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

1.1. 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 the 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 move 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: Java Development Kit

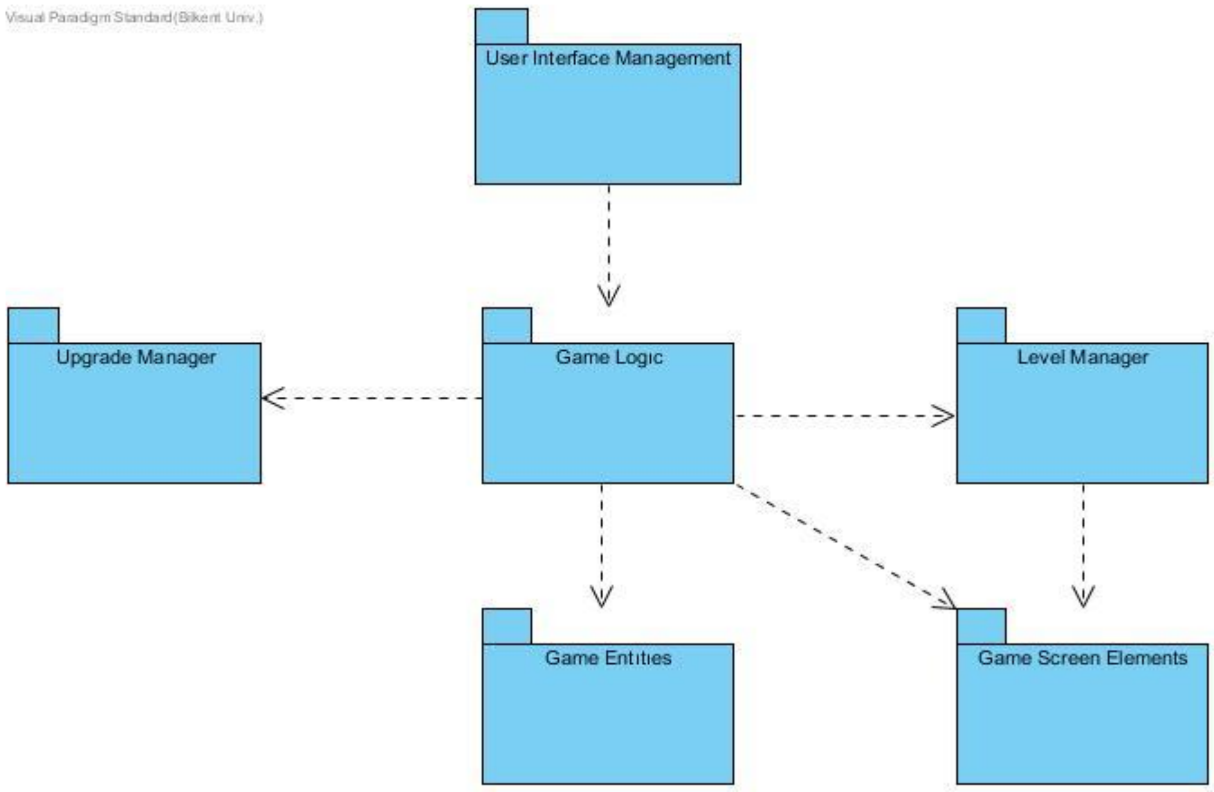
JVM: Java Virtual Machine

FPS: frames per second

2. Software Architecture

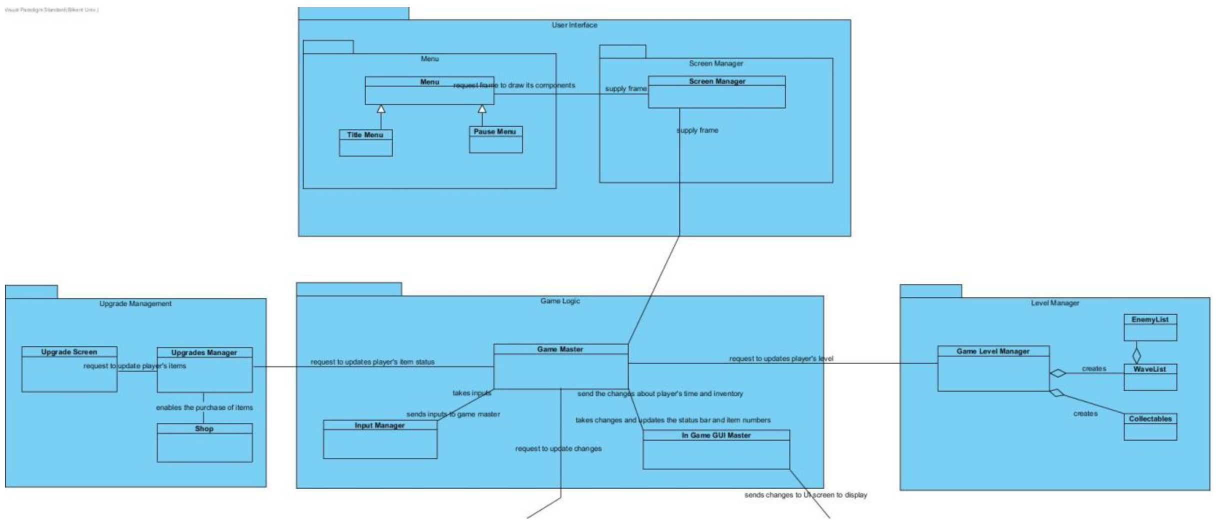
2.1 Subsystem Decomposition

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.



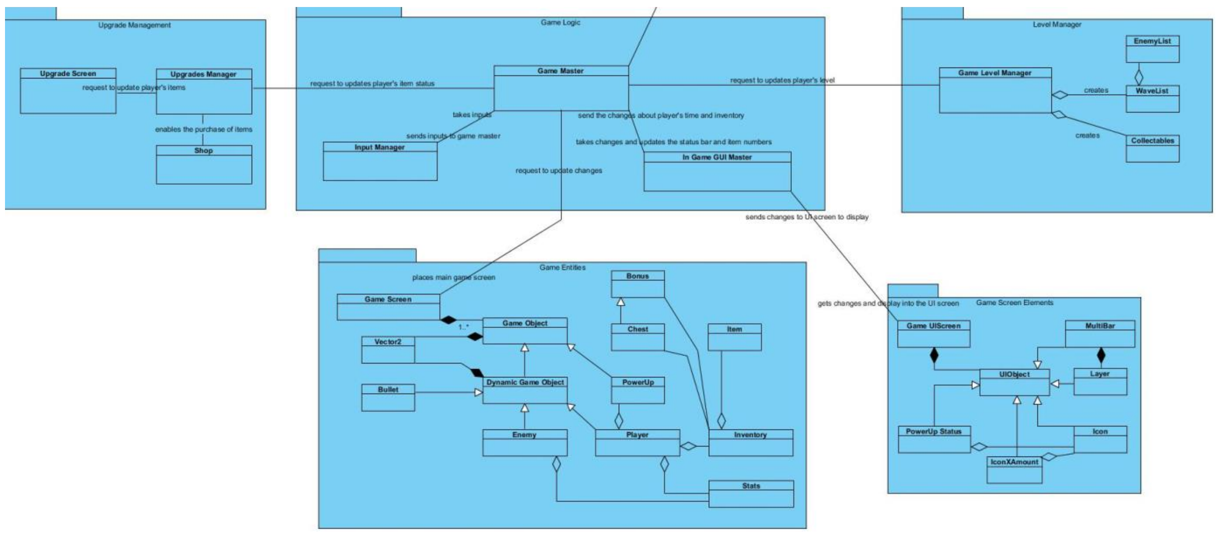
Subsystem Decomposition

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.


Visual Paradigm Standard(Bilkent Univ.)
User Interface
Menu
Menu
request frame to draw its components
Title Menu
Pause Menu
Screen Manager
supply frame
Screen Manager
supply frame
Upgrade Management
Upgrade Screen
request to update player's items
Upgrades Manager
enables the purchase of items
Shop
Game Logic
request to updates player's item status
Game Master
takes inputs
sends inputs to game master
Input Manager
request to update changes
send the changes about player's time and inventory
takes changes and updates the status bar and item numbers
In Game GUI Master
sends changes to UI screen to display
request to updates player's level
Level Manager
Game Level Manager
EnemyList
creates
WaveList
creates
Collectables

Relation Between Layer 1 and Layer 2

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 is 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 2 and Layer 3

In the data tier, the third layer of the decomposition, utility classes will operate game entities and screen elements. Game Entities package contains the classes that are responsible for 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.

2.2 Hardware / Software Mapping

Our game will require the Java Runtime Environment to be executed since it will be developed in Java. 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. Average computer standards will be enough to handle the game.

2.3 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 onto the memory.

