Bilkent University

Department of Computer Engineering

CS 319 – Object Oriented Software Engineering

CS 319 Project: Catch the Deadline

Design Report

GROUP 2D
Argert Boja, Bikem Çamli, Hareem Larik, Umut Ak

Course Instructor: Bora Güngören Progress/Final Report

Design Draft Report October 21, 2017

Contents

1. Introduction

1.1 Purpose of the System

“Catch the Deadline” is a 2D arcade-style platformer game. This game is designed to entertain the players, hence in order to do this we have included a variety of power-ups to collect and various obstacles to overcome. In addition, we designed our leveling system in difficulty order to make player’s gaming experiences more enjoyable. Therefore, the main purpose of the “Catch The Deadline” game is to make players have a nice time while playing this game.

The game will be implemented using Java and is designed for PC. In the following sections of this report, we will provide the goals of the design, tradeoffs and different diagrams about the System Decomposition of the game. Furthermore, the class diagram and explanation for each class will also be described in the following sections. This report plays a key role in the implementation of the system. All diagrams provided in the upcoming sections, are created using Visual Paradigm software.

1.2 Design Goals

In this section, we have determined our design goals to be followed. These design goals are inferred from the non-functional requirements, which we have stated in our analysis report. In the process of designing our system we will be focusing on the below mentioned criteria.

1.2.1 Performance Criteria

Response Time: Response time is a crucial criterion for our game to provide graphical smoothness and provide enjoyable game experience for the user. The game should response the player's moves instantly and exhibit related animations and graphics.

1.2.2 Dependability Criteria

Robustness: Java makes it easier for us to deal with this feature since this language can be run in all OS's. Still, while implementing our game, we will also consider the cases when the player presses a wrong input, so that the game continues and not crash. Furthermore the presence of Database will also be one of the cases where we will make sure of the proper working of the game. Hence, if the connection to the database is not possible, the user will still have the possibility to play the game as a guest.

Fault Tolerance: Project should operate in situations that can cause errors. We will test our program in as many aspects as we can to increase the fault tolerance of the system.

1.2.3 Maintenance Criteria

Extensibility: Since we use object-oriented architecture, new features such as power-ups and obstacles can be easily added to our game. For instance, with creating a new class that will extend our weapon class, we can add a new weapon without modifying the rest of the classes. This can also be done to our power-up and enemy classes. Therefore, with the help of a few modifications our game will be easily extensible.

Modifiability: In order to give a better game experience, we have designed our system to be feasible for new features and progressions. By reducing the coupling between the subsystems in our design, we are not only making it easier to fix the bugs and eliminate the errors from our game, but also to add or modify different classes in this game. The system is designed in such a way that the operations of one class have a minimum impact on the other classes.

Portability: Having a large variety of devices and Operating Systems we decided to implement our game using Java which can be run in almost all devices and operating systems. Java has JVM to enable platform independence [2] and works in most commonly used operating systems (Windows, Linux, MacOS, Polaris) because they include updated versions of JDK and Java [3]. These properties of Java increase the portability of our system.

Furthermore, implementing this game in Java, will make it easier to also implement this game for Android devices. Since Android also works with Java, in our project we will only have to modify the first layer which is the User Interface. In addition, the local database that we are using, SQLite, is also supported in Android.

1.2.4 End User Criteria

Usability: We will create explicit user interface design so that the users can be able to understand the gameplay and the menus easily. Keeping the UI in a basic, non-complex level, and also providing hints on how to navigate through the game, will help the user to easily play this game.

1.2.5 Trade-Offs

User Friendliness vs. Functionality: The purpose of the game is to give an enjoyable time to the user. Since one of the criteria for our game is to be user-friendly, our priority is to make this game clear and simple. Therefore, in the design of “Catch The Deadline”, we chose user-friendliness to functionality. Thus, the game will not include complex functionalities but it will be easy and enjoyable.

Response Speed vs. Memory: Being a game, the response speed is one of the main features because graphical smoothness and quality of the animations are the properties that improve the game experience. Hence while implementing our game we will concentrate more on the response speed and smoothness of the graphics rather than on the memory consumption.

Delivery time vs. Functionality: Having a very strict deadline to finish this project, we have removed some of the functionality features which could increase the performance of the game a lot more. One of the main features would be making the game multiplayer, hence it could be run in several different machines in the same time and the players could play with friends. This would make the game more enjoyable for the users. Furthermore, if more time would be provided we would also be able to add more levels and game features which would increase the functionality of the game. However a complete easy project is always better than a difficult non-working project.

Portability vs. Efficiency: Portability is the feature that we chose as the most important one compared to efficiency since we decided to implement the game using Java. This programing language requires JVM which should be set up in the device before the user can play the game. Furthermore, the JVM also affect the performance of the game since there will be a virtual environment between OS and our game. However since the game is not very complex, this effect will not be very significant.

1.3 Definitions, Acronyms, Abbreviations

1.3.1 Acronyms and Abbreviations:

JDK: Java Development Kit [3]

JVM: Java Virtual Machine [2]

OS: Operating System

1.3.2 Definitions:

2D Arcade-style platformer game: A 2D action game in which protagonist of the character runs and jumps from platforms, collect power-ups and overcome the obstacles.

JDK: Java Development Kit which includes JVM and necessary sources for development of Java [3] .

JVM: Java Virtual Machine is a CPU that Java Compilers keep the code. [2]

Database: A system used in order save and get data in an efficient way

1.4 References

[1] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 2nd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2004, ISBN: 0-13-047110-0.

[2] M. Roulo, “Java's three types of portability,” Java's three types of portability. [Online]. Available: https://www.javaworld.com/article/2076944/java-s-three-types-of-portability.html. [Accessed: 19-Oct-2017].

[3] “Java Development Kit,” Java Development Kit. [Online]. Available: https://en.wikipedia.org/index.php?q=aHR0cHM6Ly9lbi53aWtpcGVkaWEub3JnL3dpa2kvSmF2YV9EZXZlbG9wbWVudF9LaXQ. [Accessed: 19-Oct-2017].

1.5 Overview

In the introduction part, we defined our game’s purpose as giving good gaming experience to the users. To maintain this purpose, we determined several design goals in four different aspects which are: the dependability, maintenance, performance of the system and user-friendliness. We make concessions from memory, functionality and efficiency to achieve the design goals we determined.

2. Software Architecture

2.1 Overview

In this section, we describe the subsystem decomposition of our system. The subsystem decomposition has been designed to address our design goals. The architectural style of our system is the three-tier architectural style, since it best suited our system. We have decomposed our system into 4 subsystems, those being, UserInterface, GameManager, AccessManager and Database.

2.2 Subsystem Decomposition

The Subsystem Decomposition of our system is influenced by design goals, in order to increase delivery time, extensibility, modifiability and portability, we tried to reduce coupling and increase cohesion among subsystems and create a quality product for the users. We have decomposed our system into four subsystems; UserInterface, GameManager, AccessManager and Database subsystems as shown in Figure – 1.

