CS 319 - Object-Oriented Software Engineering

Project Design Report

LOST

GROUP 2-D

Eren Bilaloğlu - 21401603

Gunduz Huseynli - 21402821

Onur Sönmez - 21300749

Yasin İlkağan Tepeli - 21301707

1. Introduction

1.1. Purpose of the System

The purpose of our game, LOST, is to entertain its users. Open world, survival map allows players to interact with various objects, and experience difference scenarios in the game, and decide strategically.

1.2. Design Goals

Maintainability

Game design will be based on object oriented programming concepts which will allow us to implement the game in an extensible manner. Game's subsystems will have hierarchical structure so that new features will be added easily without huge changes in the upper systems. Additionally code will be written in a way that can be easily understood by others.

User - Friendliness

LOST will have user friendly interface design which will enable users to easily grasp the game basics and controls. Considering the human's cognitive abilities, game objects' icons and control icons will have similarities with real life for easy understanding and memorizing. Besides, a help wiki will be available online, accessible from the main menu which will explain the game in a detailed way.

Robustness

In order for our players to enjoy, our game should not have any major bugs that will disrupt the flow of the game. We will achieve this by limiting the user's interactions with the game through user interface, and handling exceptions in a safe manner.

Performance

Players, while playing our game, will not want to wait for long every time they perform a certain action. As a result, we will design the game in a way that player's actions will result in less than 2 seconds on average.

Trade-Offs:

Cost vs Portability

With limited time and energy, we are only implementing the game for desktop operating systems that runs java. As a result, players will not be able to play our game in mobile platforms.

Memory vs Performance

Since our game is designed with object oriented principles, we are not very concerned about memory usage. This allows us to gain run time efficiency in the design, by creating, accessing and changing every object on the go.

Functionality vs Robustness

Being an open world game, with various objects and scenarios it is easy for user to try an invalid combination of actions. However, by carefully detecting and handling exceptions, we are also maintaining the robustness of the code.

2. Software Architecture

2.1. Subsystem Decomposition

We decided use the three tier architecture which fits to design goals of our project. The first layer is user interface layer. Middle layer is the application layer, which includes the game engine and entity objects. Last layer is the data persistency layer, which will store the game data in local and cloud. By implementing a closed architecture, which only allows the layers to access the layers below them, we are making the game more maintainable. Additionally, we limit the subsystems to communicate with each other over a

single interface, so that they can not directly access the subcomponents of other subsystems. This decision favors the maintainability as well as robustness of our game.

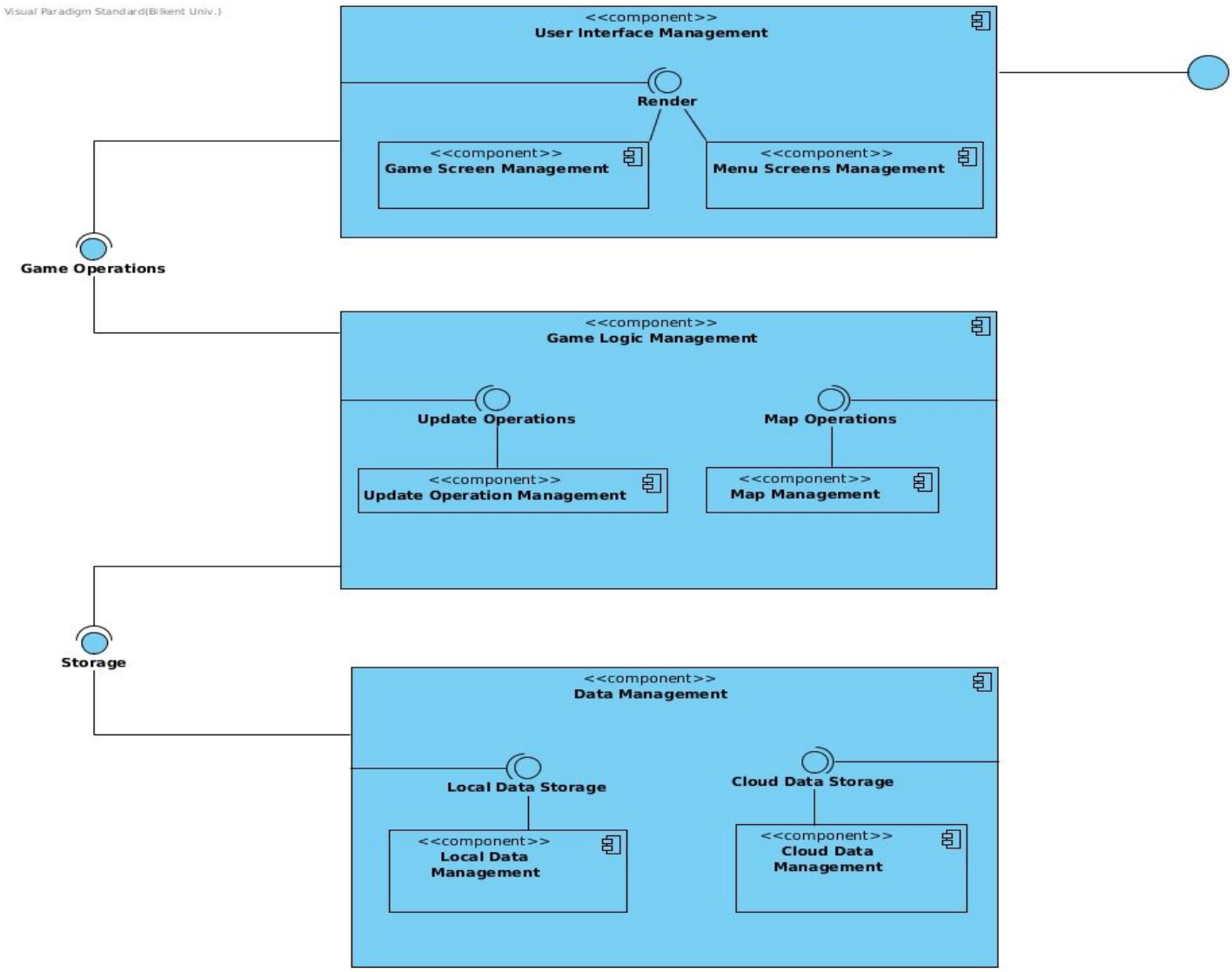


Figure 1: component diagram

2.2. Hardware / Software Mapping

Our game, LOST, does not require any specific purpose built hardware in order to run. Since, the game logic is not that complicated, it can be carried out by the general purpose hardware fast enough. However, in order to play our game, user will need a pc, a keyboard, a mouse, and a monitor. Our

game will run in Linux, Mac, and Windows environments which has Java Runtime Environment(JRA).

