Bilkent University

Department of Computer Engineering

CS 319 Course Project

Group 1C-SS

Design Report

- Alperen Öziş
- Arda Göktoğan
- Cemal Gündüz
- Eren Şenoğlu
- Duygu Nur Yaldız
- Yavuz Faruk Bakman

Supervisor: Eray Tüzün

Contents

1. Introduction

1.1. Purpose of the System

The purpose of this project is to make the extended version of the game Slay the Spire. Slay the Spire is a single-player, turn-based strategy game. The purpose of this game is to complete all the challenges on the map. The player starts the game from the beginning of the map with the chosen character. The map consists of different locations, which are rooms of hostile enemies, resting areas, treasure rooms, unknown rooms or merchants. The player has a deck of cards that can be extended via the merchants, treasures or fights. The deck is used during the turn-based fights. In each turn, several cards are randomly distributed to the player's hand from the deck. The game is over when the player slays the boss at the final room of the map or when the player dies.

Our main concerns in this system's design is reliability, performance, cost, maintainability and modifiability. We use a layered architecture in our design.

1.2. Design Goals

Reliability:

Providing a game with very little likelihood of encountering a bug is one of our main design goals. Throughout the implementation process we are aiming to test for possible bugs by different members of team, separately. For the possible bugs that might be overlooked, we have the possibility of examining and experiencing different cases of our game and had the chance of correcting bugs that we encountered. The fact that our project is a small it is possible to discover and fix bugs by exhaustion. After applying reliability tests, our prior aim is to reach reliability magnitude of 0.95 at least.

Performance:

Although we are designing one of the most basic games, it should provide the user a smooth gaming experience. Thus, we will implement the game in a way that it runs fast and correctly without minimum system requirements. Java is a fast language so it will help us a lot to achieve this goal.

Cost:

We expect our system to be low cost in terms of development time. Since we have limited time for design and implementation of the project, we don't want to have complex design that can not be handled. Also since we have no budget, we don't want to use any software, database or other components that are not free.

Maintainability:

We want our design to be maintainable for new card, relic and potion designs. It should be easy to design your own card with its rules and add to our game. This will provide ability of extension to our game and it will not be boring after a long time of playing if our software is maintainable.

Modifiability:

We want our design to be modifiable. It means that if we need to change a functionality, or add a new one it should be easy to adapt it. If our game is modifiable, making these things will not require to change the current design significantly and will not be much challenging.

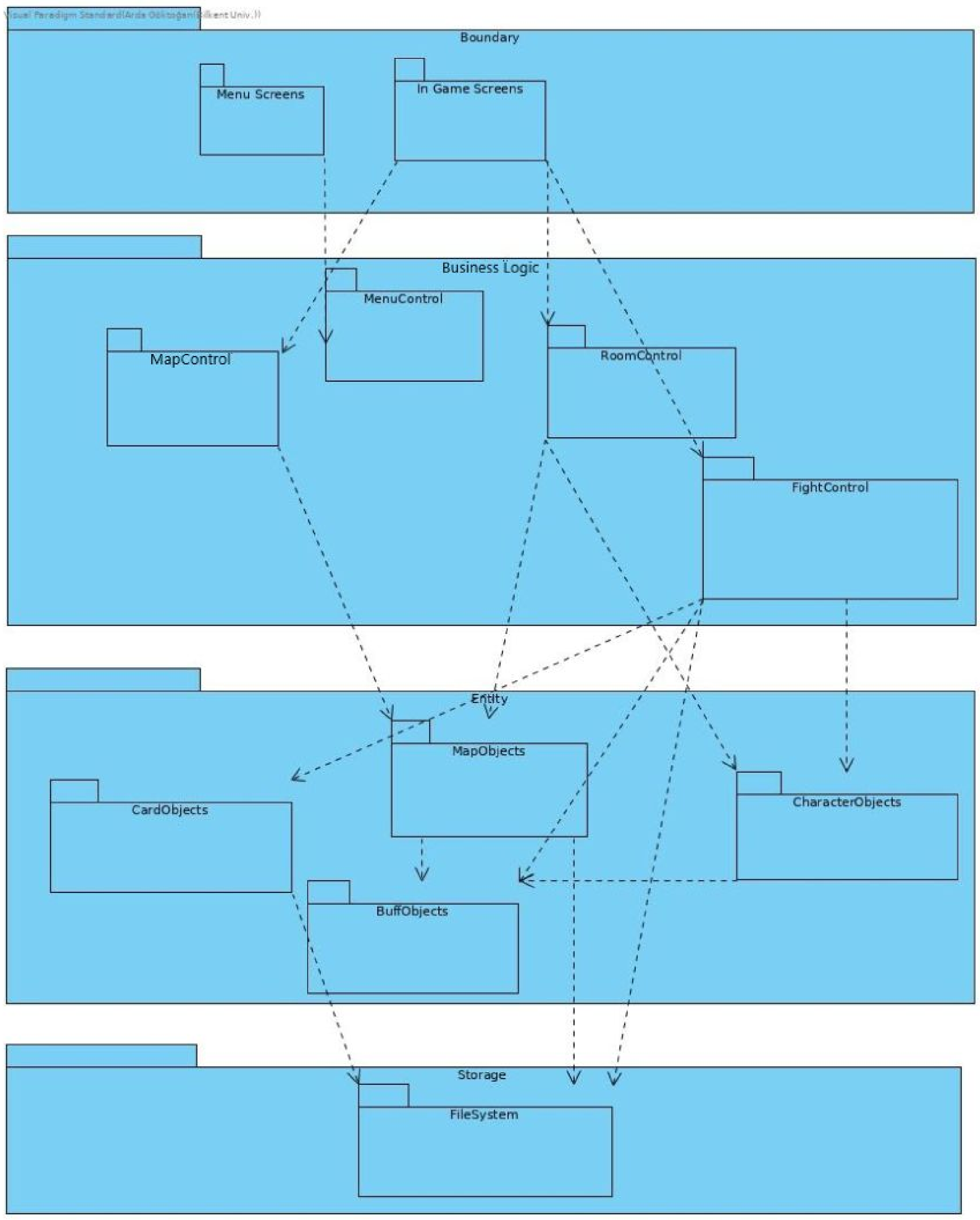
Portability:

We are implementing our game using Java Programming language, this means that Java's JVM will provide us a chance to implement our game in different portals since it is executable on any machine that supports it.

User Friendliness: The system provides a very simple user-interface that every player can understand what is going on. Every components in the game such as cards, relics, potions etc. has explanation, if the player clicks the component, it learns what is the component, how it works etc. Also, the game mechanics does not require any complicated stuff. The player play its movement by easily applying a sequence of simple events. These events are very intuitive such that the player even don't need to think about it. Also, the settings of the game can be easily changed by the user therefore the user can rearrange its name in the game or the window size and therefore it creates a better game experience.

2. High-Level Software Architecture

2.1. System Decomposition



In our system architecture, we used 4 layer architecture. In the first later We have storage classes to communicate with file system to store persistent data to initialize entity objects. Also we have card rules in the file system that are available for the usage of FightControl subsystem. In the second system we have entity objects. In the third layer, we have the business logic. Most of the functionality to play game like traveling around map and actions in the rooms are handled. It consist of 4 subsystems. MenuControl is responsible for menu options like saving game or loading game, starting new game, or changing the user. MapControl is responsible for managing map, traveling of character in the map and directing to the right room when the character wants to advance in the map during the game. RoomControl is responsible for the actions that happens in the Rooms. FightControl is a separate subsystem from RoomControl because complicated events are possible during the fights. To be able to handle all possible interactions of cards,relics and potions; FightControl subsystem will be useful.

2.2. Hardware Software Mapping

Whole program will run on only one system.

2.3. Persistent Data Management

Our game will be offline game so we don’t need online database to store our game data. We will store our data locally by using “.json” files. We are planning to store game saves which is character information and map information as well as players in the game. The reason we use JSON files instead of a database system is we do not modify data multiple times in the game and our game saving so not require complicated structures. JSON files is easy and fast to read and write. Therefore, we decided to use JSON files to keep track of the persistent data.

2.4. Access Control and Security

Since our game is not an online game, it will not connect to the internet. Also, our game does not require any personal information, access control and security is not a major concern for us.

2.5. Boundary Conditions

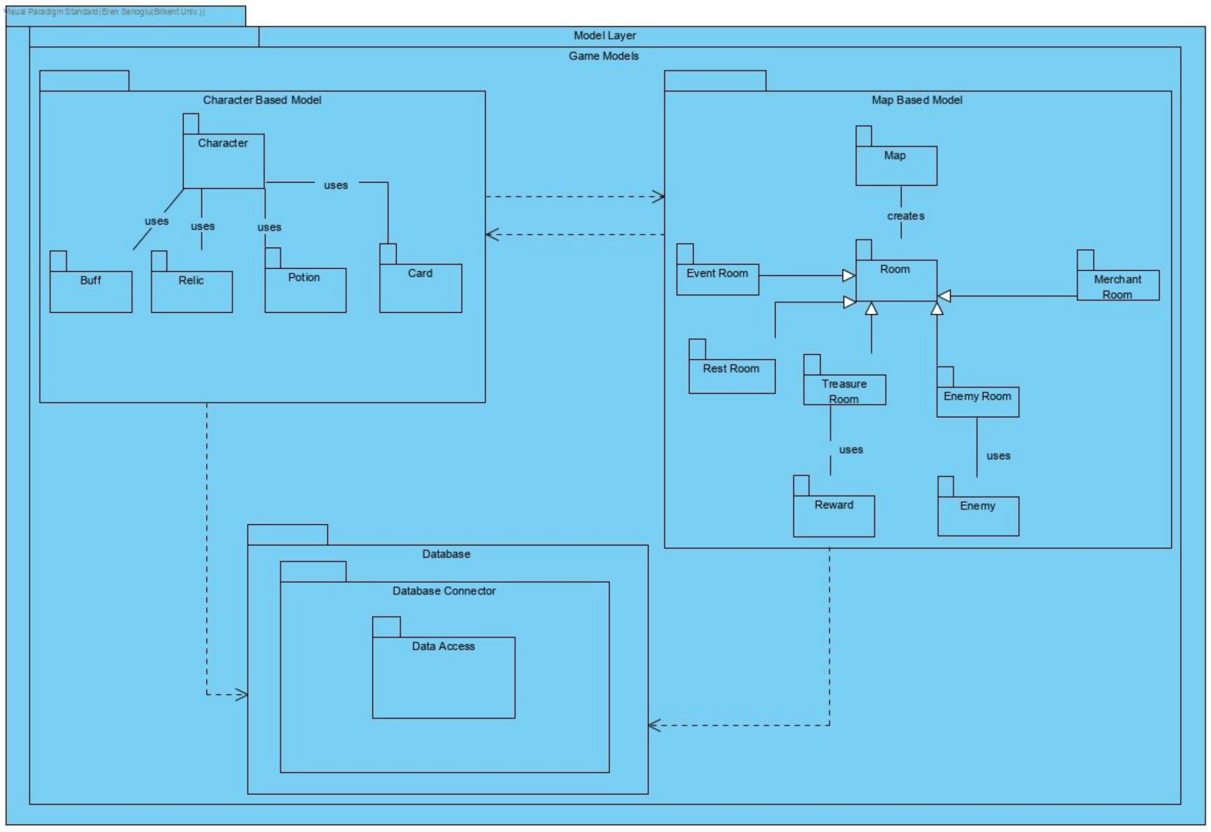
Initialization:
The start of the program is done through the jar file. The initialization of the saved game is done by the main menu buttons. If the player wants to continue with the saved game, the program get contacted with the database and load necessary game documents. Also, the specific objects belonging to the rooms are initialized when the player enters the room in the game.

