  
  
  
  
CS 319 - Object-Oriented Software Engineering  
Design Report  
  
  
Project Name: Survival in Bilkent

Group 2-M

PelinElbinGünay - 21402149

Kübra Nur Güzel - 21400946

AlperŞahıstan - 21501207

SemihTeker– 21300964

Table of Contents

[1. Introduction 3](#_Toc496356686)

[1.1. Purpose of the system 3](#_Toc496356687)

[1.2 Design Goals 3](#_Toc496356688)

[End User Criteria: 3](#_Toc496356689)

[Maintenance Criteria: 3](#_Toc496356690)

[Performance Criteria: 3](#_Toc496356691)

[Trade Offs: 4](#_Toc496356692)

[1.3 Definitions, acronyms, and abbreviations 4](#_Toc496356693)

[2. Software Architecture 5](#_Toc496356694)

[2.1 Subsystem Decomposition 5](#_Toc496356695)

[2.2 Hardware / Software Mapping 7](#_Toc496356696)

[2.3 Persistent Data Management 7](#_Toc496356697)

[2.4 Boundary Condition 8](#_Toc496356698)

# 1. Introduction

## Purpose of the system

Survival in Bilkent is a top to down 2D shooter game. The purpose of the game is to entertain the player. In order to design more enjoyable game, we created a variety of enemies such as bugs, assignments, quizzes, labs, midterms, and finals. These are the obstacles that the players shoot. Additionally, we designed our game that includes 4 levels. These levels’ difficulty increases from first to last level. Because of that, the game is more enjoyable and challenging.

## 1.2 Design Goals

### End User Criteria:

**User-Friendliness:** The game can be played easily. In other words, the mechanics are generic and self-explanatory that seen in many 2-D shooter games. Additionally, we create an understandable user interface.

### Maintenance Criteria:

**Extensibility:** Reusability and extendibility are crucial for software projects. Especially, we will add new upgrade items to the project. To addition, Survival in Bilkent can be modified and re-used in another project easily.

**Portability:** Our game will work in different software environments and we will implement our project in Java because, it has JVM that provides an opportunity to work correctly.

### Performance Criteria:

**Game Performance**: The game should move instantly. Average FPS should be greater than 30. Otherwise, the game cannot be playable and enjoyable.

### Trade Offs:

**Performance vs. Memory**:Performance is important for our games. We want our game will run quickly. In order to moves our game smoothly, we should increase the memory space to give better game experience.

**Efficiency – Reusability:** We indicated that our game will be reusable. We will try to write the code as much reusable we can However, if the game’s efficiency is affected in a bad way, we will try to write the code more efficiently rather than reusability.

