Object-Oriented

Software Engineering

System Design Report

Group 3K

Deniz Alkışlar

Ekinsu Bozdağ

Eliz Tekcan

Serhat Aras

Selen Haysal

1. Introduction

1.1. Purpose of the System

Roomie is a story-based, single player Android game designed to simulate the university life of a student. With Roomie, we aim to make users enjoy and understand how the university life can actually be, and how our choices matter with the different scenarios Roomie offers. The user-friendly design of Roomie makes it simpler to play. As the game starts, the user creates his/her unique character with entering its name and gender, with this feature we aim to make the game user-specific. Roomie is designed to keep the user occupied with the notifications and unexpected tasks during the day, even if the game is not on. This feature of Roomie makes it more challenging compared to other story-based games, and it helps Roomie to preserve its users interest. Roomie also includes a status bar with four different labels: health, money, sociality and grades. With this feature our aim is to keep the user engaged and more competitive while making choices since their choice will increase or decrease the levels of these statuses. Roomie also aims to display the user its interesting features in their first few visits, so that the user will preserve interest.

1.2. Design Goals

Prior to building the system, our aim is to identify the design goals to clarify what the system will focus on. Below are our design goals, many of them are included in the analysis stage of Roomie, in non-functional requirements.

End User

1. **Reliability**

   The reliability of the system is dependent to the reliability of Android.

Exception handling yapacağız

2. **Usability**

   We aim to plan how the users of Roomie will engage with the app, looking at Roomie from their perspective, to see how they will understand the app and use it correctly. This will lead Roomie to create a positive user experience. Another approach we are going to take to create a positive user experience is to make Roomie require as few steps as possible. We will implement Roomie in a way that the user will go through fewer steps, pages and buttons. We plan to design the app in a way that all options make sense to the user, spending time to figure out the best structure and see how all features of Roomie will fit together in an organized way.

3. **Extensibility**

   Since the game is designed to be object-oriented, it can easily be modified and extended. The design architecture reduces the malfunctioning possibility and makes it possible to extend and change particular features without having to modify other classes. Its design makes Roomie highly extensible. Roomie is also suitable to be reused and extended for further versions and other works.

4. **Adaptability**

   We are using Android to fulfill adaptability feature, Roomie will be available on Android devices with Android version 6.0 and above.

5. **Ease of learning**

   Our design will be simple and effective so that the user can easily understand and perform actions. We plan to make the user interface easy to use, and use icons to guide the users and use iconography to lead the users, this makes Roomie easier to learn and play, even for user without prior knowledge of the game.

6. **Portability**

   Roomie is highly portable since it will run on any Android device with Android version 6.0 and above.

7. **Availability**

   Since Android API 23 (Marshmallow) is going to be used in the project development, Roomie will be available to those with Android version 6.0 and above.

Developer

1. **Minimum # of errors**

   The design of Roomie aims to have minimum number of errors.For this purpose, we will use the libraries that Android provides. Complicated and long codes will be avoided which will result in easier code to debug. Many classes will be utilized to have a more stable structure in our implementation. We will employ unit tests to validate that every class works fine. To develop a more reliable system, we will make sure that necessary exception handlings are done.

2. **Modifiability**

   This aspect is not a must for Roomie. However, it is intended that Roomie will be highly modifiable. To fulfill this aspect, our aim is to reduce the number of modules that are directly affected by change. To achieve this, the coupling of the subsystems will be minimized so that a change in a system component does not have a huge effect in the system.

3. **Reusability**

   It is aimed to make Roomie suitable to be reused and extended for further versions and other works, but it is not a priority.

Tradeoffs

1. **Efficiency** - Reusability

   Efficiency is an aspect that matters for Roomie. Roomie is implemented in an efficient way by using graph data structure. The events are kept in a graph design since Roomie is an event-based story game. Arrays are not utilized in the fundamental design to keep the implementation efficient. Also, when the players save a game on Roomie, the whole game is not saved. Instead, only the key points that are changed during the game will be saved, and the game will be loaded with those changed state savings. The game is implemented by adopting from what Android has to offer which makes the implementation more efficient. Roomie's code could be reused in the case of extending the game and building on the design. The storyline might be enlarged by modifying the graph structure and other features might be added.

2. Functionality - **Usability**

   Roomie's prior concern is to have a simple usage rather than having a complex functionality. The game will not be too complex to play, since this would decrease the interest in Roomie. The game will have a simple user interface and easy to learn actions that the user can perform. The user will not have to spend time understanding what can be performed, the game will proceed easily without confusing the player. With its minimal features and functionality, Roomie will be as user-friendly as possible.

3. Space - **Speed**

   Roomie does not occupy a large space since the game contains of only source code files and multimedia files that are used in the user-interface. Also the game will save and load into the Internal Storage Device, again not taking up storage space. Roomie tends to be a dynamically working, high performance game and provide its users a fast and vital game environment.

1.3. References

[1] “Storage Options.” Android Developers, 8 Nov. 2016, developer.android.com/guide/topics/data/data-storage.html#pref.

[2] Rogers, Rick. Learning Android Game Programming: A Hands-on Guide to Building Your First Android Game. Addison-Wesley, 2012.

[3] “Design Patterns MVC Pattern.” Www.tutorialspoint.com, Tutorials Point, 15 Aug. 2017, www.tutorialspoint.com/design_pattern/mvc_pattern.htm.

1.4. Overview

In the Introduction section of this Design Report, the purpose of Roomie, which is to simulate the life of an university student is explained in depth. The design goals of Roomie are detailed. Reliability, usability, extensibility, adaptability, ease of learning, portability and availability compose our prior concerns regarding Roomies design. Among efficiency and reusability Roomie favors efficiency, since our prior concern is to develop an efficient game rather than writing reusable code. Our user-friendly design and not complex functionalities of Roomie leads Roomie to choose usability over functionality. Regarding speed and space, Roomie opts for speed, it is designed in a way that lessens the storage use and is a fast game environment.

2. Software Architecture

2.1. Overview

In this chapter, the architecture of our design will be revealed in more detail. The aim is to define an abstract framework for the components of Roomie and to set a firm structural organization. Along with structural design, more details about user access, data management and boundary conditions will be disclosed.

2.2. Subsystem Decomposition

In this section, the aim is to decompose our system into smaller subsystems to have a more stable and understandable structure. The whole system of our project is divided into three main sub-categories. This structural design meets our non-functional requirements such as modifiability and extendibility. Their connection with each other, and their independent roles makes the whole implementation easier and logical. This makes any possible further modifications easier to implement.

The first subsytem is called User Interface, and it holds viewpoints. The ActivityManager in this system responds the requests that are coming from the lower subsystem which is called Game Manager. Game Manager is responsible for receiving user input and evaluating these inputs, Game Manager sends appropriate requests to User Interface to update the screen view. Along with these two subsystems, Game subsystem provides the main logic of the game. All of the game events, player objects, and other components that are related to the internal logic of the game are hold in this subsystem. See the figure that is below for more detailed view of these three subsystems’ decomposition:

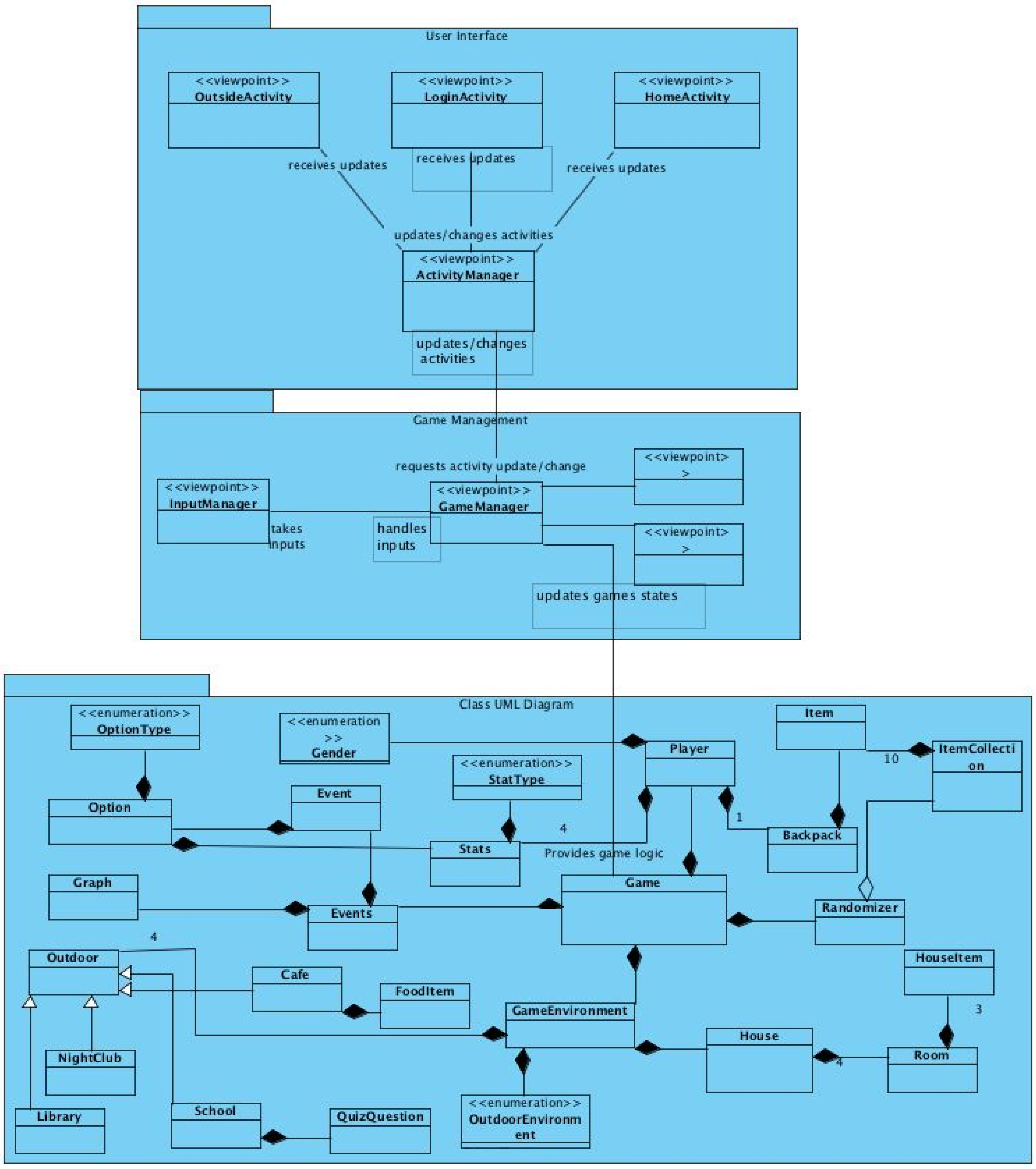


Figure 1: Basic Subsystem Decomposition

2.3 Architectural Design

Roomie follows Model-View-Controller Architectural Pattern in its design. Using this model, the separation between the UI/Interaction layer and the whole logic layer becomes apparent. There is also another layer that arranges the relationship between these two other layers, the controller layer. The view layer will be responsible for the visualization of the model layer [3]. The model layer contains all the objects that contain data about the game. The control layer is to update the view layer whenever there is change in data of the model layer. The corresponding components in our design are listed below:

1) View Layer - User Interface: This layer is about the UI that is displayed to user while playing the game.
2) Controller Layer - GameManager Class: This class is to provide connection between the view layer and the model layer. It can be thought as an action listener, its duty is to listen user inputs and direct those inputs to view layer for displaying appropriate screens.
3) Model Layer - Game Class: The class that implements the whole internal logic and flow of the game. All of the game related objects (e.g Player, Stats, School) are implemented in this layer of the architecture.

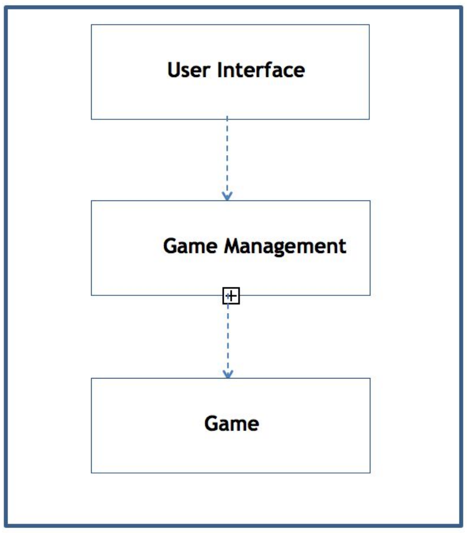


Figure 3: Three layers of MVC Design Pattern

2.4 Hardware / Software Mapping

Roomie will be implemented in Java programming language for Android. JDK 1.8 (Java Development Kit) will be used. As hardware setup, a working Android phone is enough to run Roomie. The touchscreen of the phone, its speakers and microphone are the components where the game will take input from. Roomie does not require Internet/ Network connection to operate, and therefore it is an offline game.

2.5 Persistent Data Management

To manage data, Android's SharedPreferences class will be utilized. This class lets applications to "save and retrieve persistent key-value pairs of primitive data types." [1] Using this class, the objects of the game will be saved as strings. Also, the point that the player leaves the game will be saved along with the player's status points. Since Roomie is implemented using graph data structure, the design will be able to observe and understand the event that user leaves the game. Using this class, all of this data information will be directly saved on the device's internal storage. These files that are stored are private and the player cannot access them. If the player uninstalls Roomie, all of these files will be removed.

2.6 Access Control and Security

In Roomie, there will not be any access control. Anyone who installs the game could play the game. Since there is no network connection, user registration is also not possible. However, the player will be able to create a character for himself before starting the game. There are not security concerns about the game, considering that the content of the game and its limitations are not matters of security.

2.7 Boundary Conditions

Initialization

Since Roomie is an Android game, it will require an installment from Google Play. After installing the game, touching on the Roomie icon is enough to initiate the game.

Termination

Roomie can be terminated / closed by clicking “Home” button of the phones, the main menu of the game will also let the players close the game. If player wants to quit during game play, system provides an automatic save and the player is able to continue the game from the same point.

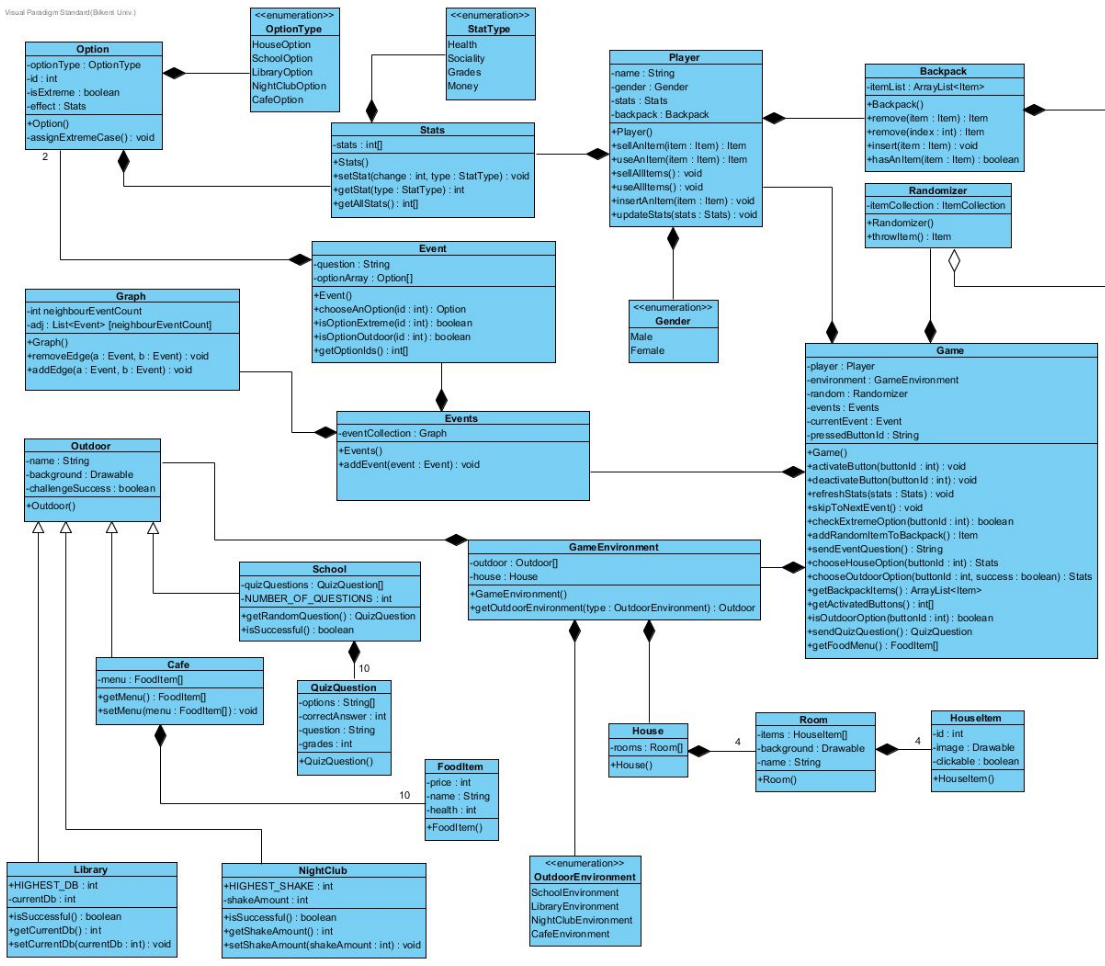
When the user completes a game cycle, main menu screen will be directed to the player.

Error

If program does not respond because of a performance/compatibility issue, the player will need to restart the game and lose all of the current data.