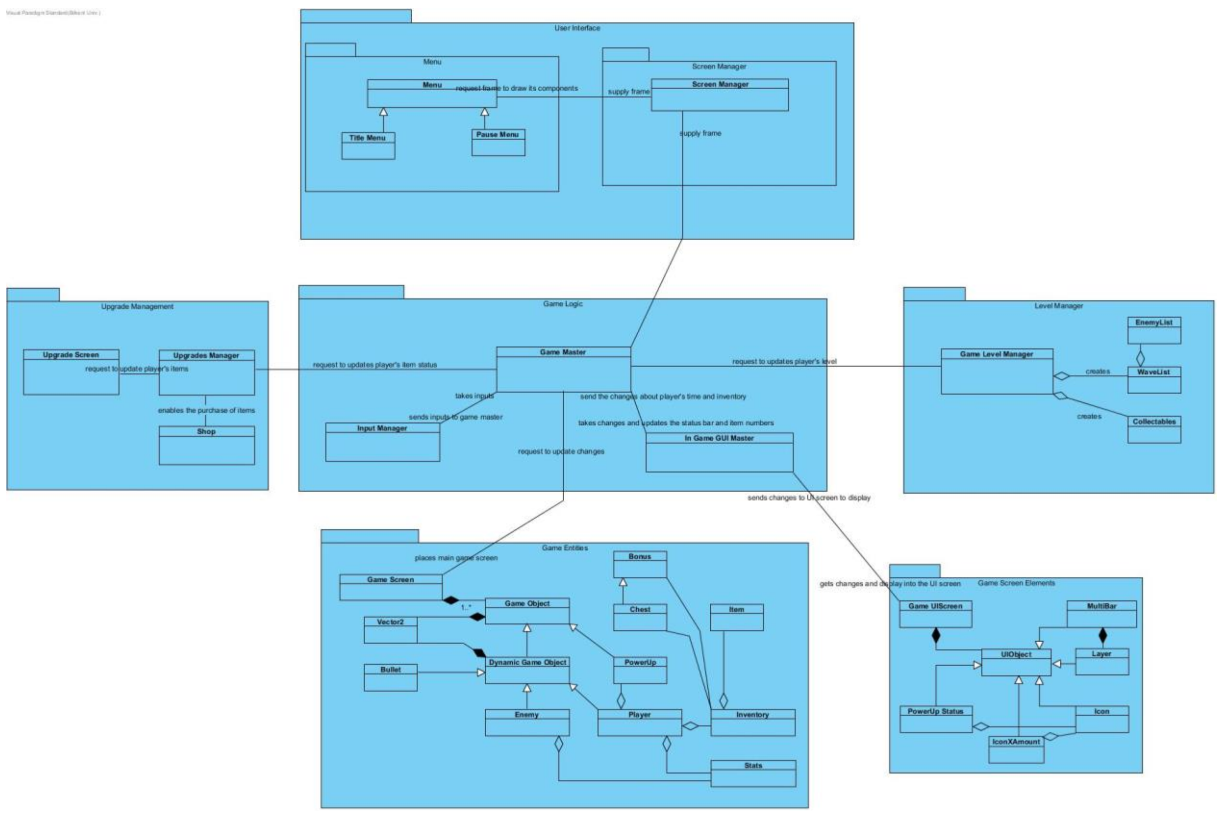
2.4 Boundary Condition

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

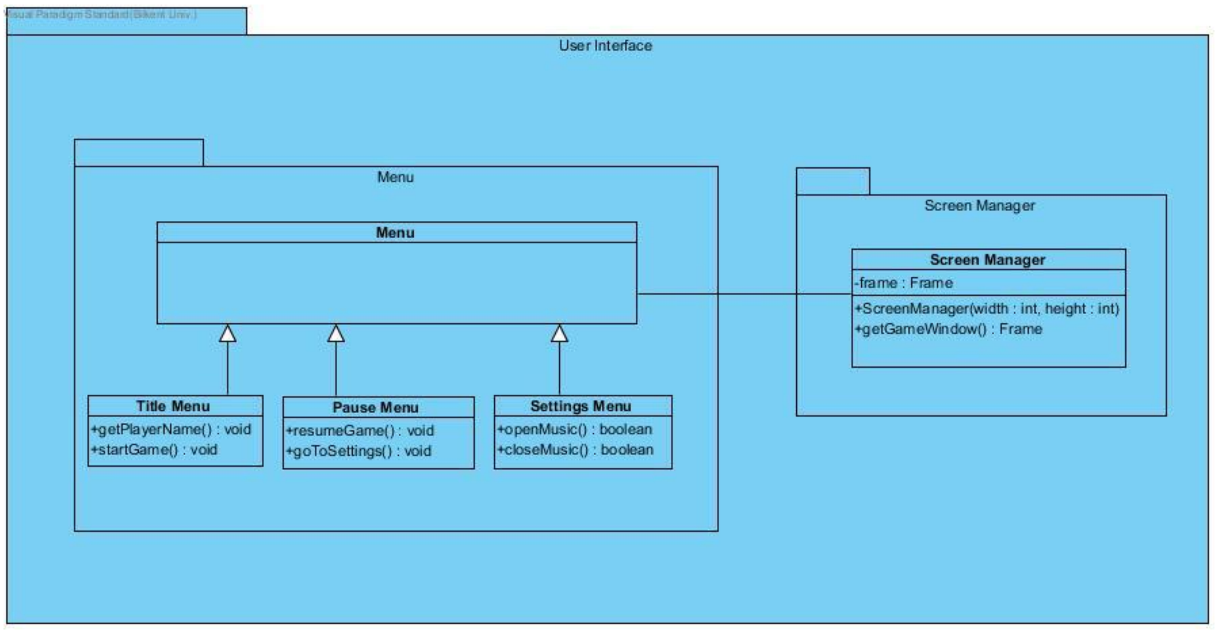
3 Subsystem Services

Detailed Object Design

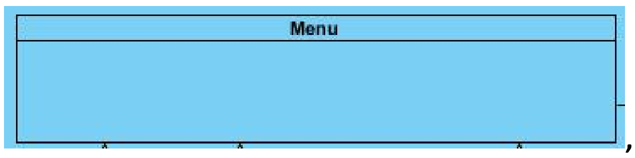
The overall class diagram will provide a better understanding of the subsystem and classes inside the packages. With the help of this diagram, it will be easier to comprehend the design of our project.



User Interface Subsystem



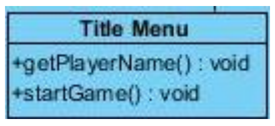
Menu Class



,

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

Title Menu Class

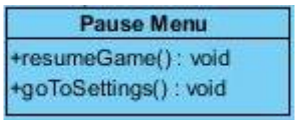


Methods:

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

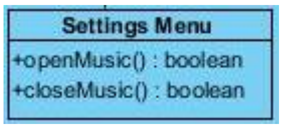


Methods:

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

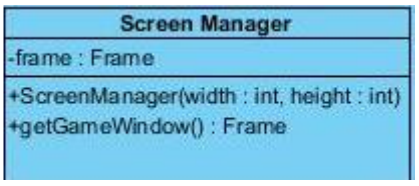


Methods:

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.

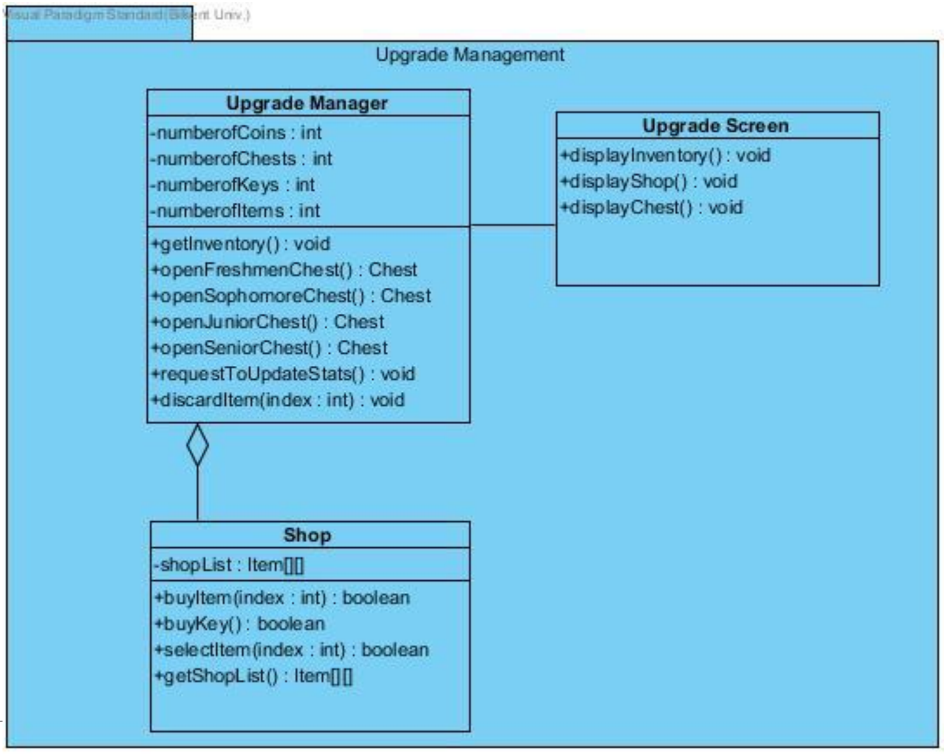
Methods:

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

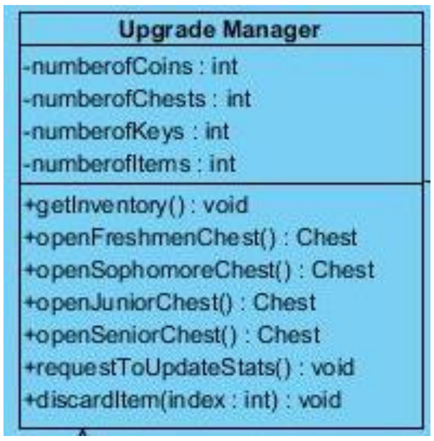
3.3 Upgrade Management Subsystem

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


Visual Paradigm Standard(Bilkent Univ.)
Upgrade Management
Upgrade Manager
-numberofCoins : int
-numberofChests : int
-numberofKeys : int
-numberofItems : int
+getInventory() : void
+openFreshmenChest() : Chest
+openSophomoreChest() : Chest
+openJuniorChest() : Chest
+openSeniorChest() : Chest
+requestToUpdateStats() : void
+discardItem(index : int) : void
Upgrade Screen
+displayInventory() : void
+displayShop() : void
+displayChest() : void
Shop
-shopList : Item[][]
+buyItem(index : int) : boolean
+buyKey() : boolean
+selectItem(index : int) : boolean
+getShopList() : Item[][]

Upgrade Manager Class


Upgrade Manager
-numberofCoins : int
-numberofChests : int
-numberofKeys : int
-numberofItems : int
+getInventory() : void
+openFreshmenChest() : Chest
+openSophomoreChest() : Chest
+openJuniorChest() : Chest
+openSeniorChest() : Chest
+requestToUpdateStats() : void
+discardItem(index : int) : void

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 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 a sophomore type of chest. To open the sophomore chest, it 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 a junior type of chest. To open the junior chest, it 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.

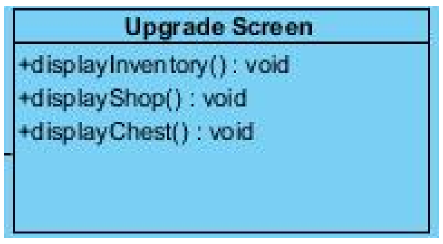
Therefore, according to these possibilities, this method gives the item and returns it.

public Chest openSeniorChest(): This method opens a junior type of chest. To open the junior chest, it 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 removes the item that right clicks on it and returns the index of this item.

Upgrade Screen Class



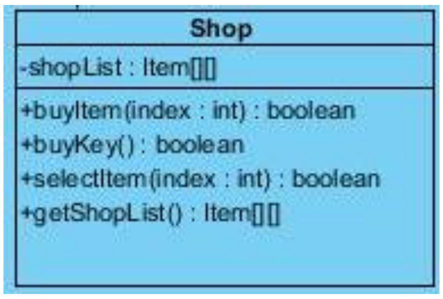
Methods:

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


Shop
-shopList : Item[][]
+buyItem(index : int) : boolean
+buyKey() : boolean
+selectItem(index : int) : boolean
+getShopList() : Item[][]

Attributes:

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

Methods:

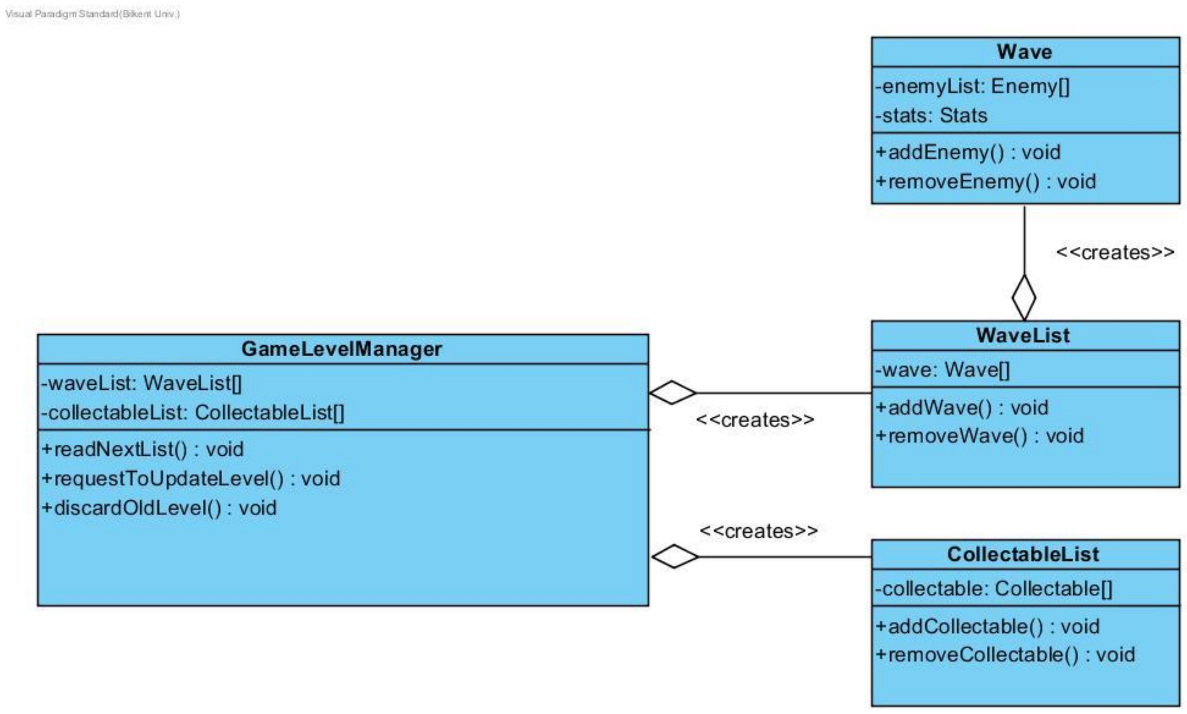
public boolean buyItem(index: item): This method buys the item and it returns true if the item is successfully bought or false if the item is not bought.

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.

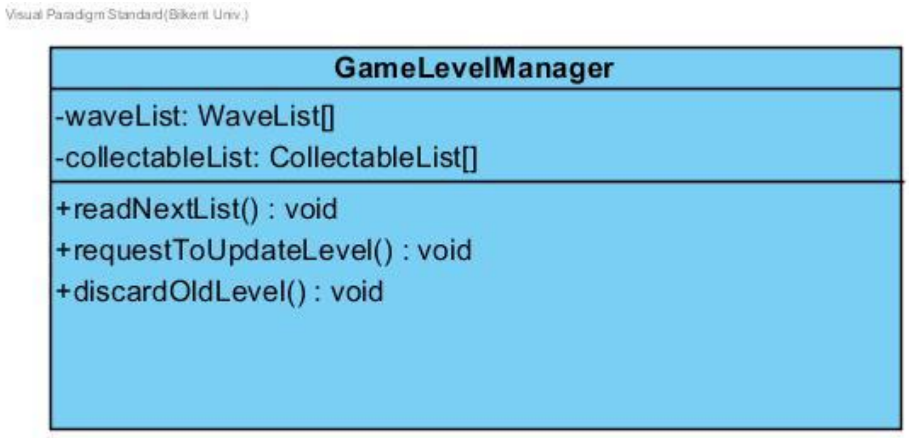
public boolean selectItem(index: item): This method selects the item whose index taken in the parameter and it returns true if the item is successfully selected or false if the item is not selected.

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

3.4 Level Manager Subsystem



GameLevelManager Class


Visual Paradigm Standard(Bilkent Univ.)
GameLevelManager
-waveList: WaveList[]
-collectableList: CollectableList[]
+readNextList() : void
+requestToUpdateLevel() : void
+discardOldLevel() : void

Attributes:

private WaveList[] waveList: it is an instance of WaveList class and used for connecting waveList class with the GameLevelManager.

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

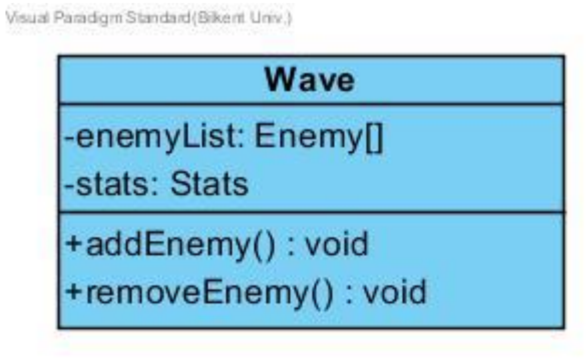
Methods:

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

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

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

Wave Class



Attributes:

private Enemy[] enemyList: This attribute is a list of Enemy objects.

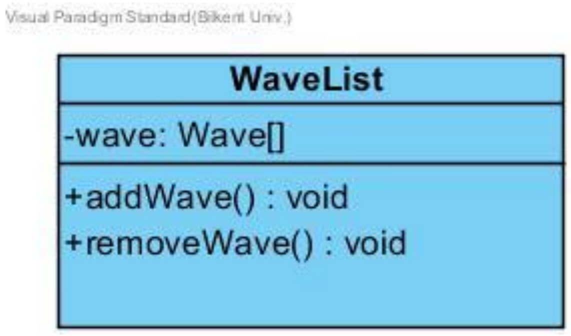
private Stats stats: This attribute is used to hold the stats of each enemy object.