## 1.3 Definitions, acronyms, and abbreviations

MVC: Model View Controller

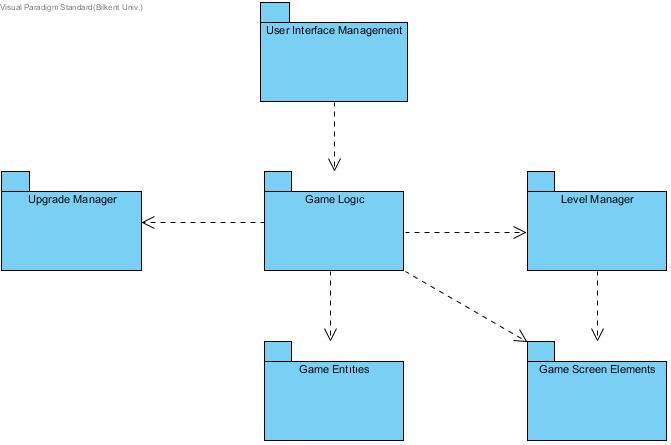
JDK: JavaDevelopment Kit

JVM: Java Virtual Machine

FPS: frames per second

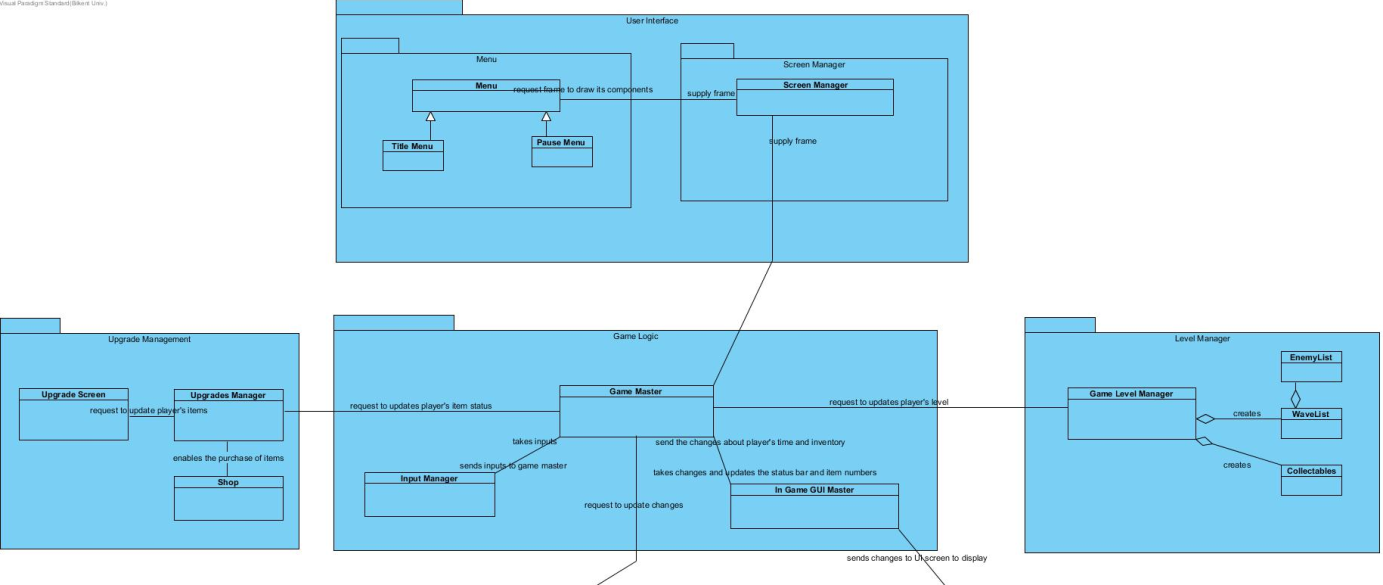
# 2. Software Architecture

## Subsystem Decomposition

We choose to construct a three-tier architectural decomposition for our projects system because it is the most suitable option for this game. Subsystem decomposition includes User Interface Management, Game Logic, Upgrade Manager, Level Manager, Game Entities, and Game Screen Elements.