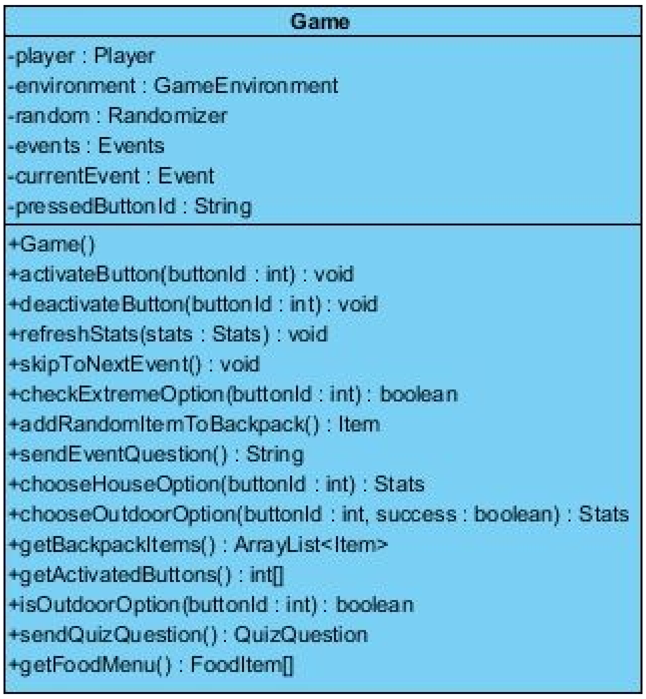
3. Subsystem Services

3.1 Game(Model)



Visual Paradigm Standard(Bilkent Univ.)
<<enumeration>> OptionType
HouseOption
SchoolOption
LibraryOption
NightClubOption
CafeOption
<<enumeration>> StatType
Health
Sociality
Grades
Money
Option
-optionType : OptionType
-id : int
-isExtreme : boolean
-effect : Stats
+Option()
-assignExtremeCase() : void
2
Stats
-stats : int[]
+Stats()
+setStat(change : int, type : StatType) : void
+getStat(type : StatType) : int
+getAllStats() : int[]
Player
-name : String
-gender : Gender
-stats : Stats
-backpack : Backpack
+Player()
+sellAnItem(item : Item) : Item
+useAnItem(item : Item) : Item
+sellAllItems() : void
+useAllItems() : void
+insertAnItem(item : Item) : void
+updateStats(stats : Stats) : void
Backpack
-itemList : ArrayList<Item>
+Backpack()
+remove(item : Item) : Item
+remove(index : int) : Item
+insert(item : Item) : void
+hasAnItem(item : Item) : boolean
Item
-name : String
-effect : Stats
-image : Drawable
+Item()
10
Randomizer
-itemCollection : ItemCollection
+Randomizer()
+throwItem() : Item
ItemCollection
-ITEMS : Item[]
+ItemCollection()
+getItems() : Item[]
Event
-question : String
-optionArray : Option[]
+Event()
+chooseAnOption(id : int) : Option
+isOptionExtreme(id : int) : boolean
+isOptionOutdoor(id : int) : boolean
+getOptionIds() : int[]
<<enumeration>> Gender
Male
Female
Graph
-int neighbourEventCount
-adj : List<Event> [neighbourEventCount]
+Graph()
+removeEdge(a : Event, b : Event) : void
+addEdge(a : Event, b : Event) : void
Events
-eventCollection : Graph
+Events()
+addEvent(event : Event) : void
Game
-player : Player
-environment : GameEnvironment
-random : Randomizer
-events : Events
-currentEvent : Event
-pressedButtonId : String
+Game()
+activateButton(buttonId : int) : void
+deactivateButton(buttonId : int) : void
+refreshStats(stats : Stats) : void
+skipToNextEvent() : void
+checkExtremeOption(buttonId : int) : boolean
+addRandomItemToBackpack() : Item
+sendEventQuestion() : String
+chooseHouseOption(buttonId : int) : Stats
+chooseOutdoorOption(buttonId : int, success : boolean) : Stats
+getBackpackItems() : ArrayList<Item>
+getActivatedButtons() : int[]
+isOutdoorOption(buttonId : int) : boolean
+sendQuizQuestion() : QuizQuestion
+getFoodMenu() : FoodItem[]
Outdoor
-name : String
-background : Drawable
-challengeSuccess : boolean
+Outdoor()
GameEnvironment
-outdoor : Outdoor[]
-house : House
+GameEnvironment()
+getOutdoorEnvironment(type : OutdoorEnvironment) : Outdoor
School
-quizQuestions : QuizQuestion[]
-NUMBER_OF_QUESTIONS : int
+getRandomQuestion() : QuizQuestion
+isSuccessful() : boolean
10
Cafe
-menu : FoodItem[]
+getMenu() : FoodItem[]
+setMenu(menu : FoodItem[]) : void
QuizQuestion
-options : String[]
-correctAnswer : int
-question : String
-grades : int
+QuizQuestion()
FoodItem
-price : int
-name : String
-health : int
+FoodItem()
10
House
-rooms : Room[]
+House()
4
Room
-items : HouseItem[]
-background : Drawable
-name : String
+Room()
4
HouseItem
-id : int
-image : Drawable
-clickable : boolean
+HouseItem()
Library
+HIGHEST_DB : int
-currentDb : int
+isSuccessful() : boolean
+getCurrentDb() : int
+setCurrentDb(currentDb : int) : void
NightClub
+HIGHEST_SHAKE : int
-shakeAmount : int
+isSuccessful() : boolean
+getShakeAmount() : int
+setShakeAmount(shakeAmount : int) : void
<<enumeration>> OutdoorEnvironment
SchoolEnvironment
LibraryEnvironment
NightClubEnvironment
CafeEnvironment

3.1.1. Game Class



```
Game
-player : Player
-environment : GameEnvironment
-random : Randomizer
-events : Events
-currentEvent : Event
-pressedButtonId : String
+Game()
+activateButton(buttonId : int) : void
+deactivateButton(buttonId : int) : void
+refreshStats(stats : Stats) : void
+skipToNextEvent() : void
+checkExtremeOption(buttonId : int) : boolean
+addRandomItemToBackpack() : Item
+sendEventQuestion() : String
+chooseHouseOption(buttonId : int) : Stats
+chooseOutdoorOption(buttonId : int, success : boolean) : Stats
+getBackpackItems() : ArrayList<Item>
+getActivatedButtons() : int[]
+isOutdoorOption(buttonId : int) : boolean
+sendQuizQuestion() : QuizQuestion
+getFoodMenu() : FoodItem[]
```

Attributes:

- **private Player player:** This attribute holds an instance of a Player class which holds the player related information and actions of the Roomie.
- **private GameEnvironment environment:** This attribute is containing the game environments in which implemented as GameEnvironment Class.
- **private Randomizer random:** This element holds the randomizer reference for randomizing the items which dropped to the player's Backpack.
- **private Events events:** This attribute holds the events for the game.
- **public Event currentEvent:** This attribute holds the current event of the Roomie.
- **public String pressedButtonId :** This attribute stores the pressed button's ID which represents the choosen option.

Constructors:

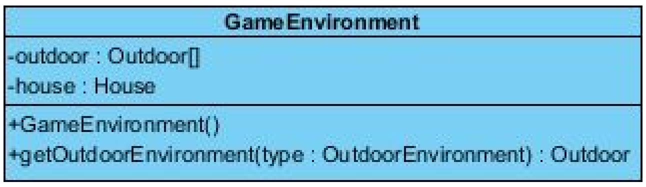
- **public Game():** Initialize the game instance for the Roomie which holds the information of the player, environment, events, current event, pressed button and randomizer class.

Methods:

- **public void activateButton(String buttonId):** This method activates a button with the give button ID. So that the player can press this button to make a choice.
- **public void deactivateButton(String buttonId):** This method deactivates a button with the give button ID. This means that the button with the given id cannot be used for making a choice.
- **public void refreshStats(Stats stats):** This method refreshes the player stats. It takes a Stats instance as a parameter. It is used after the player presses a button.
- **public void skipToNextEvent():** This method skips to the next event in the events graph. It is called after the previous event finishes and is not an extreme event.
- **public boolean checkExtremeOption(int buttonId):** This method checks the choosen option to see if it is an extreme option. If it is an extreme option and player chooses it, the game is over.
- **public Item addRandomItemToBackpack():** This method generate a random item by using the Randomizer and adds it to the player’s backpack. It is used when the player is successful in an outside challenge.
- **public String sendEventQuestion():** This method returns the current event’s question in String type. The question is shown in a dialogbox during the game.
- **public Stats chooseHouseOption(int buttonId):**This method is used during making a house choice by the player. It returns a Stats instance to refresh the stats bar in the user interface.
- **public Stats chooseOutdoorOption(int buttonId, boolean success):** This method is used during making an outdoor choice by the player. It is used after an outdoor challenge finishes and uses the challenge’s success as a parameter to calculate new stats. It returns a Stats instance to refresh the stats bar in the user interface.
- **public ArrayList<Item> getBackpackItems():** This method returns the items in the backpack as an ArrayList. The returned ArrayList is used as an ArrayAdapter for the listview.

- **public int [] getActivatedButtons():** This method returns button ids to be activated. Activating a button means that making the button clickable.
- **public boolean isOutdoorOption(int buttonId):** This method check the pressed option to see if it is an outdoor option. It used to start the outside challange by launching a new activity.
- **public QuizQuestion sendQuizQuestion():** This method returns a random quiz question. It is used when School challenge is launched.
- **public FoodItem[] getFoodMenu():** This method returns the food menu. It is used when Cafe challange is launched.

3.1.2. Game Environment Class



Attributes:

- **private Outdoor[] outdoor:** This attribute holds the outdoor Game Environment.
- **private House house:** This attribute holds the House object.

