The Quest + Quest of Legends

Requirements/Details:

Abstract

- ☑ ~~The heroes and the monsters live in a world represented by a **square grid of fixed dimensions.**~~
- ☑ ~~The heroes are able to fight the monsters using weapons, armors, potions and spells.~~

- [x] ~~The heroes can also augment their powers by purchasing items to assist in their quest.~~
- [x] ~~Every time the heroes win, they gain some experience and some money.~~
- [x] ~~When they accumulate enough experience they level up which means that their skills become stronger.~~
- The goal of the game is for the heroes to reach a very high level of experience.

The Market

- [x] ~~heroes can buy weapons, armors, spells and potions~~

Weapons

Used by hero to attack a monster, uncosumable

▼ **Attributes:**

- name
- price
- minimum hero level required → enforced during purchase
- damage inflicted
- number of hero's hands required

Armors

when worn by hero, reduces incoming damage from enemy's attack, uncosumable

▼ **Attributes:**

- name
- price
- minimum hero level required → enforced during purchase

Spells

represents magic attack and can be executed by a hero

cast of the spell, mana of hero deducted, consumable

▼ **Attributes:**

- name
- price
- minimum hero level required→ enforced during purchase
- damage range ←depends on the dexterity skill level of the hero
- magic energy (mana) required to get casted

▼ **Types:**

- ice spell → apart from the damage it causes it also reduces the damage range of the enemy
- fire spell→ apart from the damage it causes it also reduces the defense level of the enemy
- lightning spell → apart from the damage it causes it also reduces the dodge chance of the enemy

Potions

used by a hero in order to increase one of their statistics by some amount, consumable

▼ Attributes:

- name
- price
- minimum hero level required → enforced during purchase

The Heroes and Monsters

Heroes

▼ **Attributes:**

- name
- level

- health power ← uncapped, at 0 hero faints
- mana ← uncapped
- skills:
  - strength
  - dexterity
  - agility
- money
- experience → level up ⇒ levels of the skills get increased

▼ **Types:**

- warriors → *favored* strength and agility
- sorcerers → *favored* dexterity and agility
- paladins → *favored* strength and dexterity

Favored means that their starting statistics on those sectors will be increased and that every time they level up those statistics will be further boosted.

Level Up/ HP/ Mana Logistics

- [x] ~~You can assume that a hero needs to **acquire their_current_level*10 experience points to level up**.~~
- [x] ~~You can assume that **hp of both heroes and monsters** can be calculated as: **100*their_level**When a hero l**evels up** then this formula is used to reset calculate her/his hp.~~
- [x] ~~You can assume that when a hero **levels up** all of their skills get increased by **5% and their favored skills get an extra 5% increase**~~
- [x] ~~You can assume that the **mana** of the heroes when they **level up** can be calculated as: **current_mana + current_mana*0.1**.~~
- [x] ~~The exp becomes 0 when they level up and then they have to gather the required amount from scratch~~

Monster

▼ **Attributes:**

- name
- health power
- level
- base damage
- amount of defense which is deducted from the damage of an incoming attack
- chance to dodge an incoming attack

▼ **Types:**

- dragons → higher damage
- exoskeletons → increased defense
- spirits → higher dodge chance

The World of Play

- [x] ~~While me and you now know the keys that are used to play the game, when coding, keep in mind that you might want to give this to your little brother to play who does now know any of this... → write a proper read me~~
- [x] ~~A sample dimension size in which the game can be played is 8x8. In this size our suggestion is to have (randomly assigned) 20% non accessible cells, 30% markets and 50% common cells.~~
- [x] ~~The world of the game is represented by a grid of specific dimensions.~~
- [x] ~~Some tiles of the grid might not be accessible or might be markets or might be just common tiles/cells.~~
- [x] ~~At any given moment the team of heroes (which are at most three and at least one) is placed in a specific tile of the grid and they can move by one tile/cell up or down or left or right.~~
- [ ] The player should be able at any moment to display information related to the heroes like their level, their hp, their mana, their current exp and their skill levels.