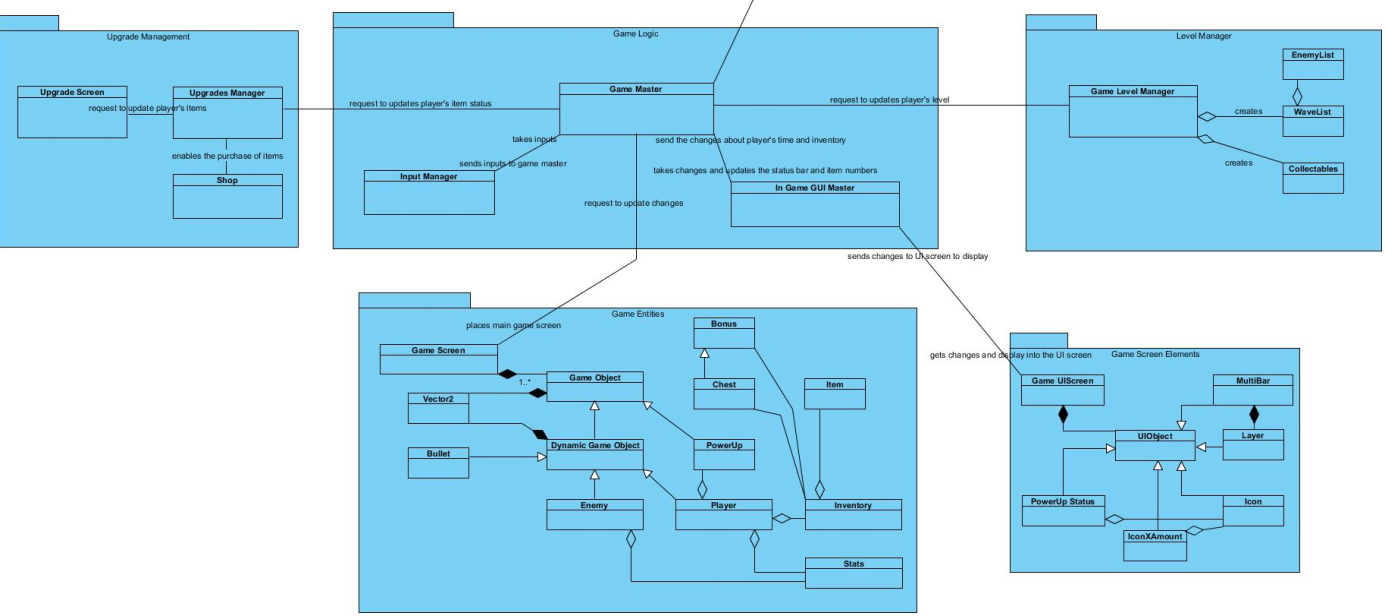
Subsystem Decomposition

In the first layer, presentation layer, we have classes responsible for presenting the interface to the user. Those classes will create the bridge between the user and the game system. In the User Interface Subsystem, there will be classes called Menu and Screen Manager, where Menu is the first interaction the user will come across. After the choices user makes, these options will be passed to the Game Logic subsystem.



In the logic layer which is the second layer of the subsystem, choices that user made in the first layer will be evaluated in the Game Logic subsystem. For example, if the user chooses the “Start Game” option, it will be constructed by the Game Master class. Also in this layer, Game Logic are in full association with Upgrade Manager and Level Manager subsystem. Level Manager is responsible for creating the layouts of the levels, then passing the created level to the Game Logic. This subsystem will update the information that passed to the Game Logic continuously due to the results of game objects.

Relation Between Layer 1 and Layer 2



Relation Between Layer 2 and Layer 3

In the data tier, third layer of the decomposition, utility classes will operate game entities and screen elements. Game Entities package contains the classes that are responsible of storing relevant information about objects in the game. Game Screen Elements on the other hand handles the In-Game User Interface that changes according to the actions occurring in the Game.

## Hardware / Software Mapping

Our game will require the Java Runtime Environment to be executed since it will be developed inJava. A keyboard and a mouse are required to play the game. In terms of graphical requirements of the program we are planning to use Slick 2D graphics library. An average computer standarts will be enough to handle the game.

## Persistent Data Management

Game data will be stored in the user’s local hard drive. Our game does not require any database system since the data that is used in the game need to be accessed in real time. Thus, all the necessary files and data will load on to the memory.