2.3. Persistent Data Management

Lost will have both user local data and cloud-based storage. At the initial state, formatted files(.txt) will be parsed and the initial data will be uploaded to the game. The number of items, characters according to their specific areas will be parsed instead of hardcoded implementation. Thus, the environment will be created. After initial data uploaded to the game, there will be two scenarios for saving the game data. First one is keeping local data in a file via parsing back to the files. This will be the default case for the game. If player has an internet connection, game data will also be saved in the cloud using Google App Engine. It will be responsible for recording persistent data into Google Cloud Platform. Instead of traditional relational database systems, highly scalable No-Sql document database system, Cloud Datastore on Java Flexible Environment(Java 8 extension) will be used for restful service. There will be data access objects in this content.

2.4. Access Control and Security

Users' high scores will be stored in an online database and other gameplay data will be stored in local memory. Users will not have access to these databases so that none of the players will have authority to manipulate game data externally. Every player will have equal access to game services. There will not be a user authentication system. User will enter his name when it is asked in the beginning of the game and his score will be displayed with that name when game is over. Therefore, game will not hold any valuable user information such as passwords. Thus, we will not deal with any user data security issues.

2.5. Boundary Conditions

2.5.1. Initialization

For the main menu, game does not require anything. After that, to continue a game by using Load Game option requires a savedFile in the system. If there is no file in system like that or file is corrupted, Load Game option will not be available. Also in order to look at the high

scores, the program needs to access database using network. If there is no available network, that option will not be clickable also.

Additionally, if user tries to open the game when there is already an opened game, There will be pop-up which says “There is already a game in process.” and the second game will not open.

2.5.2. Termination

When user decides to exit from the game while in gameplay, user can click the button “Main Menu” from the top of the screen. When user clicks Main Menu button, the program will ask whether user wants to save or not. Also when user directly clicks the close button from window frame, it will again ask user whether to save or not.

2.5.3. Failure

If the program cannot upload the sounds or images when it is requested, the program will not crash or exit. It will continue to work and let the user decide whether he/she wants to close and open game.

We don’t have any case for instant crashes due to performance issues. None of the system function will work and user will lose current data without saving.

3. Subsystem Services

3.1. Interface Layer

3.1.1. User Interface Management

This component is to display user interface, menus and handling menu and game navigation. It includes two subcomponents: Game Screen Management and Menu Screen Management. It will communicate with the Game Logic Management in the Application Layer to get the game data.

3.1.1.1. Game Screen Management

This component handles the display of game screen which will also display the map.

3.1.1.2. Menu Screen Management

This component’s responsibility is to display the main menu screens.

3.2. Application Layer

3.2.1. Game Logic Management

This component basically controls the gameplay and includes the game objects.

Firstly for control classes, its responsibility is to handle the requests of user interface layer, utilizing game logic, and as a result change the game entity objects. These classes can demand data from the GameObjects in same layer and from the Storage Layer. Then this component will use the data to carry out game logic. Additionally, it will communicate with its subcomponents for relevant services.

Secondly, entity (Game Object) classes will deliver the information requested by the Control classes about the entity objects in our game universe.

3.3. Storage Layer

3.3.1 Data Management

This component is responsible for getting the data from one of its subcomponents, parse or retrieve the data, and return it to the Game Logic Management component accordingly.

3.3.1.1 Local Data Management

This component handles the local data. It parses the game data (excluding the high scores) from the local memory. Parsed data is used from Game Logic Management.

3.3.1.2 Cloud Data Management

In Cloud Data Management, there will be data access objects and they will interact with Google Cloud Platform. Changes in game environment will be transferred by using Cloud Datastore Api.

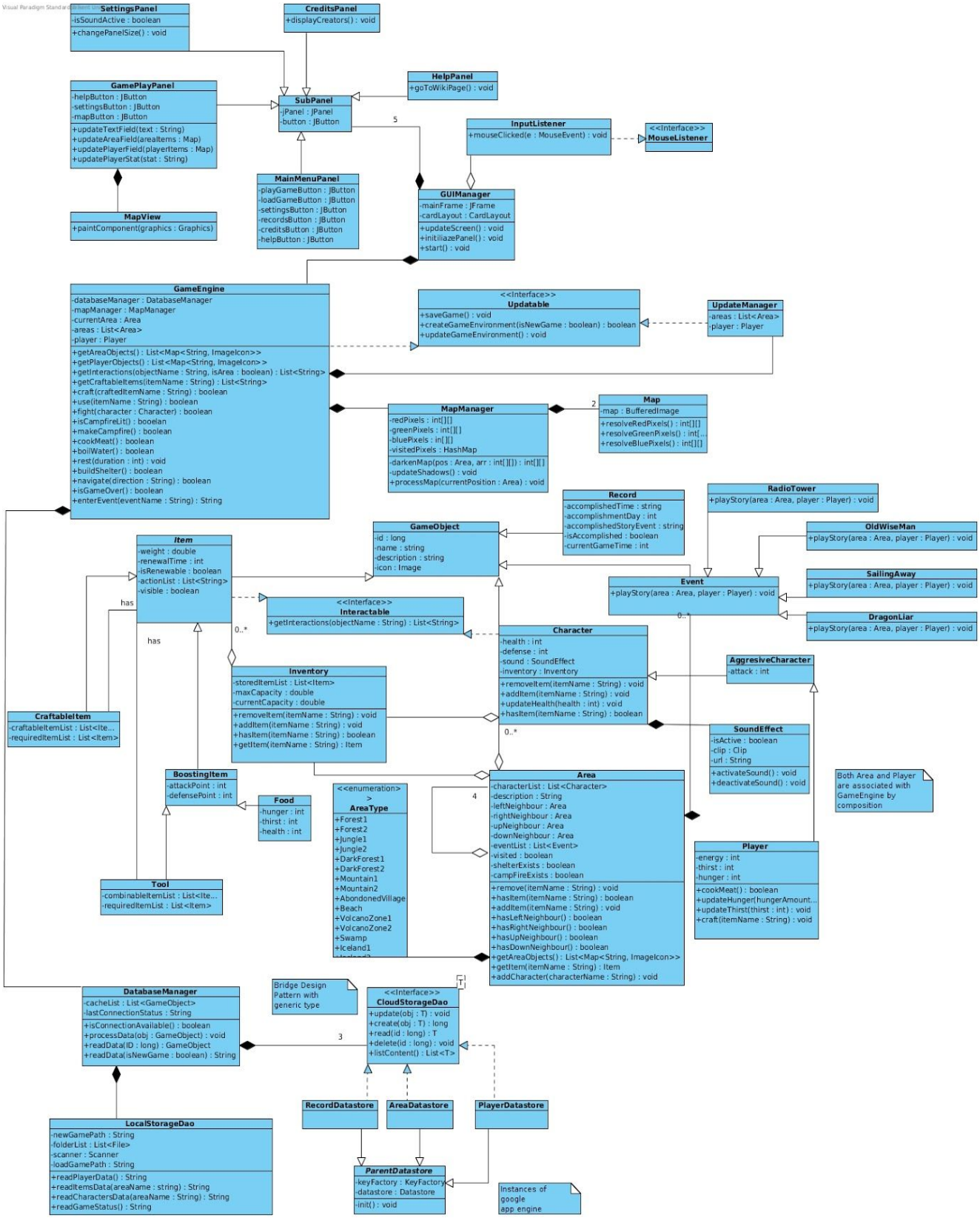
4) Low Level Design

4.1) Object Design Trade-Offs

Maintainability versus Performance: Since we are using facade design pattern in our project, we are sacrificing performance of our system to the maintainability of the code, by forcing each component to interact with each other through single classes.