Methods:

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

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

WaveList Class



Attributes:

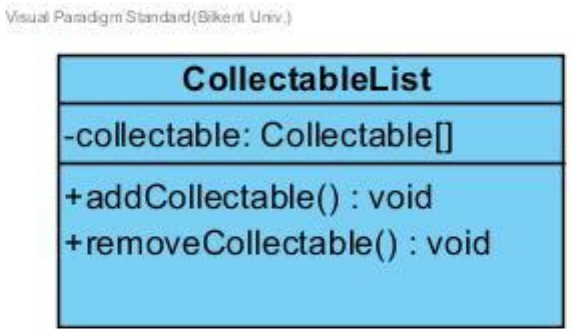
private Wave[] wave: This attribute is a list of Wave objects.

Methods:

public void addWave(): Adds an Enemy object to the list.

public void removeWave(): Removes an Enemy object from the list.

CollectableList Class


Visual Paradigm Standard(Bikent Univ.)
CollectableList
-collectable: Collectable[]
+addCollectable() : void
+removeCollectable() : void

Attributes:

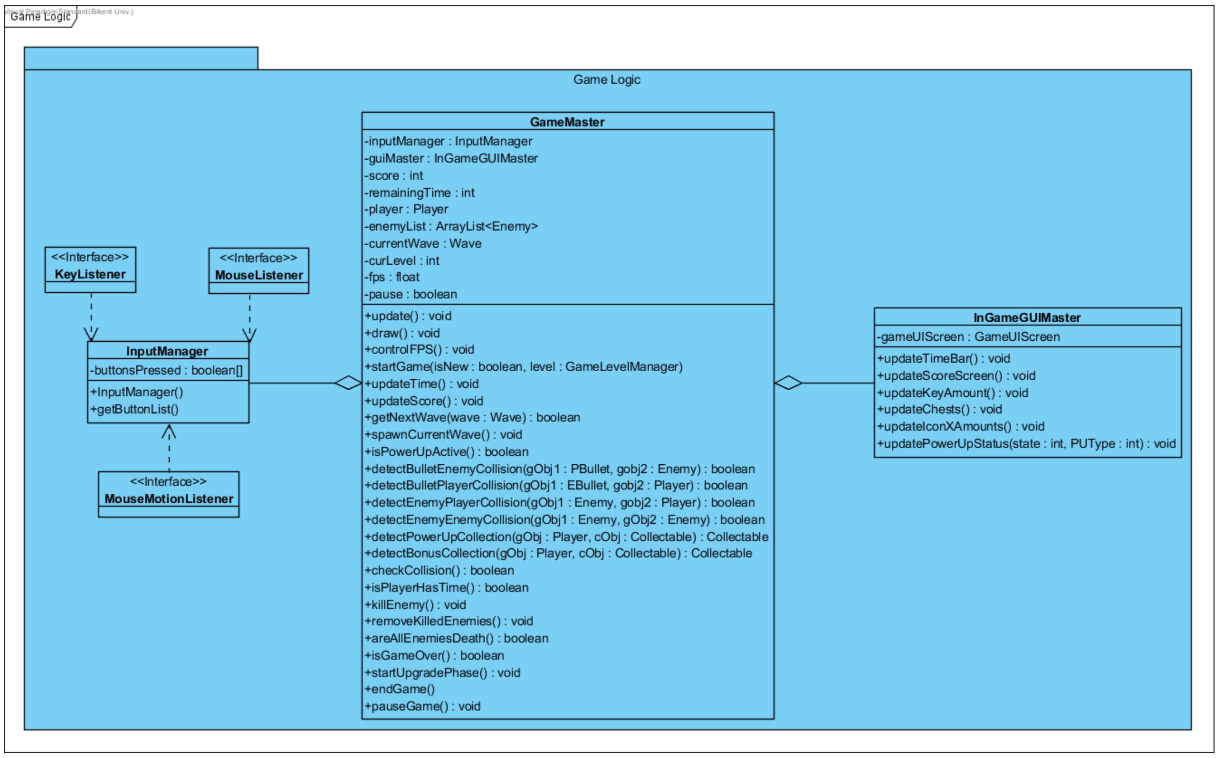
private Collectable[] collectible: This attribute is a list of Collectable objects.

Methods:

public void addCollectable(): Adds a collectible object to the list.

public void removeCollectable(): Removes a collectible object from the list.

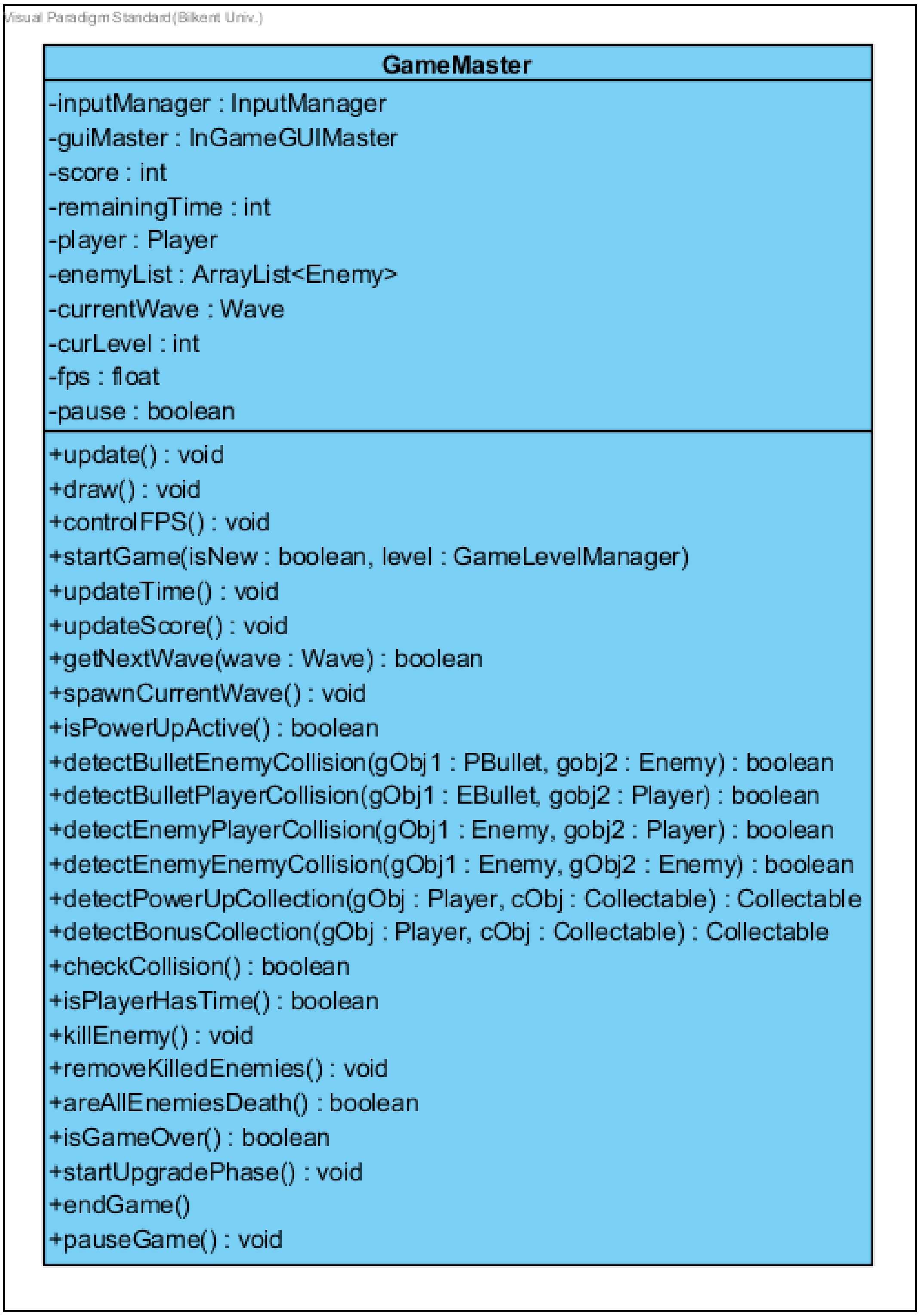
3.5 Game Logic Subsystem



Game Logic
Game Logic
GameMaster
-inputManager : InputManager
-guiMaster : InGameGUIMaster
-score : int
-remainingTime : int
-player : Player
-enemyList : ArrayList<Enemy>
-currentWave : Wave
-curLevel : int
-fps : float
-pause : boolean
+update() : void
+draw() : void
+controlFPS() : void
+startGame(isNew : boolean, level : GameLevelManager)
+updateTime() : void
+updateScore() : void
+getNextWave(wave : Wave) : boolean
+spawnCurrentWave() : void
+isPowerUpActive() : boolean
+detectBulletEnemyCollision(gObj1 : PBullet, gobj2 : Enemy) : boolean
+detectBulletPlayerCollision(gObj1 : EBullet, gobj2 : Player) : boolean
+detectEnemyPlayerCollision(gObj1 : Enemy, gobj2 : Player) : boolean
+detectEnemyEnemyCollision(gObj1 : Enemy, gObj2 : Enemy) : boolean
+detectPowerUpCollection(gObj : Player, cObj : Collectable) : Collectable
+detectBonusCollection(gObj : Player, cObj : Collectable) : Collectable
+checkCollision() : boolean
+isPlayerHasTime() : boolean
+killEnemy() : void
+removeKilledEnemies() : void
+areAllEnemiesDeath() : boolean
+isGameOver() : boolean
+startUpgradePhase() : void
+endGame()
+pauseGame() : void
<<Interface>>
KeyListener
<<Interface>>
MouseListener
InputManager
-buttonsPressed : boolean[]
+InputManager()
+getButtonList()
<<Interface>>
MouseMotionListener
InGameGUIMaster
-gameUIScreen : GameUIScreen
+updateTimeBar() : void
+updateScoreScreen() : void
+updateKeyAmount() : void
+updateChests() : void
+updateIconXAmounts() : void
+updatePowerUpStatus(state : int, PUType : int) : void

In the Game Logic subsystem, our controller objects are grouped together to manage the actual game dynamics and game logic. We have GameMaster, InGameGUIMaster and InputManager. These classes will be explained in detail, in this section.