Termination:
In the game, the objects are deleted after the player exits the current game. Also, specific objects belong to a room are terminated when the player is done with the room. When the program is closed by the user, the system connects with database and automatically save the current situation of the game. To close the system, the user can directly click the cross button on the top or use exit button in the main menu.

Errors:
When there is an error about the initialization of the objects through JSON files the system directly kick the player from the game ask the player to re-enter the game.
If there is a problem about the loading saved game from the database, the program also asks the player to re-click the load saved game.
The errors of the game will be saved to a log file to solve them further.

3. Subsystem Services

3.1. Game Model Subsystem Services

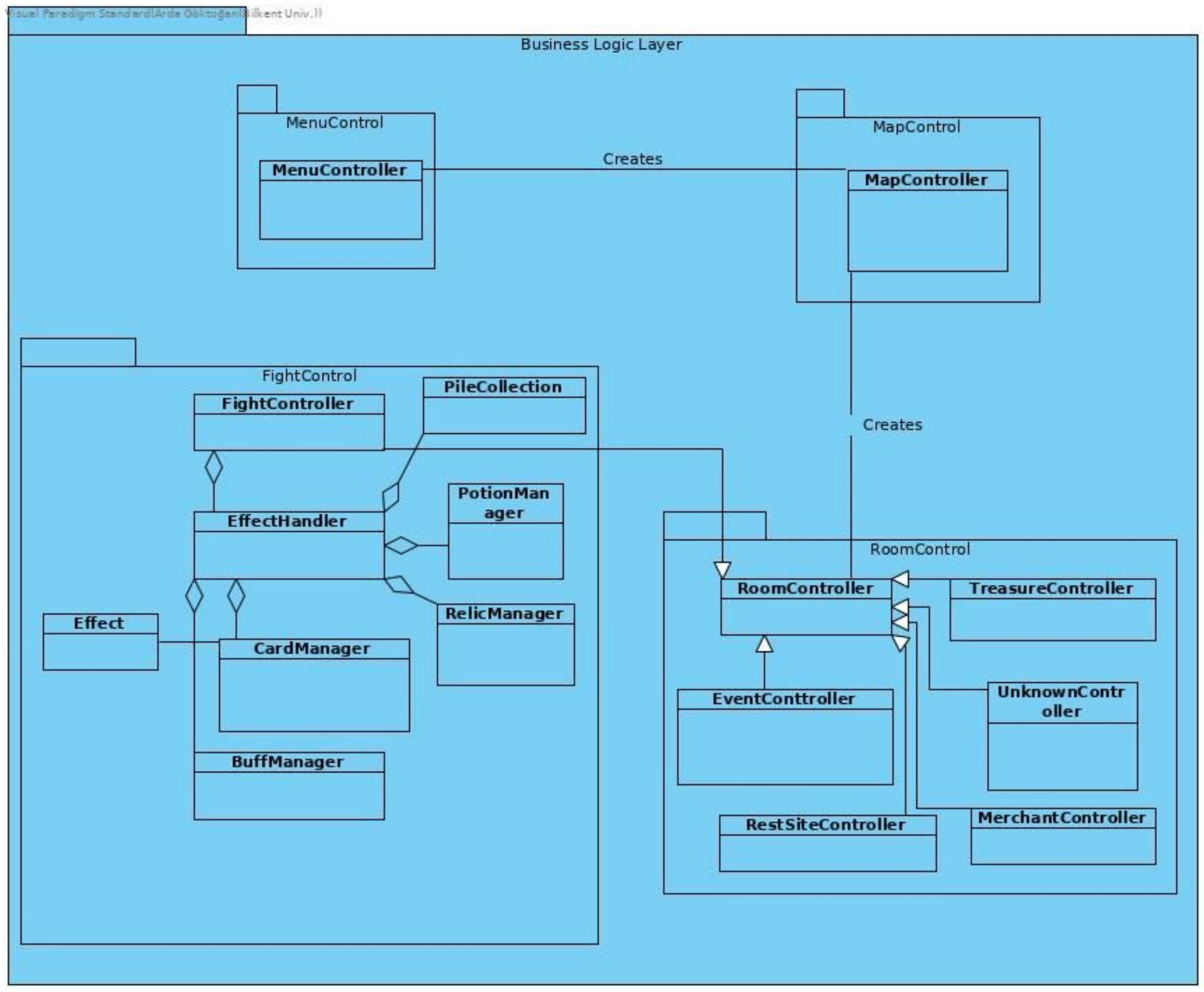


We can examine our model subsystem in two parts.

First main part is character and elements that is associated with character. Character is a core element for our game as the games is based on character's changing cards, relics, potions. Relic, Potion, Pet, Card, and even Buff can be seen as supportive classes for character class' attributes.

Other crucial part in model subsystem is the Map and Rooms. Map includes different rooms in it and we divide these rooms into different classes according to functionality and name. Unknown room can also be mentioned as it is a special room. Because it will be handled by Event class as unknown room is randomly generated and also can be turned into other room models by events.

3.2. Business Logic Layer Subsystems

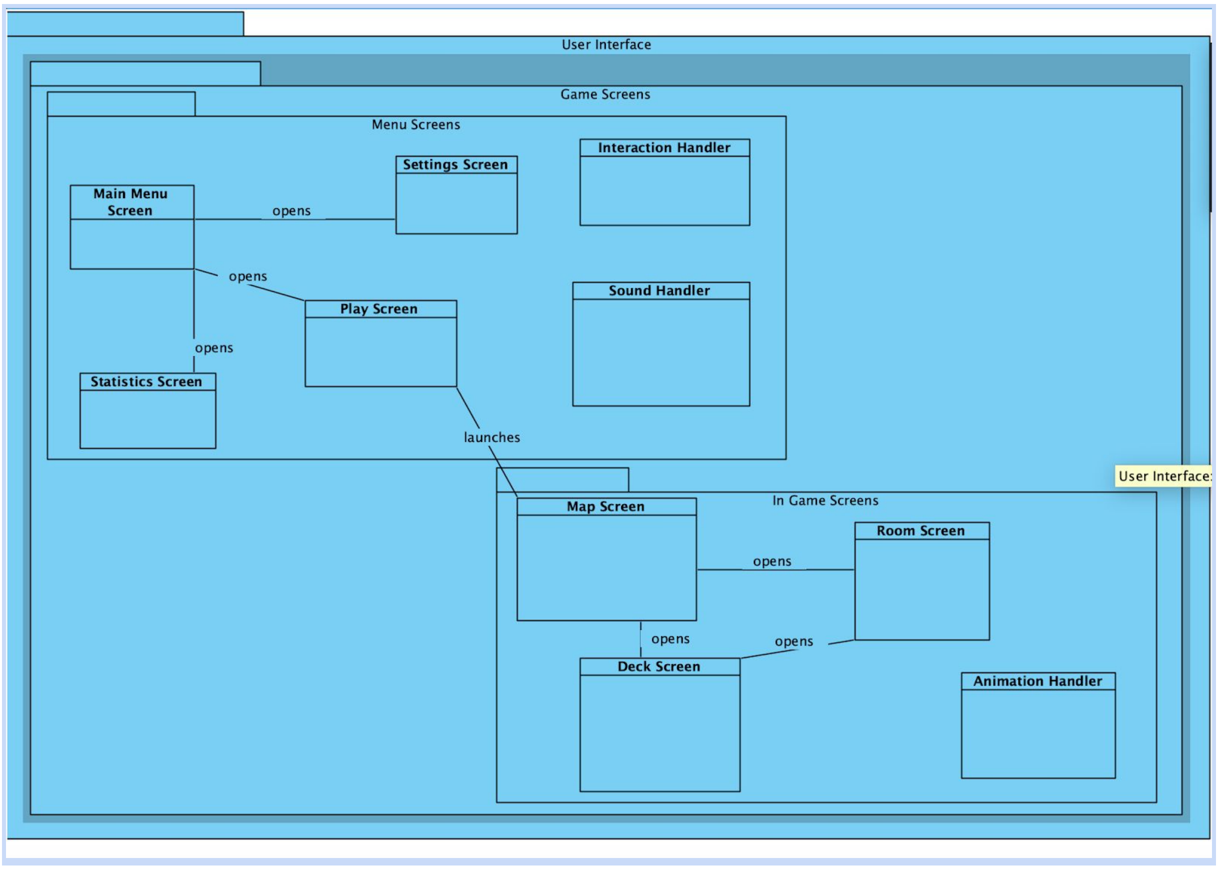


Controller layer consists of 4 subsystems. Menu Control subsystem is mainly for operations that are done in the menu of the game. When the player chooses to play game Map Control subsystem is activated. This subsystem is responsible for the operations done in the map page of the game, such as selecting the next play room.

Room Control subsystem is to handle the different room operations. When a room is chosen from the map, Room Control subsystem is activated. The operations of Treasure Room, Merchant Room, Rest Site and Event Rooms are handled in this subsystem.

Fight Control subsystem is actually a kind of a part of Room Control, however, since it is an important and big part of the game, we put it in a different subsystem. Fight Control subsystem is mainly responsible for the operations done during a combat, such as playing a card, potion or enemies playing.

3.3. View Subsystem



User Interface subsystem is responsible for creating screens according to user input. The game screens can be divided into 2 different subsystems which have basically fundamental different components. The menu screens subsystem handles the screens that happens in where the real game does not start yet. These screens are settings screen, main menu screen, statistics screen. The Interaction handler changes the displayed screen and sound handler is responsible for the background music. From the play screen, the user can launch the In Game Screens subsystem. This subsystem deals with the user interface components that exist in where real game works. The 3 fundamental part of this system are Map, Room and Deck Screen which are 3 main parts of the game screens. The animation handler is responsible for the animation events it these 3 rooms.

4. Low Level Design

4.1. Object Design Trade-offs

Efficiency v. Portability

We choose Portability because Java can run any of architectures and system if required JRE is installed.

Functionality vs. Cost
Since we have limited time until the deadline of the project, we want to keep functions as simple as possible. Core logic of the game will be designed but extra features that are in analysis report can be postponed for the second iteration. Therefore we can reduce the time that is required for software design and implementation.

4.2. Design Patterns

4.2.1. Factory Design Pattern

We used factory design pattern in several parts of the game, such as Card, Relic, Character, Potion and Pet. The reason we use this design pattern is that it makes easier to create the related object. Since there are lots of subclasses of the related parent classes, we think that Factory Design Pattern is a good choice.

4.2.2. Facade Design Pattern

We used the facade design pattern for subsystems that are complex because there are multiple independent classes to deal with. This pattern hides the complexities of the larger system and provides a simpler interface to us, so we used facade design pattern in ,for example, map control and Pile Collection.

Pile Collection Class is responsible for functionalities of Pile classes that are used during the fight. For Example, when player draws card, cards from draw pile are moved to hand pile. If drawPile is empty, cards from discard pile are shuffled and placed to draw pile. PileCollection provides easier interface for these operations.

Map Control class is responsible for dynamics in the map scene. Choosing a room and creating related Room Controller object, selecting the active pet or drinking a potion are done during the map. We collected all this functionalities in this class in order to reduce the complexity.

