Lab Activity 4

Objectives

1. Meet your teammates
2. Design character classes for your game
3. Apply polymorphism to implement your character classes

Teams

Time Estimate: 15 minutes

Your TAs will guide you through the team formation process to form teams of 2-3 students each. You will collaborate with your team to design the remaining 3 phases of your project, though all code will be developed and demonstrated individually. **Do not share code with your teammates until all lab sections have finished that demo**. You will continue to use your individual repository (No team repo) which you will not share with your teammates.

Once your team is formed, take a few minutes to meet your teammates and introduce yourself.

Character Class Design

Time Estimate: 25 minutes

At this point in the project you have a character class with a variety of stats, leveling mechanics, damage mechanics, and 2 forms of attack. Now, with your teammates, design different character types for your game which will be variations of the base Character class. Design enough types such that each teammate can implement at least 2 types during this lab activity.

Each character type you design must define the following:

- The initial stats of the character
- How each stat increases when leveling up
- Up to 4 battle options that can be chosen during a turn of battle (Attacks, spells, etc)
- At least 1 battle option that is learned at a certain level (Max of 4 battle options at max level)

Example: A fighter type with high attack power and defense, low/no magic attack and magic defense, and attack that deals damage based on their attack power, and at level 3 they learn a stronger attack that deals more damage but also damages themselves when used.

Note: You may add some randomness to your characters if you'd like, though it's not required. For example, the initial stats can be randomly chosen from a fixed range.

Character Class Implementation

Time Estimate: 60 minutes

This portion of the lab activity must be completed individually. You may continue to talk about the design of your game with your teammates, but your code must be your own.

Implement [at least] 2 of the character types designed by your team.

Character Class - In the character class, add the following abstract methods and make this an abstract class:

- A method named battleOptions that takes no parameters and returns a list of strings. This list contains strings identifying each of the battle options currently available to this character. This list should have a maximum size of 4.
- A method named takeAction that takes a string and a character as parameters and returns Unit. The string is a battle action to be taken on the input character. If the string is not a valid battle option for this character, nothing should happen. (Ex. takeAction("Physical Attack", enemyReference) or takeAction("Heal", allyReference))

Concrete Character Types - Create classes for each character type you are implementing. For each type:

- Create a new class for the type and extend Character
- Implement battleOptions and takeAction according to the design of your character type
    - While implementing takeAction you can reuse, or modify, the code for physical and magic attacks while writing additional methods for new actions
- Implement/override the stats defined for your character type (Initial stats and stat increases on level up)
- Add the ability to learn at least one new battle option after reaching a certain level

Testing - For each of your new character types, the following 5 pieces of functionality must be tested. You will demo your tests to a TA:

- The initial stats
- The initial battle options

- Stats after leveling up at least once
- Battle options after leveling up enough to learn a new option
- Taking each battle option on a character

Repository - Push your new code to your repository before leaving lab.

Lab Grading

0 points	Absent, no effort, or code not pushed to repo before the end of lab
10 points	At least 1 class created that extends Character and at least 2 testing bullets are tested
20 points	Mostly correct, but with at least one issue. This can include: Having unit testing that doesn't test **all 5** pieces of functionality for each character type, or having code that doesn't function based on the description of this lab
25 points	Satisfactorily completed all objectives

Project Phase 2 (Demo 2)

Due 24.02.20

WARNING: Do not share any of your project code with your teammates, or anyone else, until after Friday @10pm during the week of the demo. You cannot control what your teammate does with your code and if they share it with someone else in the class you may receive an academic integrity violation. Do not give anyone the opportunity to hamper your career.

The last lab section ends on Friday @10pm at which point everyone has submitted and you can freely share your work with your teammates and combine each piece of your project.

For phase 2 of the project, you and your teammates will build the user interfaces for your project. Their are two primary GUIs that must be built to make a complete game: The overworld and battle GUIs. We will also make games that simultaneously support web and desktop users.

For this phase, each member of your team will choose one of the following to implement:

- An overworld GUI for web users
- An overworld GUI for desktop users
- A battle GUI for web users
- A battle GUI for desktop users

Two team members cannot choose the same option. There are no teams with more than 3 members so no single team will build 2 full interfaces, but you will have opportunities to add the remaining pieces from other teams or the course staff.

Web GUI: The web GUI must run in a browser and be able to be served from a web server (You can use localhost for your demo). This GUI will use web technologies (HTML, CSS, JavaScript) to build the required functionality. You may use any libraries or technologies you find useful to complete your GUI including game libraries.

Desktop GUI: The desktop GUI must be written in Scala. You may use any libraries you find useful to complete your GUI including game libraries (ie. You do not have to use ScalaFX).