GameMaster Class


Visual Paradigm Standard(Bilkent Univ.)
GameMaster
-inputManager : InputManager
-guiMaster : InGameGUIMaster
-score : int
-remainingTime : int
-player : Player
-enemyList : ArrayList<Enemy>
-currentWave : Wave
-curLevel : int
-fps : float
-pause : boolean
+update() : void
+draw() : void
+controlFPS() : void
+startGame(isNew : boolean, level : GameLevelManager)
+updateTime() : void
+updateScore() : void
+getNextWave(wave : Wave) : boolean
+spawnCurrentWave() : void
+isPowerUpActive() : boolean
+detectBulletEnemyCollision(gObj1 : PBullet, gobj2 : Enemy) : boolean
+detectBulletPlayerCollision(gObj1 : EBullet, gobj2 : Player) : boolean
+detectEnemyPlayerCollision(gObj1 : Enemy, gobj2 : Player) : boolean
+detectEnemyEnemyCollision(gObj1 : Enemy, gObj2 : Enemy) : boolean
+detectPowerUpCollection(gObj : Player, cObj : Collectable) : Collectable
+detectBonusCollection(gObj : Player, cObj : Collectable) : Collectable
+checkCollision() : boolean
+isPlayerHasTime() : boolean
+killEnemy() : void
+removeKilledEnemies() : void
+areAllEnemiesDeath() : boolean
+isGameOver() : boolean
+startUpgradePhase() : void
+endGame()
+pauseGame() : void

GameMaster class is the Façade class of the Game Logic subsystem, it performs the proper operations according to the requests that came from User Interface subsystem, and also this class runs the game in a loop.

Attributes:

private InputManager inputManager: this attribute is used for detecting user actions in the game.

private InGameGUIMaster guiMaster: this attribute is used for updating graphical user interface on the game screen.

private int score: it is used for player's score to represent the success in the game.

private int remainingTime: it is used for determining the game time representing both real-time and player's health.

private Player player: this attribute initializes a player object to use in the game.

private ArrayList<Enemy> enemyLists: this attribute initialize enemy list taking from the level subsystem.

private Wave currentWave: this attribute initialize a wave of the enemy by taking the enemy list when the player killed them

private int curLevel: this attribute holds the which level is called.

private float fps: this attribute takes the frame per second to determine the movements of game objects.

private boolean pause: this attribute is used for whether the game is paused or not.

Methods:

public void update(): runs a loop in which the system is updated continuously until the breakpoints(such as pause, game over, or finish game).

public void draw(): draws the all game objects and UI on the game screen according to the level manager.

public void controlFPS(): holds the average frame per second until the 30.

public void startGame(isNew: boolean, level: GameLevelManager): starts a new game by taking level information (to call level's enemy list).

public void updateTime(): updates the time by decreasing while the game processes and when the player takes damages.

public void updateScore(): updates the score when the player success to kill enemies.

public boolean getNextWave(wave: Wave): calls the new wave of enemies when all enemies in the previous wave are killed by the player.

public void spawnCurrentWave() : if getNextWave is true, this method creates current wave enemies.

public boolean isPowerUpActive(): checks whether the player has activated any power-ups or not.

public boolean detectBulletEnemyCollision(gObj1: PBullet, gobj2: Enemy): detects the Player Bullet and Enemy collision, in this type collision enemy takes damage and its health decreases. After the collision, bullet disappears.

public boolean detectBulletPlayerCollision(gObj1: EBullet, gobj2: Player): detects the Enemy Bullet and Player collision, in this type collision player takes damage and its health decreases. After the collision, bullet disappears.

public boolean detectEnemyPlayerCollision(gObj1: Enemy, gobj2: Player): detects the Enemy and Player collision, in this type collision both enemy and player takes damage and their health decreases.

public boolean detectEnemyEnemyCollision(gObj1: Enemy, gObj2: Enemy): detects the Enemy and Enemy collision, in this type collision both enemies hits and removes each other without any damages.

public Collectable detectPowerUpCollection(gObj : Player, cObj : Collectable) : detects the Player and Power-ups collision. After the collision, the player takes the power up and isPowerUpActive is returned true. And also, the power up disappears on the game arena then it goes to the power-up box on the corner of the game screen to be used later in the game.

public Collectable detectBonusCollection(gObj : Player, cObj : Collectable) : detects the Player and collectable item(key, chest or coin) collisions. After the collision player takes the items and one of these InGameGUIMsster methods (updateKeyAmount, updateChests, updateIconXAmounts) is called and this item is added player’s inventory. And also, the item disappears on the game arena.

public boolean checkCollision(): checks any collision, if there is any call the detectCollision methods and return true, else return false.

public boolean isPlayerHasTime(): checks whether the player has the time or not. If the time is over then returns false, otherwise, returns true.

public void killEnemy(): changes the death-flag to true if the enemy has no life.

public void removeKilledEnemies(): checks death-flags of all enemies and removes them if the flags are true.

public boolean areAllEnemiesDeath(): checks the death-flags of enemies in the list of the enemy, if all of them are true then this method is also return true, otherwise, return false.

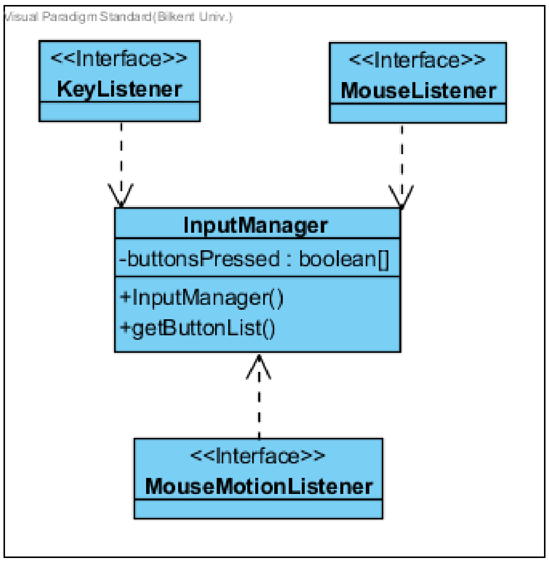
public boolean isGameOver() : checks the player has enough time (if isPlayerHasTime is true) or not. This method returns true if isPlayerHasTime is false, otherwise, returns false.

public void startUpgradePhase(): before each level, this method is called and upgrade screen appears.

public void endGame(): exits the game when the player presses the “exit the game” button.

public void pauseGame(): pauses the game and call the pause screen when the player presses the pause button.

InputManager Class


Visual Paradigm Standard(Bilkent Univ.)
<<Interface>> KeyListener
<<Interface>> MouseListener
InputManager
-buttonsPressed : boolean[]
+InputManager()
+getButtonList()
<<Interface>> MouseMotionListener

InputManager class is design to detect and designate the user actions performed by mouse (such actions to turn the barrel to shoot enemies) and keyboard (such actions to move the player in the game arena and to pause the game). Therefore, InputManager class implements MouseListener, MouseMotionListener and KeyListener interfaces of Java.

Attributes:

private boolean buttonsPressed: this attribute checks whether the specified buttons are pressed or not.

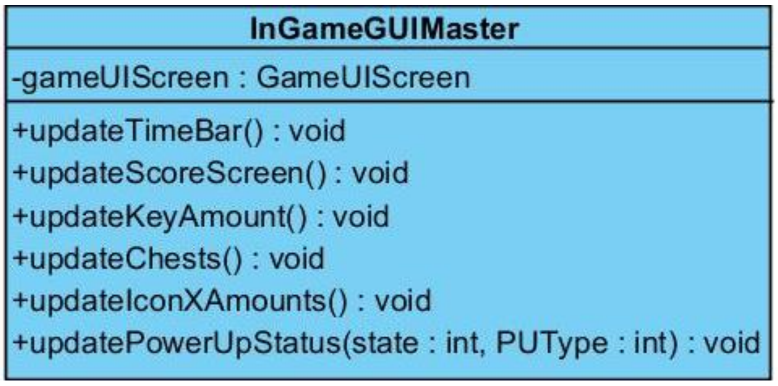
Constructors:

public InputManager(): it initializes the attributes of the GameManager for the first run of the system.

Methods:

public getButtonList(): this method specified the list of buttons in the game.

InGameGUIMaster Class


InGameGUIMaster
-gameUIScreen : GameUIScreen
+updateTimeBar() : void
+updateScoreScreen() : void
+updateKeyAmount() : void
+updateChests() : void
+updateIconXAmounts() : void
+updatePowerUpStatus(state : int, PUType : int) : void

Attributes:

private GameUIScreen gameUIScreen: User Interface screen for the in-game GUI. It allows the connection between the Game Screen Elements package's GameUIScreen class and Game Logic package's InGameGUIMaster.