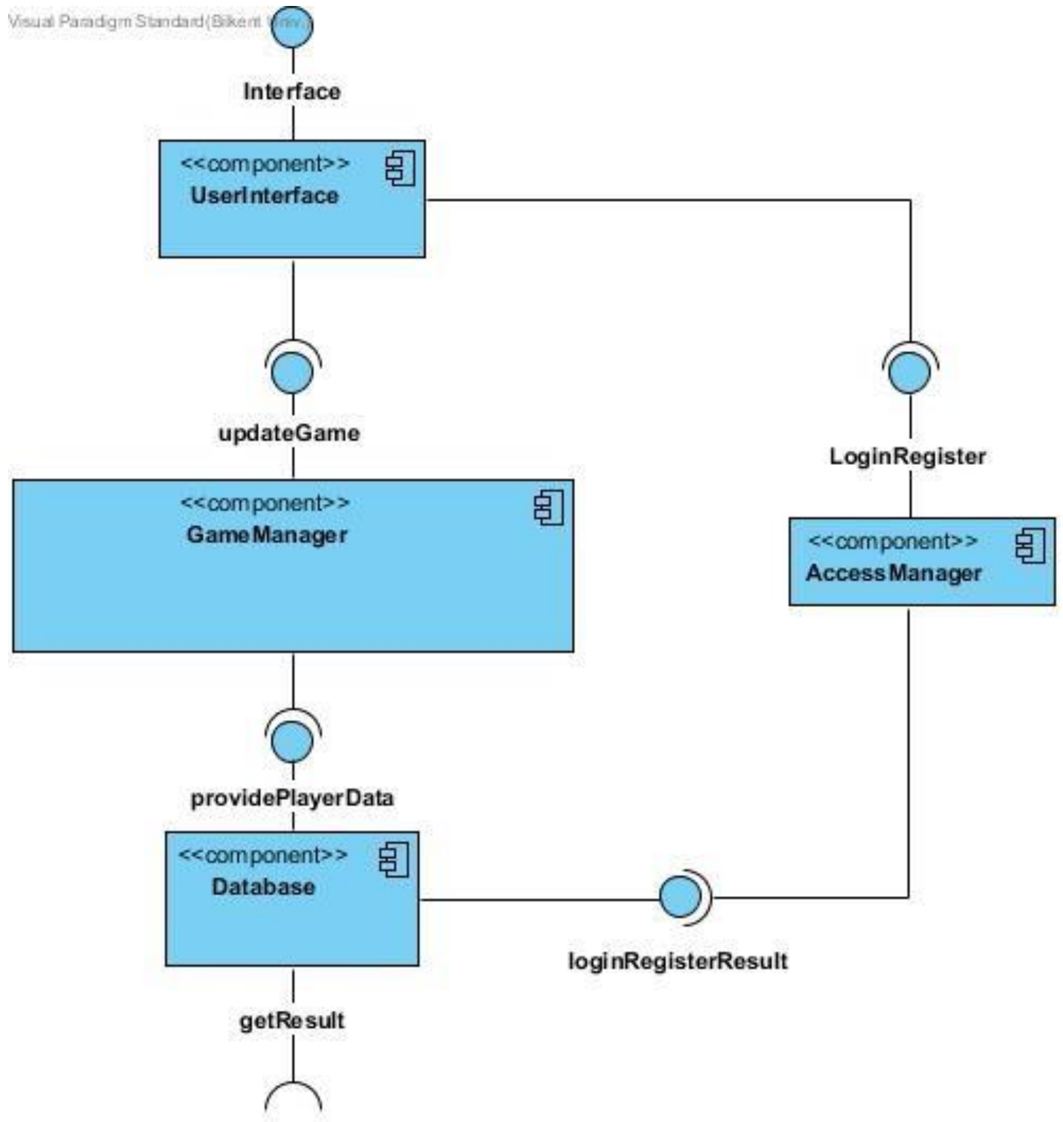


Figure 1- High-Level Representation of Subsystem Decomposition

As we can see from Figure – 1, in the first layer we have UserInterface subsystem, which is responsible for the interface of the game, and it is dependent on the GameManager subsystem to provide information to keep updating the game and on AccessManager subsystem to grant access to the user through login or registration. This layer will include all the classes dealing with the UI whereas all the logical part that will provide data to these classes will be handled in the second layer.

The second layer consists of GameManager and AccessManager subsystems. The GameManager subsystem is responsible for initializing the control and entity objects, constructing the game layout, and rule checking and realizing the processing throughout the game play. The AccessManager subsystem is responsible for the identification and authorization of the user to his respective account and data. Both of these subsystems on the second layer are dependent on the Database subsystem to provide formerly stored player data and the authorization result. Furthermore, both of these subsystems interact with the first layer by taking data while the user inserts new inputs for getting access and playing the game and also by sending data for the game updates.

The third layer consists of the Database subsystem, which is responsible for the storage and retrieval of information from the local database. It deals with storing the data at the end of each game, updating the high scores and storing the authentication information of each user.

2.3 Architectural Style

We chose the three-tier architectural style for our system, because the three layers perfectly correspond to the needs of our system. As it can be seen in Figure – 2, the first layer is the top layer and the Interface layer, includes all the boundary objects for interaction with the user. This layer consists of the User Interface subsystem. The second layer is the application logic layer which consists of the Game Management subsystem and the Access Manager subsystem. It includes all the entity and control objects and it is responsible for rule checking, control of the access and flow of the game. The third layer is the storage layer and the bottom layer of the system which consists of the Database subsystem. It deals with storage and retrieval of data from the local database. This is overall a closed architecture.

Furthermore, we chose this type of architecture because it enables us to create a system which can be can easily extended in the future. Hence, as we mentioned previously, if we will decide to implement this game even for Android devices we will only consider the modification of the first layer. If we decide to add new functionalities in our game such as multiplayer or online database, we will only have to modify the second and bottom layer, without having to modify the UserInterface subsystem. Therefore, this type of architecture best suits our goals.