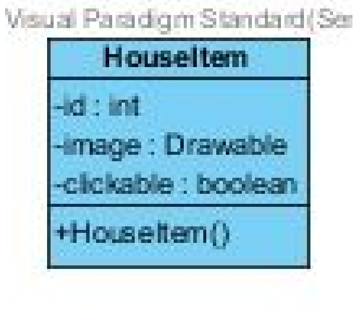
Constructors:

- **public GameEmvironment():** Constructs the game environments.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public Outdoor getOutdoorEnvironment(OutdoorEnvironment type):** Returns an outdoor instance from the Outdoor array with a given OutdoorEnvironment type.

3.1.4. HouseItem Class



Attributes:

- **private String name:** This variable stores the name of the item.
- **private String id:** This attribute holds the id of the house item.
- **private Drawable image:** This drawable variable holds the image of the house item.
- **private Boolean clickable:** This Boolean attribute stores the condition of the clickable case of the item.

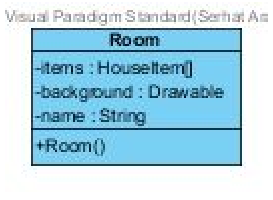
Constructors:

- **public HouseItem():** Construtor for the House Item Class.

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.5. Room Class



Attributes:

- **private HouseItem[3] items:** Stores the house item in an HouseItems array.
- **private Drawable background:** Holds the background of the room.
- **private String name:** Holds the name of the room.

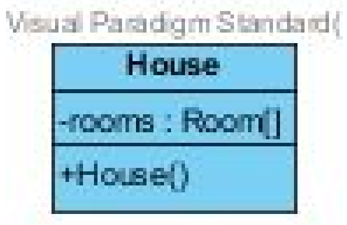
Constructors:

- **Public Room():** Construct the room with the items and background.

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.6. House Class



Attributes:

- **private Room[4] rooms:** Holds the Rooms in the house.

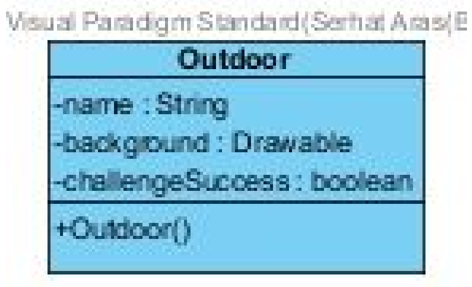
Constructors:

- **public House():** The constructor of the House class initiates the rooms.

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.7. Outdoor Class


Visual Paradigm Standard(Serhat Aras(E
Outdoor
-name : String
-background : Drawable
-challengeSuccess : boolean
+Outdoor()

Attributes:

- **private String name:** This attribute holds the name of the outdoor.
- **private Drawable background:** This attribute holds the background of the Outdoor.
- **private boolean challengeSuccess:** This attribute stores the flag whether the challenge is accomplished successfully or not. If challenge is successfully accomplished the value of challengeSuccess is true, otherwise it is false.

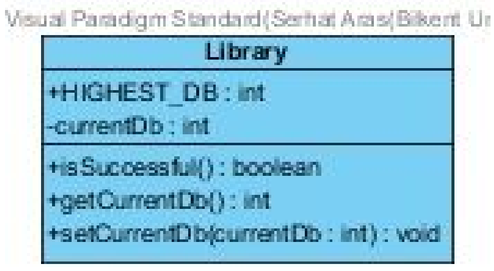
Constructors:

- **public Outdoor():** This constructor initiates the outdoor object.

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.9. Library Class


Visual Paradigm Standard(Serhat Aras(Bilkent U
Library
+HIGHEST_DB : int
-currentDb : int
+isSuccessful() : boolean
+getCurrentDb() : int
+setCurrentDb(currentDb : int) : void

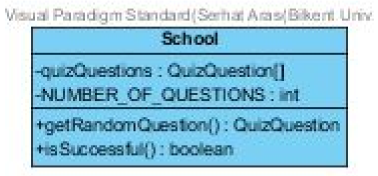
Attributes:

- **public final int HIGHEST_DB:** This attribute holds the highest decibel that is considered to be silent for library challenges. It is defined as constant since it will remain the same during game execution.
- **private int currentDb:** This attribute holds the current decibel input via users microphone.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public void setCurrentDb(int currentDB):** This method sets the current decibel and is used by the GameManager class. Using the microphone input, the decibel is calculated and stored in the attribute currentDB.
- **public int getCurrentDb():** The integer value created and converted by the listener of the sensors. Overwrites the currentDb attributes.

3.1.10. School Class


Visual Paradigm Standard(Serhat Aras(Bilkent Univ.
School
-quizQuestions : QuizQuestion[]
-NUMBER_OF_QUESTIONS : int
+getRandomQuestion() : QuizQuestion
+isSuccessful() : boolean

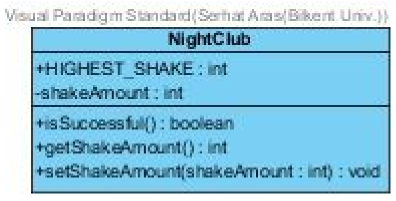
Attributes:

- **private QuizQuestion[] quizQuestions:** This attribute holds the questions that will be asked in the quiz when the player is in the School Outdoor Environment.
- **private final int NUMBER_OF_QUESTIONS:** This attribute holds the maximum number of questions that is stored in the School class.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public void getRandomQuizQuestion():** This method is used to get a random quiz question among possible quiz questions stored in quizQuestions array.
- **public boolean isSuccessful():** This method is used to determine whether the given answer to the prompted question in a Quiz performed in School Environment is true. It returns true if the answer is correct, otherwise false.

3.1.11. NightClub Class


Visual Paradigm Standard(Serhat Aras(Bilkent Univ.))
NightClub
+HIGHEST_SHAKE : int
-shakeAmount : int
+isSuccessful() : boolean
+getShakeAmount() : int
+setShakeAmount(shakeAmount : int) : void

Attributes:

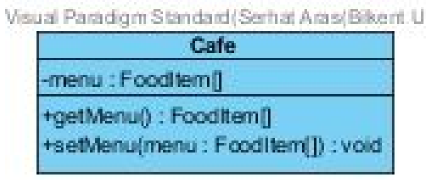
- **private int shakeAmount:** This attribute holds the latest shake amount that is performed by the user shaking the phone in NightClub challenges.
- **public final int HIGHEST_SHAKE:** This attribute holds the maximum amount of shake that can be performed during the game.

Methods:

- Getter and setter methods for each attribute are available for this class.

- **public void setShakeAmount(int shakeAmount):** This method is used by the GameManager class to update the shakeAmount attribute to users latest shakeAmount. It is used in NightClub challenges.
- **public boolean isSuccessful():** This method is used to determine whether the current shakeAmount is bigger than HIGHEST_SHAKE. It returns true if it is higher, otherwise false.

3.1.12. Cafe Class


Visual Paradigm Standard(Serhat Aras(Bilkent U
Cafe
-menu : FoodItem[]
+getMenu() : FoodItem[]
+setMenu(menu : FoodItem[]) : void

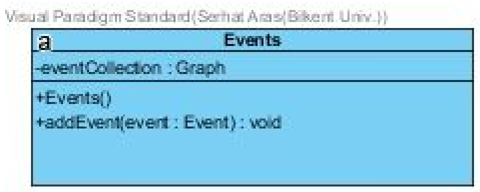
Attributes:

- **private FoodItem[] menu:** This attribute holds the menu of the cafe.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public FoodItem[] getMenu():** Returns the array that contains the instance of the FoodMenu Objects. Each of them predefined within the class so that in each call, returns the same array.
- **public void setMenu(FoodItem [] menu):** Sets the menu to the given parameter.

3.1.13. Events Class


Visual Paradigm Standard(Serhat Aras(Bilkent Univ.))
Events
-eventCollection : Graph
+Events()
+addEvent(event : Event) : void

Attributes:

- **private Graph eventCollection:** Holds the graph data structure for storing the events.

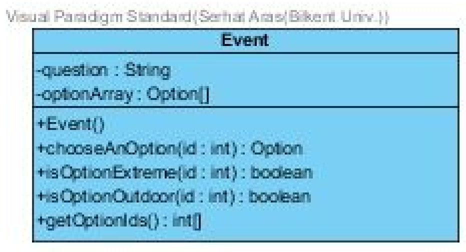
Constructors:

- **public Events():** Constructs the Events Class by the events graph. Events are initialized in the beginning of the game and do not change during game execution.

Methods:

- **public void addEvent(event: Event):** This function add the event which is taken from the parameter to the eventCollection graph.
- Getter and setter methods for each attribute are available for this class.

3.1.14. Event Class


Visual Paradigm Standard(Serhat Aras(Bilkent Univ.))
Event
-question : String
-optionArray : Option[]
+Event()
+chooseAnOption(id : int) : Option
+isOptionExtreme(id : int) : boolean
+isOptionOutdoor(id : int) : boolean
+getOptionIds() : int[]

Attributes:

- **private String question:** This String stores the question.
- **private Option[2] optionArray:** This Option array stores the possible answer to the question.