- [x] ~~**Let’s all agree to a specific common set of keys that will be used** :~~
  ~~**W/w:** move up~~
  ~~**A/a:** move left~~
  ~~**S/s:** move down~~
  ~~**D/d:** move right~~
  ~~**Q/q:** quit game~~

- [x] ~~**I/i:** show information. I~~
  ~~f we are **not in a fight** this should show information about the heroes (more specifically **their level, their hp, their mana, their current exp, their money and their skill levels**).~~
  ~~If we are on a fight this should show information about the heroes (more specifically **their level, their hp, their mana and their currently equipped weapons and armors**) and in separate section information about the monsters (more specifically their level, their hp, their defense and their damage)~~

- [ ] m/M The player should be able at any moment to display the map (grid). There should a comprehensive way of visually representing all the tiles and their properties (whether the heroes are in a specific tile, if a tile is accessible and if it contains a market).

- [x] ~~The player should be able at any moment to quit the game. ← I implemented this so that players can only quit when they are traversing the map/ after they choose to leave a tile~~

- [x] ~~The heroes can buy items from the market if they have enough money to do so.~~

- [x] ~~Each hero has her/his **private wallet** and **does not wish to share her/his money with any other hero**. Moreover, they can **sell items to the half of the price they were bought**.~~

- [x] ~~Those **transactions must take place through a special menu** where the player will be able to **see all the items available for selling or buying as well as the information of those items** (price, required hero level etc.).~~

- [x] ~~When the **heroes do not buy or sell items and do not fight they should be able to check their inventories, to change the weapon they hold and the armor they wear. They should also be able to consume a potion.**~~

- [x] The **fight starts automatically as soon as the heroes move to that tile** (if it is one of their unlucky tiles).
- [x] **The monsters that "live" in a cell are not created when the map is created.** Every time the heroes visit a cell we "roll a dice" and if they are "unlucky" then at that moment we create the monsters and initiate the fight.
- [x] every time the heroes **visit a common tile** there is a **chance that they will engage in a fight with monsters** that will **have the same level as the level of the highest leveled hero**.
- [x] The heroes fight **always** monsters that their level is equal to the level of the "most experienced" hero of the team. The monsters' level does not increase, you just pick every time a monster from the helper files that has the level that you are looking for. Also, to make sure that I will be able to find some monsters to fight and test your code please **set the probability of encountering monsters on a common cell >= 75%** in your final submission

The Fight

- [x] A fight takes place between heroes and monsters
- [x] The first to attack is always the team of heroes

- [x] A fight consists of multiple rounds
- [x] Make sure that during a round of a fight **you show clearly to the user who caused how much damage to whom**.
- [x] During a round of the fight, when it is the turn of the heroes, the player chooses for each hero whether they will do a **regular attack** or if they will **cast a spell** or if they **will use a potion** or **if they will change their armor/weapon.**
  - [x] You can assume that the **damage** a hero causes with an attack with their weapon can be calculated as: **(strength + weapon damage)*0.05**.
  - [x] A **spell's final damage** can be calculated by the following formula: **spells_base_damage +(dexterity/10000)*spells_base_damage.**
  - [x] You can assume that the level of the **enemy's skill deterioration that is caused from each of the spells is equal to 10%.**

- [x] ~~You can assume that a hero has a **probability of dodging an attack** which can be calculated as: **agility*0.02**.~~

- [x] ~~The heroes regain some of their hp at every round if they are still alive.~~
- [x] ~~You can assume that during **every round of a fight the heroes regain 5% of their hp and 5% of their mana.**~~
- [x] ~~At each round a hero can perform only one of the above~~
- [x] ~~At each round the player can display the stats of a hero or a monster.~~
- [x] ~~You can assume that all of the fights will be either 1v1, 2v2 or 3v3 depending on the number of the heroes the player wants to start their game with.~~