Memory versus Performance: To use the object oriented design fully, we created and used objects as much as possible. Although this approach allowed us to implement the game, in a more object oriented manner, it also decreased the overall runtime performance.

4.2) Final Object Design



Imgur link to download: http://imgur.com/a/tdCvS

This image is also in the github with the name fulldesign.jpg
(https://github.com/yitepeli/CS319Project_Group2D_LOST)

4.2.1) Facade Design Pattern

In Application Layer, GameEngine class serves as facade class. It provides services for Interface Layer. In Storage Layer, Database Manager class serves as facade class. It provides services for Application Layer.

4.2.2) Bridge Design Pattern

In Storage Layer, generic interface CloudStorageDao is used as a bridge and it accesses the actual implementations of RecordDatastore, AreaDatastore, PlayerDatastore. It enables run-time binding and gives simple usage for database management. Instead of writing separate interfaces for every updatable object, four basic functions of persistent storage CRUD(create, read, update, delete) will be bridged by generic interface.

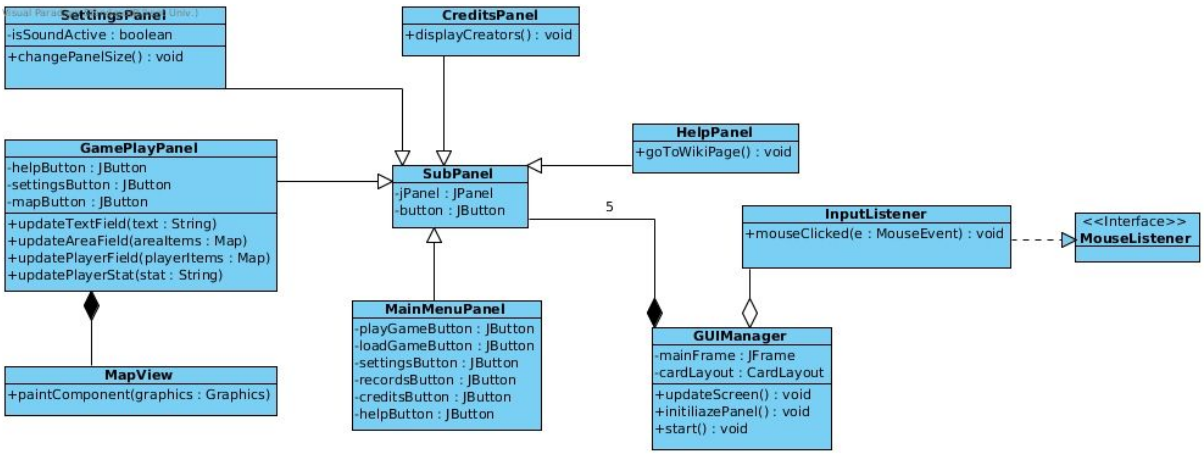
4.2.3) Composite Design Pattern

In Item classes, composite design pattern is used. CraftableItem class and Tool class store lists of Item objects as precondition list, postcondition list inside its implementation.

4.3) Layers

Our game consists of 3 layers, which represents the three layer architecture. Each layer, has only access to the layer below, and all communication in each layer is controlled

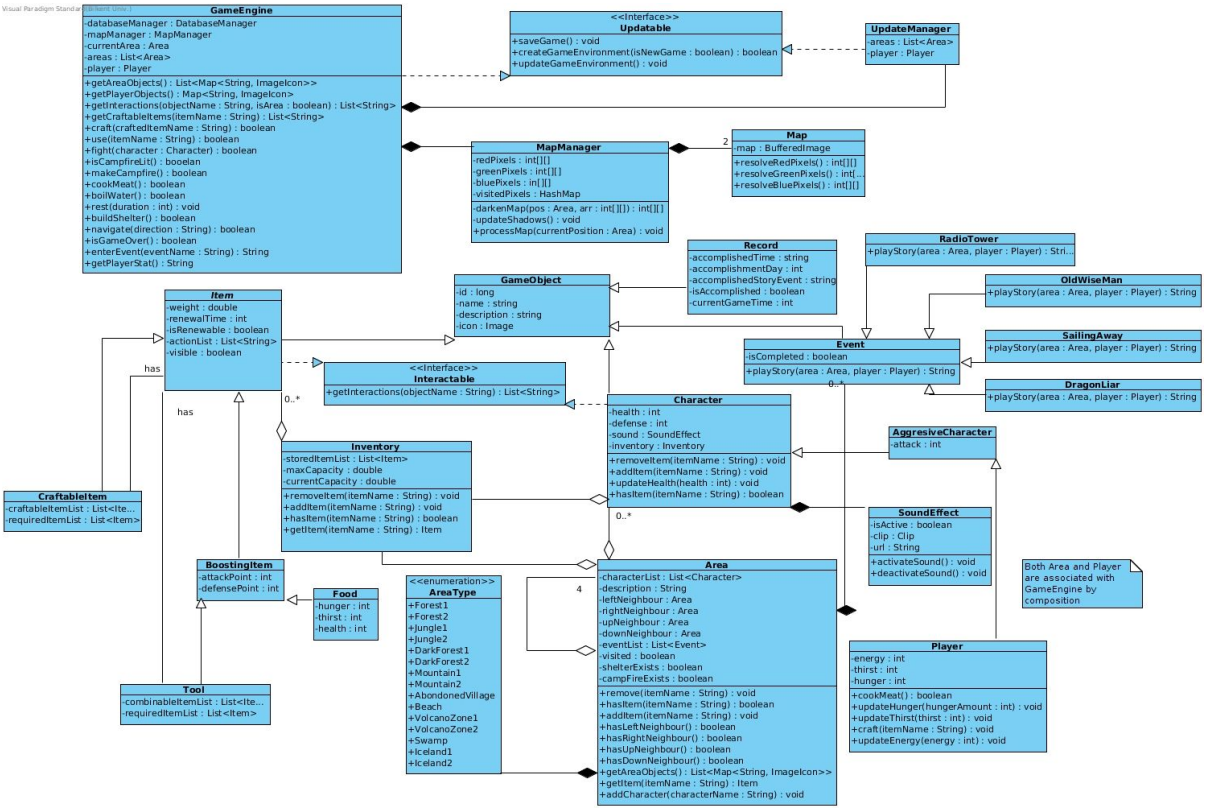
4.3.1) User Interface Management Layer



This layer acts as a boundary object between the user and the application layer. All classes in this layer will be handled by the GUIManager class. Also GUIManager is the only