Constructors:

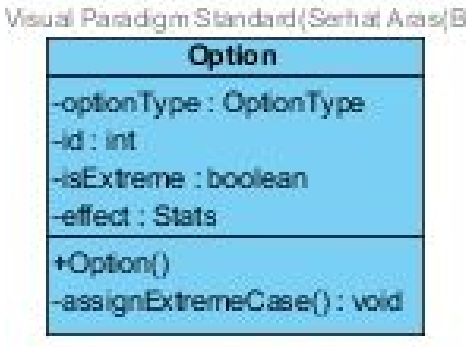
- **public Event():** Constructs the question and the options by reading them from a text file. This attributes initialized in the beginning of the game and do not change during game execution.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public Option chooseAnOption(int id):** This function chooses the option with the given id and returns the chosen Option object.
- **public boolean isOptionExtreme(int id):** This function chooses the option with the given button id and returns if the chosen Option is an extreme option. It returns 1 if it is extreme, 0 otherwise.

- **public boolean isOptionOutdoor(int id):** This function chooses the option with the given button id and returns if the chosen Option is an outdoor option. It returns true if it is outdoor, false otherwise. It is used to determine whether GameEnvironment will be updated in GameManager.

3.1.15. Options Class



Attributes:

- **private OptionType optiontype:** This attribute holds the optionType.
- **private String id:** This variable holds the id of the Option.
- **private boolean isExtreme:** This variable is the flag variable for the extreme case since the options are going to treated differently for extreme cases.
- **private StatType type:** This variable holds the StatType attribute of the option
- private int effect: This integer holds the effect amount in case of use this option.

Constructors:

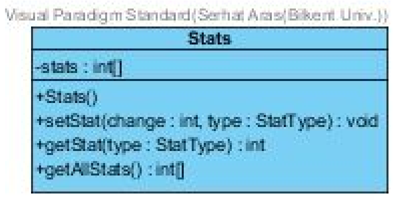
- **public Option():** This constructor initialize the Option object. The attributes of this object are initialized in the beginning of the game and do not change during game execution.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public boolean assignExtremeCase():** This method is used to assign extreme cases to Options. This is done randomly, it is less likely that an Option is an extreme case. This function is implemented in the following way: In the beggining of function execution random integer in the range 1 to 10 is initialized for each Option. If it is not

10, the Option is considered not extreme, if the value of the random integer variable is 10 an extreme case is assigned to current Option Object.

3.1.16. Stats Class


Visual Paradigm Standard(Serhat Aras(Bilkent Univ.))
Stats
-stats : int[]
+Stats()
+setStat(change : int, type : StatType) : void
+getStat(type : StatType) : int
+getAllStats() : int[]

Attributes:

- **private int stat:** This parameter stores the value of the stat as integer

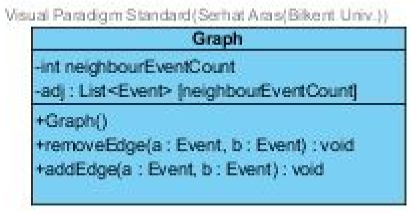
Constructors:

- **public Stats():** Constructs the Stats Object.

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.17. Graph Class


Visual Paradigm Standard(Serhat Aras(Bilkent Univ.))
Graph
-int neighbourEventCount
-adj : List<Event> [neighbourEventCount]
+Graph()
+removeEdge(a : Event, b : Event) : void
+addEdge(a : Event, b : Event) : void

Attributes:

- **private int neighbourEventCount:** This integer stores the adjacent edges on the graph. These edges are manifested as the neighbourEventCount.
- **private List<Event> [neighbourEventCount] adj:** Hold the linked edges to the one event as in the Graph implementation.

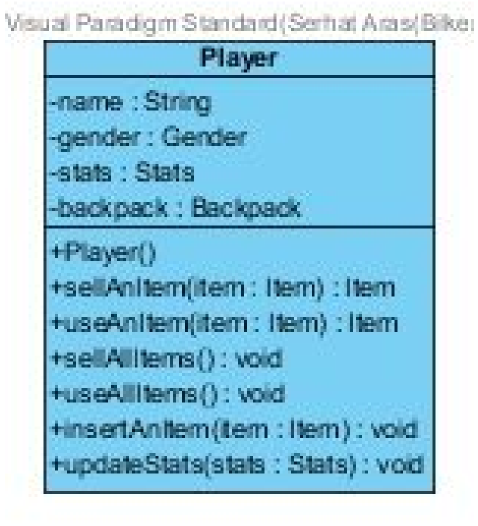
Constructors:

- **public Graph():** Constructs the Graph by the Attributes.

Methods:

public void removeEdge(a: Event, b:Event): Removes vertexes between event a and event b(events are treated as vertex)
public void addEdge(a: Event, b: Event): Adds vertexes between event a and event b(events are treated as vertex)

3.1.18. Player Class


Visual Paradigm Standard(Serhat Aras(Bilke
Player
-name : String
-gender : Gender
-stats : Stats
-backpack : Backpack
+Player()
+sellAnItem(item : Item) : Item
+useAnItem(item : Item) : Item
+sellAllItems() : void
+useAllItems() : void
+insertAnItem(item : Item) : void
+updateStats(stats : Stats) : void

Attributes:

- **private String name:** This variable holds the Name of the Player.
- **private boolean gender:** This variable holds the gender of the player.
- **private Stats[] stats:** This array stores the stats of the player.
- **private Backpack backpack:** This attribute denoted the itemCollection, known as BackPack (Inventory)

Constructors:

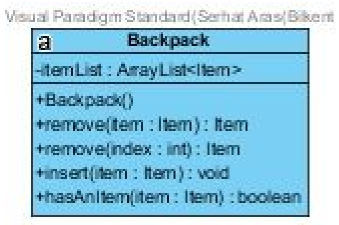
- **public Player():** Constructs the Player.

Methods:

- Getter and setter methods for each attribute are available for this class.
- **public void sellAnItem(Item item):** Sell an item from the BackPack which is going to worth for some money.

- **public void useAnItem(Item item):** Use an item from the Backpack to increase/decrease the stats.
- **public void sellAllItems():** Sell all items in the BackPack which is going to worth for some money.
- **public void useAllItems():** Use all item in the Backpack to increase/decrease the stats.
- **public void insertAnItem(Item item):** This method is used to insert an item to the BackPack.
- **public void updateStats(Stats stats):** This method updates the stats of the user with respect to the stats parameter input.

3.1.19. Backpack Class


Visual Paradigm Standard(Serhat Aras(Bilkent
Backpack
-itemList : ArrayList<Item>
+Backpack()
+remove(item : Item) : Item
+remove(index : int) : Item
+insert(item : Item) : void
+hasAnItem(item : Item) : boolean

Attributes:

- **Private ArrayList<Item> itemList:** This arraylist stores the items in the Backpack

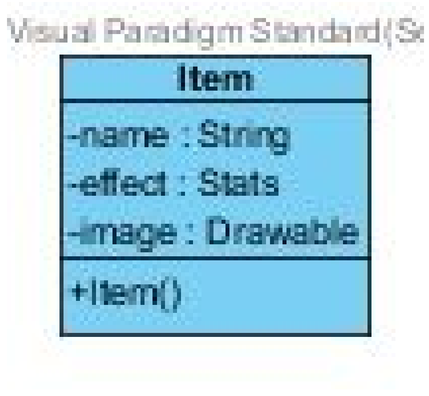
Constructors:

- **public Backpack():** Creates the Backpack

Methods:

- **public void remove (Item item):** Removes an item from the backpack
- **public void remove(int index):** Remove an item from the backpack with the given index.
- **public void insert(Item item):** Inserts the item to the Backpack.
- Getter and setter methods for each attribute are available for this class.

3.1.20. Item Class



Attributes:

- **Private String name:** This attribute holds the name of the item.
- **private Drawable image:** Stores the image of the Item.
- **private effect Stats:** This attribute stores the value of the effect of the particular item.

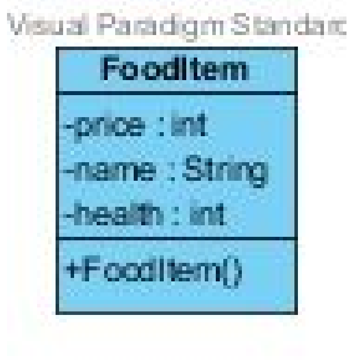
Constructors:

- **public Item():** Create and initialize the Item

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.21 Foot Item Class



Attributes:

- **Private int price:** This attribute holds the name of the item.
- **private String name:** This attribute holds the name of the item.
- **private int health:** This attribute hold the value of the health of the FoodItem.

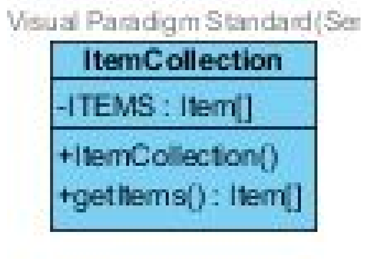
Constructors:

- **public FoodItem():** Create and initialize the Item

Methods:

- Getter and setter methods for each attribute are available for this class.

3.1.22. Item Collection Class



Attributes:

- **Private Item[10] items:** This array stores all items in the Roomie

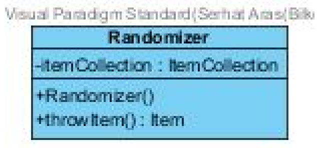
Constructors:

- **public ItemCollection():** creates the items and itemCollection.