- [x] ~~The fight ends only when the hp of either all the monsters or all the heroes is zeroed. **NO FLIGHT**~~
- [x] ~~If the heroes win the fight then they earn some money based on the level and the number of monsters that they faced in that fight.~~
- [x] ~~"You can assume that after every successful fight each hero who did not faint gains 100*monsters_level coins and 2 exp for their troubles".~~
- [x] ~~If the monsters win the fight the heroes lose half of their money.~~
- [x] ~~If after the end of a fight (won by the heroes) a hero's hp is zero the hero gets revived by the other hero(es) and gets back half of his/her hp but doesn't gain any exp~~
- [x] ~~It is an acceptable variation to end the game if all of the heroes die - someone could argue that there is no one to revive them. So whether you decide to end the game only on the user decides to do so or on the death of all the heroes as well it is correct. → I'll do revive by 0.25~~
- [x] ~~(So, for example if the player decided to start their adventure with two heroes, then every time the heroes get in a fight the fight will be against exactly two monsters. For more simplicity you can assume that the first hero will always attack the first monster and the second hero the second monster. However, do not forget to think of the case that one of them faints and the alive hero has to fight both of the monsters – or the opposite.)Implementation~~

Implementation

TheQuestGameEngine.java `class`

▼ Fields

All fields are private

- `final int` rowsize → Map row size
- `final int` col size → Map column size
- `final double` probabilityInaccessible → Map probability of Inaccessible tiles
- `final double` probabilityMarket → Map probability of Market tiles
- `final double` probabilityCommon → Map probability of Common tiles
- `final double` probabilityEncounter → Map probability of encountering monster on common tiles
- `final Merchant` merchant → merchant of the game
- `final List<Hero>` heroes → all possible heroes of the game
- `final List<Monster>` monsters → all possible monsters of the game
- `final` Map → Map of the game
- `final` Team → Team of heroes selected by user
- `int` location → current location of the team of heroes in the map

▼ Methods

Public Methods

- Constructor
  - initialize fields according to game default values found in `Defaults.java`
  - initialize Map by calling `Map.java`'s constructor
  - initialize Team by calling `selectTeam` method
  - calls `startGame` method

Private Methods

- numTeam
  - prompts user for number of heroes within the team 1-3
- selectTeam
  - calls `numTeam` method
  - prompts the user to select a hero from the selection of default heroes for each team member
- startGame
  - assigns location field to top left market tile by calling `Map.java`'s `getSafeStart` method
  - calls `Map.java`'s place method to place team on the specific location field
- checkMove
  - check if user entered a valid move, valid here implies not beyond the area of the map.
- validateTile
  - checks if the user entered a valid move by calling `Map.java`'s `getTile` and checking tile type, valid here implies not an inaccessible tile
- computeLocation
  - computes the new location resulted from the user entered move
- move
  - prompts the user to select next move or quit game
  - input validates user move by calling `checkMove` and `validateTile`
    - if user chooses to quit, return
    - else assigns location field to user entered move by calling `computeLocation`
    - calls `Map.java`'s place method to place team on the specific location field

        - calls `move` again (recursion)

Map.java `class`

▼ Fields

All fields are private

- `final int` rowsize → Map row size
- `final int` col size → Map column size
- `final Tile[][]` map → Map
- `final double` probabilityInaccessible → Map probability of Inaccessible tiles
- `final double` probabilityMarket → Map probability of Market tiles
- `final double` probabilityCommon → Map probability of Common tiles
- `final double` probabilityEncounter → Map probability of encountering monster on common tiles

▼ Methods

Public Methods

- Constructor
    - initialize fields accordingly to values passed from `TheQuestEngine.java`
    - calls `setMap` method
- getTile
    - returns tile of the map on given a location
- getSafeStart
    - returns location of first market tile occurrence (top left)
- display
    - prints out map
- place
    - displays map appropriately
    - given a location, retrieve tile on that location and activate the tile