4.3. Packages

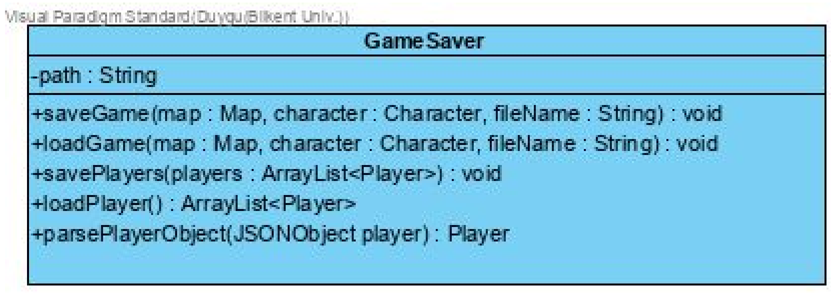
We use JSON files to manage persistent data. In order to achieve this, we use JSON.simple library.
We use JavaFX to construct the GUI of our game. The list of some packages that we will use and their simple explanations is below:

- javafx.scene : Provides the core set of base classes for the JavaFX Scene Graph API
- javafx.animation : Provides the set of classes for ease of use transition based animations
- javafx.application : Provides the application life-cycle classes
- javafx.fxml : Contains classes for loading an object hierarchy from markup.
- javafx.stage : Provides the top-level container classes for JavaFX content
- javafx.event : Provides basic framework for FX events, their delivery and handling.

4.4. Class Interfaces

4.4.1. Database Connection Classes

Game Saver

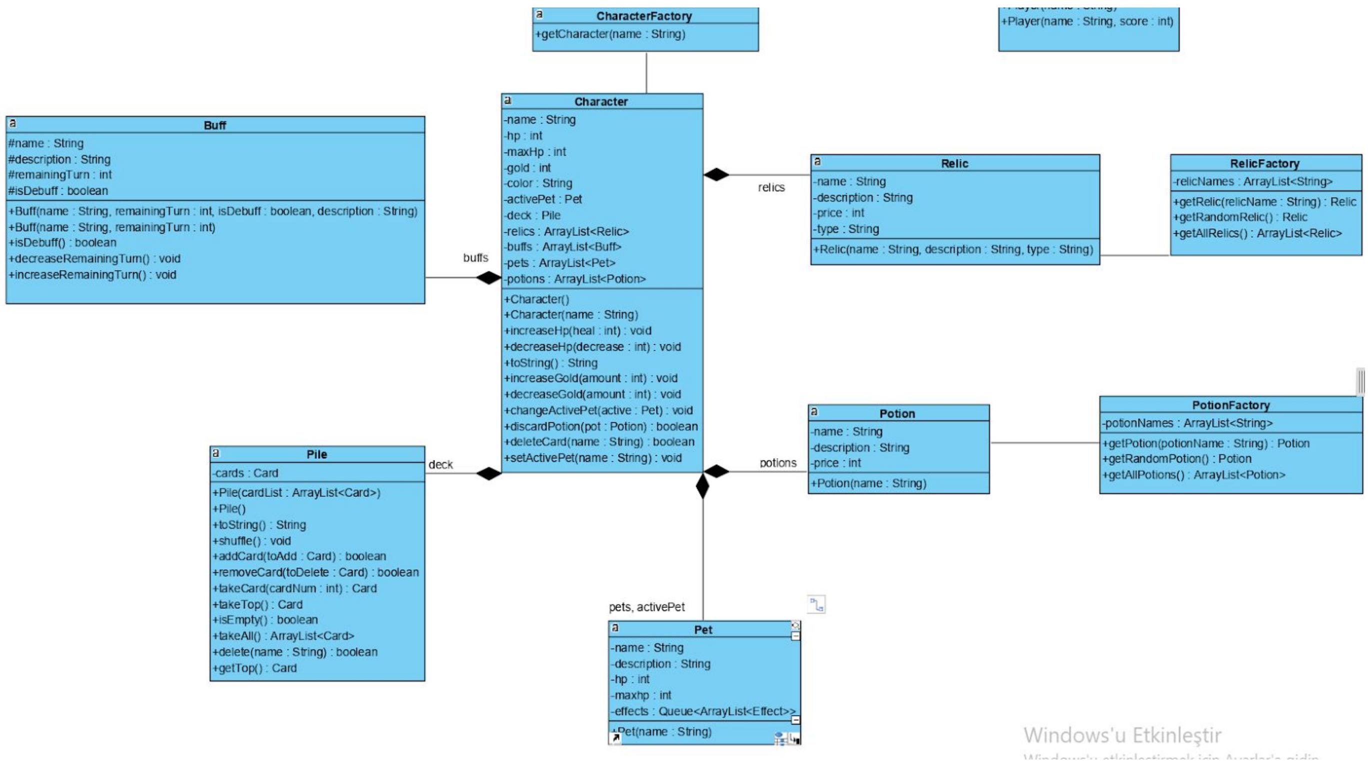


Visual Paradigm Standard(Duygu(Bilkent Univ.))
GameSaver
-path : String
+saveGame(map : Map, character : Character, fileName : String) : void
+loadGame(map : Map, character : Character, fileName : String) : void
+savePlayers(players : ArrayList<Player>) : void
+loadPlayer() : ArrayList<Player>
+parsePlayerObject(JSONObject player) : Player

Game saving and reading saved game information is done by this class. Game saving option basically writes the character and map information to a JSON file. This class also saves the players in the game and reads their information from a JSON file. parsePlayer method is a private method that we use while reading saved players from the file.

4.4.2. Entity Classes

<u>Character Subsystem</u>



CharacterFactory
+getCharacter(name : String)
+Player(name : String, score : int)
Buff
#name : String
#description : String
#remainingTurn : int
#isDebuff : boolean
+Buff(name : String, remainingTurn : int, isDebuff : boolean, description : String)
+Buff(name : String, remainingTurn : int)
+isDebuff() : boolean
+decreaseRemainingTurn() : void
+increaseRemainingTurn() : void
buffs
Character
-name : String
-hp : int
-maxHp : int
-gold : int
-color : String
-activePet : Pet
-deck : Pile
-relics : ArrayList<Relic>
-buffs : ArrayList<Buff>
-pets : ArrayList<Pet>
-potions : ArrayList<Potion>
+Character()
+Character(name : String)
+increaseHp(heal : int) : void
+decreaseHp(decrease : int) : void
+toString() : String
+increaseGold(amount : int) : void
+decreaseGold(amount : int) : void
+changeActivePet(active : Pet) : void
+discardPotion(pot : Potion) : boolean
+deleteCard(name : String) : boolean
+setActivePet(name : String) : void
relics
Relic
-name : String
-description : String
-price : int
-type : String
+Relic(name : String, description : String, type : String)
RelicFactory
-relicNames : ArrayList<String>
+getRelic(relicName : String) : Relic
+getRandomRelic() : Relic
+getAllRelics() : ArrayList<Relic>
potions
Potion
-name : String
-description : String
-price : int
+Potion(name : String)
PotionFactory
-potionNames : ArrayList<String>
+getPotion(potionName : String) : Potion
+getRandomPotion() : Potion
+getAllPotions() : ArrayList<Potion>
deck
Pile
-cards : Card
+Pile(cardList : ArrayList<Card>)
+Pile()
+toString() : String
+shuffle() : void
+addCard(toAdd : Card) : boolean
+removeCard(toDelete : Card) : boolean
+takeCard(cardNum : int) : Card
+takeTop() : Card
+isEmpty() : boolean
+takeAll() : ArrayList<Card>
+delete(name : String) : boolean
+getTop() : Card
pets, activePet
Pet
-name : String
-description : String
-hp : int
-maxhp : int
-effects : Queue<ArrayList<Effect>>
+Pet(name : String)
Windows'u Etkinleştir

Player

Attributes:
-name : String
Holds the player’s name.
-score : int
Holds the score of player.
Methods:
+Player(name:String)
constructor method, sets score to 0.
+Player(name:String, score:int)
constructs an object with given parameters.

Character

Attributes:
-relics : ArrayList<Relic>
List of relics that character has.
-name : String
The name of the character.
-potions : ArrayList<Potion>
List of potions that character has.
-hp : int
Current hit point of character.
-deck : Pile
List of cards that character has which is called Pile.
-maxHP : int
Maximum hit point of character.
-pets : ArrayList<pet>
List of pets character has.
-activePet : Pet
The currently active pet for character.
-gold : int
The amount of gold character has.
-color : string
Color indicator that used for character cards’ and energy orb’s color.
-buffs : ArrayList<buff>
List of currently active buffs on the character.

Methods:
+Character() :
Constructor method.
+Character(name String) :
Constructor method.

+increaseGold(int amount) : void
Increases the gold amount of player by given parameter.
+decreaseGold(int amount) : void
Decreases the gold amount of player by given parameter.
+changeActivePet(Pet active) : boolean
To change the active pet of ccharacter.
+changeHP(int newHP) : void
Change the curent hp of character to parameter.
+changeMaxHP(int maxHP) : void
Set the max HP of character to parameter value
+discardPotiont(Potion pot) : bool
Discard the current potion that character has.
+deleteCard(name:String):void
deletes the wanted card from player’s deck.

Pile

Attributes:
-cards : ArrayList<Card>
Pile holds the array list of cards.

Methods:
+shuffle(): void
Shuffles the player’s card pile.
+addCard(Card toAdd): boolean
Adds a card to player’s pile. Returns true if add operation is successful.
+removeCard(Card toDelete): boolean
Removes a card from player’s pile. Returns true if remove operation is successful.
+takeCard(cardNum : int) : Card
returns the requested card.
+takeTop() : Card
returns the card at the top and removes it from the pile
+takeAll() :ArrayList<Card>
returns all off the pile ad removes the all cards in it.
+getTop() : Card
returns the card at the top of the pile.

Pet

Attributes:

-name:String
holds the name of the pet.
-desctiption:String
holds the description of the pet.

-hp:int
current Health point.
-maxHp:int
maximum Health point.
-block:int
stores the amount of blocked attack
.**-effects: Queue<ArrayList<Effect>>**
stores the effect list with the pattern.

Potion

Attributes:
-name : String
Name of the potion.
-description: String
Description of the potion including effects on character and it's duration.
-price : int
The amount that the potion will be sold in merchant rooms.
Methods:
+Potion(name: String)
constructor method.
+getPrice(): int
Returns the value of price.

Relic

Attributes:
-name : String
Name of the relic.
-description: String
Description of the relic including effects on character and it's duration.
-price : int
The amount that the relic will be sold in merchant rooms.
-type: String

Methods:
+Relic(name : String, description : String, type : String)
constructor method.

PotionFactory

Attributes:
-potionNames: ArrayList<String>
holds all the potion names in the game.

Methods:
+getPotion(potionName: string): Potion
returns the potion with the given name.
+getRandomPotion():Potion
returns a random potion
+getAllPotions():ArrayList>Potion>
returns an arraylist that contains all the potions.