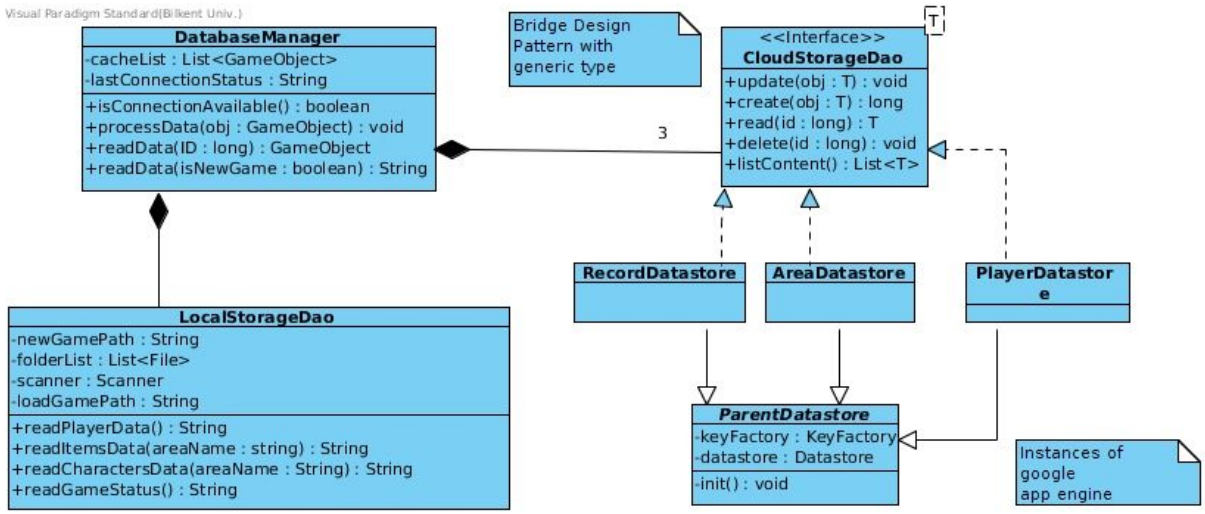
class in this layer that will communicate with the Game Logic Management layer. GUIManager will change the panels according to the user's actions, open up the GamePlayPanel for the main game screen. Also, with each action carried out by the user, GUImanager will gather game information and update the relevant screens.

4.3.2) Game Logic Management Layer



This layer represents the Application Logic layer in our game. The facade class, which is GameEngine class, provides services to the upper User Interface Management Layer. This layer is responsible for getting game data from the below layer, creating the game universe, and handling the requests of the upper layer, and returning relevant information. During these requests, GameEngine class accesses and manipulates the Game Entity objects.

4.3.3) Data Management Layer

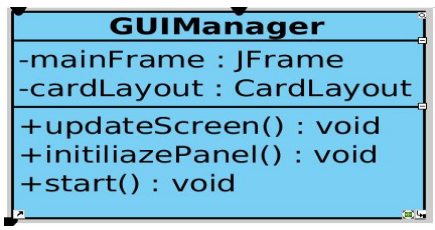


This layer represents the data persistency layer in our game. It is responsible for saving and retrieving the game data into both local and cloud storage.

4.4) Class Interfaces

4.4.1) User Interface Layer Class Interfaces

4.4.1.1) GUIManager



GuiManager will be main layout for other views. It will be main working space and it will both interact with middleware and other gui components.

4.4.1.1.1) Attributes:

Private CardLayout cardLayout: CardLayout will be collection of panels and will supply working panel for main frame.

Private JFrame mainFrame: actual working space

4.4.1.1.2) Operations:

public void updateScreen(): Updates panel accordingly.

public void start(): Starts game panel.

public void initiliazePanel(): Initiliaze the panel,locates the buttons and text areas in panel.

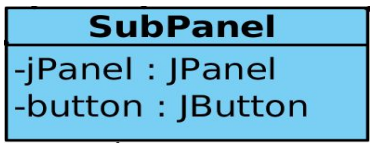
4.4.1.2) InputListener

It will act like a mouse listener. Mouse events will be tracked by InputListener.

4.4.1.2.1) Operations:

mouseClicked(e:MouseEvent): Will override the mouseClicked method of MouseListener.

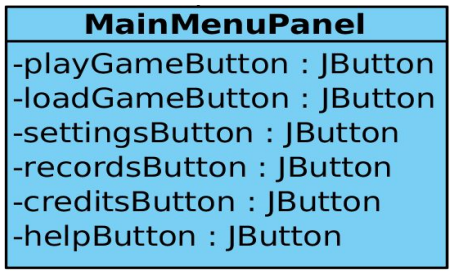
4.4.1.3) SubPanel



It will inherit the JPanel, and button for all subclasses.

4.4.1.3.1) Attributes: JPanel,button

4.4.1.4) MainMenuPanel



Main menu panel will let user to connect subpanels of system.

4.4.1.5) CreditsPanel

4.4.1.5.1) Operations:

public void displayCreators(): It will display information about the creators of Lost in panel.

4.4.1.6) HelpPanel

4.4.1.6.1) Operations:

public void goToWikiPage(): It will direct user to web-base wiki page and user can observe detailed guidelines about the game.

4.4.1.7) SettingsPanel

User can change the settings including the sound mode, panel size via this view.

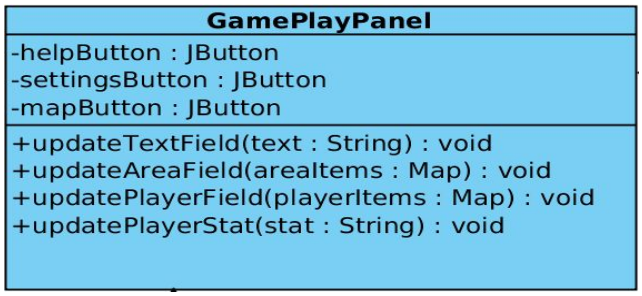
4.4.1.7.1) Attributes:

private boolean isSoundActive: It hols whether the sound effects active on the game or not.

4.4.1.7.2) Operations:

public void changePanelSize(): Will change panel size.

4.4.1.8) GamePlayPanel



4.4.1.8.1) Attributes:

Holds reference for buttons and they will direct panel according to the button clicked.

4.4.1.8.2) Operations:

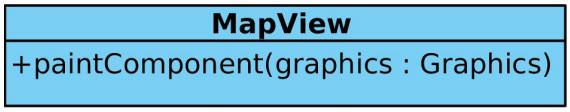
public void updateTextField(String text): Will update the text area

public void updateAreaField(Map areaItems): Updates map in every navigation of user. It processes data accordingly.

public void updatePlayerField(Map playerItems): Updates map in every navigation of user. It processes data accordingly.

public void updatePlayerStat(String stat): Updates map in every navigation of user. It processes data accordingly.