- if market tile
    - prompts if the user would like to leave or not
        - while no
            - prompts if the user would like to perform a transaction or explore inventory or none
                - call `Team.java`'s `transaction` method
                - call `Team.java`'s `exploreInventory` method
            - prompts if the user would like to leave or not
                - if yes break
        - if yes
            - deactivate tile, return
- if common tile
    - checks if the current common tile monster is awake by calling `CommonTile.java`'s `monsterAwake` method
    - if monster is awake, get enemies (list of monster) from common tile by calling `CommonTile.java`'s `getMonster` method and passing in `Team.java`'s `getnC` and `getMaxLeve` methods
    - start battle by calling `Team.java`'s `battle` method
    - prompt if the user would like to leave or not
        - if no
            - call `Team.java`'s `exploreInventory` method
        - if yes
            - deactivate tile, return
- deactivate tile

Private Methods

- genOptions
    - creates a list of various types of tiles based on the appropriate probability

    - calls constructor of each tile type and passing the appropriate values (MarketTile with merchant, CommonTile with monsters and probabilityEncounter)
    - shuffles the list before returning it
- setMap
    - calls `genOptions` to get list of tiles
    - sets each tile Map field to appropriate tile
    - sets location of each tile object by calling `Tile.java`'s `setLocation`

Package: utils

Defaults.java `class`

IOConstants.java `class`

ColouredOutputs.java `class`

ErrorMessage.java `class`

Package: tile

Tile.java `abstract class`

TileType.java `enum`

MarketTile.java `class`

CommonTile.java `class`

InaccessibleTile.java `class`

Package: character

Character.java `abstract class`

Team.java `class`

▼ Fields

all fields are private

- `List<character.hero.Hero>` team

▼ Methods

Public Methods

- getnC
  - return number of team members
- getMaxLevel
  - return level of the highest leveled hero
- exploreInventory
  - prompts the user to select a hero to explore inventory by calling `Hero.java`'s `exploreInventory` method on the selected hero
  - prompts user if user would like to explore another hero's inventory
    - if yes
      - call `exploreInventory` (recursion)
    - if no
      - return
- transaction
  - prompts the user to select a hero to talk to the merchant by calling `Hero.java`'s `talkToMerchant` method on the selected hero and merchant, also calls Merchant.java's
  - prompts user if user would like to have another hero talk to the merchant
    - if yes
      - call `transaction` (recursion)

        - if no
            - return
- battle
    - while heroes are alive or monsters are alive, checked by calling `everyoneFainted` and `defeatedMonsters` methods
        - for 1- team size
            - if current hero not dead, gets a turn
                - prompts the user to select, display info or change weapon/armor or use a single potion or attack or cast spell.
                - if display info
                    - display appropriate info (`Hero.java`'s `battleDisplay` method, `displayEnemies`)
                    - prompts the user to select, change weapon/armor or use a single potion or attack or cast spell.
                - if change weapon/armor
                    - calls `Hero.java`'s `change` method
                    - calls `Hero.java`'s `battleDisplay` method
                - if use potion
                    - if the hero has potion, checked by calling `Inventory.java`'s `numPotions` method on `Hero.java`'s `getInventory` method
                        - calls `Hero.java`'s `usePotion` method
                        - calls `Hero.java`'s `battleDisplay` method
                    - if hero has no potion
                        - turn wasted
                - if attack
                    - hero attack monster and get an AttackResult

- current hero calling `Hero.java`'s `doBasicAttack` method
- current enemy calling `Monster.java`'s `receiveBasicAttack` method

- if AttackResult DEAD
  - NEEDS FIXING... didn't consider case all monster are dead currently in an infinite loop fml (although might never happen since if kill, checks and break but further testing required)
  - prompts the user to select another alive monster
  - Attack the selected monster the same way
- if AttackResult DODGED
- if AttackResult SUCCESS
- if AttackResult Kill
  - check if all monster are dead, if yes break

- if spell
  - if the hero has spell, checked by calling `Inventory.java`'s `numSpell` method on `Hero.java`'s `getInventory` method
  - similar to attack except
  - hero attack monster and get an AttackResult
    - current hero calling `Hero.java`'s `castSpell` and `castSpellDamage` method
    - current enemy calling `Monster.java`'s `receiveSpell` method
  - if hero has no spell
    - turn wasted