Overworld GUI: In the overworld each party can move freely in a large open area. If two parties collide while on the overworld, they will enter a battle.

View

- Display the overworld and all parties at their appropriate locations. This will be done by receiving a JSON string for the map itself, then receiving constant updates of JSON strings containing all the party locations on the map. Players will be able to move freely on the overworld, but a tile-based grid system will be used to indicate the terrain of the map. Most importantly for each tile the GUI will indicate whether or not a party can move through that tile (this functionality doesn't have to be implemented, just visualized)
- Map JSON format - {"mapSize":{"width": 500, "height": 500}, "tiles":[[tile]]}
    - mapSize is in tiles (The size will not always be 500x500)
    - tiles contains information for every tile in the overworld (this can be a very large JSON string). This is an array of arrays where each inner array represents a row of tiles on the map
        - Each tile is in the format - {"type":"", "passable": true/false}
        - You don't have to do anything with the tile type, but passable needs to be visualized in some way
- Write a setup method that takes a map formatted JSON string and renders the map with no parties
- Parties JSON format - {"playerParty": party, "otherParties":[party]}
    - Each party is in the format - {"location":{"x":4.36, "y": 107.85}, "level":5, "inBattle": false}
- Write an update method that takes a parties formatted JSON string and displays all parties on the overworld1
    - The player party must be indicated in some way so the player know where they are even while surrounded by other parties
    - Other parties must have some way of visualizing their level so a player knows how dangerous a battle may be
    - The locations are floating point numbers allowing parties to move freely between tiles. Parties should be shown at their exact location, not in the center of the closest tile

    - Parties in a battle must be indicated in some way

Controller

- The player can control their party in 4 directions and see the movement on the map
    - For this phase you must be able to fully move the player in some way without communicating with a server. Later in the semester, the raw inputs will be sent to your server for processing and a new JSON with your new location will be sent over the network for rendering
    - You can allow players to move through tiles that cannot be passed through (This functionality will be added to the server)
- As the player moves, create/update the JSON string representing the overworld with the new player location and call your update method
- The screen must scroll in some way as the player moves around
    - The player's party must always be on the screen
    - You cannot display the entire overworld in your GUI at the same time (Assume that the world will be large enough to make this impractical)
    - The method of scrolling is up to you (Keeping the player centered is acceptable)

Battle GUI: When a battle begins, the characters in each party will choose battle actions until all of the characters in one party reach 0 HP. The surviving party will gain the appropriate amount of experience points, based on demo 1 code, and return to the overworld. This GUI must contain the following features:

View

- Each character in both parties must be displayed on the screen (Squares or circles are fine). The information for the battle will be provided as a JSON string containing the 2 parties and name, hp, max hp, and battle options for each character in the battle (Later in the semester, this JSON string will be received after each update from the server)
    - You can assume parties are limited to 4 characters
    - The JSON will contain two keys - {"playerParty":party, "enemyParty":party}
    - Each party is in the format - {"characters":[{"name":"", "hp":50, "maxHP":70, "battleOptions": ["","","",""]}]}
- Write an update method that takes a JSON formatted string and renders all the characters on the screen with their names displayed
- If a character has 0 HP, this must be indicated in some way (Changing color is fine)
- A method named animate(String, String, int) which takes two character names as inputs and indicates the first input is taking a battle action on the second and an int indicating the change in hp of the second character. When this is called the GUI must indicate the interaction between these two characters in some way (Having each flash different colors is fine). There are 3 different animations that can happen on an action:
    - If the int is negative (dealing damage) indicate this as attack (eg. flashing red)
    - If the int is positive (healing) indicate this as a healing action

    - If the int is 0, indicate this as a neutral action (buff, debuff, miss)

Controller

- A method named takeTurn(String) must be implemented which indicates the name of a player character who is ready to take a turn. When this method is called, the player must be given a way to choose one of that character's battle options, as well as a target character for that option. Once the user chooses an option and target, call another method indicating the choices made (Later in the semester, this method will send the choices to the server for processing. For now, you can have this method print the choices to the screen/console for testing and for your demo).
    - While choosing the options, the player being controlled should be indicated in some way

Testing: There is no required unit testing for this demo. You should be prepared to demonstrate all required functionality to a TA. The burden of proof is on you to show that you've completed all the required functionality.

Submission

Push your code to your repository. You should push your code often while working on your project to backup you work. You latest push that's before the start of your lab on the due date will be your official submission.

Do not share your code with anyone, including your teammates, until after Friday @10pm during the week of the demo.

Be prepared to demo your project to a TA during lab on the due date.