Methods:

public void updateTimeBar(): Updates the time bar of the player depending on the events occurring in the Game Master.

public void updateScoreScreen(): Updates the score screen in the game depending on the player's score collection taken from the Game Master.

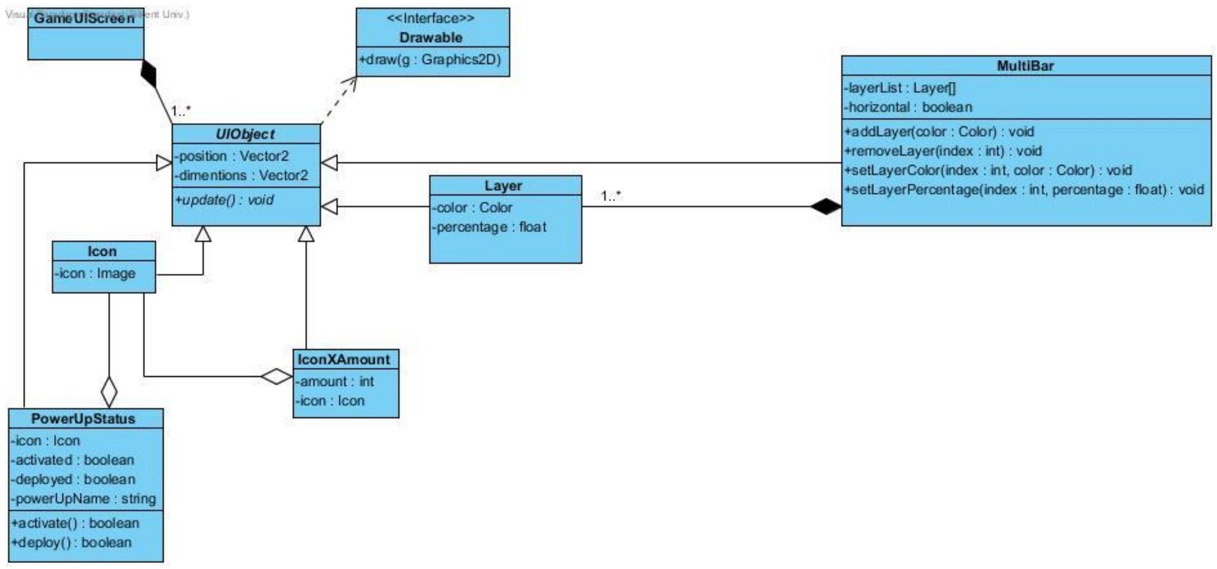
public void updateKeyAmount(): Updates a number of keys in the game depending on the Level Manager's Collectables class.

public void updateChests(): Updates the chests in the game depending on the Level Manager's Collectables class.

public void updateIconXAmounts(): Updates the number of Items (Number X "Icon of Object") in the user's inventory.

public void updatePowerUpStatus(state: int, PUType: int): Updates the current PowerUp icon on the screen by taking its states and types.

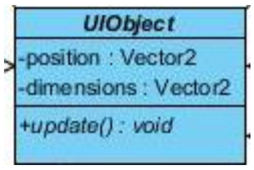
3.6 Game Screen Elements Subsystem



GameUIScreen Class

It holds the list of UI Objects.

UIObject Class



Attributes:

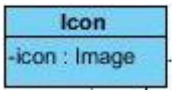
private Vector2 position: It keeps the position of the objects.

private Vector2 dimensions: It keeps the size of the objects.

Methods:

public void update():

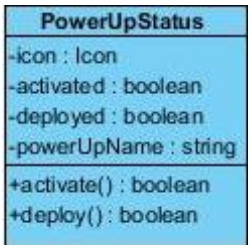
Icon Class



Attributes:

private Image icon: It holds an image of the icons.

PowerUpStatus Class



Attributes:

private Icon icon: It holds an icon of the power-ups.

private boolean activated: It holds whether the power-up is activated or not.

private boolean deployed: It holds whether the power-up is activated or not.

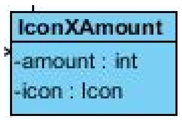
private string powerUpName: It holds power up's name.

Methods:

public boolean activate(): This method activates the power up and it returns whether the power-up is activated or not.

public boolean deploy(): This method deploys the power up and it returns whether the power-up is deployed or not.

IconXAmount Class

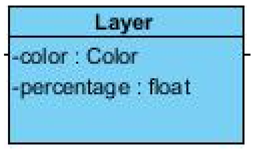


Attributes:

private Icon icon: It holds an icon the items.

private int amount: It holds an amount of the items.

Layer Class

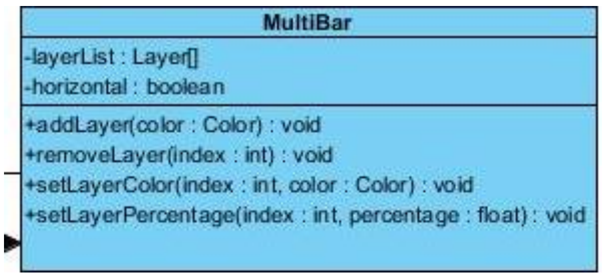


Attributes:

private Color color: It holds a color of the layer.

private float percentage: It holds a percentage of the layer.

Multibar Class


MultiBar
-layerList : Layer[]
-horizontal : boolean
+addLayer(color : Color) : void
+removeLayer(index : int) : void
+setLayerColor(index : int, color : Color) : void
+setLayerPercentage(index : int, percentage : float) : void

Attributes:

private Layer[] layerList: It holds a list of the layer.

private boolean horizontal: It holds the direction of the layer.

Methods:

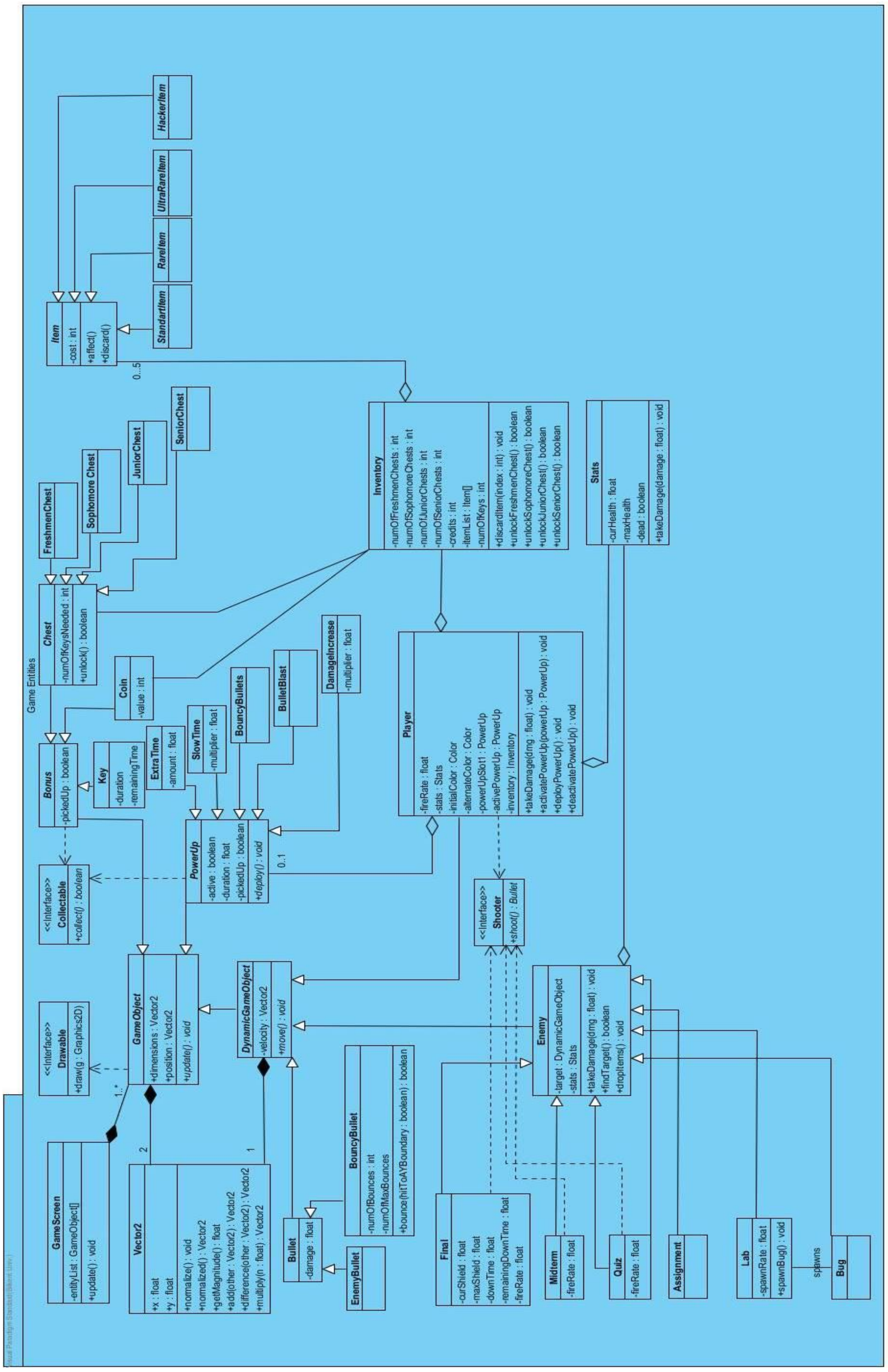
public void addLayer(color: Color): This method adds the new layer and it does not return anything.

public void removeLayer(Index: int): This method removes the existing layer and it does not return anything.