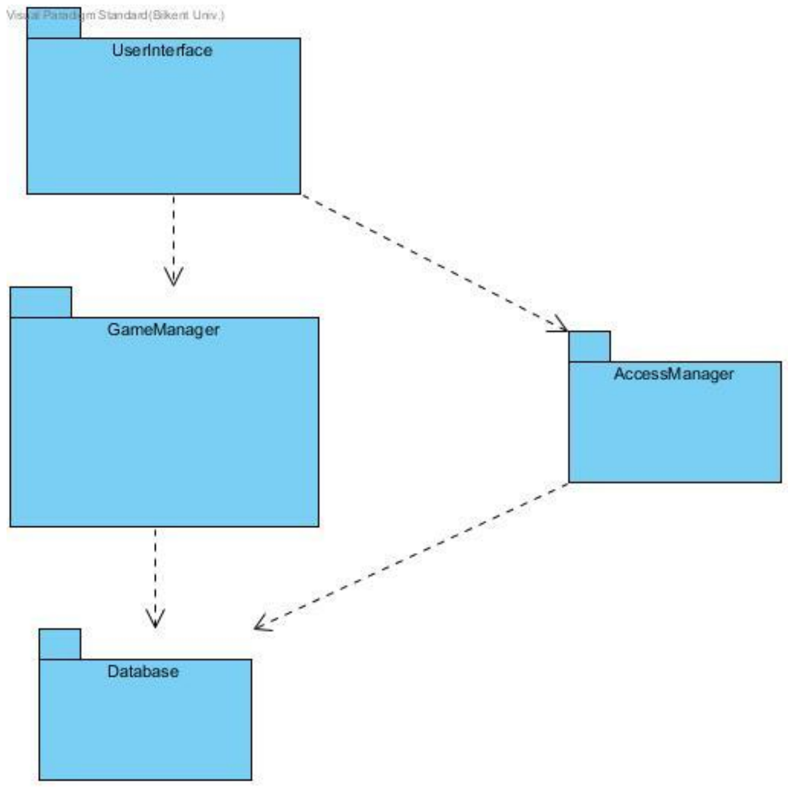


Figure 2- Three-Tier Architectural Style

2.4 Hardware/Software Mapping

For hardware configuration, we will use arrow keys to control the position and movement of the protagonist of the game and the space button will be used for jumping and letter keys ‘A’, ’S’, ’D’ for attacking with different types of weapons. In addition, mouse pointer will be used in main menu and other menus for selection. *Catch the Deadline* will be implemented in Java. Therefore, to run this game, the user will need a computer which has an operating system that supports Java Runtime Environment such as Windows, Linux, MacOS or Polaris. Additionally, latest version of JDK and a Java compiler will be needed. There is no requisite for any network connection since for storing the accounts and each account’s data we will use local database. Due to the portability of Java and basic needs of hardware, standard computers will be enough to run “*Catch the Deadline”*.

2.5 Persistent Data Management

Being an offline game, all the data of the game will be stored in a local database, SQLite, such as account data and also the high scores and level progress. Furthermore, all the static data such as sprites, background music, sounds and background images will be stored in the local storage of the user’s device. If in the future the game is extended to work with an online database again the above mentioned static data will again be stored in the user’s device since fetching them from the online server would drastically decrease the performance of the game.

2.6 Access Control and Security

Being a local database there is not many security issues related to our project. Furthermore, there are no data such as username of server and database that could be an issue. In the future, if we will make our game online, we will have to take care of problems such as dealing with the possibility of remote controls. However, for the moment, we don't have to consider this case. In addition, we will do our best to hide each account's personal data, the passwords, so that other players of the same device won't use other accounts without permission.

2.7 Boundary Conditions

2.7.1 Initialization

Catch the Deadline will be a .jar file. Hence, if the operating system of the user supports JRE and has a JDK set up, the user will only have to double click the file in order to run the game and play with it.

2.7.2 Termination

User can exit in several ways:

- Catch the Deadline can be terminated by pressing the "x" button in the upper right-hand corner of the window.
- The user can exit through the main menu by clicking the "Quit Game" button.
- During the game play, the game can be terminated by pressing the pause button and then clicking "Quit Game" button in the pause game menu.

2.7.3 Failure

- Unexpected problems that occur in JRE can cause errors. We will only take care of the exceptions for which we are responsible for during our implementation.

- In the case of a hardware failure, the data of the game, which was not saved during the game play may be lost.

3. Subsystem Services

3.1 Detailed System Design

Below is the diagram for the detailed System Design of our program.