- if current monster not dead, gets a turn
  - similar to attack

        - post round regenration
            - for all heroes that is alive call `Hero.java`'s `regen` method
    - check for which team win by calling `defeatedMonsters` method
        - if heroes win call `endBattle` method with true parameter
        - else call `endBattle` method with false parameter

Private Methods

- display
- talkToMerchant
    - COULD BE REFACTORED INTO `Hero.java`, but I think here is fine too
    - prompts the user if the selected hero would like to buy or sell items
        - if buy
            - call `Hero.java`'s `buy` method
        - if sell
            - call `Hero.java`'s `sell` method
    - prompts if the user is done talking to merchant
        - If yes
            - return
        - if no
            - call `talkToMerchant` (recursion)
- everyoneFainted
    - return true if no heroes in team is alive
- defeatedMonsters
    - returns true if no monsters in enemies alive
- displayEnemies
- endBattle
    - if win

        - reward alive heroes accordingly
        - revive fainted heroes accordingly
    - if lose
        - tax heroes
        - revive fainted heroes accordingly

Transaction.java `interface`

- implemented by Hero and Merchant
- ▼ abstract methods
    - buy takes in a list of items → no operation in Merchant
    - sell returns a list of items

Battle.java `interface`

- implemented by Hero and Monster
- ▼ abstract methods
    - receiveBasicAttack takes in double damage, returns AttackResult
    - receiveSpell takes Spell spell and double spellDamage, returns AttackResult in → no operation in Hero
    - doBasicAttack returns double
    - castSpell returns Spell → no operation in Monster
    - castSpellDamage returns double→ no operation in Mnster
    - regen → no operation in Monster

AttackResult.java `enum`

- ▼ Types

  DEAD → if the attacked is dead

  DODGE → if the attacked dodge

SUCCESS → if the attacked received intended damage

KILL → if the attacked died from intended damage

Package : character.merchant

Merchant.java `class`

Package: character.hero

Hero.java `class`

HeroType.java `enum`

Inventory.java `class`

ItemQuantity.java `class`

Package: character.monster

Monster.java `class`

MonsterType.java `enum`

Package : character.items

Item.java `class`

ItemType.java `enum`

Package: character.items.weapons

Weapons.java `class`

Package: character.items.armors

Armors.java `class`

Package: character.items.spells

Spell.java `class`

SpellType.java `enum`

Package: character.items.potions

Potion.java `class`

PotionType.java `enum`

Feedback from TA

Good deeds

- did a great job understanding the OO, and the use of interface and inheritance relation.
- Your design is really complete and great.
- It’s great that you think of having merchant which implements the transaction interface.
- Knowing that item as abstract class which gets extended.
- Monster and hero using the battle interface.
- The code and design is clear and organized.

Room for improvement

- Heroes, Spells, Monsters,Potions could be improved in terms of an OO design perspective through using inheritance instead for scalability

- ☑️ ~~Refactor Hero Class~~
- ☑️ ~~Refactor Spells~~
- ☑️ ~~Refactor Monsters~~

- [x] ~~Refactor Items~~
- [x] ~~fix no operations on interface base on new lecture materials~~
    - HeroBattle Interface implements Battle Interface
    - MonsterBattle Interface implements Battle Interface
- [x] ~~Refactor Potions~~

-What if a potion can increase multiple attribute in the future?

- https://stackoverflow.com/questions/4254182/inheritance-vs-enum-properties-in-the-domain-model
- https://stackoverflow.com/questions/41001932/oo-design-inheritance-vs-type-enum-variable

---

bugs

- [x] ~~fix weapon infinite loop issue~~
- [x] ~~fix bounty distribution display~~
- [ ] fix cast spell mana issue
    - mana validation before choosing spell?.. to prevent wasting turns?
    - mana is 0 remains 0 after regen...
- [x] ~~fix transaction interface into single responsibility~~

Part 2

The Quest of Legends