RelicFactory

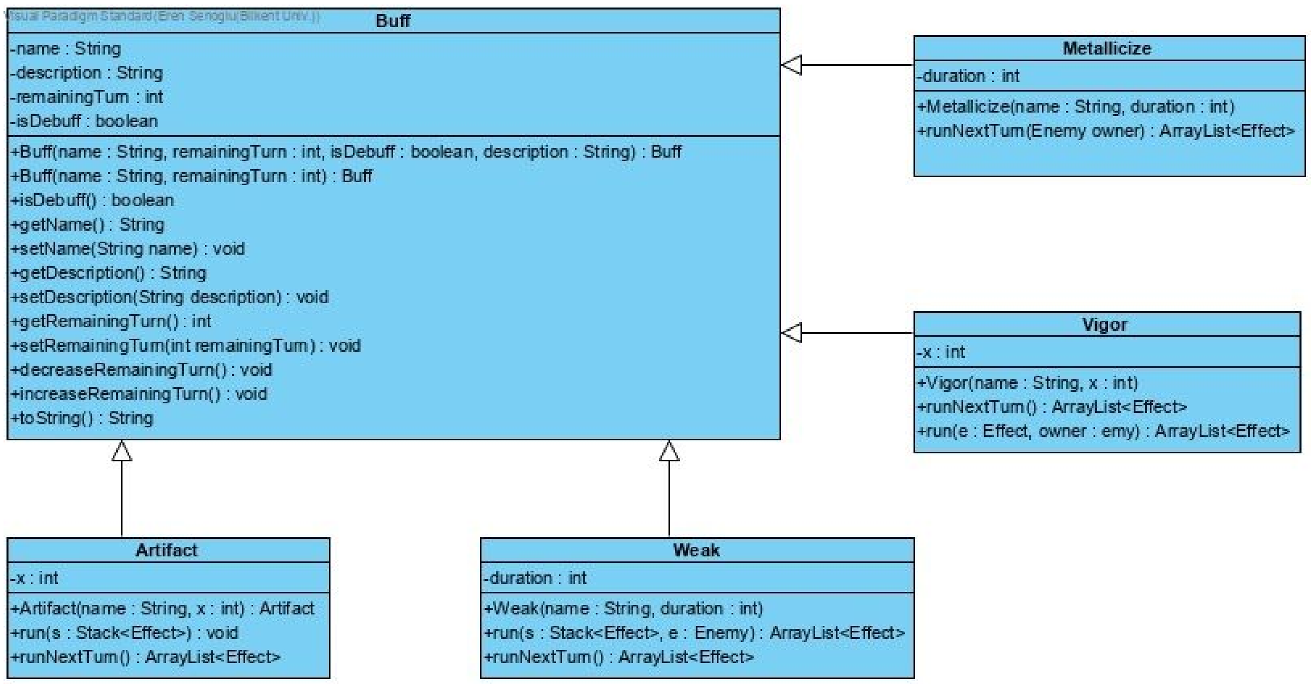
Attributes:
-relicNames: ArrayList<String>
holds all the relic names in the game.

Methods:
+getRelic(relicName: string): Relic
returns the relic with the given name.
+getRandomRelic():Relic
returns a random relic.
+getAllRelics():ArrayList<Relic>
returns an arraylist that contains all the relics.

CharacterFactory

Methods:
+getCharacter(name:String): CHaracter
returns the character with the given name.

Buff Subsystem


Visual Paradigm Standard(Eren Senoglu(Bilkent Univ.))
Buff
-name : String
-description : String
-remainingTurn : int
-isDebuff : boolean
+Buff(name : String, remainingTurn : int, isDebuff : boolean, description : String) : Buff
+Buff(name : String, remainingTurn : int) : Buff
+isDebuff() : boolean
+getName() : String
+setName(String name) : void
+getDescription() : String
+setDescription(String description) : void
+getRemainingTurn() : int
+setRemainingTurn(int remainingTurn) : void
+decreaseRemainingTurn() : void
+increaseRemainingTurn() : void
+toString() : String
Metallicize
-duration : int
+Metallicize(name : String, duration : int)
+runNextTurn(Enemy owner) : ArrayList<Effect>
Vigor
-x : int
+Vigor(name : String, x : int)
+runNextTurn() : ArrayList<Effect>
+run(e : Effect, owner : emy) : ArrayList<Effect>
Artifact
-x : int
+Artifact(name : String, x : int) : Artifact
+run(s : Stack<Effect>) : void
+runNextTurn() : ArrayList<Effect>
Weak
-duration : int
+Weak(name : String, duration : int)
+run(s : Stack<Effect>, e : Enemy) : ArrayList<Effect>
+runNextTurn() : ArrayList<Effect>

Buff

Attributes:

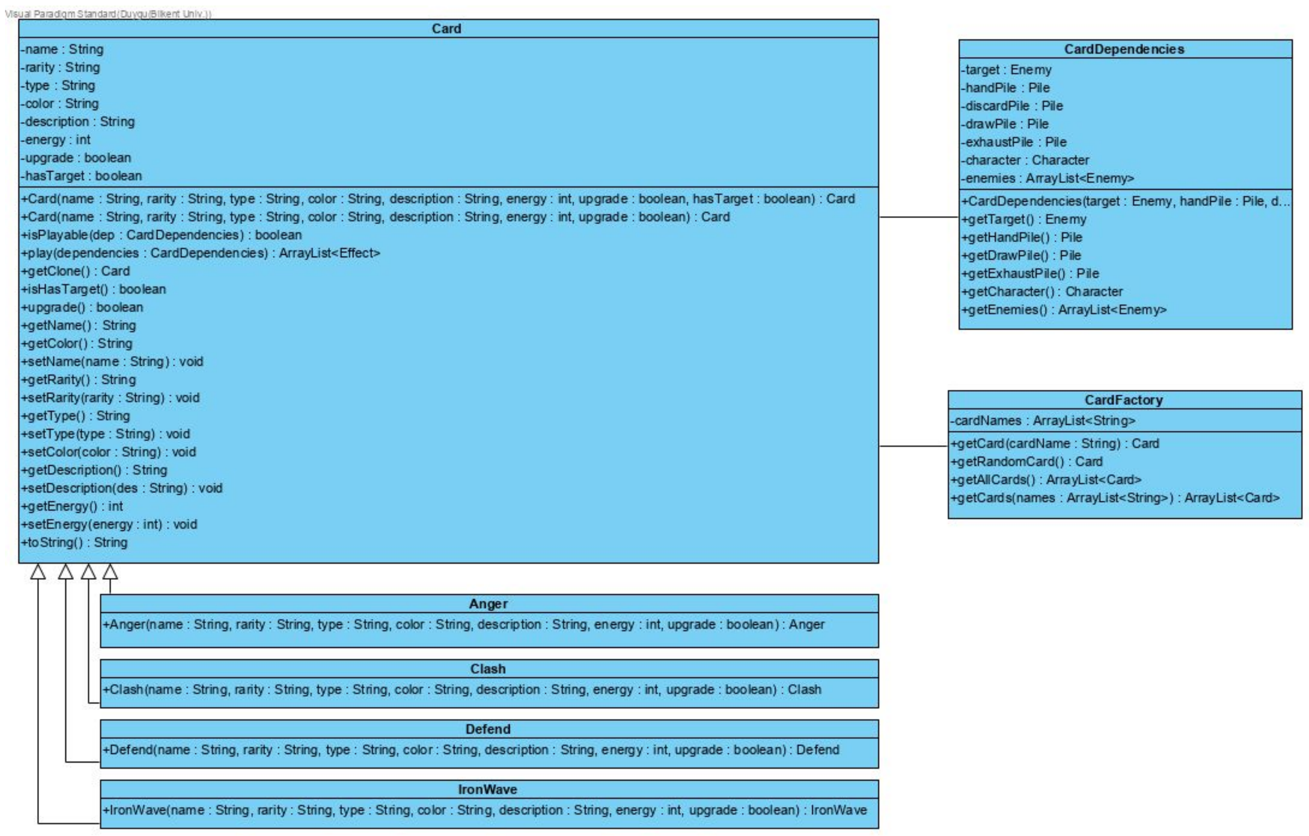
-name : String
Name of the buff.
-description : String
Understandable description of buff's effects.
-remainingTurn : int
Remaining turn to buff to become inactive or disappear completely.
-isDebuff : boolean
Boolean that holds the information of card is being debuff or not.

Methods:

+Buff(String name, int remainingTurn, boolean isDebuff, String description): Buff
Creates a buff object with the given parameters.
+Buff(String name, int remainingTurn): Buff
Overloaded buff constructor for child classes.
+isDebuff(): boolean
Checks if buff is debuff.
+getName(): String
Returns the name of the buff.
+setName(String name): void
Sets the name to the given parameter.
+getDescription(): String
Returns the description of the buff.
+setDescription(String description): void
Sets the description to the given parameter.
+getRemainingTurn(): int
Returns the remaining turns of the buff.
+setRemainingTurn(int remainingTurn): void
Sets the remaining turn to the given parameter.
+toString(): String
Print method for buff objects.
+increaseRemainingTurn(): void
Increases the remaining turns of the buff by one.
+decreaseRemainingTurn(): void
Decreases the remaining turns of the buff by one.

In order to prevent making complicated and complex diagram, we limited the presented subclasses of buff. Vigor, Weak, Artifact and Metallicize generally presents the overall view of the other subclasses. Attributes, methods and constructors of the remaining buff subclasses are analogous to the four subclasses that are represented above.

Card Subsystem


Visual Paradigm Standard(Duygu(Bilkent Univ.))
Card
-name : String
-rarity : String
-type : String
-color : String
-description : String
-energy : int
-upgrade : boolean
-hasTarget : boolean
+Card(name : String, rarity : String, type : String, color : String, description : String, energy : int, upgrade : boolean, hasTarget : boolean) : Card
+Card(name : String, rarity : String, type : String, color : String, description : String, energy : int, upgrade : boolean) : Card
+isPlayable(dep : CardDependencies) : boolean
+play(dependencies : CardDependencies) : ArrayList<Effect>
+getClone() : Card
+isHasTarget() : boolean
+upgrade() : boolean
+getName() : String
+getColor() : String
+setName(name : String) : void
+getRarity() : String
+setRarity(rarity : String) : void
+getType() : String
+setType(type : String) : void
+setColor(color : String) : void
+getDescription() : String
+setDescription(des : String) : void
+getEnergy() : int
+setEnergy(energy : int) : void
+toString() : String
CardDependencies
-target : Enemy
-handPile : Pile
-discardPile : Pile
-drawPile : Pile
-exhaustPile : Pile
-character : Character
-enemies : ArrayList<Enemy>
+CardDependencies(target : Enemy, handPile : Pile, d...
+getTarget() : Enemy
+getHandPile() : Pile
+getDrawPile() : Pile
+getExhaustPile() : Pile
+getCharacter() : Character
+getEnemies() : ArrayList<Enemy>
CardFactory
-cardNames : ArrayList<String>
+getCard(cardName : String) : Card
+getRandomCard() : Card
+getAllCards() : ArrayList<Card>
+getCards(names : ArrayList<String>) : ArrayList<Card>
Anger
+Anger(name : String, rarity : String, type : String, color : String, description : String, energy : int, upgrade : boolean) : Anger
Clash
+Clash(name : String, rarity : String, type : String, color : String, description : String, energy : int, upgrade : boolean) : Clash
Defend
+Defend(name : String, rarity : String, type : String, color : String, description : String, energy : int, upgrade : boolean) : Defend
IronWave
+IronWave(name : String, rarity : String, type : String, color : String, description : String, energy : int, upgrade : boolean) : IronWave