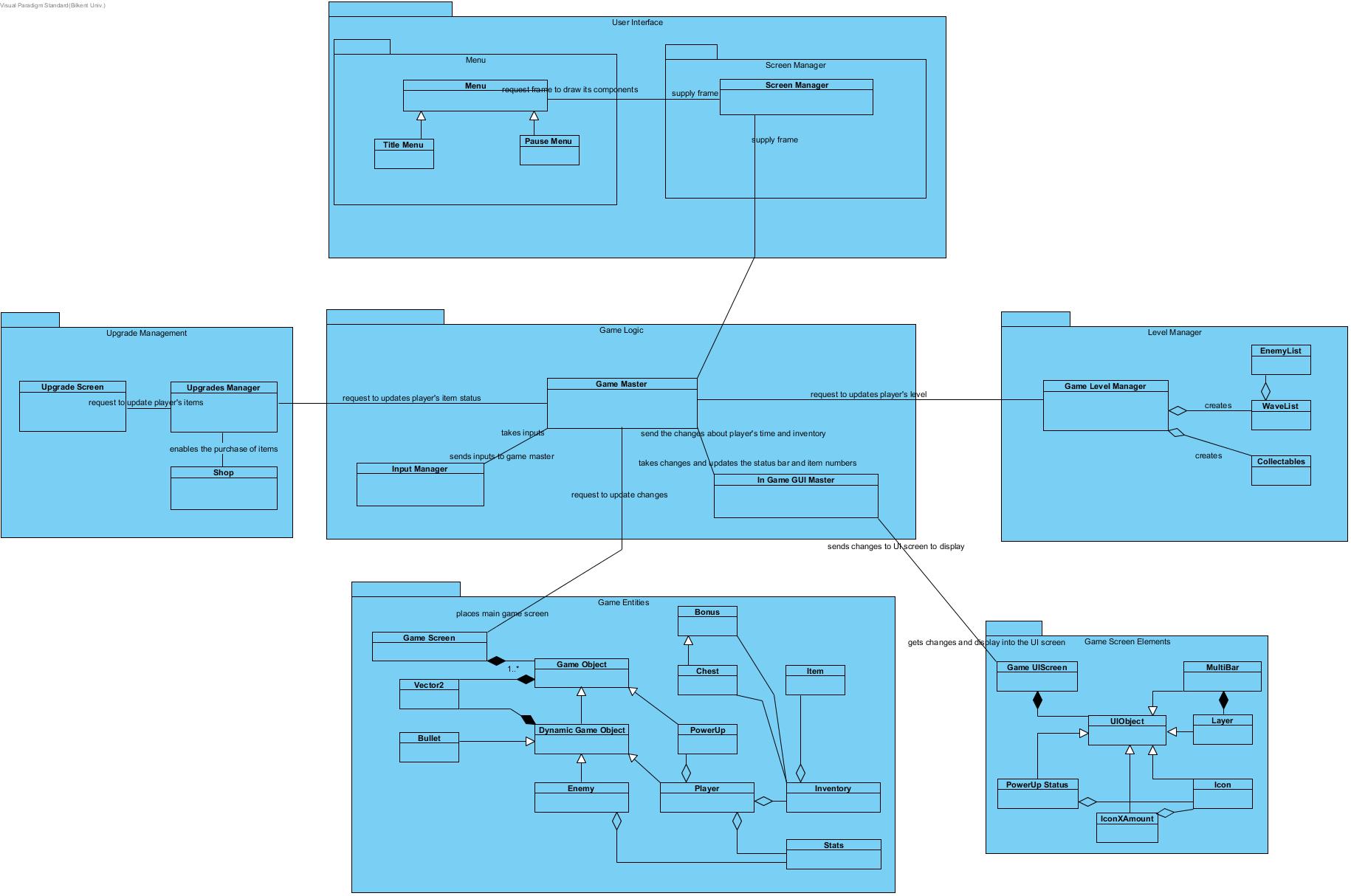
## Boundary Condition

The game will give an error if the file is corrupted and will delete its content. Game will return to the main menu if all lives of the player are gone. The game has finite number of levels so if the player will be able to complete the game will return to the main menu again.

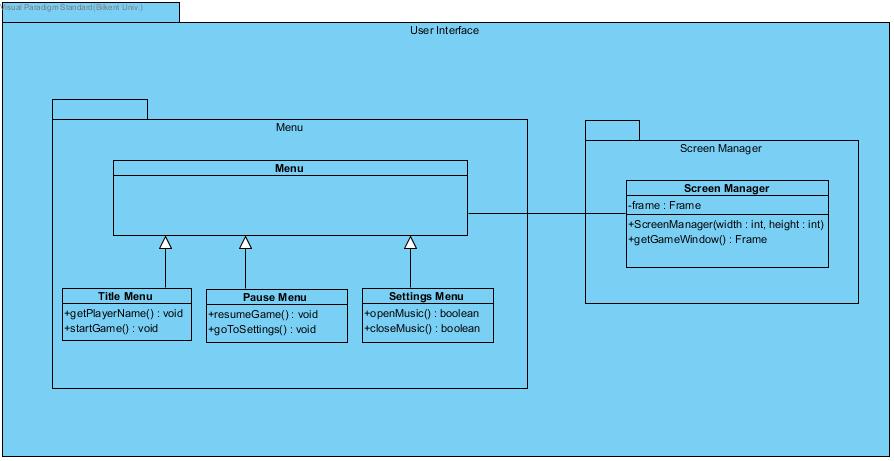
# Subsystem Services

## Detailed Object Design

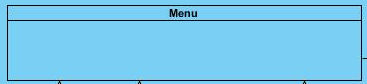
The overall class diagram will provide a better understanding of the subsystem and classes inside the packages. With the help of this diagram it will be easier to comprehend the design of our project.



# User Interface Subsystem



Menu Class

****

This class organizes Title Menu Class, Pause Menu Class, and Settings Menu Class.

Title Menu Class

****

*Methods:*

**public void getPlayerName():** This method gets the player’s name or nick-name. It does not return anything.

**public void startGame():** This method starts the game and opens the main game screen. It does not return anything.

**Pause Menu Class**

****

*Methods:*

**public void resumeGame():** This method stops the game. In other words, it stops the game flow. It does not return anything.

**public void goToSettings():** This method displays the setting menu to the user. It does not return anything.

Setting Menu Class

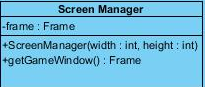
****

*Methods:*

**public boolean openMusic():** This method plays the audio. It returns true if the audio plays, otherwise it returns false if the audio could not be played.

**public boolean closeMusic():**This method stops the audio. It returns true if the audio stops, otherwise it returns false if the audio could not be stopped.