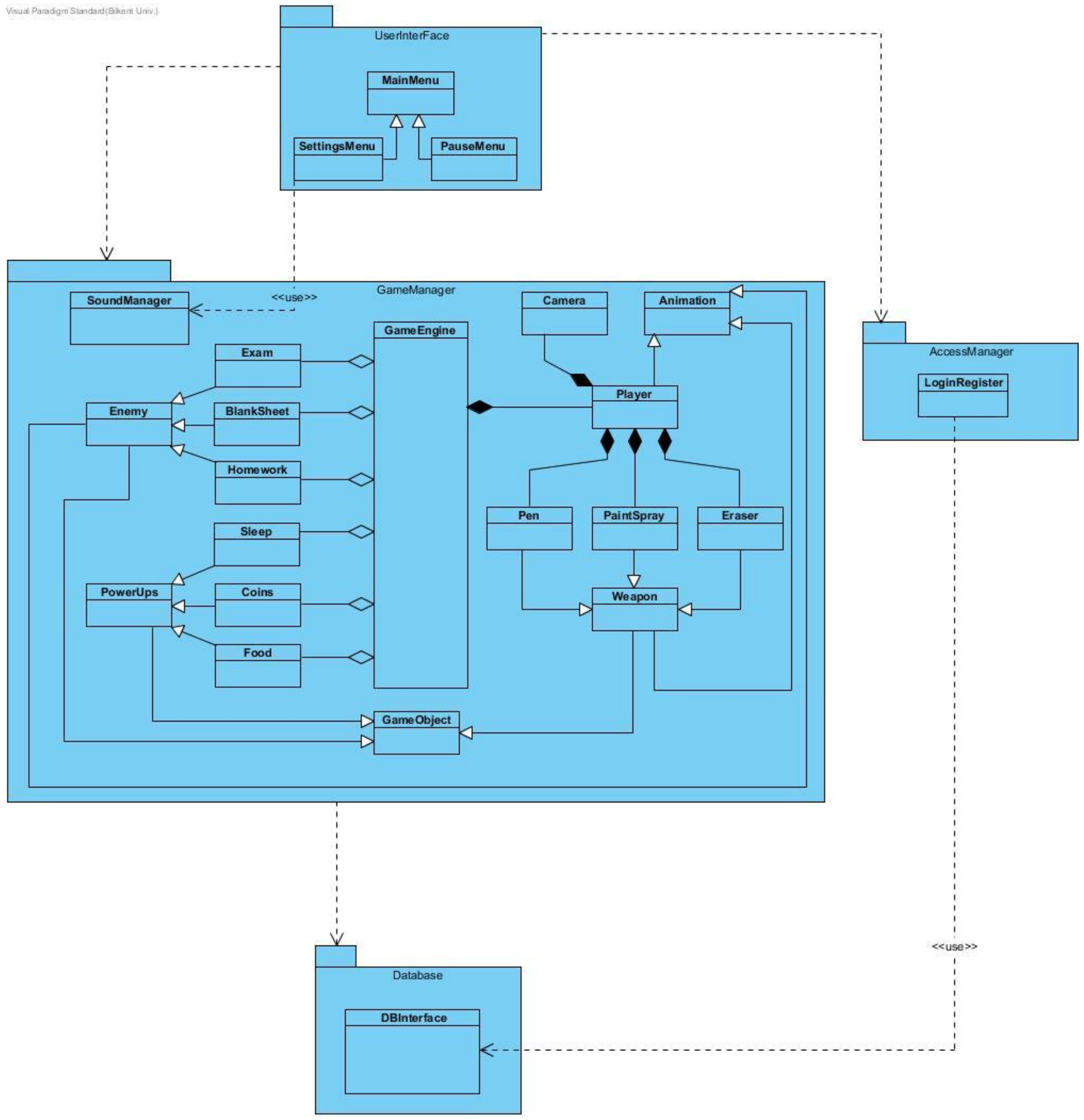


Figure 3- Detailed System Design

3.2 Final Object Design

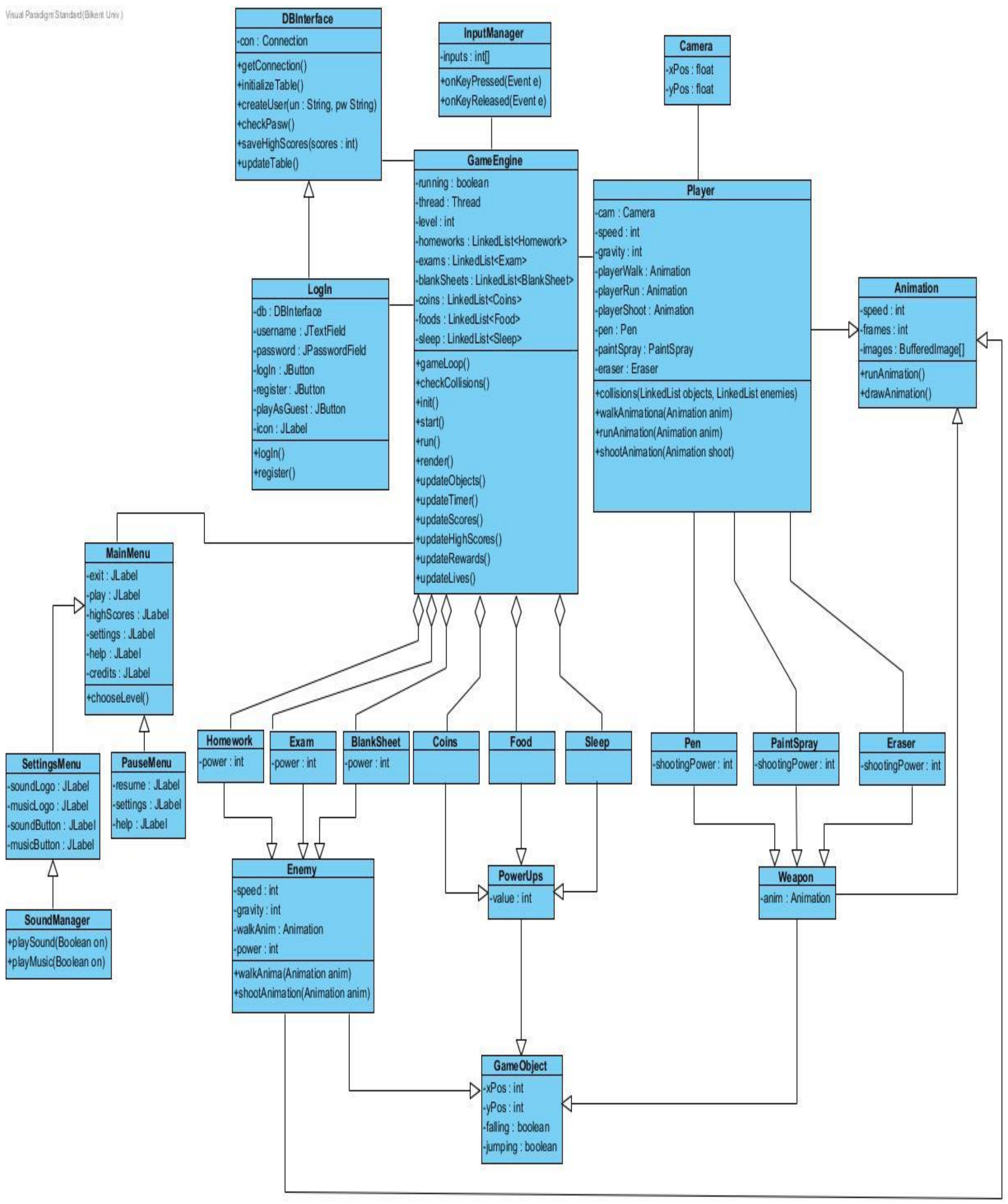


Figure 4 Final Object Design

3.3 User Interface Subsystem

The User Interface Subsystem is responsible for the creation of the interface for the user. It consists of the MainMenu, SettingsMenu and PauseMenu classes. This subsystem is responsible for displaying the main menu when the user first runs the game, the pause menu during the game play, the settings menu for navigating through the settings.

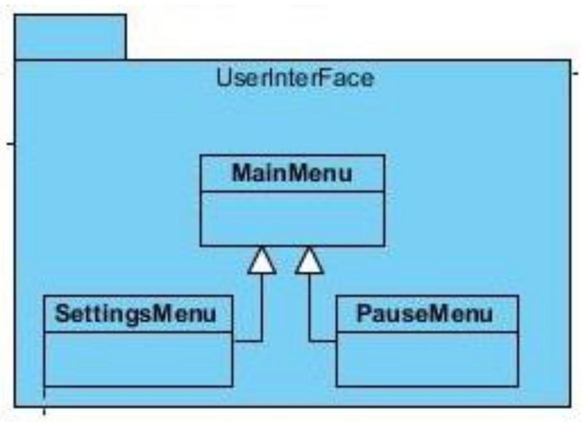
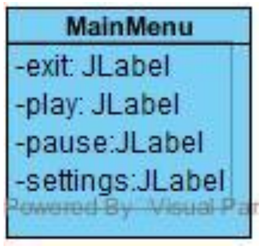


Figure 5- User Interface Subsystem

MainMenu class



This is the user interface class of the Main Menu. It has JLabel properties to create the Main Menu screen.

Constructor

public MainMenu(): This constructor is used for initializing the exit, play, pause and settings JLabels. This is the first frame that will be opened when user enters the game.

Attributes

private JLabel play: It is a JLabel property, which is used for displaying the “Play” label.

private JLabel exit: It is a JLabel property, which is used for displaying the “Exit” label.

private JLabel settings: It is a JLabel property, which is used for displaying the “Settings Menu” screen.

private JLabel help: It is a JLabel property, which is used for displaying the “Help Menu” screen.