CardDependencies

-target:Enemy

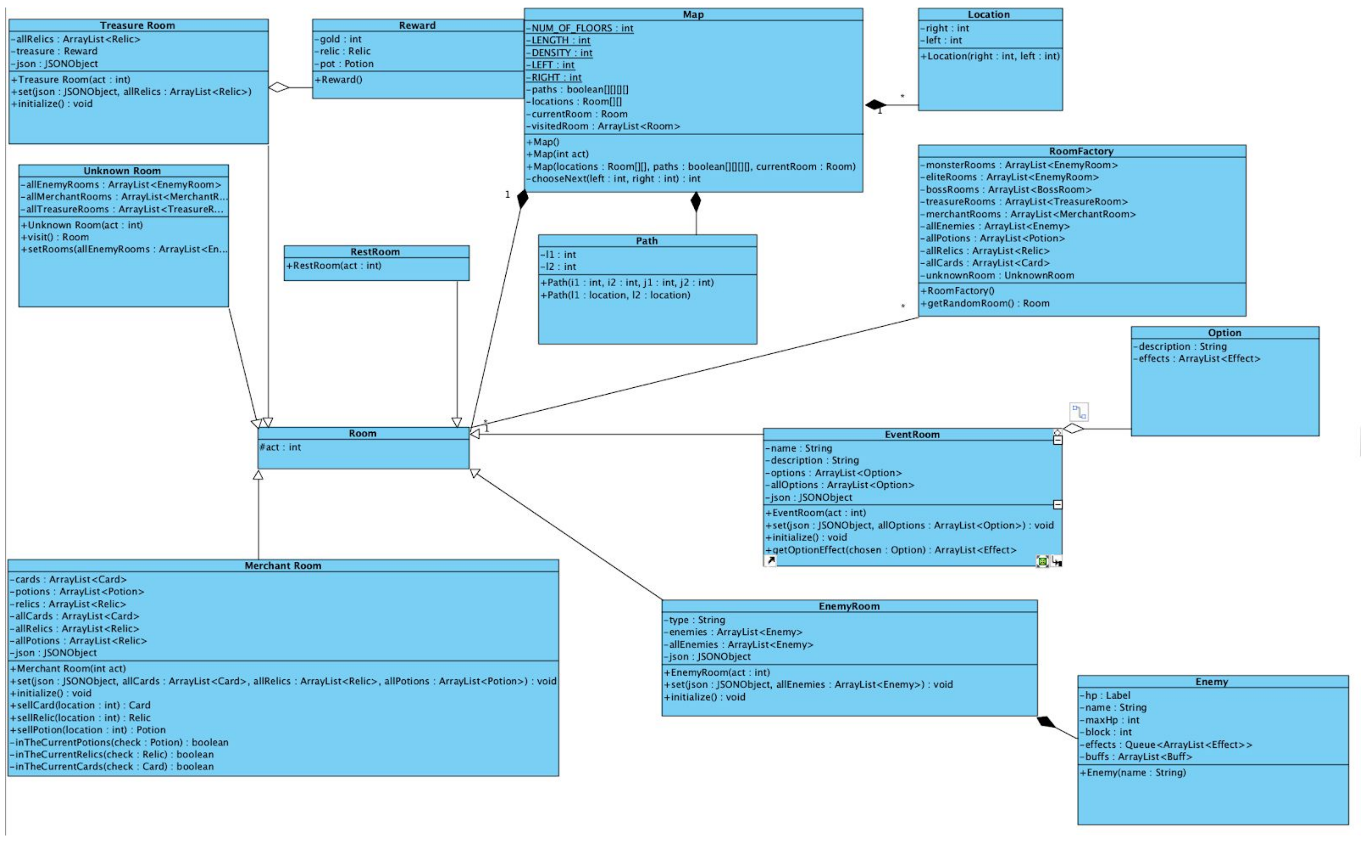
target of the card
-handPile:Pile
cards of the player in the fight scene
-discardPile:Pile
discarded cards pile in the fight scene
-drawPile:Pile
draw pile in the fight scene
-exhaustPile:Pile
exhausted cards pile in the game
-character:Character
-enemies:ArrayList<Enemy>
enemy list in the fight scene

CardFactory
-cardNames:ArrayList<String>
static and final list. holds all the cards name in the game
+getCard(String cardName): Card
return the card object according to the given name
+getRandomCard():Card
return a random card object
+getAllCards():ArrayList<Card>
return a list of the all card objects in the game.
+getCards(ArrayList<String> names):ArrayList<Card>
returns list of the cards whose names are given.

Card
-name : String
Name of the card.
-description : String
Detailed information of how cards work and what is its effect.
-energy : int
Indicator that how much energy is the card cost.
-type : String
Type of the card. A card can be attack, skill, power, status or curse type.
-color:String
color of the card
-hasTarget: boolean
holds if the card has a target or not
+isPlayable(CardDependencies dep): boolean
returns if the cards can be played or not
+getClone():Card
return a new copy of the card
+upgrade():void
upgrades the card

Anger, Clash, Defend and IronWave are some child classes of the parent class Card. Since they have different effects they are implemented as different classes. There are more child classes of the Card class similar to these classes, however, because of the abundance of them, we did not include then in the diagram.

Map Subsystem



Treasure Room
-allRelics : ArrayList<Relic>
-treasure : Reward
-json : JSONObject
+Treasure Room(act : int)
+set(json : JSONObject, allRelics : ArrayList<Relic>)
+initialize() : void
Reward
-gold : int
-relic : Relic
-pot : Potion
+Reward()
Map
-NUM_OF_FLOORS : int
-LENGTH : int
-DENSITY : int
-LEFT : int
-RIGHT : int
-paths : boolean[][][][]
-locations : Room[][]
-currentRoom : Room
-visitedRoom : ArrayList<Room>
+Map()
+Map(int act)
+Map(locations : Room[][], paths : boolean[][][][], currentRoom : Room)
-chooseNext(left : int, right : int) : int
Location
-right : int
-left : int
+Location(right : int, left : int)
*
1
Unknown Room
-allEnemyRooms : ArrayList<EnemyRoom>
-allMerchantRooms : ArrayList<MerchantR...
-allTreasureRooms : ArrayList<TreasureR...
+Unknown Room(act : int)
+visit() : Room
+setRooms(allEnemyRooms : ArrayList<En...
1
RestRoom
+RestRoom(act : int)
Path
-l1 : int
-l2 : int
+Path(i1 : int, i2 : int, j1 : int, j2 : int)
+Path(l1 : location, l2 : location)
RoomFactory
-monsterRooms : ArrayList<EnemyRoom>
-eliteRooms : ArrayList<EnemyRoom>
-bossRooms : ArrayList<BossRoom>
-treasureRooms : ArrayList<TreasureRoom>
-merchantRooms : ArrayList<MerchantRoom>
-allEnemies : ArrayList<Enemy>
-allPotions : ArrayList<Potion>
-allRelics : ArrayList<Relic>
-allCards : ArrayList<Card>
-unknownRoom : UnknownRoom
+RoomFactory()
+getRandomRoom() : Room
*
Option
-description : String
-effects : ArrayList<Effect>
Room
#act : int
*
1
EventRoom
-name : String
-description : String
-options : ArrayList<Option>
-allOptions : ArrayList<Option>
-json : JSONObject
+EventRoom(act : int)
+set(json : JSONObject, allOptions : ArrayList<Option>) : void
+initialize() : void
+getOptionEffect(chosen : Option) : ArrayList<Effect>
Merchant Room
-cards : ArrayList<Card>
-potions : ArrayList<Potion>
-relics : ArrayList<Relic>
-allCards : ArrayList<Card>
-allRelics : ArrayList<Relic>
-allPotions : ArrayList<Relic>
-json : JSONObject
+Merchant Room(int act)
+set(json : JSONObject, allCards : ArrayList<Card>, allRelics : ArrayList<Relic>, allPotions : ArrayList<Potion>) : void
+initialize() : void
+sellCard(location : int) : Card
+sellRelic(location : int) : Relic
+sellPotion(location : int) : Potion
-inTheCurrentPotions(check : Potion) : boolean
-inTheCurrentRelics(check : Relic) : boolean
-inTheCurrentCards(check : Card) : boolean
EnemyRoom
-type : String
-enemies : ArrayList<Enemy>
-allEnemies : ArrayList<Enemy>
-json : JSONObject
+EnemyRoom(act : int)
+set(json : JSONObject, allEnemies : ArrayList<Enemy>) : void
+initialize() : void
Enemy
-hp : Label
-name : String
-maxHp : int
-block : int
-effects : Queue<ArrayList<Effect>>
-buffs : ArrayList<Buff>
+Enemy(name : String)

Map

Attributes:
-LENGTH:int
length of the pap
-paths:boolean[][][][]
this attribute is holds if two room in the map are connected or not. paths[x1][y1][x2][y2] is true if there is a path between rooms whose locations are (x1,y1) and (x2,y2)
-locations:Room[][]
this attribute is to hold the place of the rooms in the map.
-currentRoom:Room
this is the current active room in the map.
-VisitedRoom:ArrayList<Room>
this is the list of the all visited rooms in the map.

Methods:
-chooseNext(left:int,right:int) : int
this is a private method to use when the map is constructed randomly.

Room

Attributes:
#act : int
indicates the act of the room

Treasure Room

Attributes:
-treasure:Reward
Holds the rewards from treasure.
-allRelics:ArrayList<Relic>
Holds all relics to select propers.
-json:JSONObject
Necessary file type to load a treasure room

Methods:
+set(json:JSONObject, allRelics:ArrayList<Relic>):void
sets database file to object to initialize
+initialize():void
It initializes the objects

Merchant Room

Attributes:
-cards : ArrayList<Card>
List of cards that is selled in merchant room.
-potions : ArrayList<Potion>
List of potions that is selled in merchant room.
-relics : ArrayList<Relic>
List of relics that is selled in merchant room.
-allCards : ArrayList<Card>
List of all cards that is selled in merchant room.
-allPotions : ArrayList<Potion>
List of all potions that is selled in merchant room.
-allRelics : ArrayList<Relic>
List of all relics that is selled in merchant room.
-json:JSONObject
Necessary file type to load a merchant room

Methods:
+sellPotion(location:int): Potion
To sell specific potion in the merchant room.
+sellCard(location:int): Card
To sell specific card in the merchant room.
+sellRelic(location): Relic
To sell specific relic in the merchant room.
.

Unknown Room

Methods:
+visit(): void
When unknown room is visited this method will start event process or room change process.

Event Room

Attributes:
-name : String
Name of the event.
-description : String
Understandable detailed information of event.
-options : ArrayList<Option>
Choices that player can choose to determine the effect of event.
-allOptions : ArrayList<Option>
All choices that player can choose to determine the effect of event.
-json:JSONObject

Necessary file type to load a Event room

Methods:
+getOptionEffect(Option chosen): Effect
Returns the effect of the event cause from option that is choosed.

Enemy Room

Attributes:
-roomType : String
Stores the type of the room
-enemies : ArrayList<Enemy>
Stores the list of enemies in the room.
-allEnemies : ArrayList<Enemy>
Stores the list of all enemies .

Room Factory

Attributes:
-monsterRooms : ArrayList<EnemyRoom>
holds all monster rooms
-eliteRooms : ArrayList<EnemyRoom>
holds all elite rooms
-bossRooms : ArrayList<EnemyRoom>
holds all boss rooms
-merchantRooms : ArrayList<MerchantRoom>
holds all merchant rooms
-treasureRooms : ArrayList<TreasureRoom>
holds all treasure rooms

Methods:
+getRandomRoom():Room
returns a random room.