4.4.1.9) MapView

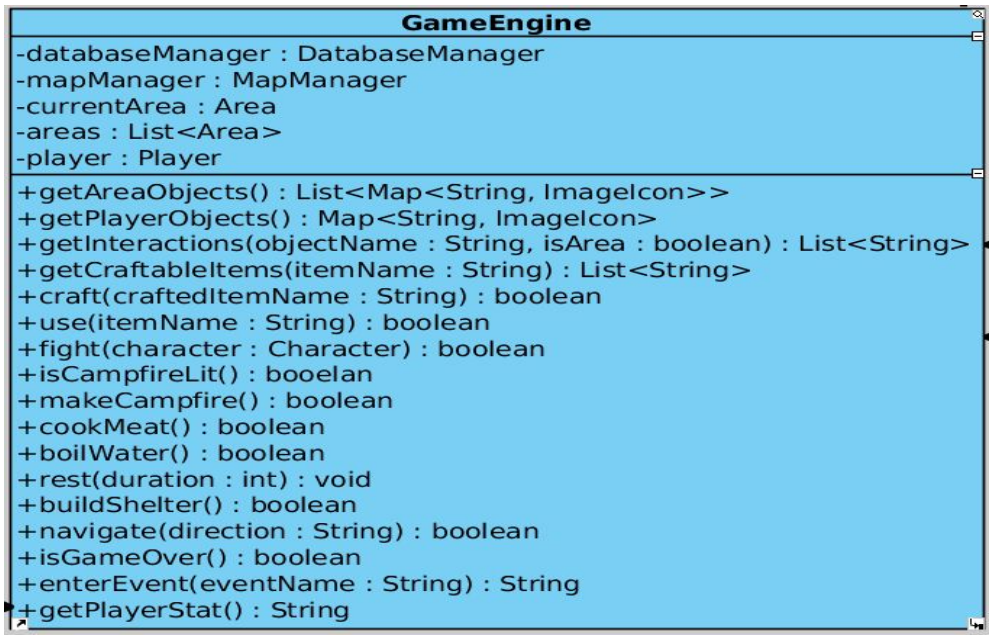


4.4.1.9.1) Operations:

Public void paintComponent(Graphics graphics): It will override ImageObserver interface and will demonstrate the processed map which is managed by Map Manager in panel.

4.4.2) Application Layer Class Interfaces

4.4.2.1) GameEngine (Facade Class)



4.4.2.1.1) Attributes: databaseManager,mapManager,currentArea,player. They are instances of game application layer.

4.4.2.1.2) Operations:

public List<Map<String,ImageIcon>> getAreaObjects(): This operation will retrieve character list,items and events from area. Takes image and name of the objects then creates a list with 3 indexes including event, item and character. Thus the text fields according to their specific areas in GamePanel will be updated accordingly.

public Map<String,ImageIcon> getPlayerObjects(): It will process same operation for player instead of area. Acquires the data in inventory of user and matches item names and icons. It will be reflected for player area text field.

public List<String> getInteractions(String objectName, boolean isArea) : this method returns a list of interactions that can be done with a certain Game Object, given a name of the object as a String. If the object is in the area, and is an item, the only available option is to take the item. In other scenarios GameEngine asks retrieves the information from the object itself.

public List<String> getCraftableItems(String itemName) : this method retrieves the craftable items list of the item in the player's inventory. If the list is empty returns null.

public boolean craft(String craftedItemName) : this method removes the required items that is needed to create new item from the player's inventory. Then adds the new created item to the inventory

public boolean use(String itemName) : this method is for using the items. If the item is a tool, or an armor, user equips the item, and relevant bonus is added to the player's status. If the item is food, the item is consumed and relevant health/thirst or energy bonus is added.

public boolean fight(Character character): this method gets the attacked character, and its attack and defense rates. Then by using these and player's rates, this method changes the healths of the character accordingly. If the attacked character dies, player takes all the items of the character automatically.

public boolean isCampfireLit(): this method simply checks whether there is a already lit campfire in the current area

public boolean makeCampfire(): this method checks whether the player has necessary items to start a fire, and removes these items if the player has them. Also this method changes the fire status in the current area to true.

public boolean cookMeat(): This operation is to cookMeat when there is fire in the area. If player has raw meat and there is fire in the area, it cooks meat and returns true, otherwise returns false. The cooked meat is added to the player's inventory

public boolean boilWater(): This operation is to boil water when there is fire in the area. If player has dirty water and there is fire in the area, it boils the water and returns true, otherwise returns false.

public void rest(int duration): Player rests for the duration time. It calls operation from Player to rest and updates the energy stats.

public boolean buildShelter(): This option simply changes the shelter status of the area to true.

public boolean navigate(String direction): This method changes the current area, with its neighbour in the given direction.

public boolean isGameOver(): It returns true if the game has ended. Game ends if the player reaches 0 health

public String enterEvent(String eventName): This method calls the story event's function, passing it a reference of area and player. Then returns the resulting String to the User Interface Layer. Also if the event has ended, it changes the event's attribute accordingly.

public String getPlayerStat(): It gets the player stats (Health, energy, hunger, thirst) from the Player Object and returns as String.

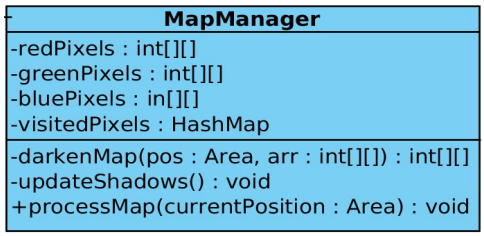
4.4.2.2) UpdateManager

4.4.2.2.1) Attributes:

private List<Area> areas: It has the list of all areas.

Private Player player: It stores the Player as instance.

4.4.2.3) MapManager



4.4.2.3.1) Attributes:

private int[][] redPixels: Keeps red pixels of image as a multidimensional array.

private int[][] greenPixels: Keeps green pixels of image as a multidimensional array.

private int[][] bluePixels: Keeps blue pixels of image as a multidimensional array.

private HashMap visitedPixels: Holds visited pixels in game for shadowing that part of map. It will provide user-specific observation for game.

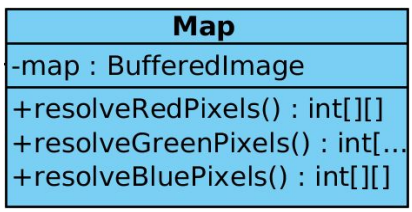
4.4.2.3.2) Operations:

private int[][] darkenMap(Area pos, int[][] arr): It darkens all parts expect from the range that user located.

private void updateShadows(): Shadowing visited parts and shows the user passed this area before.

public void processMap(Area currentPosition): When the user navigate his/her position, this method should be called for updating map according to the current position.

4.4.2.4) Map



4.4.2.4.1) Attributes:

private BufferedImage map: Keeps original image.

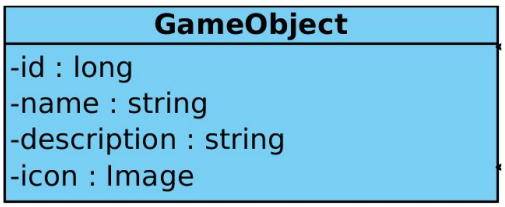
4.4.2.4.2) Operations:

public int[][] resolveRedPixels(): Gets red pixels of image as a multidimensional array.

public int[][] resolveGreenPixels(): Gets green pixels of image as a multidimensional array.

public int[][] resolveBluePixels(): Splits blue pixels of image as a multidimensional array.

