Inventory Class

This class is used for organizing the inventory objects of the player.

Attributes:

private int weightLimit: This shows the maximum weight that player's inventories can achieve. Players can not exceed this weight limit.

private Item[] itemList: This shows the list of items that user currently have.

private int currentWeight: This attribute holds the current weight of the player's items to compare the whether the user exceeds the weight limit or not.

Operations:

public boolean deleteFromInventory(Item): This is used for deleting an item from the inventory. When the user drops an item or gives it to NPC item will be deleted from his inventory with this function.

public boolean isFull(): This function shows whether the current items in player's inventory is equal to the weightLimit or not. It operates with the addToTheInventory() function.

public void addToTheInventory(Item): This is used for adding an item to the inventory. If isFull() function returns false, then user can add item. Otherwise, he can't.

public void draw(): This function draws each item on the console that player currently have. When the user calls changeToInventoryView(), which is an operation of GameEngine class, this function will be called and console will show the current inventory.

Item Class

This class depends on the Inventory class. It keeps the operations and attributes of each item in the player's inventory.

Attributes:

Private String Description: This is used for describing the item's features.

private int weight: It keeps the weight of items since each item has specific weight. This attribute will be used when calculating the currentLimit and checking weightLimit of inventory.

Private Image icon: It shows the icon of the item in the inventory view in console.

Operations:

public void draw(): This is used for drawing the item onto the console when the changeToInventoryView() is called.

WearableItem Class

This class depends on the Item class. It keeps the wearable items in the inventory which are Weapons and Armors.

Weapon Class

This class holds the attribues of weapons.

Attributes:

private int baseDamage: This valueholds the damage of the selected weapon. This value is added to the player's attackDamageValue which is an attribute of Charachter class..

Armor Class

This class holds the attribues of armors.

Attributes:

Private int baseStrength: This attribute holds the base sterngth value of selected armor. When player use an armor, the str value of the player will increase.

private int baseProtection: This attribute holds the base protection value of selected armor. When player use an armor, the protection value of the player will increase.

QuestItem Class

This class depends on the Item class. It keeps the quest items in the inventory that are used during a quest.

ConsumableItem Class

This class depends on the Item class. It keeps the consumable items in the inventory.

TemporaryEffect Class

Skill Class

Class Class

BlockObject Class

BlockObject is an object that only blocks the way of the player.

It overrides all the methods of the GameObject Class.

Character Class

Private int[] stats: An integer list of size 4. Each integer corresponds to Strength, Intelligence, Dexterity, Constitution. These all have impact on the damage a player can deal and the damage absorption that player can have.

Private int Health: The health of character. More health means harder to kill.

Private int mana: Mana of character. If a character has more mana, he/she can cast more skills.

Private int Stamina: Stamina of character.

Private TerrainCell mapCell: The Cell that character is on.

Private Class Classs: the Class that character belongs to.

Private temporaryEffect[] tempEffectList: current temporary effects that character has at that moment.

Private Zone currentZone: the Zone the character is on.

Private double ArmorValue: the Armor value that player has.

Private double attackDamageValue: The amount of damage that player can hit with normal Attack.

Operations:

Enemies Class

Enemy class has the attributes of the enemies. It extends the Character Class.

Attributes:

Private String [] ScriptedDialogues: Dialogue list of the Enemy.

Private Item[] droppableItems: List of Items that can be given to the Player if he/she kills the enemy.

Private int expPointValue: the amount of experience points that the player will get after killing this enemy.

Operations:

Public void talk(): Randomly selects one of the scripted dialogues and displays it on the User’s screen.

Public void create(String s): Takes the name of Enemy to be created as parameter and returns the Corresponding enemy.

**There are 4 types of enemies. They are extending the Enemies class. They are : Ogre,Boss,HumanEnemy,Ghoul. They have their own create methods.**

Player Class

Private Inventory inventory : This is the Inventory of the Player. It contains the items of player.