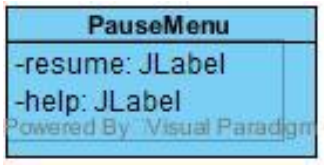
private JLabel highScores: It is a JLabel property, which is used for displaying the “High Scores Menu” screen.

private JLabel credits: It is a JLabel property, which is used for displaying the “Credits Menu” screen.

Methods

public void chooseLevel(): This method is used for choosing the level of the game which will be displayed after the user gets access to the game.

PauseMenu class



This is the user interface class of the Pause Menu. It has JLabel properties to create the Pause Menu screen.

Constructor

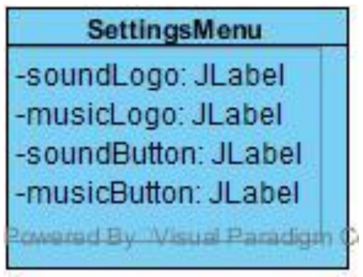
public PauseMenu(): This constructor is used for initializing the resume and help JLabels.

Attributes

private JLabel resume: It is a JLabel property, which is used for returning to the gameplay.

private JLabel help: It is a JLabel property, which is used for displaying the hints that will help the user to navigate through the game menus and also it will show some hints on how to play the game.

SettingsMenu class



This is the user interface class of the Settings Menu. It has JLabel properties to create the Settings Menu screen.

Constructor

public SettingsMenu(): This constructor used for initializing the soundLogo, musicLogo, soundButton and musicButton JLabels.

Attributes

private JLabel soundLogo: It is a JLabel property, which is used for displaying the logo of sound.

private JLabel musicLogo: It is a JLabel property, which is used for displaying the logo of music.

private JLabel soundButton: It is a JLabel property, which is used for displaying the on/off button of sound. This attribute will be controlled by the ActionEvent.

private JLabel musicButton: It is a JLabel property, which is used for displaying the on/off button of music. This attribute will be controlled by the ActionEvent.

3.4 Game Management Subsystem

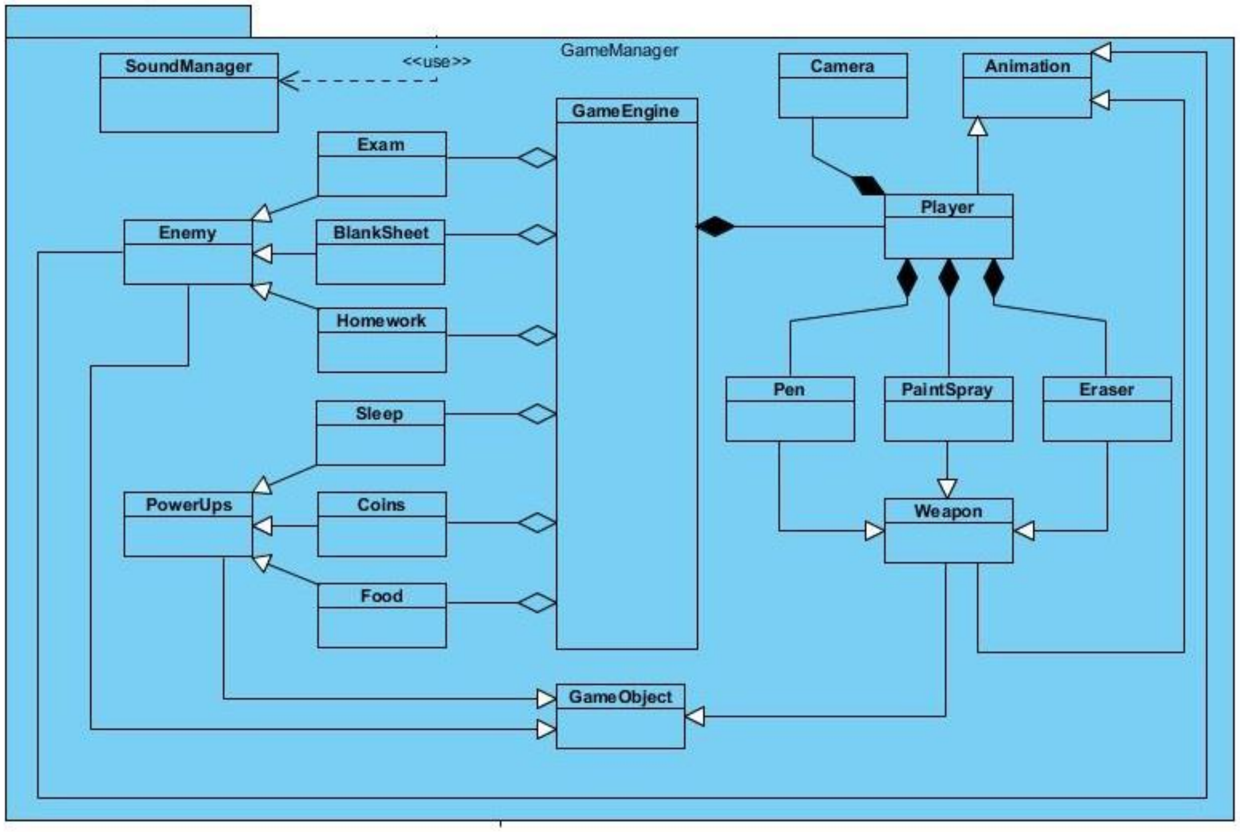
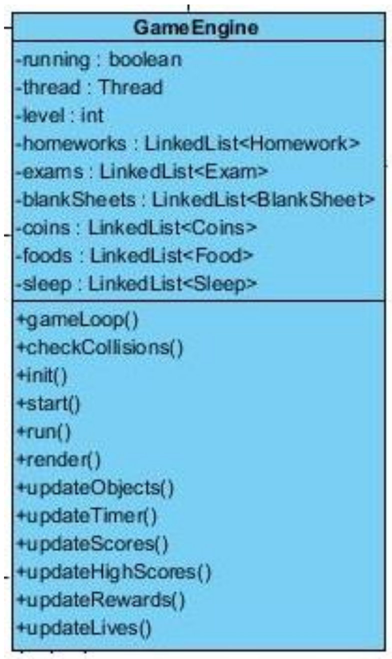


Figure 6- GameManager Subsystem

GameEngine class



This class is the base class for all the entity and control objects, it is responsible for initializing all the power-ups and enemy objects as well as setting up the background and the obstacles for each stage. It takes input from the user through the InputManager, and passes that information to the Player for movement.

Constructor

Public GameEngine(): this constructor is responsible for initializing all the entity and control objects of the game and for setting up display for the game play.

