
* - currentFloor: int
* - maxDifficulty: int
* - exploreDifficulty: int
* - currentCharactersFound: int
* - damgeDealt: int
* - enemiesDefeated: int

# Method

* + Explore(player: Player) – creates an explore object.
* + getDamageDealt(): int – get how much damage has been dealt.
* + getEnemiesDefeated(): int – get how many enemies have been defeated.
* + getCurrentFloor(): int – gets the current floor.
* + getMaxDifficulty(): int – gets the max difficulty.
* + getCurrentFloorDifficuatly(): int – gets the current floor difficulty.
* + getEnemyTeam(): ArrayList<CharacterBase> : int – gets the enemy’s team.
* + setDamageDealt(damage: int) – sets how much damage has been dealt.
* + setDefeatedEnemies(enemies: int) – sets the number of defeated enemies.
* + setMaxDifficulty(max: int) – sets the max difficulty.
* + setExploreDifficulty(difficulty: int) – sets the explore difficulty.
* + incrementFloor() – increment the current floor by 1.
* + resetFloors() – resets the current expedition.
* + increaseMaxDifficulty() – increase the max difficulty by 10 until max level.
* + randomEvent() – gives a random or predetermined event. 0 – nothing, 1 – new character, and 2 – fight. If the player doesn’t get a new character within a run, then give them a new character on floor then on floor 9 give the player a new character. On the last floor, the player fights a boss.
* + fight(): int – have the player and enemy team fight one round. -1 the enemy wins, 0 both teams are alive, 1 the player wins
* + attackTeam(attackingManager: CharcterManger, defendingManager: CharacterManger): int[] – have two parties fight a round and return how much damage and how many were defeated by the attacking party.
* + generateEnemyTeam() – generates a standard enemy team.
* + genrateBossTeam() – generates a boss team.
* + genrateFinalBossTeam() – generates a final boss team.
* + genrateCharacter(): CharacterBase – generates a character with the current floor difficulty.
* distributeXPAndEnemies(ArrayList<CharacterBase> team) – distributes experience and kills to the team. The experience split is damageDealt / 2 / teamSize.

jFX package

The jFX package is used to create the games UI. Most of the java files are used just for the game. There are 3 files that can be used outside of the game.

SceneController Class

# Variables

* - panes: HashMap<String, Pane>
* - parentPane: BorderPane
* - stage: Stage

# Methods

* + SceneController(stage Stage) – creates a scene controller.
* + getPanesHashMap(): HasMap<String, pane> - returns the hash map of the controller.
* + setPane(paneNane: String) – changes the pane to a pane within the hash map. If the pane implements Refresh, then the refresh method in the pane is activated.

JRPGFXPane Class extends BorderPane Class

A default abstract class that incorporates the basic needs of most panes used by the sceneCotnroller

# Variables

* - sceneController: SceneController
* - backBtn: Button

# Methods

* + JRPGFXPane(sceneController: SceneController, pageTitle: String) – creates a page with a page title.
* + JRPGFXPane(sceneController: SceneCotnroller, pageTitle: String, buttonText: String) – creates a page with a title and alternate button text.
* + JRPGFXPane(sceneController: ScneController, pageTitle: String, object EventHandler<ActionEvent>) – crate a page title and action event.
* + JRPGFXPane(sceneController: ScneController, pageTitle: String, buttonText: String, object EventHandler<ActionEvent>) – create a page title and action event.
* - setUpPane(sceneController: SceneController, pageTitle String) – set up page.
* + getSceneController(): SceneController – get the scene controller.
* + setSceneController(sceneContoller, SceneController) – set the scene controller.
* - setBackButtonOnAction(object EventHandler<ActionEvent>) – set the back button’s action

Refresh Interface

Allows for panes to refresh

# Methods

* + *refresh*()