Private int level: this indicates the level of the player. Player’s enemies will be created according to the player’s level. Like 2 level Player will encounter with enemies with level 1 and level 2.

Private int experiencePoints: this is the amount of experience Points that player has.

Private Item equipments[]: These are the items that are equipped by the player. Equipped Items will be changing the stats of the Player.

Private ArrayList<Quest> unfinishedQuests: These are the quests that Player has not taken yet by not talking to the related NPC.

Private ArrayList<Quest> activeQuests: These are the ongoing Quests of the Player.

Private ArrayList<Quest> finishedQuests: List of finished Quests. These quests are the finished. Number of finished Quests will be used for displaying statistics.

private int enemyKillCount: this is the total number of killed by enemy.

Private Player player: This is the instance of the Player. Essential since we do not want multiple copies of the player. Part of using Singleton Design Pattern.

Private int[] expReq: List of required experience points for all the levels.

Operations:

Public void changeTemporaryStat(String,int amount):Changes temporary Stat with the given amount of amount.

Public void levelUp(): Increases the level of the Player if it exceeds the required amount of leveling up.

Public void addExp(int exp): adds the given amount of experience Points to the player’s total experience points.

Public boolean equip(Item I): Adds the item to the equipments if it is in the inventory and returns true. Returns false if it is not in the inventory.

Public void displayInventory(): displays the contents of the inventory.

Public Item findItem(String itemName): Takes the item name and returns the item from the inventory.

Public void updateAttackArmorValue(): Attack and Armor values of the player will be updated each time he/she equips an item. This method is called by equip method.

Private Player(): Private Constructor. Part of Singleton pattern.

Public static Player getInstance(): returns the player instance if it is already created. Creates a new player instance if it is not created yet

BattleStarter Class

BattleStarter is a kind of object that if Player collides with it , a battle is engaged. It is a GameObject so it overrides all the methods of the GameObject class. Overriden methods are not listed below.

Attributes:

Private BattleMap battleMapReference: This is the reference battleMap that the battle will occur.

Private BattleFactory battleFactory: this is the BattleFactory object that both creates enemies and the corresponding battleMap

Operations:

Public BattleMap startBattle(): This method calls for readFile and createBattleMap methods. Then returns the battleMap that created by calling the listed 2 methods.

private String[] readFile(String filename): This method reads the enemy information and the map information stored on the map from the file whose name is passed as parameter. And returns all the information in a String list.

private BattleMap createBattleMap(String [] infoList):This method takes the information list that is returned by the readFile method, calls the methods of battleFactory to get a BattleMap and Enemies. It sends the information of enemies to the createEnemy() method of BattleFactory. Takes an empty BattleMap and fills it with player and the enemies.

EnemyAI Class

EnemyFactory Class

This factory creates Enemies with given names. This is part of using Factory Method design pattern.

Private EnemyFactory instance: instance of EnemyFactory. Part of using Singleton Design Pattern.

Operations:

Public static EnemyFactory getInstance(): If the instance is created, returns the instance. If it is not created, creates a new Enemy Factory instance and returns it.

Public Enemy getEnemy(String s): creates appropriate Enemy and returns it to the caller back.

BattleFactory Class

This class creates the Battle Components.

Operations:

Public BattleMap createBattleMap(): Creates an empty Battle map and returns it.

Public Enemies createEnemy(String s): creates and returns an enemy with given name.

BattleMap Class

BattleMap extends Map class so it has all the attributes that Map Class has. In addition, this map has the instance of Player on the BattleMap and has instances of Enemies that Player will face on the BattleMap as attributes.

Operation:

Public draw():overrides the draw method of Map class in a way that BattleMap with appropriate images will be drawn.

Map Class

Map is an abstract class. Map is the place that player will be wandering on mostly. There are 2 kinds of Maps. One is BattleMap that player is placed when the player is engaged in a battle. The other one is the Zone, the map that player is on when he/she is not engaged in a battle.