Attributes

private boolean running: this attribute holds the boolean value to see if the game is running or not. This attribute will be used to stop the game engine for creating a new Thread for the game in the game loop.

private Thread thread: this attribute holds the thread object that is going to start during the game play.

private int level: this attribute holds the value for the level that is going to be chosen by the player before game play.

private LinkedList<Homework> homework: this attribute holds a linked list of Homework objects, that are going to appear in the game as enemies.

private LinkedList<Exam> exam: this attribute holds a linked list of Exam objects, that are going to appear in the game as enemies.

private LinkedList<BlankSheet> blankSheet: this attribute holds a linked list of BlankSHeet objects, that are going to appear in the game as enemies.

private LinkedList<Coins> coins: this attribute holds a linked list of Coin objects, that are going to appear in the game as power-ups.

private LinkedList<Food> food: this attribute holds a linked list of Food objects, that are going to appear in the game as power-ups.

private LinkedList<Sleep> sleep: this attribute holds a linked list of Sleep objects, that are going to appear in the game as power-ups.

Methods

public void gameLoop(): this method is the loop for running the game continuously by creating new obstacles and enemies as the player moves forward and updating the objects, lives, rewards, scores of the player.

public boolean checkCollisions(): this method checks the collisions of player with obstacles or enemies. If the object collides with an obstacle, then it will stand over it and not fall. If the player collides with an enemy, then his/her life points will be reduced

public void init(): this method is responsible for the initialization of all the game entity objects.

public void start(): this method initiates the game play and load the display. It is executed after the player has pressed the "play game" button.

public void run(): this method is responsible for starting the threads.

public void render(): this method will perform the rendering of the animations through the BufferedImage so that we increase the performance of the game.

public void updateObjects(): this method is responsible for updating the obstacles, enemies and powerUps that are going to appear in the gameplay.

public void updateTimer(): this method increments the timer during the game play.

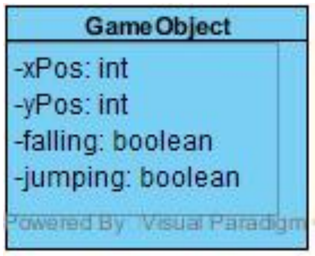
public void updateScore(): this method updates the score at the end of each level.

public void updateHighScore(): this method updates the high scores at the end of each level.

public void updateRewards(): this method updates the rewards that collide with player during the game play.

public void updateLives(): this method decrements the lives of the player, if the player has finished up all his powerUps.

GameObject class



The GameObject is the parent object for the Enemy, PowerUp and Weapon class. It is going to be implemented for future modifiability and extensibility. This class determines the position of the weapons, power-ups and enemies in the game.

Constructor

public GameObject(): This constructor initializes the objects for GameObject class.

Attributes

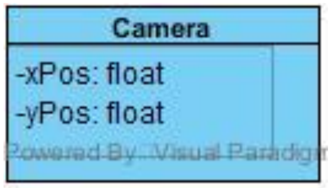
private int xPos: This attribute holds the horizontal position of the object on the frame as an integer.

private int yPos: This attribute holds the vertical position of the object on the frame as an integer.

private boolean falling: This attribute holds the boolean value of the object if it is in the state of falling or not.

private boolean jumping: This attribute holds the boolean value of the object if it is in the state of jumping or not.

Camera class



The camera class is basically the frame of the game play that keeps moving according to the movement of the player, and updates the background on the screen.

Constructor

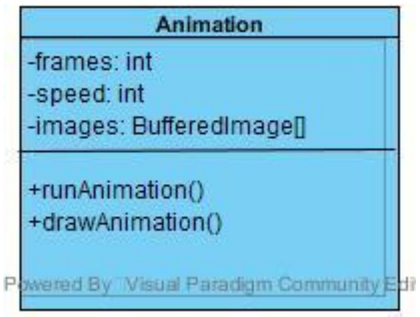
public Camera(): this constructor initializes the camera object and creates the default frame for the beginning of the game.

Attributes

private float xPos: this attribute holds the horizontal position of the player on the screen and keeps updating as the player moves.

private float yPos: this attribute holds the vertical position of the player on the screen and keeps updating as the player moves.

Animation class



The Animation class is responsible for the initialization and creation of animation and the smooth flow of images. It is the base class for producing the movement of player, weapons and enemies.

Constructor

public Animation(): This constructor initializes the Animation object, thus drawing all the objects for the first time and it loads the images to be displayed for the movement of objects.

Attributes

private int frames: this attribute holds the integer value of frames to be used for each object animation.

private int speed: this attribute holds the value for the number of the frames to be changed per second.

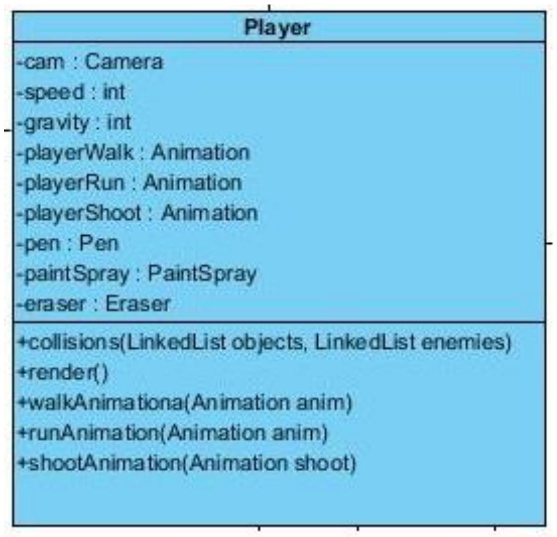
private BufferedImage[] images: this attribute holds the image icons for the animation of each object in an image array to be repeated in a loop continuously.

Methods

public void runAnimation(): this method is responsible for displaying the image icons from the images array so as to animate the movement of the objects.

public void drawAnimation(): this method basically draws the graphics onto the screen and keeps on updating them.

Player class



The Player class is responsible for creating the player object, which represents our protagonist. This class is responsible for providing the control of player, display information and the rule checking of the interaction of player with other objects. The player class provides this data to the game engine to be displayed.

Constructor

public Player(): this constructor creates the player object, by initializing the player with default values for the first time and initializing the object instances of Animation for walking, running and shooting, the Camera object for positioning the screen and the weapon objects.

Attributes

private Camera cam: this attribute holds the calculated value of the position of the screen according to location and movement of the player.

private int speed: this attribute holds the value of the speed at which the player moves, the speed is determined by the input given by the user.

private int gravity: this attribute holds the speed with which the player falls down.

private Animation playerWalk: this attribute holds the instance of the Animation object for the walking animation of the player.

private Animation playerRun: this attribute holds the instance of the Animation object for the running animation of the player.

private Animation playerShoot: this attribute holds the instance of the Animation object for the shooting animation of the player.

private Pen pen: this attribute holds the instance object for Pen weapon.

private Eraser eraser: this attribute holds the instance object for Eraser weapon.

private PaintSpray paintSpray: this attribute holds the instance object for PaintSpray weapon.