Enemy

Attributes:
-buffs : ArrayList<Buff>
List of buffs that enemy has currently.
-hp:int
current Health point.
-maxHp:int

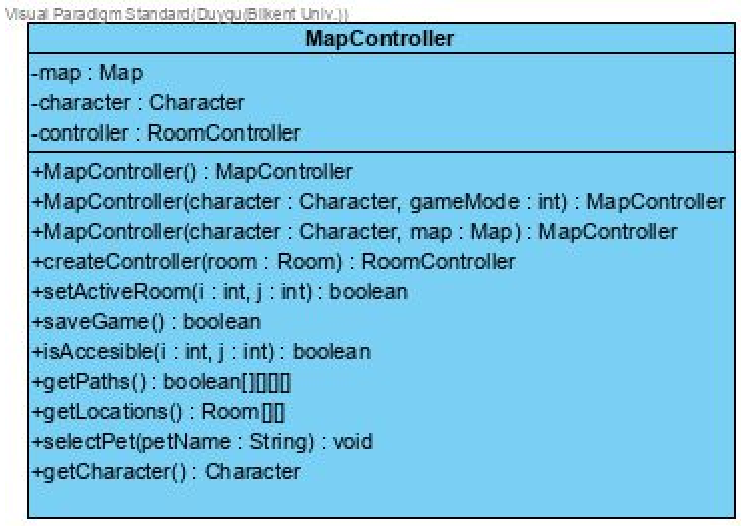
maximum Health point.
-block:int
stores the amount of blocked attack
.**-Queue<ArrayList<Effect>>**
stores the effect list with the pattern
-buff:ArrayList<Buff>
stores the list of buffs

Option

Attributes:
-description : String
holds the description of the object.
-effects:ArrayList<effect>
holds the all effects of the option

4.4.3. Business Logic Layer Classes

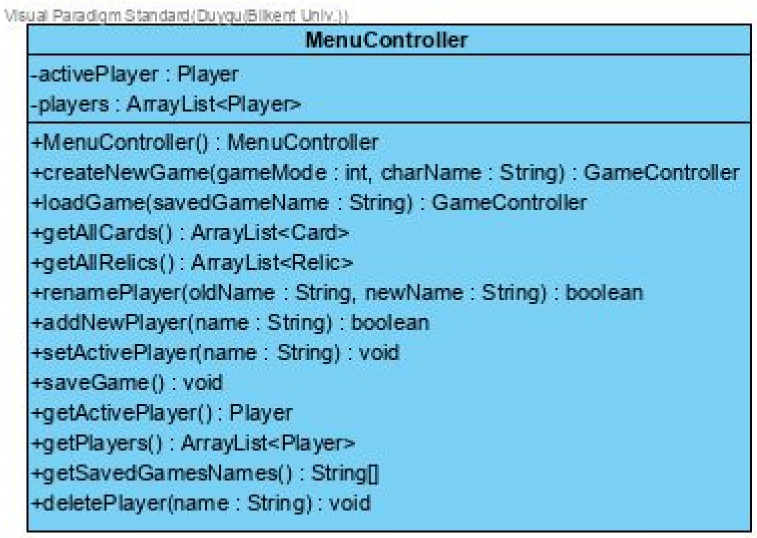
4.4.3.1. Map Control Classes


Visual Paradigm Standard(Duygu(Bilkent Univ.))
MapController
-map : Map
-character : Character
-controller : RoomController
+MapController() : MapController
+MapController(character : Character, gameMode : int) : MapController
+MapController(character : Character, map : Map) : MapController
+createController(room : Room) : RoomController
+setActiveRoom(i : int, j : int) : boolean
+saveGame() : boolean
+isAccesible(i : int, j : int) : boolean
+getPaths() : boolean[][][]
+getLocations() : Room[][]
+selectPet(petName : String) : void
+getCharacter() : Character

MapController
-map: Map of the game
-character: Character that current user is playing.
-controller: This controller is responsible when player enters any room.
+MapController():MapController
default constructor of the class
+MapController(Character character, int gameMode):MapController
creates a new MapController object with a random map according to the gameMode and given character choice.
+MapController(Character character, Map map):MapController
this function is to create a map controller from saved game. it creates a MapController object that has given character and map objects.

+createController(): Controller
this method is called whenever any player visits room and returns the controller that will be used for that room.
+setActiveRoom(int i, int j): boolean
this will set the current room in the map according to the given position.
+saveGame(): void
This method is called if the user saves the game. It will save the game to file including current map situation, cards, relics and potions of the character.
+isAccessible(int i, int j):boolean
it returns if a room in given location can be accessed/clicked or not.
+getPaths():boolean[][][][]
returns the paths between room in the map to show in the GUI.
+getLocations():Room[][]
returns the rooms in the map to show in the GUI
+selectPet(Pet pet):void
changes the active pet of the character to the given pet.

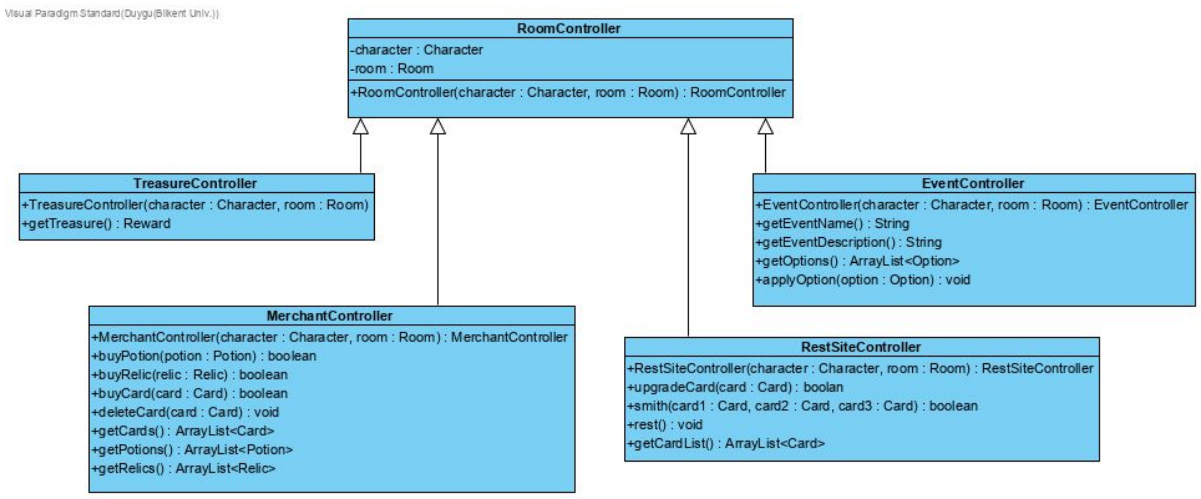
4.4.3.2. Menu Control Classes


Visual Paradigm Standard(Duygu(Bilkent Univ.))
MenuController
-activePlayer : Player
-players : ArrayList<Player>
+MenuController() : MenuController
+createNewGame(gameMode : int, charName : String) : GameController
+loadGame(savedGameName : String) : GameController
+getAllCards() : ArrayList<Card>
+getAllRelics() : ArrayList<Relic>
+renamePlayer(oldName : String, newName : String) : boolean
+addNewPlayer(name : String) : boolean
+setActivePlayer(name : String) : void
+saveGame() : void
+getActivePlayer() : Player
+getPlayers() : ArrayList<Player>
+getSavedGamesNames() : String[]
+deletePlayer(name : String) : void

MenuController
-activePlayer: Player
Active player of the current game
-players:ArrayList<Player>
available player list in the game
+MenuController():MenuController
constructor of the class
+createNewGame(int gameMode, String charName)
creates a new MapController object acorrding to the given parameters.
+loadGame(String savedGameName)
This function loads a saved game.
+getAllCards(): ArrayList<Card>
This function returns the all available cards in the game.
+getAllRelics():ArrayList<Relic>

This function returns the all available relics in the game.
+renamePlayer(String oldName, String newName)
This function is for editing an existing player profile.
+addNewPlayer(String name)
This function is for adding a new player profile to the game
+setActivePlayer(String name)
It sets the active player of the game
+saveGame()
this function saves the game to a file.
+getSavedGameNames():ArrayList<String>
returns the list of the names of the all saved games to show in GUI.
+deletePlayer(String name):void
deletes the given player in the game.

4.4.3.3. Room Control Classes


Visual Paradigm Standard(Duygu(Bilkent Univ.))
RoomController
-character : Character
-room : Room
+RoomController(character : Character, room : Room) : RoomController
TreasureController
+TreasureController(character : Character, room : Room)
+getTreasure() : Reward
EventController
+EventController(character : Character, room : Room) : EventController
+getEventName() : String
+getEventDescription() : String
+getOptions() : ArrayList<Option>
+applyOption(option : Option) : void
MerchantController
+MerchantController(character : Character, room : Room) : MerchantController
+buyPotion(potion : Potion) : boolean
+buyRelic(relic : Relic) : boolean
+buyCard(card : Card) : boolean
+deleteCard(card : Card) : void
+getCards() : ArrayList<Card>
+getPotions() : ArrayList<Potion>
+getRelics() : ArrayList<Relic>
RestSiteController
+RestSiteController(character : Character, room : Room) : RestSiteController
+upgradeCard(card : Card) : boolan
+smith(card1 : Card, card2 : Card, card3 : Card) : boolean
+rest() : void
+getCardList() : ArrayList<Card>

Room Controller
-character: this is reference to current character.
-room: this is reference to current room.

MerchantController
+buyRelic(Relic r): reduces the gold of the character and adds given relic to character’s relics.
+buyPotion(Potion p):reduces the gold of the character and adds given potion to character’s potions.
+buyCard(Card c): void
Reduces the gold of the character and adds given card to character’s pile.
+deleteCard(Card c):void
Reduces the gold of the character and removes given card from character’s pile.
+getCards():ArrayList<Card>
returns the cards for sale.
+getPotions():ArrayList<Potion>
returns the potions for sale.

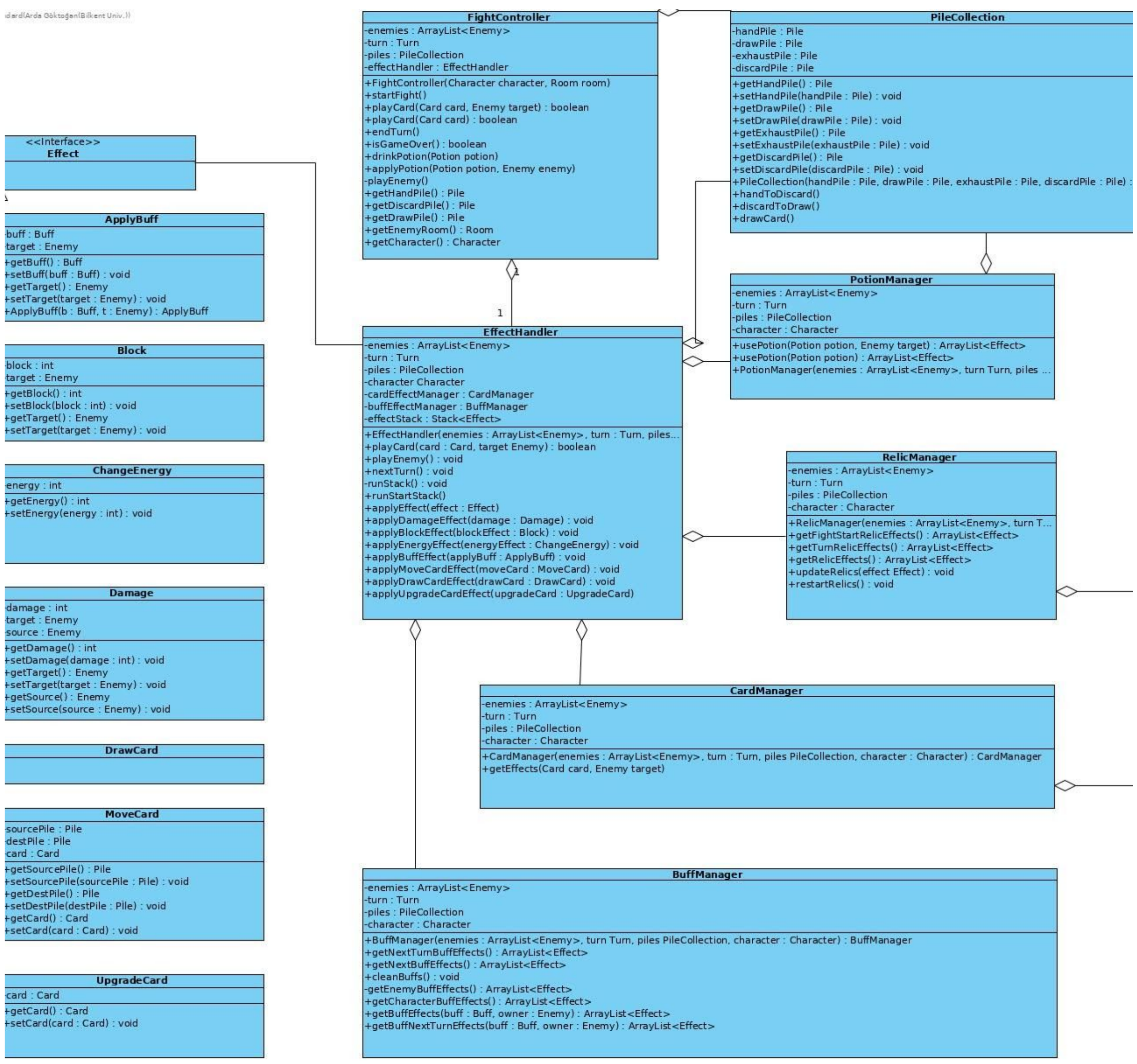
+getRelics():ArrayList<Relic>
returns the relics for sale.