4.4.2.5) GameObject



4.4.2.5.1) Attributes:

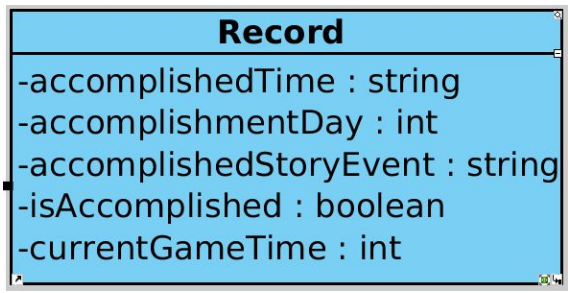
private long id: Every object have a unique id.

private String name: Every object have a name.

private String description: Every object have a description about what use it has..

private Image icon: Objects will have icon to reflect on screen.

4.4.2.6) Record



This class is used for creating record (high score) instances.

4.4.2.6.1) Attributes:

private String accomplishedTime: Real time date in which record is made.

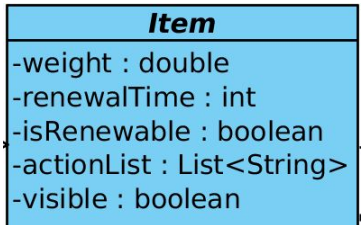
private int accomplishmentDay: Elapsed game time for the record.

private String accomplishedStoryEvent: Holds the title of accomplished story event.

private boolean isAccomplished: Checks if player completed the game or not.

private int currentGameTime: Holds current game time. Value held in this is assigned to accomplishmentDay if player accomplishes a story event.

4.4.2.7) Item



4.4.2.7.1) Attributes:

private double weight: Every item has a weigh. If the total weight of the items exceed inventory limits, player cannot take this item.

private int renewalTime: Some items in the game has a renewalTime which means after they are taken by player, they will disappear from areas but every GameTime%renewalTime==0 situation, they will reappear again.

private boolean isRenewable: It shows whether an item is renewable or not. Some special items are just appears once.

private ArrayList<String> actionList: It holds the actions as string such as use, eat, craft, drop

private boolean visible: It is a value of boolean to indicate whether item will be shown in GUI or not.

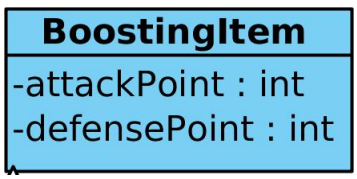
4.4.2.8) CraftableItem

4.4.2.8.1) Attributes:

private ArrayList<String> craftedItemList: This list includes the Items which uses this item to craft themselves.

private ArrayList<String> requiredItemList: This list includes items that is required to craft this Item.

4.4.2.9) BoostingItem



Items that can contribute to the player.

4.4.2.9.1) Attributes:

private int attackPoint: It indicates how many points it will contribute to the players attack.

private int defensePoint: It indicates how many points it will contribute to the players defense.

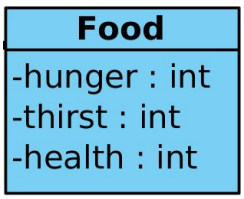
4.4.2.10) Tool

It is BoostingItem which may be crafted by others.

4.4.2.10.1) Attributes:

private ArrayList<Item> requiredItemList: This list includes items that is required to craft this Tool.

4.4.2.11) Food



It is not craftable and it is eatable or drinkable things that reduce the thirst or hunger.

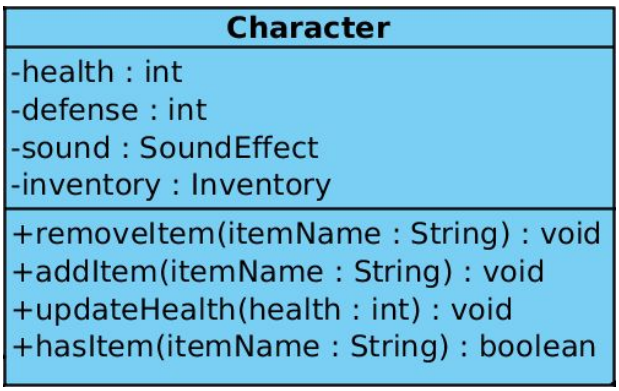
4.4.2.11.1) Attributes:

private int hunger: How much the food will reduce the hunger.

private int thirst: How much the food will reduce the thirst.

private int health: How much the food will change health.

4.4.2.12) Character



4.4.2.12.1) Attributes:

private int health: The attack power of the character.

private int defense: The defense power of the character.

private SoundEffect sound: It is an object of Sound which belongs to character.

private Inventory inventory: It is the Inventory object which has Items in it of Character.

4.4.2.12.2) Operations:

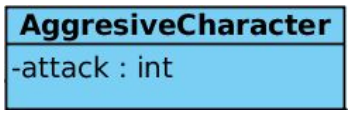
public void removeItem(String itemName): This operation is to remove the items from the character inventory. Inside of it, it calls an operation from Inventory Object.

public Boolean hasItem(String itemName): It calls the Inventory operation hasItem(Item item) to look whether there is the “item” in the inventory.

public void addItems(String itemName): It calls the Inventory operation addItem(Item item) to add the “item” in the inventory.

public void updateHealth(int health): When there is fight or player as a character eats something or there is hunger and thirst for player, it will update the health.

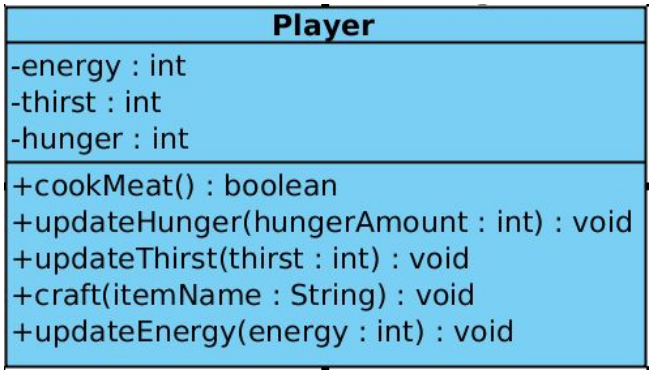
4.4.2.13) AggresiveCharacter



4.4.2.13.1) Attributes:

private int attack: It returns the damage power of character.

4.4.2.14) Player



4.4.2.14.1) Attributes:

private int energy: The amount of thirst the player has.

private int thirst: The amount of thirst the player has.

private int hunger: The defense power of the character.

4.4.2.14.2) Operations:

public boolean cookMeat(): It is an operation to cookMeat if there is a fire near player.