Attributes:

Private TerrainCell terrainCellList[]: This is the list of terrain cells on the map. Every map is consisting of terrain cells that hold the game objects on them.

Private Character characterList[]: This is the list of characters that are on the map.

Operations:

Public void draw(): This operation draws the Map on the Graphical User Interface.

GameObject Class

This is an abstract class.Game Objects are the objects that player can interact or collide with. There are 4 types of GameObjects: Character ,NPC,Block Object and BattleStarter.

Attributes:

Private String name: Name of the game object.

Private Image Icon: this is the Image of the object that is displayed on the map.

Private terrainCell cell: this is the cell that game object is on.

Private boolean interactiveness: If a game object is not block object, then it is interactable so it is true. It is false if the object is a block object.

Operations:

Public void move(String direction): It used to move the object to the given

Public void draw(): Draws the gameObject on the map with its icon displayed on the corresponding TerrainCell.

Public boolean isInteractable(): returns true if the object is interactable. False if it is not interactible.

TerrainCell Class

This class is used for dividing maps into cells and putting each a GameObject to Cells when it is needed.

Attributes:

Private Point position: It has the position of the given TerrainCell.

Private boolean blocked: If the cell contains a gameObject it is true, otherwise it is false.

Private GameObject gameobject: this is the GameObject that is on the specified cell. it is NULL, if the TerrainCell is empty and it is not blocked. If it is not NULL, then it means specified TerrainCell is not empty.

Operations:  
public void draw(): It draws the TerrainCell on the map.

Public boolean isBlocked(): It returns whether the cell is blocked or not. Returns true if blocked, false otherwise.

Zone Class

Zone class extends the Map class. Zone is the map that player can wander if he/she is not in a battle. NPCs, and other staff will be here.

Attributes:   
Private NPC Npclist[]: This is the list of NPCs on the zone.

Operations:

Public draw(): overrides the draw method of Map class in a way that Zone Map with appropriate images will be drawn.

NPC Class

NPC is a class is a class that is responsible for giving quests to the Player and giving rewards to the Player if the player has completed a quest. NPC is a game object. So it overrides the methods of the GameObject Class. The overriden methods are not listed below.

Attributes:

Private String name: this is the name of the NPC

Private String ScriptDialogue[]: This is the dialogue List of of the NPC. If the player selects to get a quest, Quest related dialogue will be displayed, otherwise introductory dialogue will be displayed

Private Quest QuestList[]: This is the list of the Quests that the NPC can give. Player can select any of them(if available)

Private TerrainCell zoneCell: This is the cell of the NPC on the map.

Operations:

Public giveQuest(Player p1,int choice): the chosen quest on the list will be added to the Player’s ongoing quests. Int Choice will be the index of the Quest on the list.

Public talk(): this will be menu method that displays the introductory dialogue of the NPC. IT will ask for the user whether he wants to take a quest or exit the dialogue. The appropriate action will be taken according to the user’s input. This

Quest Class

Private Description: Type is string. short description of the quest. Contains information about the quest like how many monster to kill/ where to go to complete the quest etc.

Private Goals: Type is a list of Strings. The goals for completing the quest will be stored here. If the player completes a goal, it will be deleted from the goal List.

Private currentGoal: Type is string. It is the current stage of quest that the player is on.

Private rewardExp:Type is int. The amount of experience point that will be given as reward to the player for completing the quest. This will be taken when the quest I completed by the player.

Operations:

SwitchGoal(): This operation passes the current Goal to the next goal on the list.This method will be used when the player passes the current Goal.

TurnManager Class

CollisionHandler Class

Collision Handler class is responsible for checking and handling the collisions that occur when the user wants his/her player to move towards a direction. In such a case, If a player is moving to another direction, move method of Game Engine class is being called and move() method of game engine calls for the methods of CollisionHandler Class. So that if a collision occurs, the appropriate action taken by the game Engine.This class has no attributes, instead it has several operations.