Screen Manager Class

****

*Attributes*:

**private Frame frame:** This is the frame of the screen where the game displays.

*Constructors:*

**public ScreenManager(width : int, height : int):** This constructor creates the screen according to the width and height in the parameter.

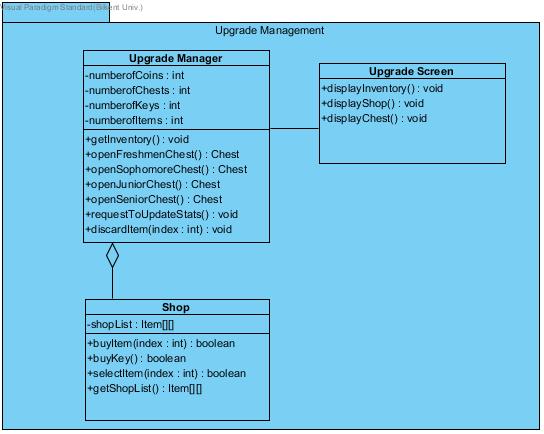
*Methods:*

**public Frame getGameWindow():** This method displays the game screen. It returns Frame of the game screen.

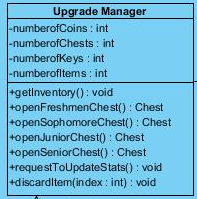
3.3 Upgrade Management Subsystem

Upgrade Management Subsystem is one of the major subsystems in the design process. This subsystem includes Upgrade Manager, Upgrade Screen and Shop classes. These classes have following functions;

* Upgrade Manager class handles the situation of player’s items.
* Upgrade Screen class displays the changes into the main screen.
* Shop class handles the purchase and removal of the items



Upgrade Manager Class

****

#### Attributes:

**private int credits**: It keeps the coin number that user has

**private int numberOfChests:** It keeps the chest number that user has

**private int numberOfKeys**: It keeps the key number that user has

#### Methods:

**public void getInventory():** This method gets player’s inventory. In other words, it shows how many keys, chests, coins does player has. It does not return anything.

**public Chest openFreshmenChest():**This method opens freshmen type of chest. To open freshmen chest, it is requires 1 key. It has 90% chance of giving a standard tier item, 7% chance of giving a rare tier item and 3% chance of giving an ultra-rare tier item. Therefore, according to these possibilities, this method gives the item and returns it.

**public Chest openSophomoreChest():**This method opens sophomore type of chest. To open sophomore chest, it is requires 2 key. It has 50% chance of giving a standard tier item, 30% chance of giving a rare tier item and 20% chance of giving an ultra-rare tier item. Therefore, according to these possibilities, this method gives the item and returns it.

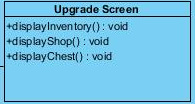
**public Chest openJuniorChest():**This method opens junior type of chest. To open junior chest, it is requires 3 key. It has 30% chance of giving a standard tier item, 35% chance of giving a rare tier item, 25% chance of giving an ultra-rare tier item and 10% chance of giving a “hacker” tier item.. Therefore, according to these possibilities, this method gives the item and returns it.

**public Chest openSeniorChest():**This method opens junior type of chest. To open junior chest, it is requires 3 key. It has 18% chance of giving a standard tier item, 25% chance of giving a rare tier item, 32% chance of giving an ultra-rare tier item and 25% chance of giving a “hacker” tier item.. Therefore, according to these possibilities, this method gives the item and returns it.

**public void requestToUpdateStats():**This method sends an upgrade request to game master in order to update the player’s stats. It does not return anything.

**public int discardItem(index : int):**This method is removes the item that right clicks on it and returns the index of this item.

Upgrade Screen Class

****

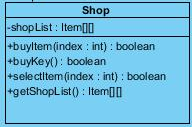
#### Methods:

**public void displayInventory():**This method displays the inventory of the player on the screen. It does not return anything.

**public void displayShop():** This method displays the shop on the screen in order to show the purchasable items from the shop. It does not return anything.