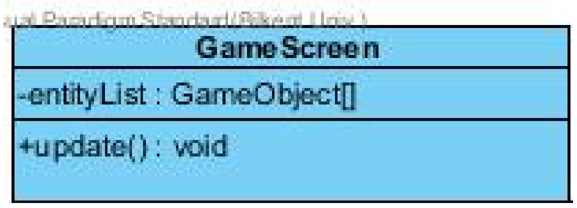
public void setLayerColor(Index: int, color: Color): This method sets the color of the existing layer and it does not return anything.

public float setLayerPercentage(Index: int, percentage: float): This method sets the percentage of the existing layer and it returns the percentage.

3.7 Game Entities Package



GameScreen Class



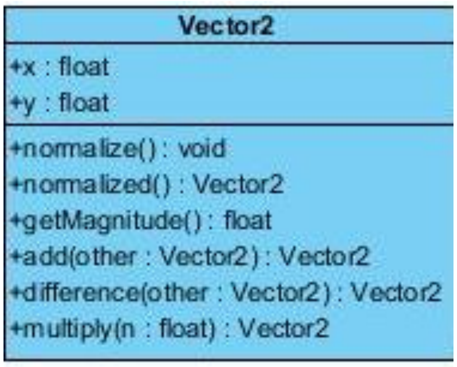
Attributes:

private ArrayList<GameObject> entityList: Holds the entity(GameObjects) objects of the game.

Methods:

public void update(): calls the update method of every entity object.

Vector2


Vector2
+x : float
+y : float
+normalize() : void
+normalized() : Vector2
+getMagnitude() : float
+add(other : Vector2) : Vector2
+difference(other : Vector2) : Vector2
+multiply(n : float) : Vector2

Attributes:

public float x: holds the x value to a 2D space.

public float y: holds the y value to a 2D space.

Methods:

public void normalize(): transforms this vector2 to a “unit” vector2.

public Vector2 normalized(): returns a unit vector2 with the same direction of this.

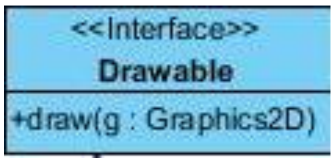
public float getMagnitude(): returns the magnitude ((x^2+y^2)^½) of this.

public Vetor2 add(Vector2 other): adds this Vector2 to other and returns a new Vector2.

public Vetor2 difference(Vector2 other): subtracts this vector2 from other and returns a new Vector2.

public Vector2 multiply(float n): Multiplies x and y with n and returns a new Vector2.

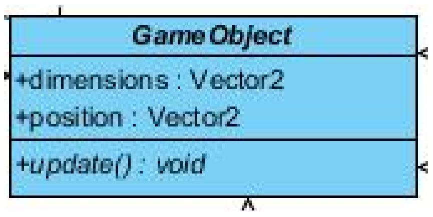
Drawable (interface)



Methods:

public void draw(): overridden by gameObjects a simple draw method to visualize objects.

GameObject Class (abstract)



Attributes:

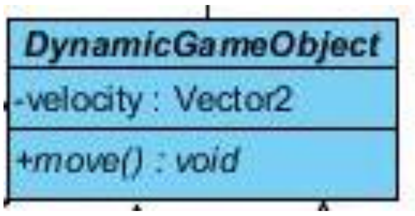
public Vector2 dimensions: rectangle sizes of a GameObject.

public Vector2 position: the center position of a GameObject.

Methods:

public void update(): updates the information accordingly to the internal commands.

DynamicGameObject (Abstract)



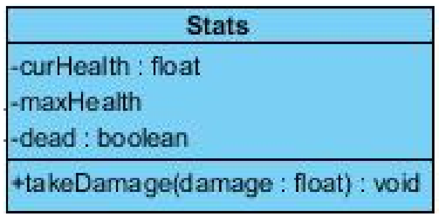
Attributes:

private Vector2 velocity: holds 2D velocity of a moving object.

Methods:

private void move(): updates the location by velocity.

Stats Class



Attributes:

private float curHealth:

private float maxHealth:

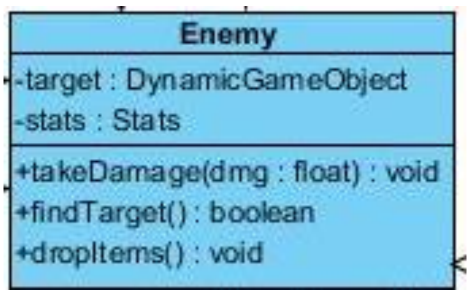
These are all values for health stats.

private boolean dead(): flag value for removal from the gameScreen.

Methods:

public void takeDamage(float damage): decrements the curHealth by damage. Checks dead flag true if curHealt<0.

Enemy Class



Attributes:

private DynamicGameObject target: the target of the enemy which will be attacked and possibly damaged.

private Stats stats: stats objects for the enemy.

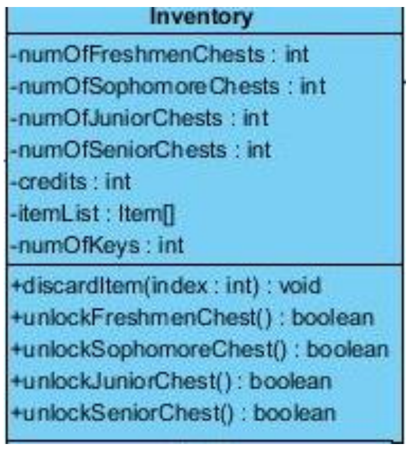
Methods:

public void takeDamage(float dmg): the amount of Damage that will be applied to Enemy if enemy is killed it will be marked for removal.(calls stats.takeDamage(dmg))

public boolean findTarget(): Enemy seeks target.

public void dropItems(): Enemy drops items before getting removed because of death.

Inventory Class


Inventory
-numOfFreshmenChests : int
-numOfSophomoreChests : int
-numOfJuniorChests : int
-numOfSeniorChests : int
-credits : int
-itemList : Item[]
-numOfKeys : int
+discardItem(index : int) : void
+unlockFreshmenChest() : boolean
+unlockSophomoreChest() : boolean
+unlockJuniorChest() : boolean
+unlockSeniorChest() : boolean

Attributes:

private int numOfFreshmenChests:

private int numOfSophomoreChests:

private int numOfJuniorChests:

private int numOfSeniorChests:

these specify the number of chests from each type which are collected and not opened.

private int credits: number of coins collected.

private Item[] itemList: 5 items that are owned by the player.

Private int numOfKeys:

Methods:

public void discardItem(int index): removes the item at the given index.

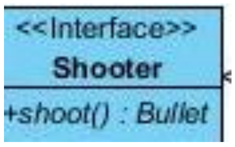
public boolean FreshmenChest(): instantiates and opens a freshmenChest.

public boolean SophomoreChest(): instantiates and opens a SophomoreChest.

public boolean JuniorChest(): instantiates and opens a JuniorChest.

public boolean SeniorChest(): instantiates and opens a SeniorChest.

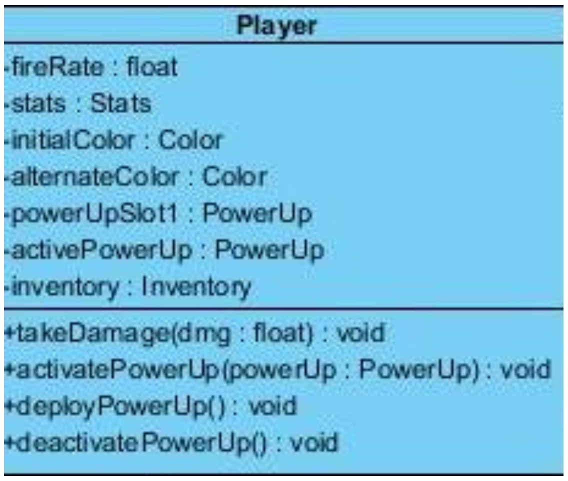
Shooter(interface)



Methods:

public Bullet shoot(): An abstract shoot method for shooting player and enemies.

Player Class


Player
-fireRate : float
-stats : Stats
-initialColor : Color
-alternateColor : Color
-powerUpSlot1 : PowerUp
-activePowerUp : PowerUp
-inventory : Inventory
+takeDamage(dmg : float) : void
+activatePowerUp(powerUp : PowerUp) : void
+deployPowerUp() : void
+deactivatePowerUp() : void

Attributes:

private float fireRate: To control the density of bullets.

private Stats stats: players stats as a Stats object.

private Color initialColor:

private Color alternateColor:

Color variables are supplied for a drawn method to indicate player objects state changes.

private PowerUp powerUpSlot1: holds the primary and only(unless there is an item that specifies an extra slot) power up.

private PowerUp activePowerUp: holds a reference to an active power-up since there might be another one supplied by an item.

private Inventory inventory: player's inventory that holds items and num of chests keys credits…

Methods:

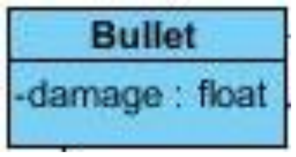
public void takeDamage(float dmg): the amount of Damage that will be applied to Player if player is killed it will be marked for removal.(calls stats.takeDamage(dmg))

public void activatePowerUp(PowerUp powerUp): activates the powerup specified in the parameter.

public void deployPowerUp(): deploys the activated powerUp.(calls activePowerUp.deploy()).

public void deactivatePowerUp(): deactivates the active power up.