Methods

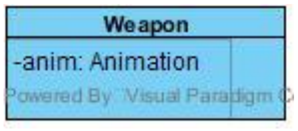
public boolean collisions(LinkedList objects, LinkedList enemies): this method is responsible for checking if the player has collided with an enemy or an object(i.e, powerUp or obstacle). It returns true if the player has collided and false otherwise.

public void walkAnimation(Animation anim): this method is responsible for running the animation of the player walking upon given input.

public void runAnimation(Animation anim): this method is responsible for running the animation of the player running upon given input.

public void shootAnimation(Animation anim): this method is responsible for running the animation of the player shooting upon given input.

Weapon class



Weapon class implements the weapons. When the students catch some of the weapons, user will gain a power and tool to fight with enemies.

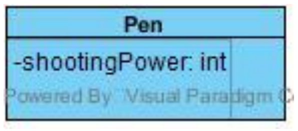
Constructor

public weapon(int shootingPower): Weapon constructor initialize shootingPower parameter of this class according to the weapon type.

Attributes

private Animation anim: This attribute is for showing an animation in compliance with the weapon type.

Pen class



Pen class extends the weapon class. Pen has all of the features and methods of weapon class however, it differs to the shooting power that it has. After the student catch this weapon, player gains ability to shoot enemies with a pen for a limited time.

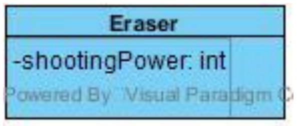
Constructor

public pen(int shootingPower): Pen constructor initialize shootingPower parameter of this class. In addition, the image of the pen weapon will be initialized in the constructor.

Attributes

private shootingPower: This attribute holds the shooting power of the pen.

Eraser class



Eraser class extends the weapon class. Eraser has all of the features and methods of Weapon class however, it differs to the shooting power that it has. After the student catch this weapon, player gains ability to shoot enemies with an eraser for a limited time.

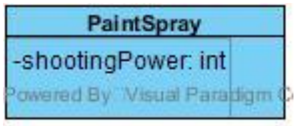
Constructor

public eraser(int shootingPower): Eraser constructor initialize shootingPower parameter of this class. In addition, the image of the eraser weapon will be initialized in the constructor.

Attributes

private shootingPower: This attribute holds the shooting power of the eraser.

PaintSpray class



PaintSpray class extend the weapon class. Pen has all of the features and methods of Weapon class however, it differs to the shooting power that it has. After the student catch this weapon, player gains ability to shoot enemies with a pen for a limited time.

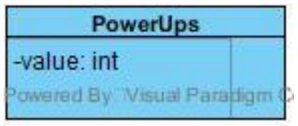
Constructor

public paintSpray(int shootingPower): PaintSpray constructor initialize shootingPower parameter of this class. In addition, the image of the paint spray weapon will be initialized in the constructor.

Attributes

private shootingPower: This attribute holds the shooting power of the paint spray.

PowerUps class



Power-up class implements the power- ups. It will hold the amounts of power-ups. User can improve their weapons by collecting power-ups.

Constructor

public power-up(int value): Power-up constructor initialize the value parameters of each power-up according to the power up types.

Attributes

private int value: The integer value will show the value of the power-ups. All of the power-ups will be initialized in the amount of 3. In the game, as user collect the power-up this value will be increased and in each fight this value will be decreased.

Food class

Food class extends the power-up class. Food has all of the features and methods of power-up class however, it is privatized with the help of its own constructor. If the food amount of the student will be zeroed, the game will end.

Constructor

public food(int value): the constructor of this class will initialize the value of this power up.

Sleep class

Sleep class extends the power-up class. sleep has all of the features and methods of power-up class however, it is privatized with the help of its own constructor. If the sleep amount of the student will be zeroed, the game will end.

Constructor

public sleep(int value): the constructor of this class will initialize the value of this power up.

Coins class

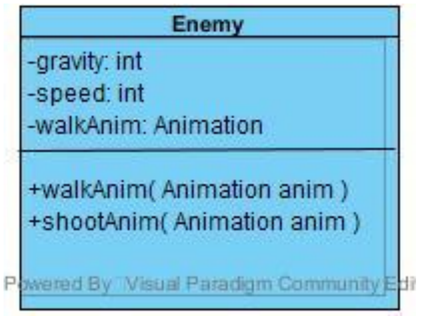
Coins
Powered By Visual Paradigm C

Coins class extends the power-up class. Coins has all of the features and methods of power-up class however, it is privatized with the help of its own constructor. The amount of the coins are not decreased with fights and user can buy new with the help of this coins.

Constructor

public coins (int value): the constructor of this class will initialize the value of this power up.

Enemy class


Enemy
-gravity: int
-speed: int
-walkAnim: Animation
+walkAnim(Animation anim)
+shootAnim(Animation anim)
Powered By Visual Paradigm Community Edi

Enemy class implements the enemies. There are various of enemies a student should fight to end the game.

Constructor

public int enemy(int gravity, int speed, int power): Enemy constructor initializes gravity, speed and power parameter of this class according to the enemy type.

Attributes

private int power: This attribute shows the amount of the power an enemy has. In the case of fight, it decrease and when it is zeroed, the enemy will die.

private int gravity: This attribute shows the value of the gravity an enemy has. In the case of jumps, it increase and in the case of falling it decrease.

private int speed: This attribute shows the value of the speed an enemy has.

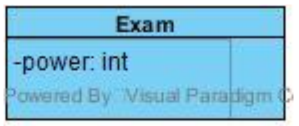
private Animation walkAnim: This attribute is an animation in compliance with the enemy type.

Methods

public walkAnim(Animation anim) : This method's purpose is to show a walking enemy.

public shootAnim(Animation anim) : This method's purpose is to show a shooting enemy.

Exam class



Exam class extends the enemy class. Exam has all of the features and methods of enemy class however, it is privatized with the help of its own constructor.

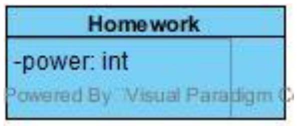
Constructor

public int exam (int gravity, int speed, int power): Exam constructor initialize gravity, speed and power parameter of this class.

Attributes

private int power: This attribute shows the amount of the power an exam has. In the case of fight, it decrease and when it is zeroed, the enemy will die.

Homework class



Homework class extends the enemy class. Homework has all of the features and methods of enemy class however, it is privatized with the help of its own constructor.

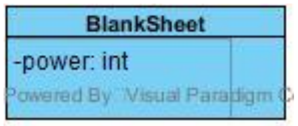
Constructor

public int homework (int gravity, int speed, int power): Homework constructor initialize gravity,   speed and power parameter of this class.

Attributes

private int power: This attribute shows the amount of the power a homework has. In the case of fight, it decreases and when it is zeroed, the enemy will die.

BlankSheet class



BlankSheet class extends the enemy class. Blanksheet has all of the features and methods of enemy class however, it is privatized with the help of its own constructor.