Rest Site Controller
+getCardList(): This will return the cards of the character
+upgrade(Card c): This will be called if the user wants to upgrade card.
+smith(Card c1,Card c2,Card c3): This method takes 3 cards as input, removes them from character's deck and adds a single stronger card to character's pile.
+rest(): This function heals the character.

TreasureController
+getTreasure(): This will return Reward object that character finds from the treasure.

EventController:
This class will be used if unknown room turns into a event room.
+getOptions(): Returns the options of the event.
+applyOption(Option o):This function will apply the chosen option effect to player.
+getEventDescription():Returns the event description.
+getEventName():Returns the event name

4.4.3.4. Fight Control Classes


FightController
-enemies : ArrayList<Enemy>
-turn : Turn
-piles : PileCollection
-effectHandler : EffectHandler
+FightController(Character character, Room room)
+startFight()
+playCard(Card card, Enemy target) : boolean
+playCard(Card card) : boolean
+endTurn()
+isGameOver() : boolean
+drinkPotion(Potion potion)
+applyPotion(Potion potion, Enemy enemy)
-playEnemy()
+getHandPile() : Pile
+getDiscardPile() : Pile
+getDrawPile() : Pile
+getEnemyRoom() : Room
+getCharacter() : Character
PileCollection
-handPile : Pile
-drawPile : Pile
-exhaustPile : Pile
-discardPile : Pile
+getHandPile() : Pile
+setHandPile(handPile : Pile) : void
+getDrawPile() : Pile
+setDrawPile(drawPile : Pile) : void
+getExhaustPile() : Pile
+setExhaustPile(exhaustPile : Pile) : void
+getDiscardPile() : Pile
+setDiscardPile(discardPile : Pile) : void
+PileCollection(handPile : Pile, drawPile : Pile, exhaustPile : Pile, discardPile : Pile) :
+handToDiscard()
+discardToDraw()
+drawCard()
<<Interface>>
Effect
ApplyBuff
-buff : Buff
-target : Enemy
+getBuff() : Buff
+setBuff(buff : Buff) : void
+getTarget() : Enemy
+setTarget(target : Enemy) : void
+ApplyBuff(b : Buff, t : Enemy) : ApplyBuff
Block
-block : int
-target : Enemy
+getBlock() : int
+setBlock(block : int) : void
+getTarget() : Enemy
+setTarget(target : Enemy) : void
ChangeEnergy
-energy : int
+getEnergy() : int
+setEnergy(energy : int) : void
Damage
-damage : int
-target : Enemy
-source : Enemy
+getDamage() : int
+setDamage(damage : int) : void
+getTarget() : Enemy
+setTarget(target : Enemy) : void
+getSource() : Enemy
+setSource(source : Enemy) : void
DrawCard
MoveCard
-sourcePile : Pile
-destPile : Pile
-card : Card
+getSourcePile() : Pile
+setSourcePile(sourcePile : Pile) : void
+getDestPile() : Pile
+setDestPile(destPile : Pile) : void
+getCard() : Card
+setCard(card : Card) : void
UpgradeCard
-card : Card
+getCard() : Card
+setCard(card : Card) : void
1
1
EffectHandler
-enemies : ArrayList<Enemy>
-turn : Turn
-piles : PileCollection
-character Character
-cardEffectManager : CardManager
-buffEffectManager : BuffManager
-effectStack : Stack<Effect>
+EffectHandler(enemies : ArrayList<Enemy>, turn : Turn, piles...
+playCard(card : Card, target Enemy) : boolean
+playEnemy() : void
+nextTurn() : void
-runStack() : void
+runStartStack()
+applyEffect(effect : Effect)
+applyDamageEffect(damage : Damage) : void
+applyBlockEffect(blockEffect : Block) : void
+applyEnergyEffect(energyEffect : ChangeEnergy) : void
+applyBuffEffect(applyBuff : ApplyBuff) : void
+applyMoveCardEffect(moveCard : MoveCard) : void
+applyDrawCardEffect(drawCard : DrawCard) : void
+applyUpgradeCardEffect(upgradeCard : UpgradeCard)
PotionManager
-enemies : ArrayList<Enemy>
-turn : Turn
-piles : PileCollection
-character : Character
+usePotion(Potion potion, Enemy target) : ArrayList<Effect>
+usePotion(Potion potion) : ArrayList<Effect>
+PotionManager(enemies : ArrayList<Enemy>, turn Turn, piles ...
RelicManager
-enemies : ArrayList<Enemy>
-turn : Turn
-piles : PileCollection
-character : Character
+RelicManager(enemies : ArrayList<Enemy>, turn T...
+getFightStartRelicEffects() : ArrayList<Effect>
+getTurnRelicEffects() : ArrayList<Effect>
+getRelicEffects() : ArrayList<Effect>
+updateRelics(effect Effect) : void
+restartRelics() : void
CardManager
-enemies : ArrayList<Enemy>
-turn : Turn
-piles : PileCollection
-character : Character
+CardManager(enemies : ArrayList<Enemy>, turn : Turn, piles PileCollection, character : Character) : CardManager
+getEffects(Card card, Enemy target)
BuffManager
-enemies : ArrayList<Enemy>
-turn : Turn
-piles : PileCollection
-character : Character
+BuffManager(enemies : ArrayList<Enemy>, turn Turn, piles PileCollection, character : Character) : BuffManager
+getNextTurnBuffEffects() : ArrayList<Effect>
+getNextBuffEffects() : ArrayList<Effect>
+cleanBuffs() : void
-getEnemyBuffEffects() : ArrayList<Effect>
+getCharacterBuffEffects() : ArrayList<Effect>
+getBuffEffects(buff : Buff, owner : Enemy) : ArrayList<Effect>
+getBuffNextTurnEffects(buff : Buff, owner : Enemy) : ArrayList<Effect>

FightController

-enemies : ArrayList<Enemy>
This attributes keeps enemies during the fight.
-turn : Turn
keeps turn number
-piles : PielCollection
Keep the discard,draw and other fight related piles in a collection with their functionalities.
-effectHandler : EffectHandler
Other controller class for fight functionalities. FightController class uses it for managing the interactions between enemies and character.

+FightController(Character character, Room room)
Creates FightController object with given character and given room enemies.
+startFight()

Prepares hand pile of the character and interacts with EffectManager class for applying the game start effects.

+playCard,+playCard(Card card, Enemy target) : boolean
returns true if card is playable, false otherwise.
Applys the given card to given enemy.

+playCard(Card card) : boolean
returns true if card is playable, false otherwise. This method is used when card has no target.

+endTurn() : void
called when player presses end turn button. Changes the turn and calls the playEnemy() function.

+isGameOver() : boolean
returns true if game is over, false otherwise

+drinkPotion(Potion potion) : void
calls effectHandler methods for applying the effects of drinking given Potion.

+drinkPotion(Potion potion, Enemy enemy) : void
calls effectHandler methods for applying the effects of drinking given Potion to given Enemy.

-playEnemy() : void
calls effectHandler methods to apply enemy effects to character or enemies.

PileCollection
-handPile : Pile
Hand Pile of character

-drawPile : Pile
DrawPile of character

-exhaustPile : Pile
Exhaust Pile

-discardPile : Pile
Discard Pile

+PileCollection(handPile : Pile, drawPile : Pile, exhaustPile : Pile, discardPile : Pile) : PileCollection
constructor

+handToDiscard()
moves cards in the hand pile to discard pile

+discardToDraw()
shuffles cards in the discard pile and adds them to draw pile

+drawCard()
takes a card from draw pile and adds it to hand pile

EffectHandler
This class will responsible for managing effect objects.

-enemies: ArrayList<Enemy>
this is a reference to the current enemy list

-turn: Turn
this is a reference to the turn variable

-piles : PileCollection
reference to in-fight piles and their functionalities.(draw pile,hand pile etc.)

-cardEffectManager : CardManager
This is reference to cardmanager object and it is responsible for returning card effects when card is played.

-cardEffectManager : CardManager
This is reference to BuffManager object and it is responsible for returning buff effects before any card effect is applied.

-character: Character
This is a reference to current character.

-effectStack : Stack<Effect>
Collection of Effects of the cards, this class pops effect from this stack and applies it.

+EffectHandler(enemies : ArrayList<Enemy>, turn : Turn, piles : PileCollection, character Character) : EffectHandler
Constructs object with given params.

+playCard(card : Card, target Enemy) : boolean
applys the given card effects to given enemy. Returns true if card is playable and false otherwise.

+playEnemy() : void
applys the current enemy behaviour effects to character or enemies.

+nextTurn() : void

Applys the nextTurn effects to both sides of the fight.

+applyEffect(Effect effect) : void
calls proper apply function according the type of the given effect.

applyDamageEffect(damage : Damage) : void
applies given damage effect

+applyBlockEffect(blockEffect : Block) : void
applies given block effect

+applyEnergyEffect(energyEffect : ChangeEnergy) : void
applies given change energy effect.

+applyBuffEffect(applyBuff : ApplyBuff) : void
applies given applyBuff effect.

+applyMoveCardEffect(moveCard : MoveCard) : void
applies given move card effect.

+applyDrawCardEffect(drawCard : DrawCard) : void
draw cards for character.

+applyUpgradeCardEffect(upgradeCard : UpgradeCard)
upgades the card in the given effect.

-runStack() : void
applys the effects in the stack one by one

PotionManager
-enemies: ArrayList<Enemy>
this is a reference to the current enemy list

-turn: Turn
this is a reference to the turn variable

-piles : PileCollection
reference to in-fight piles and their functionalities.(draw pile,hand pile etc.)

-character: Character
This is a reference to current character.

+usePotion(Potion potion, Enemy target) : ArrayList<Effect>
returns the given potion effects to given target.

+usePotion(Potion potion) : ArrayList<Effect>
returns the given targetless potion effects.

+PotionManager(enemies : ArrayList<Enemy>, turn Turn, piles PileCollection, character Character) : PotionManager
construct object

RelicManager
-enemies: ArrayList<Enemy>
this is a reference to the current enemy list

-turn: Turn
this is a reference to the turn variable

-piles : PileCollection
reference to in-fight piles and their functionalities.(draw pile,hand pile etc.)

-character: Character
This is a reference to current character.

+RelicManager(enemies : ArrayList<Enemy>, turn Turn, piles PileCollection, character Character) : RelicManager
Constructer

+getFightStartRelicEffects() : ArrayList<Effect>
returns the effects of Relics if fight is just started.

+getTurnRelicEffects() : ArrayList<Effect>
returns the effects of Relics when turn passes.

+getRelicEffects() : ArrayList<Effect>
returns the effects of Relics when any card is played

+updateRelics(effect Effect) : void
updates the relic properties whenever effect is applies in the fight.

+restartRelics() : void
restarts relics when fight is over.

CardManager
-enemies: ArrayList<Enemy>
this is a reference to the current enemy list

-turn: Turn
this is a reference to the turn variable

-piles : PileCollection
reference to in-fight piles and their functionalities.(draw pile,hand pile etc.)

-character: Character
This is a reference to current character.

+CardManager(enemies : ArrayList<Enemy>, turn : Turn, piles PileCollection, character : Character) : CardManager
Constructor

+getEffects(Card card, Enemy target) : ArrayList<Effect>
returns the effects of card when card is played to given target in the current situation of the game.