Operations:

Public checkForCollusion(Character, TerrainCell): This method takes the Player instance and the Terrain Cell object that player wants to move to( Terrain Cell is a cell of the map that either contains a game object or not). Then checks for the possible collisions that the player may encounter. Player can either collide with a block object,with an NPC or with a BattleStarter object if the player is on the zone map( whenever a player collides with a battle starter object, a battle is engaged between player and his/her enemies). Or player can collide with an Enemy if the player is on a battle map. For all the cases, if the player collides with a game object, handleCollision() methods are called for each object

Public isCollusionBlock(Character, TerrainCell): This method takes the Player instance and the Terrain Cell that player wants to move to. If the TerrainCell ,that player wants to move to, has a Block object on it, it returns true. Otherwise it returns false.

Public isCollusionNPC(Character,Terrain Cell): This method takes the Player instance and the Terrain Cell that player wants to move to. If the TerrainCell ,that player wants to move to, has an NPC object on it, it returns true. Otherwise it returns false.

Public isCollusionBattle(Character,Terrain Cell): This method takes the Player instance and the Terrain Cell that player wants to move to. If the TerrainCell ,that player wants to move to, has a BattleStarter object on it, it returns true. Otherwise it returns false.

Public isCollusionEnemy(Character,Terrain Cell): This method takes the Player instance and the Terrain Cell that player wants to move to. This method is called if the player is on a battle map. If the TerrainCell ,that player wants to move to, has an enemy object on it, it returns true. Otherwise it returns false.

Public handleCollusionBlock(Character, TerrainCell): This method takes the Player instance and the Terrain Cell that player wants to move to. This method is called by checkForCollusion() method if the result from isCollusionBlock() is true. The player will be informed with appropriate message that he/she hit a block. No other action will be taken. So this method returns nothing.

Public handleCollusionNPC(Character, TerrainCell): This method takes the Player instance and the Terrain Cell that player wants to move to. This method is called by checkForCollusion() method if the result from isCollusionNPC() is true. The player will talk with NPC in such a case by calling talk() method of NPC. No other action will be taken. So this method returns nothing.

Public handleCollusionBattle(Character, TerrainCell): This method takes the Player instance and the Terrain Cell that player wants to move to. This method is called by checkForCollusion() method if the result from isCollusionBattle() is true. The player will be engaged in a battle, startBattle()operation of the BattleStarter object will be called.

Public handleCollusionEnemy(Character, TerrainCell): This method takes the Player instance and the Terrain Cell that player wants to move to. This method is called by checkForCollusion() method if the result from isCollusionEnemy() is true. Player will attack to the enemy by using normalAttack() method of the Character.

CommandInterpreter Class

ConsoleResponder Class

QuestManager Class

This Class is responsible for handling the quests that player is completing/doing and storing the quests that are active/inactive for the player.

Attributes:

Private totalQuests : Type is int. It stores the total numbers of quests in the whole gaming process. This information is used when the user wants to see his/her statistics.

Private totalQuestsFinished: Type is int. Used for preparing statistics of the user.

Private currentQuest: Type is Quest. This is the current Quest that player is doing. Whenever user wants to display his/her quest information, the information is taken from this instance.

Private ongoingQuests: this is a list of Quests. These are the quests that player hasn’t finished.

Operations:

switchCurrentQuest(Player): This operation returns nothing and switches the quest that player is doing to the next quest on the Quest List. So that whenever user wants to display information about the Quest he is doing, the switched Quest’s information will be displayed.

updateCurrentQuest(Player): updates the Player’s current Quests’ Goal to the next goal on the Quest’s Goal List.

checkQuestEnding(Player): checks the Player’s current Quest’s ending state and deletes the quest from ongoing quest List if the player has completed all the goals in the quest.

BattleManager Class

MusicPlayer Class

This class is responsible for playing the music according to the user’s preferances. It has no attributes, an instance of this class is stored in Settings Manager Class. Whenever a music will be played, the operation of the MusicPlayer instance is used.