Methods:

- **public void setItems(Item[]):** Sets the items form the parameter.
- **public Item[] getItems():** Returns the items in an array.

3.1.23. Randomizer Class


Visual Paradigm Standard(Serhat Aras(Bilk
Randomizer
-itemCollection : ItemCollection
+Randomizer()
+throwItem() : Item

Attributes:

- **private ItemCollection itemCollection:** Holds the Items which are possibly to drop.(Added to the backpack)

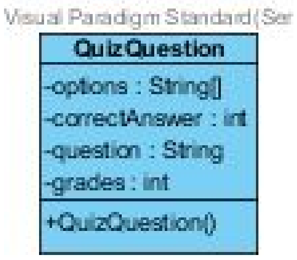
Constructors:

- **public Randomizer():** Inıtıalize the Randomizer.

Methods:

- **public Item throwItem():** Throws a random item to the backpack of the player form ItemCollection
- **public void setItemCollection(ItemCollection):** Sets predefined itemCollection which is taken from the parameter.
- **public ItemCollection getItemCollection():** Returns the itemCollection.
- Getter and setter methods for each attribute are available for this class.

3.1.24. Quiz Question Class



Attributes:

- **private String[] option:** This attribute stores the values of the
- **private int correctAnswer:**
- **private String question:**
- **private int grades:**

Constructors:

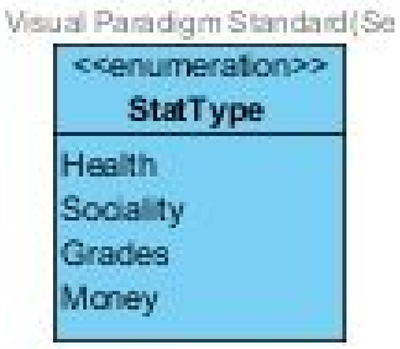
- **QuizQuestion()**: Inıtıalize the Randomizer.

Methods:

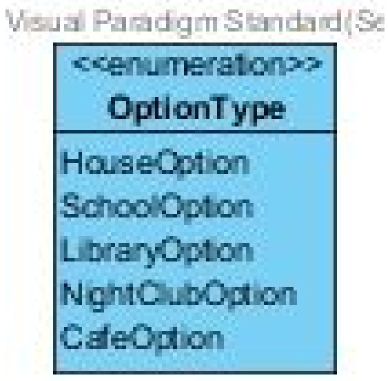
- Getter and setter methods for each attribute are available for this class.

3.1.25. Enumaration Classes:

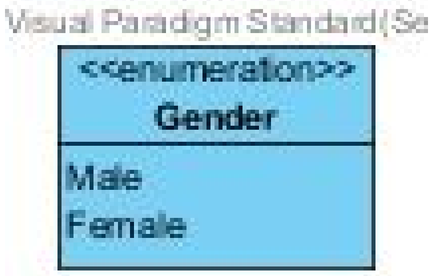
StatType



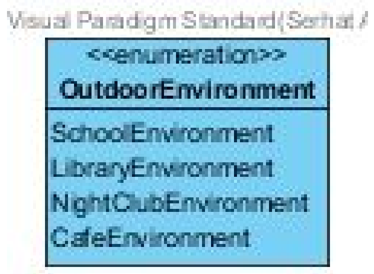
OptionType



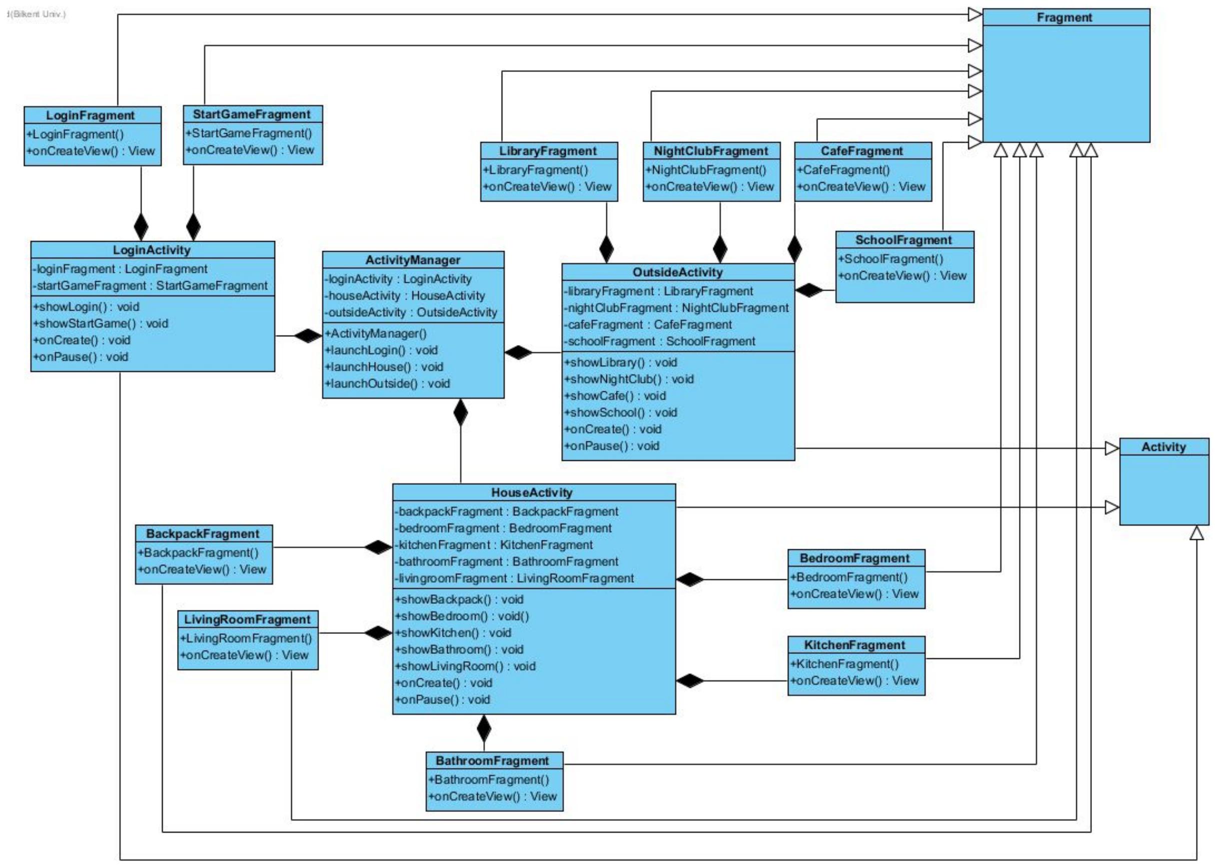
Gender



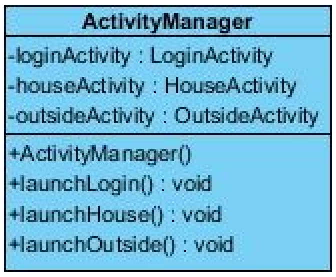
OutdoorEnvironment



3.2 User Interface(View)



3.2.1 ActivityManager Class



Attributes

private LoginActivity loginActivity: This is the activity which Roomie starts with.

private HouseActivity houseActivity: This is the activity which shows house and backpack of the player.

private OutsideActivity outsideActivity: This is the activity when an outside event is choosen.

Constructors

public ActivityManager: Initializes loginActivity, houseActivity and outsideActivity.

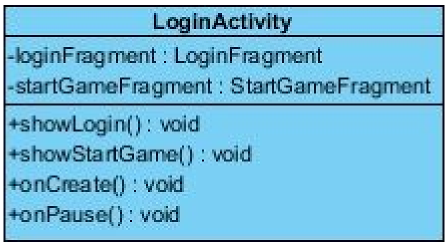
Methods

public void launchLogin(): This method lauches the login activity.

public void launchHouse(): This method lauches the house activity.

public void launchOutside(): This method lauches the outside activity.

3.2.2 **LoginActivity Class**



Attributes:

private LoginFragment loginFragment: This is the fragment when Roomie is first launched. It is constructed by some ui components such as buttons, labels and textareas.

private StartGameFragment startGameFragment: This is the fragment when Roomie is launched. It has Roomie's logo and a button to start the game.

Methods:

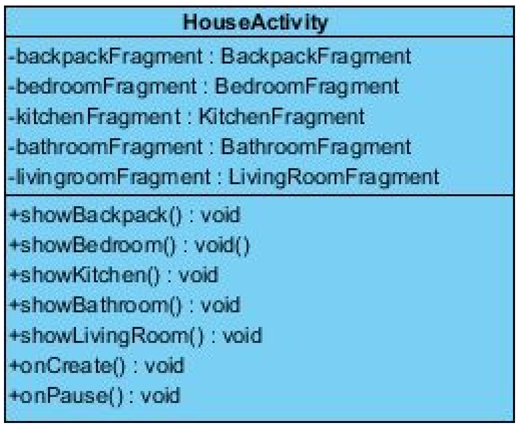
public void showLogin(): This method sends loginFragment to the front.

public void showStartGame(): This method sends startGameFragment to the front.

public void onCreate(): This is an overridden method from extended Activity class. It basically behaves like a constructor.

public void onPause(): This is an overridden method from extended Activity class. It pauses the activity when other activities is launched.