BuffManager
-enemies: ArrayList<Enemy>
this is a reference to the current enemy list

-turn: Turn
this is a reference to the turn variable

-piles : PileCollection
reference to in-fight piles and their functionalities.(draw pile,hand pile etc.)

-character: Character
This is a reference to current character.

+BuffManager(enemies : ArrayList<Enemy>, turn Turn, piles PileCollection, character : Character) : BuffManager
constructor

+getNextTurnBuffEffects() : ArrayList<Effect>
returns the effects of the buffs that are active at the begining of the turn

+cleanBuffs() : void
cleans the buffs if their function is over.

-getEnemyBuffEffects() : ArrayList<Effect>
returns the effects of enemy buffs

+getCharacterBuffEffects() : ArrayList<Effect>
returns the effects of character buffs

+getBuffEffects(buff : Buff, owner : Enemy) : ArrayList<Effect>
returns the effect of single buff.

+getBuffNextTurnEffects(buff : Buff, owner : Enemy) : ArrayList<Effect>
returns the effect of single buff if buff has any effect at the begining of the turn.

Damage
This class keeps the information of damage effect
-damage : int
damage amount
-target : Enemy
target enemy, if it is null, it means target is character
-source : Enemy
source of the damage, if it is null, it means source is character

Block
-block : int
block amount
-target : Enemy
enemy that block is going to applies. if null, block will be applied to character

ChangeEnery
-energy : int
amount of energy that is going to added to character energy.

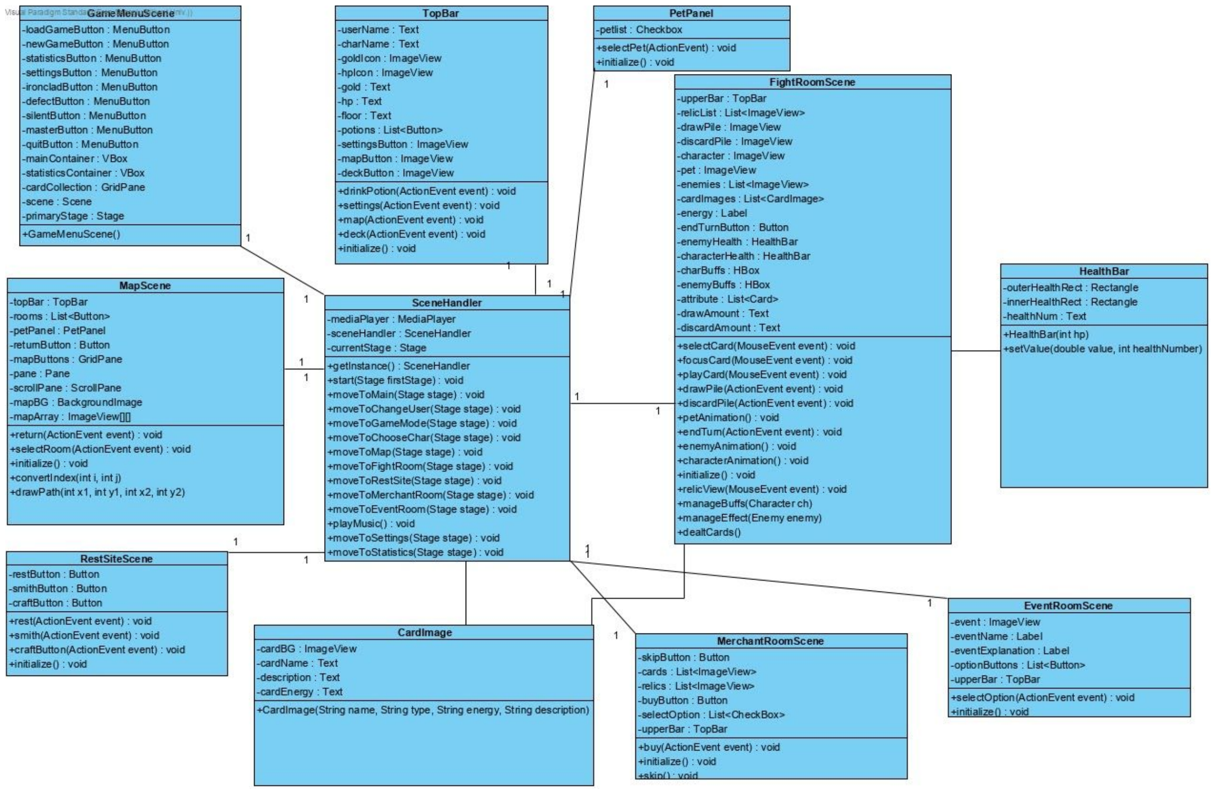
DrawCard
keeps the information that card will be drawn

MoveCard
keeps the information that card will be moved
-sourcePile : Pile
pile that the card is moved from. (if null, no card will be taken from somewhere.)
-destPile : Pile
pile that card will be added to. (if null , card will not be added anywhere)
-card : Card
card that is going to move

ApplyBuff
-buff : Buff
buff that is going to be added to buff list of target
-target : Enemy
target enemy. If null, target is character.

UpgradeCard
-card : Card
card that is going to be upgraded.

4.4.4. Interface Classes


GameMenuScene
-loadGameButton : MenuButton
-newGameButton : MenuButton
-statisticsButton : MenuButton
-settingsButton : MenuButton
-ironcladButton : MenuButton
-defectButton : MenuButton
-silentButton : MenuButton
-masterButton : MenuButton
-quitButton : MenuButton
-mainContainer : VBox
-statisticsContainer : VBox
-cardCollection : GridPane
-scene : Scene
-primaryStage : Stage
+GameMenuScene()
TopBar
-userName : Text
-charName : Text
-goldIcon : ImageView
-hpIcon : ImageView
-gold : Text
-hp : Text
-floor : Text
-potions : List<Button>
-settingsButton : ImageView
-mapButton : ImageView
-deckButton : ImageView
+drinkPotion(ActionEvent event) : void
+settings(ActionEvent event) : void
+map(ActionEvent event) : void
+deck(ActionEvent event) : void
+initialize() : void
PetPanel
-petlist : Checkbox
+selectPet(ActionEvent) : void
+initialize() : void
FightRoomScene
-upperBar : TopBar
-relicList : List<ImageView>
-drawPile : ImageView
-discardPile : ImageView
-character : ImageView
-pet : ImageView
-enemies : List<ImageView>
-cardImages : List<CardImage>
-energy : Label
-endTurnButton : Button
-enemyHealth : HealthBar
-characterHealth : HealthBar
-charBuffs : HBox
-enemyBuffs : HBox
-attribute : List<Card>
-drawAmount : Text
-discardAmount : Text
+selectCard(MouseEvent event) : void
+focusCard(MouseEvent event) : void
+playCard(MouseEvent event) : void
+drawPile(ActionEvent event) : void
+discardPile(ActionEvent event) : void
+petAnimation() : void
+endTurn(ActionEvent event) : void
+enemyAnimation() : void
+characterAnimation() : void
+initialize() : void
+relicView(MouseEvent event) : void
+manageBuffs(Character ch)
+manageEffect(Enemy enemy)
+dealtCards()
HealthBar
-outerHealthRect : Rectangle
-innerHealthRect : Rectangle
-healthNum : Text
+HealthBar(int hp)
+setValue(double value, int healthNumber)
MapScene
-topBar : TopBar
-rooms : List<Button>
-petPanel : PetPanel
-returnButton : Button
-mapButtons : GridPane
-pane : Pane
-scrollPane : ScrollPane
-mapBG : BackgroundImage
-mapArray : ImageView[][]
+return(ActionEvent event) : void
+selectRoom(ActionEvent event) : void
+initialize() : void
+convertIndex(int i, int j)
+drawPath(int x1, int y1, int x2, int y2)
SceneHandler
-mediaPlayer : MediaPlayer
-sceneHandler : SceneHandler
-currentStage : Stage
+getInstance() : SceneHandler
+start(Stage firstStage) : void
+moveToMain(Stage stage) : void
+moveToChangeUser(Stage stage) : void
+moveToGameMode(Stage stage) : void
+moveToChooseChar(Stage stage) : void
+moveToMap(Stage stage) : void
+moveToFightRoom(Stage stage) : void
+moveToRestSite(Stage stage) : void
+moveToMerchantRoom(Stage stage) : void
+moveToEventRoom(Stage stage) : void
+playMusic() : void
+moveToSettings(Stage stage) : void
+moveToStatistics(Stage stage) : void
RestSiteScene
-restButton : Button
-smithButton : Button
-craftButton : Button
+rest(ActionEvent event) : void
+smith(ActionEvent event) : void
+craftButton(ActionEvent event) : void
+initialize() : void
CardImage
-cardBG : ImageView
-cardName : Text
-description : Text
-cardEnergy : Text
+CardImage(String name, String type, String energy, String description)
MerchantRoomScene
-skipButton : Button
-cards : List<ImageView>
-relics : List<ImageView>
-buyButton : Button
-selectOption : List<CheckBox>
-upperBar : TopBar
+buy(ActionEvent event) : void
+initialize() : void
+skip() : void
EventRoomScene
-event : ImageView
-eventName : Label
-eventExplanation : Label
-optionButtons : List<Button>
-upperBar : TopBar
+selectOption(ActionEvent event) : void
+initialize() : void
1

SceneHandler:
This class basically change the screen by moveTo methods. It has the Singleton design pattern to provide just one SceneHandler instance for the client-side system as can be observed through the sceneHandler attribute and getInstance function. The sound effects and the audio description features are also provided with this class. It has the mediaPlayer attribute and playMusic function for the sound effects. Also, the object holds the current screen by the current stage attribute.

GameMenuScene:
This class handles the main menu screen. There are buttons in the main menu that are used for moving other game screens which are Game mode screen, settings screen, edit name screen, statistics screen. Moving screens is done by adding and removing panes. Also, by the quit button, the player can terminate the program.
TopBar:
This upper bar provides a short-cut tool to reach the basic stuffs in the game. It appears different scene of the game similarly. Therefore, we create this class individually and put the objects where it appears. It basically shows the username, character name, hp, current floor, gold by labels. There is also relics, deck and map buttons. When user hover mouse on of them it gets detailed information about it. Also, there is settings button where user can directly access the setting page.

FightRoomScene:
This class handles the fight room screen and its animations. It has all the elements which are buttons, images, labels etc. that the fight room needs. It handles the all results of the button by corresponding function of the buttons. It handles all the visual components of the fight including health bar, card dealt, discard pile, draw pile, buffs, effects and other minor visual components.

MapScene:
This class handles map screen. It has a list of buttons which are locations, since map is randomly generated, scene gets map input from model, according to that random generation buttons are placed on the screen and all other attributes and effects of buttons are handled. By the clicks on the room buttons system handles the new scene and room. It also has a return button to turn back to the main menu.

PetPanel:
This class handles the pet panel in the Map scene. User’s petList is displayed here and he will be able to choose which one he’s bringing in to the fight via checkBoxes.

EventRoomScene:
This class handles the Event room screen. The event name, image, explanation and the options of this event are displayed. The user will select the best option by clicking on it with the help of selectOption() function.

MerchantRoomScene:
This class handles the Merchant room screen. The items on sale are displayed. User can select whichever items he wants to buy by clicking on the checkboxes below the items, and purchase them by clicking on buy button. It is also possible to just skip by clicking on the skip button.

RestSiteScene:
This class handles the Rest site screen. Basically user has 3 options: rest, smith or craft. He will be able to do one them by clicking on one of these buttons.

CardImage:
This class handles the visuality of the cards. With the required inputs , name, background, description and energy, a card image is processed with all onClick and onMouseEnter effects as well.

HealthBar:
This class handles the health bars of the fighters in the fight which includes enemies, user and pet. Draws an health bar according to current and max health. Updates the bar according to the changes.