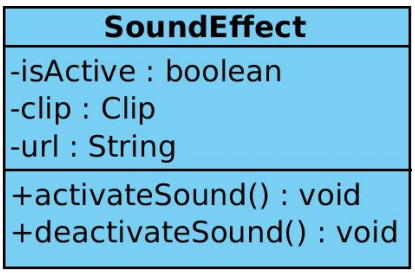
public void updateHunger(int hungerAmount): It is the method for updating the hunger stat of the player.

public void updateThirst(int hungerThirst): It is the method for updating the hunger stat of the player.

public void updateEnergy(int hungerEnergy): It is the method for updating the hunger stat of the player.

public void craft(String itemName): It does the crafting and arrange the items in inventory.

4.4.2.15) SoundEffect



4.4.2.15.1) Attributes:

private boolean isActive: It is true if Sound of the character is active, false if not.

private Clip clip: It is an instance of Java Clip class.

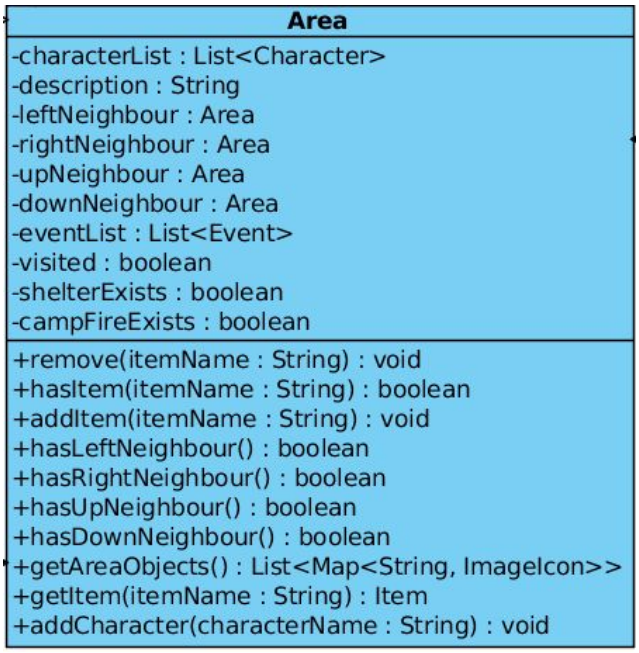
private String url: If it is founded from internet, which is online, there should be the url instance of this Object.

4.4.2.15.2) Operations:

public void activateSound(): It changes isActive to true.

public void deactivateSound(): It changes isActive to false.

4.4.2.16) Area



4.4.2.16.1) Attributes:

private ArrayList<Character> characterList: Holds all the characters who lives in area as ArrayList.

private String description: It is a brief description of the Area such as “Forest is the basic area of the game where has some animals like rabbit etc.”.

private Area leftNeighbour: It holds the area which is on the left of this area. Null if it is a border on map.

- **private Area rightNeighbour:** It holds the area which is on the right of this area. Null if it is a border on map.
- **private Area upNeighbour:** It holds the area which is on the top of this area. Null if it is a border on map.

private Area downNeighbour: It holds the area which is on the bottom of this area. Null if it is a border on map

private ArrayList<Event> eventList: This list holds the Event Objects, which is used to finish the game, in area.

private boolean visited: It shows whether player came to this area before or not.

private boolean shelterExists: It shows whether there is shelter in the area or not. Because player needs shelter to rest and gain energy.

private boolean campFireExists: In order to cook meat, campFireExists must be true.

4.4.2.16.2) Operations:

public void remove(String itemName): It removes the item with name “itemName” from the area inventory.

public boolean hasItem(String itemName): Checks whether there is item with name “itemName” in the inventory.

public void addItem(String itemName): It adds the item with name “itemName” from the area inventory.

public boolean hasLeftNeighbour(): It returns false if there is no area on left.

public boolean hasRightNeighbour(): It returns false if there is no area on right.

public boolean hasUpNeighbour(): It returns false if there is no area on up.

public boolean hasDownNeighbour(): It returns false if there is no area on down.

public List<Map<String,ImageIcon>> getAreaObjects(): It returns a List of Map which includes name and icon of GameObject.

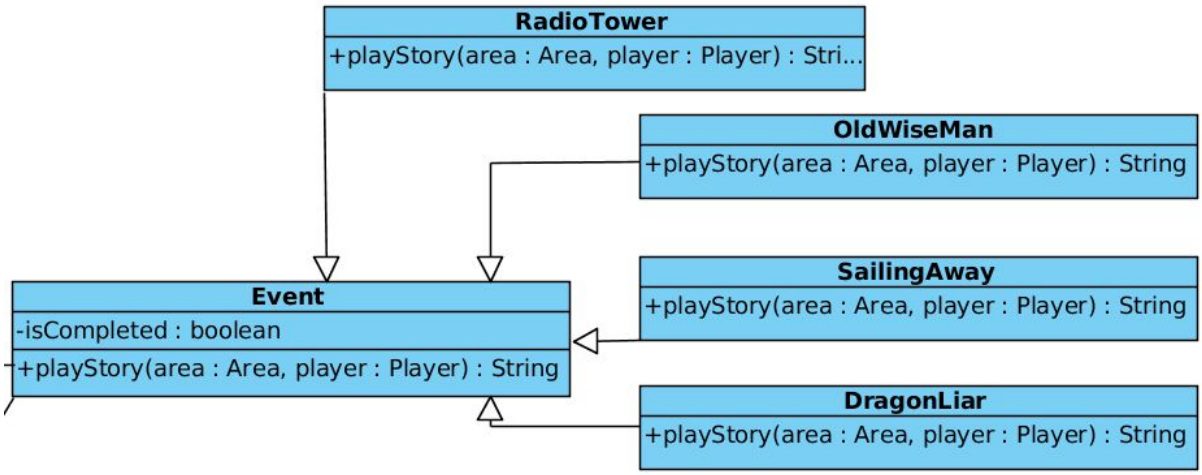
public Item getItem(String itemName): It returns the Item object with the name “itemName”.

public void addCharacter(String characterName): It adds a character to the characterList.

4.4.2.17) AreaType (Enumeration)

All area types are stored as an enumeration for simplicity. Lots of area types are available. So data management is easier that hardcoded integer enums.Also provides better run-time performance.

4.4.2.18) Event



DragonLiar: There is aggregation between Event and this DragonLiar class. It just overwrites the playStory() operation.

OldWiseMan: There is aggregation between Event and this DragonLiar class. It overwrites the playStory() operation. It also has second story in it. Because of that, we have playAltStory() operation.

SailingAway: There is aggregation between Event and this SailingAway class. It just overwrites the playStory() operation.

RadioTower: There is aggregation between Event and this RadioTower class. It just overwrites the playStory() operation.

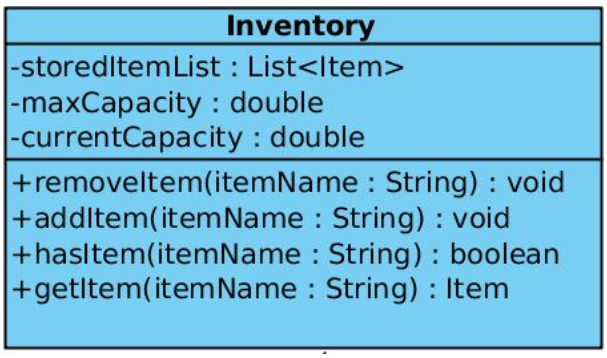
4.4.2.18.1) Attributes:

public boolean isCompleted: It shows whether the event is completed or not.