3.2.3 **HouseActivity Class**


HouseActivity
-backpackFragment : BackpackFragment
-bedroomFragment : BedroomFragment
-kitchenFragment : KitchenFragment
-bathroomFragment : BathroomFragment
-livingroomFragment : LivingRoomFragment
+showBackpack() : void
+showBedroom() : void()
+showKitchen() : void
+showBathroom() : void
+showLivingRoom() : void
+onCreate() : void
+onPause() : void

Attributes:

private BackpackFragment backpackFragment: This fragment shows player's backpack. The backback has some items in it. It is constructed by some buttons and a listview.

private BedroomFragment bedroomFragment: This fragment shows house's bedroom. It has some buttons with images to show items of the room.

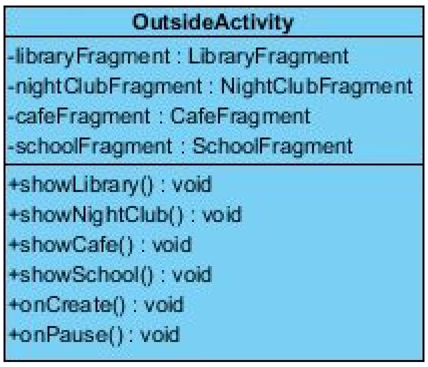
private KitchenFragment kitchenFragment: This fragment shows house's kitchen. It has some buttons with images to show items of the room.

private BathroomFragment bathroomFragment: This fragment shows house’s bathroom. It has some buttons with images to show items of the room.

private LivingRoomFragment livingroomFragment: This fragment shows house’s living room. It has some buttons with images to show items of the room.

Methods:
public void showBackpack(): This method sends backpackFragment to the front.
public void showBedroom(): This method sends bedroomFragment to the front.
public void showKitchen(): This method sends kitchenFragment to the front.
public void showBathroom(): This method sends bathroomFragment to the front.
public void showLivingRoom(): This method sends livingroomFragment to the front.
public void onCreate(): This is an overridden method from extended Activity class. It basically behaves like a constructor.
public void onPause(): This is an overridden method from extended Activity class. It pauses the activity when other activities are launched.

3.2.4 **OutsideActivity Class**


OutsideActivity
-libraryFragment : LibraryFragment
-nightClubFragment : NightClubFragment
-cafeFragment : CafeFragment
-schoolFragment : SchoolFragment
+showLibrary() : void
+showNightClub() : void
+showCafe() : void
+showSchool() : void
+onCreate() : void
+onPause() : void

Attiributes:
private LibraryFragment libraryFragment: This fragment shows library and it’s event. Library event requires player to stay quite in a some amount of time. It does not have additional UI components.

private NightClubFragment nightClubFragment: This fragment shows night club and it’s event. Night club event requires player to shake the device become successful. It does not have additional UI components.

private CafeFragment cafeFragment: This fragment shows cafe and it’s event. Cafe event requires player to choose a food item from the menu. The menu is constructed by using some labels, buttons and a listview.

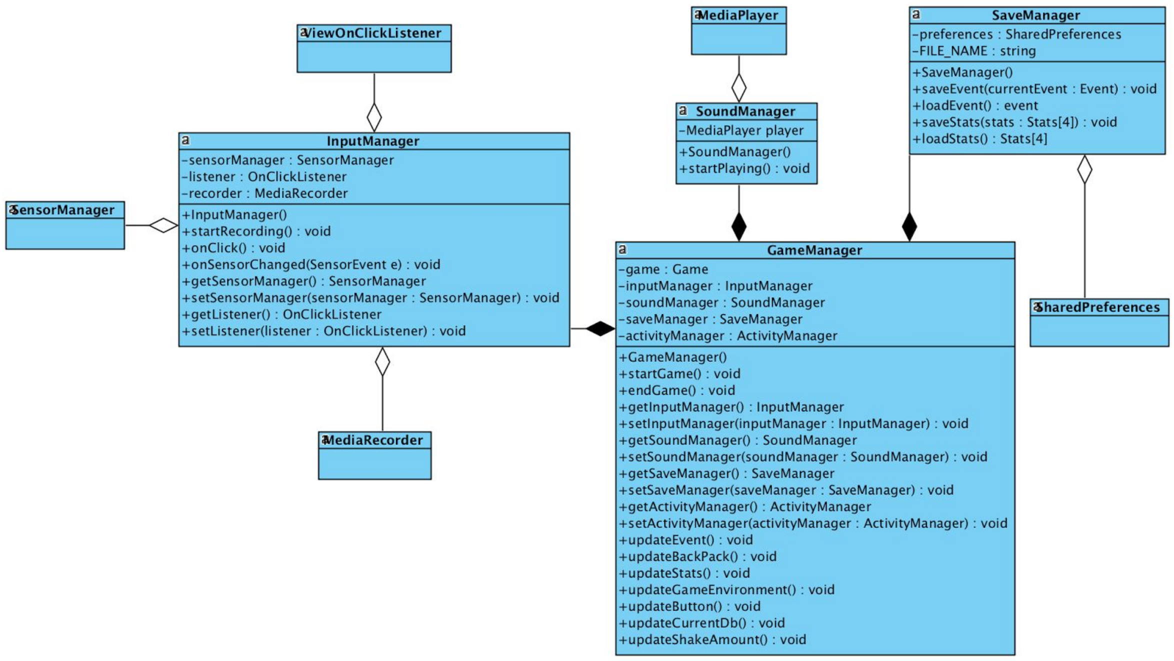
private SchoolFragment schoolFragment: This frahment shows school and it’s event. School event requires player to fill out a quiz. The quiz is constructed by using some labels and buttons.

Methods:
public void showLibrary(): This method sends libraryFragment to the front.
public void showNightClub(): This method sends nightClubFragment to the front.
public void showCafe(): This method sends cafeFragment to the front.
public void showSchool(): This method sends schoolFragment to the front.
public void onCreate(): This is an overridden method from extended Activity class. It basically behaves like a constructor.
public void onPause(): This is an overridden method from extended Activity class. It pauses the activity when other activities are launched.

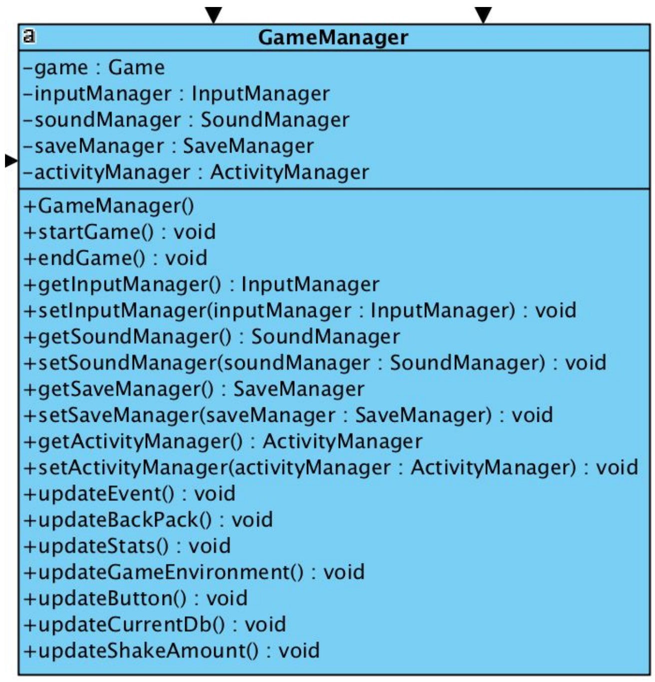
3.2.5 **Other Classes**

The other 11 classes are fragment classes. They have same functional properties. They are the subclass of the Android’s fragment class. They all have an empty constructor which is neccessary. They also have onCreateView method to initialize properties and create UI components.

3.3 Game Management(Controller)


ViewOnClickListener
InputManager
-sensorManager : SensorManager
-listener : OnClickListener
-recorder : MediaRecorder
+InputManager()
+startRecording() : void
+onClick() : void
+onSensorChanged(SensorEvent e) : void
+getSensorManager() : SensorManager
+setSensorManager(sensorManager : SensorManager) : void
+getListener() : OnClickListener
+setListener(listener : OnClickListener) : void
SensorManager
MediaRecorder
MediaPlayer
SoundManager
-MediaPlayer player
+SoundManager()
+startPlaying() : void
SaveManager
-preferences : SharedPreferences
-FILE_NAME : string
+SaveManager()
+saveEvent(currentEvent : Event) : void
+loadEvent() : event
+saveStats(stats : Stats[4]) : void
+loadStats() : Stats[4]
SharedPreferences
GameManager
-game : Game
-inputManager : InputManager
-soundManager : SoundManager
-saveManager : SaveManager
-activityManager : ActivityManager
+GameManager()
+startGame() : void
+endGame() : void
+getInputManager() : InputManager
+setInputManager(inputManager : InputManager) : void
+getSoundManager() : SoundManager
+setSoundManager(soundManager : SoundManager) : void
+getSaveManager() : SaveManager
+setSaveManager(saveManager : SaveManager) : void
+getActivityManager() : ActivityManager
+setActivityManager(activityManager : ActivityManager) : void
+updateEvent() : void
+updateBackPack() : void
+updateStats() : void
+updateGameEnvironment() : void
+updateButton() : void
+updateCurrentDb() : void
+updateShakeAmount() : void

3.3.1 GameManager Class


GameManager
-game : Game
-inputManager : InputManager
-soundManager : SoundManager
-saveManager : SaveManager
-activityManager : ActivityManager
+GameManager()
+startGame() : void
+endGame() : void
+getInputManager() : InputManager
+setInputManager(inputManager : InputManager) : void
+getSoundManager() : SoundManager
+setSoundManager(soundManager : SoundManager) : void
+getSaveManager() : SaveManager
+setSaveManager(saveManager : SaveManager) : void
+getActivityManager() : ActivityManager
+setActivityManager(activityManager : ActivityManager) : void
+updateEvent() : void
+updateBackPack() : void
+updateStats() : void
+updateGameEnvironment() : void
+updateButton() : void
+updateCurrentDb() : void
+updateShakeAmount() : void

Attributes:

private Game game: An instance of the game.

private InputManager inputManager: An instance of the inputmanager.

private SoundManager soundManager: An instance of the soundmanager.

private SaveManager saveManager: An instance of the savemanager.

private ActivityManager activityManager: An instance of the activitymanager.