Operation:

Public playFile(File f): This method returns nothing, takes a Music File that will be played and Plays the music.

FileManager Class

This class is used for file processes like saving the game, loading the game.

Attributes:

Private savePath: Type is String. This is the path that the game file will be saved at.

Operations:

Public loadGame(): This operation loads the game information stored at the file located at the savePath.

Public SaveGame(): This operation uses the Player and Map information stored at the game Engine, stores it in a data file located on the save Path of the hard disk.

Public deleteSave(): This operation deletes the save file located at the save path of the harddisk. This operation is used when the player dies while he/she was playing on hardest difficulty.

SettingManager Class

This class is responsible for changing the settings like current music,difficulty level according to the user’s preferances

Attributes:

Private currentDifficulty: Type is string.This is the current difficulty that player wants to play the game on.

Private isMusicOn: Type is boolean. This is the attributed that will be used to turn off the music or turn on the music

Private MusicList: Type is File List. Contains the list of music files to be played during the entire game.

Private musicPlayerInstance: This is an Instance of Music Player. Used for Playing the Music.

Operations:

Public turnOffMusic(): If the music is on, this method turns off the music. If the music is off, it displays an error message as “The music is already off”.

Public turnOnMusic(): If the music is off, this method turns on the music. If the music is on, it displays an error message as “The music is already on”.

Public IncreaseDifficulty(): This method increases difficulty by 1 level. There are 3 levels of difficulties on this game. If the player is on the highest difficulty it displays appropriate message and does not change the difficulty level.

Public decreaseDifficulty(): This method decreases difficulty by 1 level. There are 3 levels of difficulties on this game. If the player is on the highest difficulty it displays appropriate message and does not change the difficulty level

Public switchTheMusicToNext(): switches the music to the next music in the music list.

Game Engine Class

This class controls the flow of game by using the appropriate controllers like Map,File,Settings,Battle,Quest etc. Managers. This class is a singleton class so its constructor is private

Attributes:

Private zoneList: This is a List of Zones. Zones will be taken from this list to draw the Maps User will see and act on while he/she is not in battle.

Private Player: This is the main character.

Private currentZone: This is the zone that player is currently displaying and acting on.

Private Instance: This is the GameEngine instance. This attribute is used to avoid creating multiple instances of GameEngine.

SettingsMgr,FileMgr,QuestMgr,CommandInterprtr,CollisionHndlr,MapMgr,BattleMgr,ConsoleRspndr attributes are used for holding the appropriate controllers in the game engine.

Operations:

endGame(): If the player chooses to quit, This method is called for ending the current game.

Move(String direction): This method takes a String parameter “direction” that is taken from the CommandInterpreter Class. Moves the player to the given direction if there is no collision between the cell Player located at and the cell Player wants to go. If there exists a collision, than the appropriate actions is taken according to the Type of the collision.

changeZone(): Uses the attribute zoneList to change the Zone player acting on to the next Zone on the List.

drawMap(): draws the Map that Player should be displaying according to the Player’s status. For example if the Player is battling, it draws the battle map. If the Player is on the zone, it draws the Zone Map.

getInstance(): static method to get the instance of Game Engine. It is essential since Game Engine class is Singleton. If the Game Engine is already created, it returns the current Game Engine. If the game engine has not been created yet, it creates a new Game Engine and returns it.

GameEngine(): private constructor for Game Engine class. It is private because Singleton pattern is applied on this class so that there will not be multiple copies of the game Engine.

startBattle(BattleMap asd): It takes the BattleMap from the BattleStarter class and current Map is changed to the taken BattleMap and Battle is started. Game Engine uses the operations of Battle Manager to operate the flow of the battle.

changeToMapView(): changes the player’s view to the map view So that player can switch to the map view from inventory View easily.

changeToInventoryView(): changes the player’s view to the inventory view So that player can display his/her inventory.

changeToIntroductoryView(): Whenever Player is encountered to a battle, this method is called to display the Icon of the Enemy to the Player.