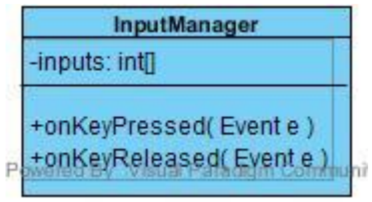
Constructor

public int blankSheet(int gravity, int speed, int power): BlankSheet constructor initialize gravity, speed and power parameter of this class.

Attributes

private int power: This attribute shows the amount of the power a blank sheet has. In the case of fight, it decreases and when it is zeroed, the enemy will die.

InputManager class



This class is for managing the inputs.

Constructor

public InputManager(): This constructor will initialize the possible inputs.

Attributes

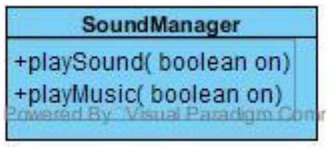
private int [] inputs: This array is for holding "space bar", "A", "S", "D", "arrow keys", and "CTRL" as the possible inputs that will be used to play the game.

Methods

public onKeyPressed(Event e): This is a method for managing the pressed input and creating an event accordingly to that.

public onKeyReleased(Event e): This is a method for managing the released input and creating an event according to that.

SoundManager class



SoundManager class is for managing the sounds and music.

Constructor

public SoundManager(): This constructor will initialize the music and sounds on.

Methods

public playMusic(boolean on): This method is for turning the music on.

public playSound(boolean on): This method is for turning the sound on.

3.5 AccessManager Subsystem

The AccessManager subsystem will take care of the login and register functions. This package will contain only one class which is LoginRegister. This is because the other classes' functionality is different from that of this class. Whenever the login operation is performed, the data that the user will provide will be checked whether they exist in the database or not. If they exist, then the user will be granted access to the game and all his records will be displayed, such as level progress and high scores. If the user decides to create a new account by signing up, the data that he will provide will again be checked to see whether they satisfy the requirements of the account, such as a unique username.

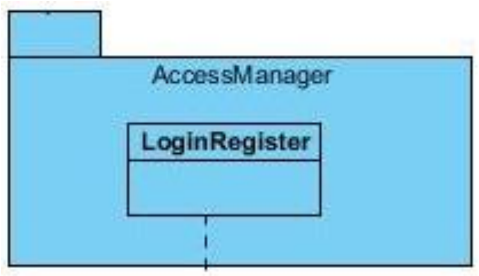
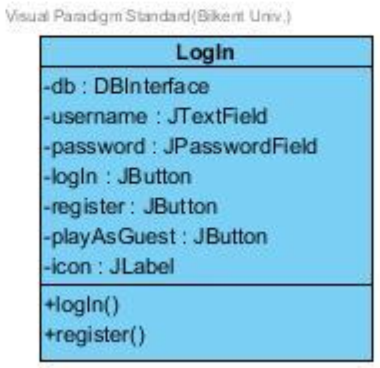


Figure 7- AccessManager Subsystem

LoginRegister class



This class is responsible for player's access to the game. It will allow the user to play with a pre-created account or offer the possibility to create a new account and start the game with no previous recorded data.

Constructor

public LoginRegister() : The constructor of this class initializes an instance of the DBInterface class which will use it in order to check the data that the user will offer, or to add new data to the database if the user decides to create a new account for the game.

Attributes

private DBInterface db: This attribute will be used in order to create a reference to the database so that the user can perform operations such as “login” or “register”

private JTextField username: this attribute will be used to get the data from the user in order to login or register.

private JPassword password: this attribute will provide the password that the user will enter before logging in or before registering.

private JButton login: this button will perform the login action by calling the method of the DBInterface which will check the provided data and if they are accepted will grant access to the user to the game.

private JButton register: this button will perform the registration operation by calling the createUser() method of the DBInterface.

private JButton playAsGuest: this button will give the opportunity to the player to play without having to create an account. In this way all the data of the game that the guest will play will not be saved in the database.

private JLabel logo: this logo will be the logo of the login window

Methods

public void logIn (): this method will perform the login operation by taking the data provided in the username and password fields and by checking them in the database. If they are not accepted, a popup will be shown to notify the user.

public void register(): this method will perform the registration operation. After taking the data provided in the username and password field, the instance of DBInterface will be used to create the new user in the system. If the data are not accepted, a popup will be shown to notify the user.

3.6 Database Subsystem

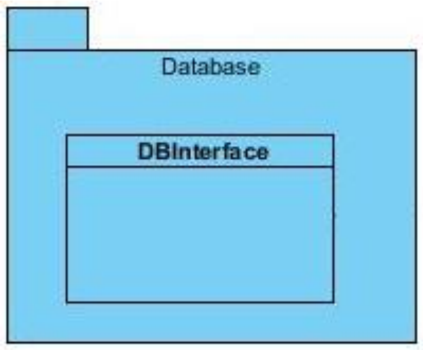
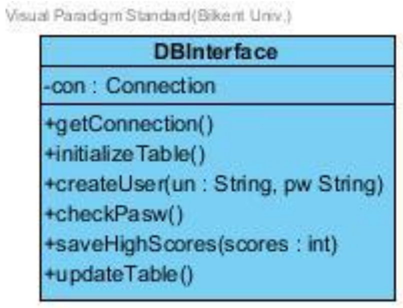


Figure 8- Database Subsystem

This package is responsible for creating the connection to the database and dealing with sending and retrieving data from the database. Even in this package there will be only one class. This subsystem also satisfies the rule of low coupling since there aren't many dependencies with the other packages.

Database class


Visual Paradigm Standard(Bilkent Univ.)
DBInterface
-con : Connection
+getConnection()
+initializeTable()
+createUser(un : String, pw String)
+checkPasw()
+saveHighScores(scores : int)
+updateTable()

This class has a direct connection to the JDBC library, hence it will be responsible for all the data operations such as account access and account registrations. Furthermore this class will also handle the level progress and also the high scores. In order to deal with the database, SQL language will be used to communicate with the database.

Constructor

public DBInterface(): the constructor of this class is only the default constructor.

Attributes

private Connection con: this variable is used to create a connection to the JDBC library

Methods

public void getConnection(): this method will create a connection to the database and also initialize the table of the database.

public void initializeTable(): this operation will initialize the table of the database by executing an SQL query.

public boolean createUser(String un, String pw): this operation will create a new user and return the result in the form of a Boolean. By taking the username and password, this operation will check whether they exist in the database or not. If yes, it will return true, if no, it will return false.

public boolean checkPasw(): this method will check whether the password that the user has provided is the same with the one in the database or not. If it is, this method will return true. If the password is not the same, then the method will return false.

public void saveHighScores(int scores): this method will save the scores of the player that is signed in the system. If the scores are higher than the ones saved before, the scores will be saved, otherwise they will be neglected.

public void updateTable(): this method will update the database table with the new provided data, whether they are scores, or data of the new registered users.