Constructors:

public GameManager(): A constructor for initializing class properties.

Methods:

public void startGame(): This method starts the game.

public void endGame(): This method ends the game.

public void updateEvent():

public void updateBackPack():

public void updateStats():

public void updateGameEnvironment():

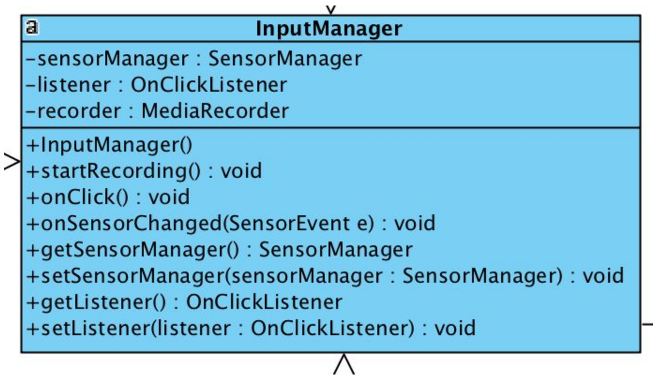
public void updateButton():

public void updateCurrentDb():

public void updateShakeAmount():

The other methods are the getters and setters.

3.3.2 InputManager Class



InputManager
-sensorManager : SensorManager
-listener : OnClickListener
-recorder : MediaRecorder
+InputManager()
+startRecording() : void
+onClick() : void
+onSensorChanged(SensorEvent e) : void
+getSensorManager() : SensorManager
+setSensorManager(sensorManager : SensorManager) : void
+getListener() : OnClickListener
+setListener(listener : OnClickListener) : void

Attributes

private SensorManager sensorManager: An instance of the sensormanager. This sensor provides gyroscope values to determine player’s shake.

private View.OnclickListener listener: An instance of the onclicklistener. This listener handles button clicks.

private MediaRecorder recorder: An instance of the mediarecorder. This recorder provides microphone input to calculate decibel level.

Constructors

public InputManager(): A constructor for initializing the properties.

Methods

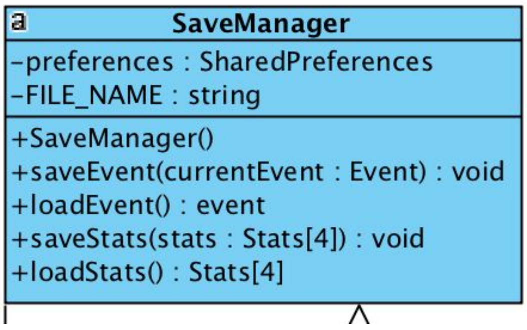
public void startRecording(): This methods starts the microphone for recording.

public void onSensorChanged(SensorEvent e): This method is called when gyroscope sensor is changed.

public void onClick(): This method handles the button clicks.

The other methods are the getters and setters.

3.3.3 SaveManager Class


SaveManager
-preferences : SharedPreferences
-FILE_NAME : string
+SaveManager()
+saveEvent(currentEvent : Event) : void
+loadEvent() : event
+saveStats(stats : Stats[4]) : void
+loadStats() : Stats[4]

Attributes

private SharedPreferences preferences: This is an Android class that helps saving and loading key values.

private String FILE_NAME: This is a constant file name to save and load the values.

Constructors

public SaveManager(): A constructor for initializing the properties.

Methods

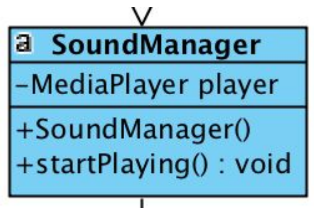
public void saveEvent(Event currentEvent): This method saves the current event of the game. This method is called when the application is stopped.

public Event loadEvent(): This method loads the current event of the game. This method is called when the application is launched.

public void saveStats(Stats [] stats): This method saves the stats of the player. This method is called when the application is stopped.

public Stats [] loadStats(): This method loads the stats of the player . This method is called when the application is launched.

3.3.4 SoundManager Class



Attributes

private MediaPlayer player: This is an Android class that plays sounds.

Constructors

public SoundManager(): A constructor for initializing the properties.

Methods

public void startPlayin(): This methods starts mediaplayer and the device starts to generate sounds.

3.4 Detaited System Design

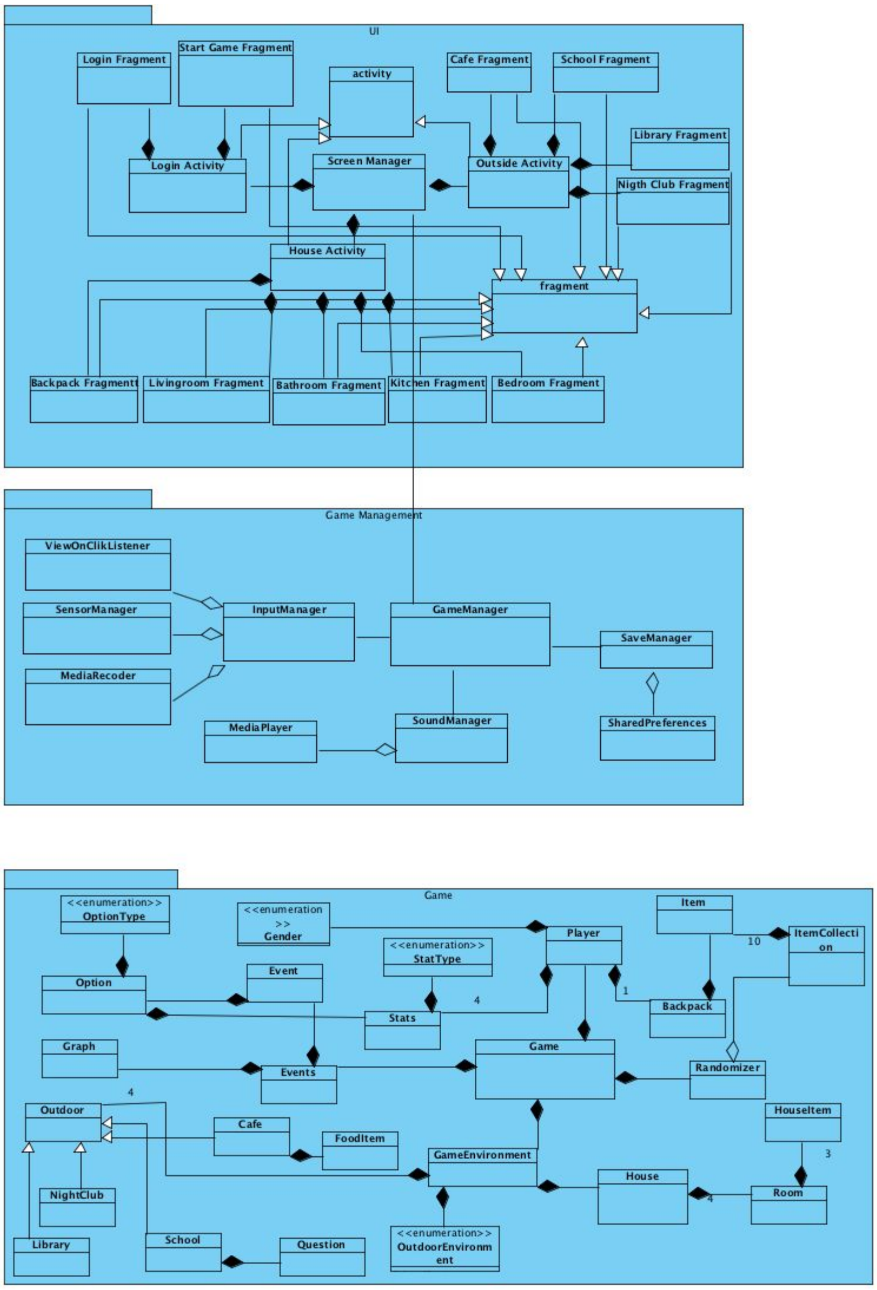


Figure 2: Detailed Subsystem Decomposition