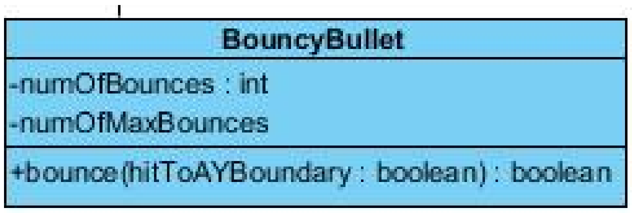
Bullet Class



Attributes:

private float damage: the amount of damage to be applied to the collided enemy.

BouncyBullet Class



Attributes:

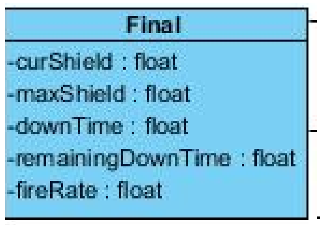
private int numOfBounces: counter for collisions with game Arena borders.

private int numOfMaxBounces: max number of bounces for a bullet.

Methods:

public boolean bounce (boolean hitToAYBoundary): First checks if numOfBounces<=macNumOfBounces then it inverts the bouncy bullet's velocity's x value if the parameter is false else it inverts y value.

Final Class



Attributes:

private float curShield: holds the current value of the shield.

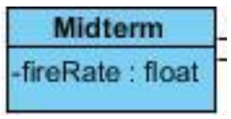
private float maxShield: holds the maximum value of the shield.

private float downTime: holds the initial value to be counted down from before the shield is back up.

private float remainingDownTime: holds the countdown time for shields to be back up.

private float fireRate: To control the density of bullets.

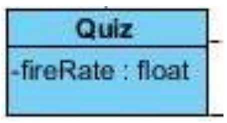
Midterm Class



Attributes:

private float fireRate: To control the density of bullets.

Quiz Class



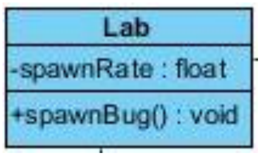
Attributes:

private float fireRate: To control the density of bullets.

Assignment Class

Assignment

Lab Class



Attributes:

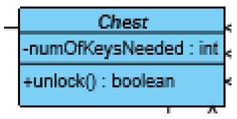
private float spawnRate: To control the density of “bug” objects spawned by the lab.

Methods:

public spawnBug(): spawns bugs to attack player

Bug Class

Chest Class



Attributes:

Private int numOfKeysNeed: It holds a number of the keys needed.

Methods:

Public boolean unlock():This method unlocks the chest and it returns whether the chest is unlocked or not.

Freshmen Chest Class

This class constructs the freshmen chest object.

Sophomore Chest Class

This class constructs the sophomore chest object.

Junior Chest Class

This class constructs the junior chest object.

Senior Chest Class

This class constructs the senior chest object.

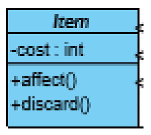
Coin Class

Coin
-value : int

Attributes:

Private int value: It holds a number of coins.

Item Class



Attributes:

Private int cost: It holds the cost of the item.

Methods:

Public void affect(): It sends the effect to the game master.

Public void discard(): It removes the item**.**

Standard Item Class

This class constructs the standard items

Ultra Rare Item Class

This class constructs the ultra-rare items

Rare Item Class

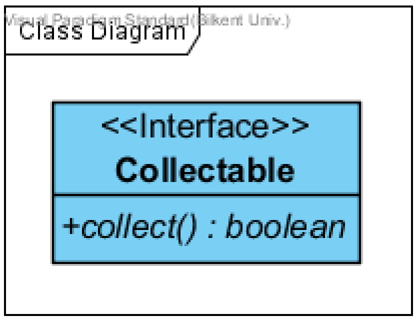
This class constructs the rare items

Hacker Item Class

HackerItem

This class constructs the hacker items.

Collectable Interface

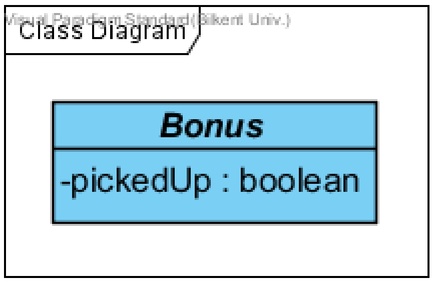


Collectable is an interface for all collectible items in the game. These are Bonus and PowerUps.

Methods:

public Boolean collect(): is called when the player collides any Collectable items in the game.

Bonus Class

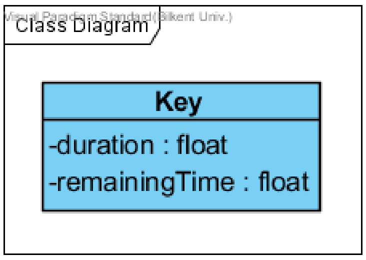


Bonus Class is a parent class and have three children classes are Chest, Coin, and Key.

Attributes:

private boolean pickedUp: is determine whether the Bonus item is picked up by player or not.

Key Class

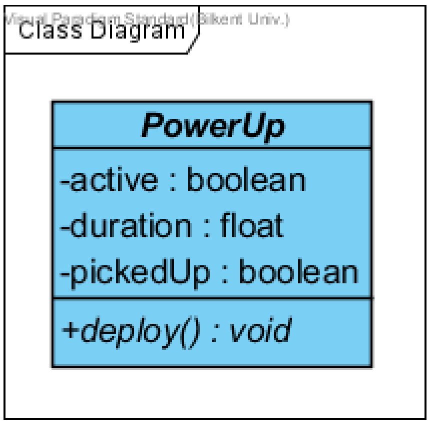


Attributes:

private float duration: it is specified the duration of existence of a key in the game.

private float remainingTime: it is specified the remaining time from the duration of the key.

PowerUp Class



PoverUp Class is a parent class and has five children classes are ExtraTime, SlowTime, BouncyBullets, BulletBlast, and DamageIncrese.

Attributes:

private boolean active: it specifies whether the power-up is active or not.

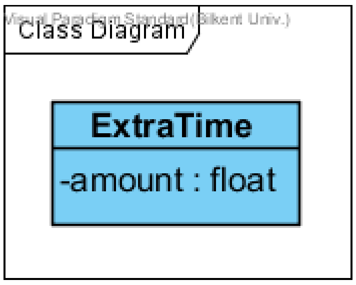
private float duration: it specifies the duration of existence of power-up in the game.

private float pickedUp: it specifies whether the power-up is picked-up by player or not

Methods:

public void deploy(): this method deploys the power-up when the player collides power-up to use in the game.

ExtraTime Class

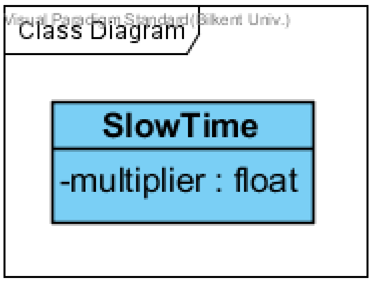


ExtraTime Class adds the extra time to player's game-time

Attributes:

private float amount: it specifies the amount of time which is added to the player.

SlowTime Class

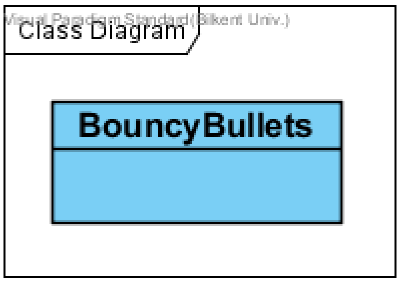


SlowTime Class slows the time for all game objects.

Attributes:

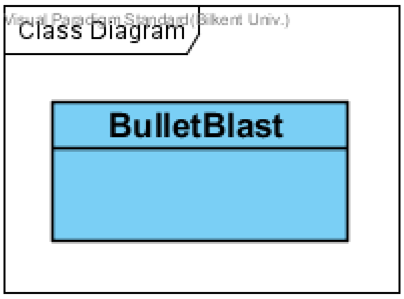
private float multiplier: it specifies the ratio of the slowing time.

BouncyBullets Class



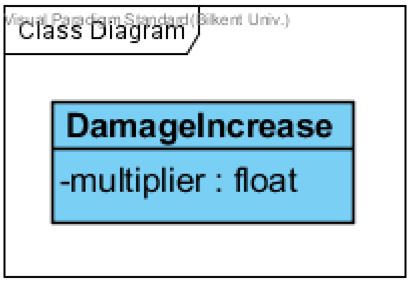
BouncyBullets Class replaces the bullets with the bouncy bullets which bounce off (3 times per bullet) from the borders of the game are instead of simply going out.

BulletBlast Class



BulletBlast Class sends out a circular group of bullets originating from the player.

DamageIncrease Class



DamageIncrease Class makes Player's bullets harder for 5 seconds.

Attributes:

private float multiplier: it specifies the ratio of the increasing damage.