**public void displayChest():**This method displays the chests of the player on the screen. It does not return anything.

Shop Class

****

#### Attributes:

**private Item[] shopList:** It keeps the shop items into the one dimensional array with its index.

#### Methods:

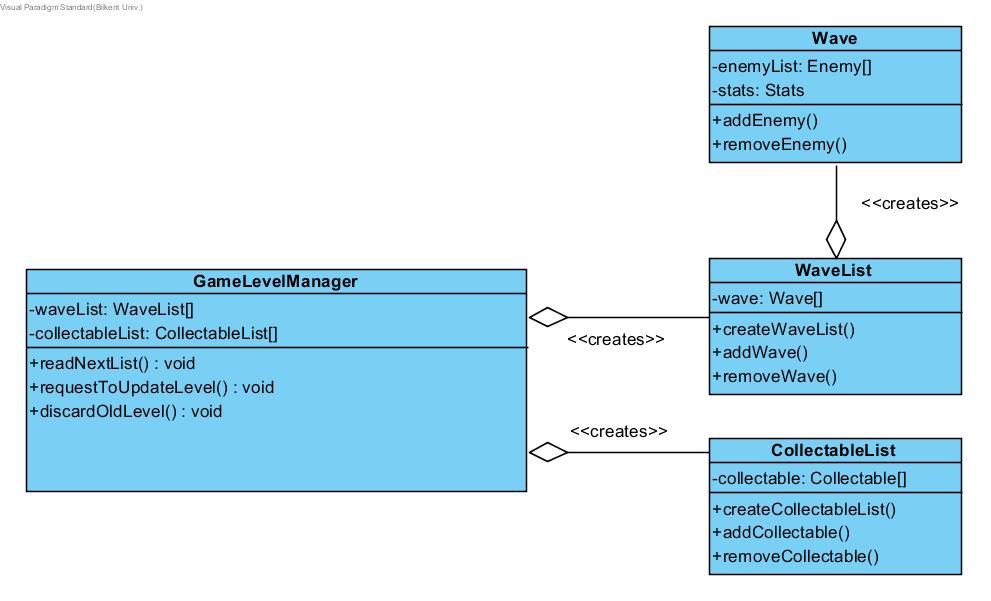
**public boolean buyItem(index : item**): This method buys the item and it returns true if the item is successfully bought or false if the item is not bought.

**public boolean buyKey():**This method buys the key in order to open a chest and it returns true if the key is successfully bought or false if the key is not bought.

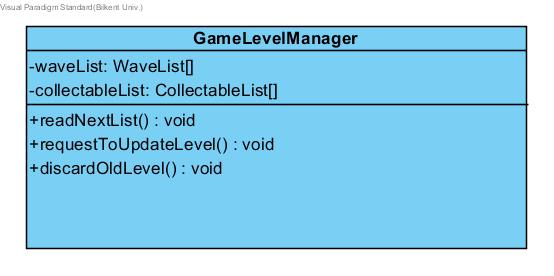
**public boolean selectItem(index : item):** This method selects the item whose index taken in the parameter and and it returns true if the item is successfully selected or false if the item is not selected.

**public Item[] getShopList():**This method gets the shop item’s list and returns them.

## 3.4 Level Manager Subsystem



### GameLevelManager Class

****

#### Attributes:

**waveList:** it is an instance of WaveList class and used for connecting wavelist class with the GameLevelManager.

**collectableList:** it is an instance of CollectableList class and used for connecting CollectableList class with the GameLevelManager.

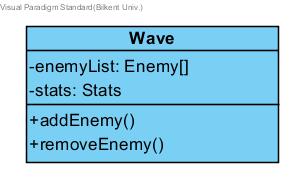
#### Methods:

**readNextList():** This method is used for reading from a text file to get the lists of objects. It used Java’s FileReader method.

**requestToUpdateLevel():**This method is for requesting to update the level from the GameMaster class in the GameLogic subsystem.

**discardOldLevel():** This method discards the level when the user completes of fails to complete a level.

### Wave Class



#### Attributes:

**enemyList:** This attribute is a list of Enemy objects.

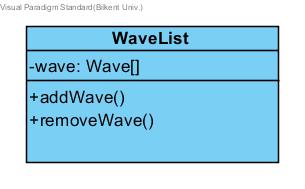
**stats:** This attribute is used to hold the stats of each enemy object.

#### Methods:

**addEnemy():** Adds an enemy object to the list.

**removeEnemy():** Removes an enemy object from the list.

### WaveList Class



#### Attributes:

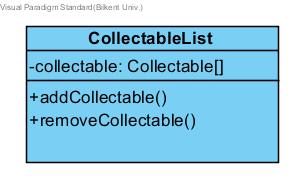
**wave:** This attribute is a list of Wave objects.

#### Methods:

**addWave():** Adds an enemy object to the list.

**removeWave():** Removes an enemy object from the list.

### CollectableList Class



#### Attributes:

**collectable:** This attribute is a list of Collectable objects.

#### Methods:

**addCollectable():** Adds a collectable object to the list.

**removeCollectable():** Removes acollectable object from the list.