4.4.2.18.2) Operations:

public boolean playStory(): It includes the stories of the event. If the event is completed it returns true, if it is not completed because of lack of required Items, it will return false.

4.4.2.19) Inventory



4.4.2.19.1) Attributes:

private ArrayList<Item> storedItemList: Holds all the items in the inventory.

private double maxCapacity: Inventory has a limit of totalWeight what we called maxCapacity.

private double currentCapacity: It shows the totalWeight of items which is in the list.

4.4.2.19.2) Operations:

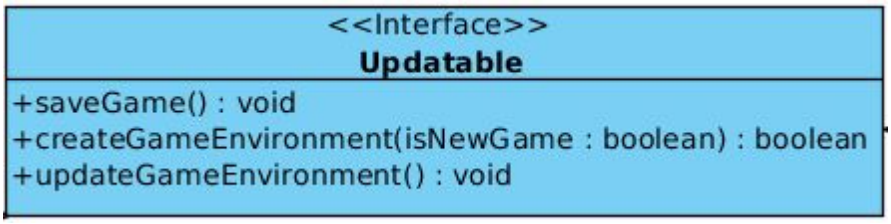
public void removeItem(String ItemName): This operation is to remove an item from the item list in the Inventory.

public boolean hasItem(String ItemName): It look whether in the list there is the “item” object and return true if there is that item.

public void addItem(String ItemName): This operation is to add an item from the item list in the Inventory.

public Item getItem(String ItemName): It returns the Item object with the name “itemName”.

4.4.2.20) Updatable (Interface)



4.4.2.20.2) Operations:

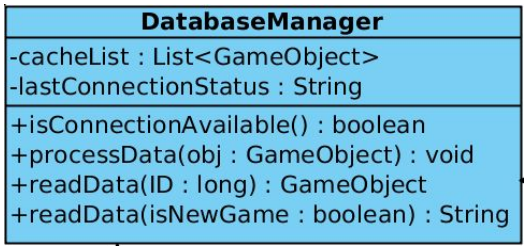
public void saveGame(): Saves the game data into txt file. If there is internet connection, game data and high scores are saved into cloud as well.

public boolean createGameEnvironment(boolean isNewGame): Checks if it should create new game from scratch or load previous game.

public void updateGameEnvironment(): Updates game environment according to game time.

4.4.3) Storage Layer Class Interfaces

4.4.3.1) DatabaseManager



4.4.3.1.1) Attributes:

private List<GameObject> cacheList: Stores last used data in a list in case of internet connection loss.

private String lastConnectionStatus: Holds last connection status.

4.4.3.1.2) Operations:

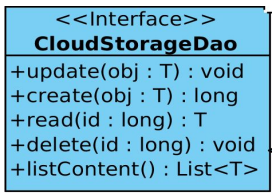
public boolean isConnectionAvailable(): Checks internet connection.

public void processData(GameObject obj): Parses data from text file or cloud.

public GameObject readData(long ID): Returns a game object which is got from cloud.

public String readData(boolean isNewGame): Returns a string data which is parsed from text file.

4.4.3.2) CloudStorageDao (Interface)



It is generic interface which provides all CRUD(create, read, update, delete) operations for database. Also it enables listing all the objects in database. Will be used as a bridge from DatabaseManager to access cloudstore classes.CloudStorage uses CloudDatastore api of Google App Engine. It is a no-sql, super scalable database system and interacts with cloud. Basically, game objects converted to entity objects that is specific object type for cloud database.

4.4.3.2.1) Operations:

public long create(obj: T): It takes generic object types as a parameter. Inside that method it makes object relational mapping for entity object, then sends newly created entity

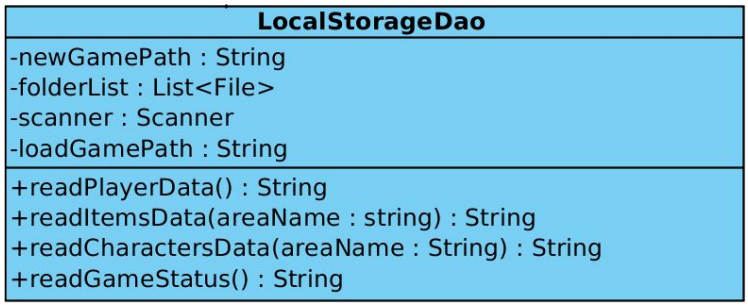
object into cloud datastore.(Note that entity is a type of Google Datastore, not related to the entity objects of game.)

public void update(obj:T): It takes generic object types as a parameter. Updates the entity object which is already created.

public T read(id:long): Seach for entity object in datastore with the id that it has. Then returns it.

public List<T>listContent(): It will return all list content which is underlined under the same keyfactory object. Meanly, takes all objects which have same key.

4.4.3.3) LocalStorageDao



Local storage will be responsible for parsing and reading data from txt files. It will be used for initialing, updating and managing data. User will be capable of playing game with internet connection or without internet connection.

4.4.3.3.1) Attributes:

private String newGamePath: Holds initial game path.

private String loadGamePath: Holds saved game path.

private List<File> folderList: It keeps files object for all components, files of parsing system.

4.4.3.3.2) Operations:

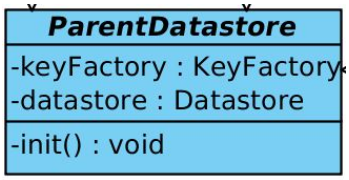
public string readPlayerData(): Returns player data as a string from file.

public string readItemsData(): Returns characters data as a string from file.

public string readCharactersData(): Returns character data in specific area.

public string readGameStatus(): Parses games status.

4.4.3.4) ParentDatastore



4.4.3.4.1) Attributes:

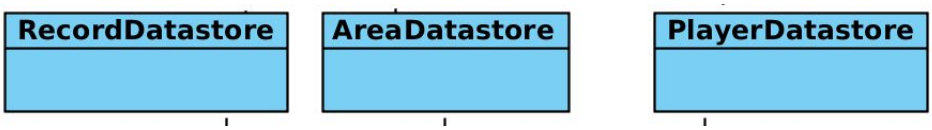
private KeyFactory keyFactory: It is a attribute of google app engine for factoring given data inside the cloud platform.

private Datastore datastore: Datastore is special access object for cloud platform.

4.4.3.4.2) Operations:

private void init(): It is called by constructor to initialize ParentDatastore.

4.4.3.5) RecordDatastore-AreaDatastore-PlayerDatastore



These classes will override interface according to the given format. Detailed information is given in CloudDatastore. RecordDatastore will take Record as a generic type. AreaDatastore will use Area as a generic type and PlayerDatastore will use player as an generic type.