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 Report December 2, 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. In order to make the game easier to be implemented in other platforms in the future, we decided to use an online database instead of a local one. In this way the player will also be able to compete with other players by looking at their highest scores.

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 gameobject class, we can add a new weapon without modifying the rest of the classes. This can also be done for power-ups and enemies. 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. Furthermore using an online database will allow the user too keep track of the scores that he/she has scored even if he/she has played in another environment.

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 userfriendliness 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 nonworking 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. Furthermore, the fact that we decided to use an online database also indicates that we chose portability over efficiency for our game.

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.

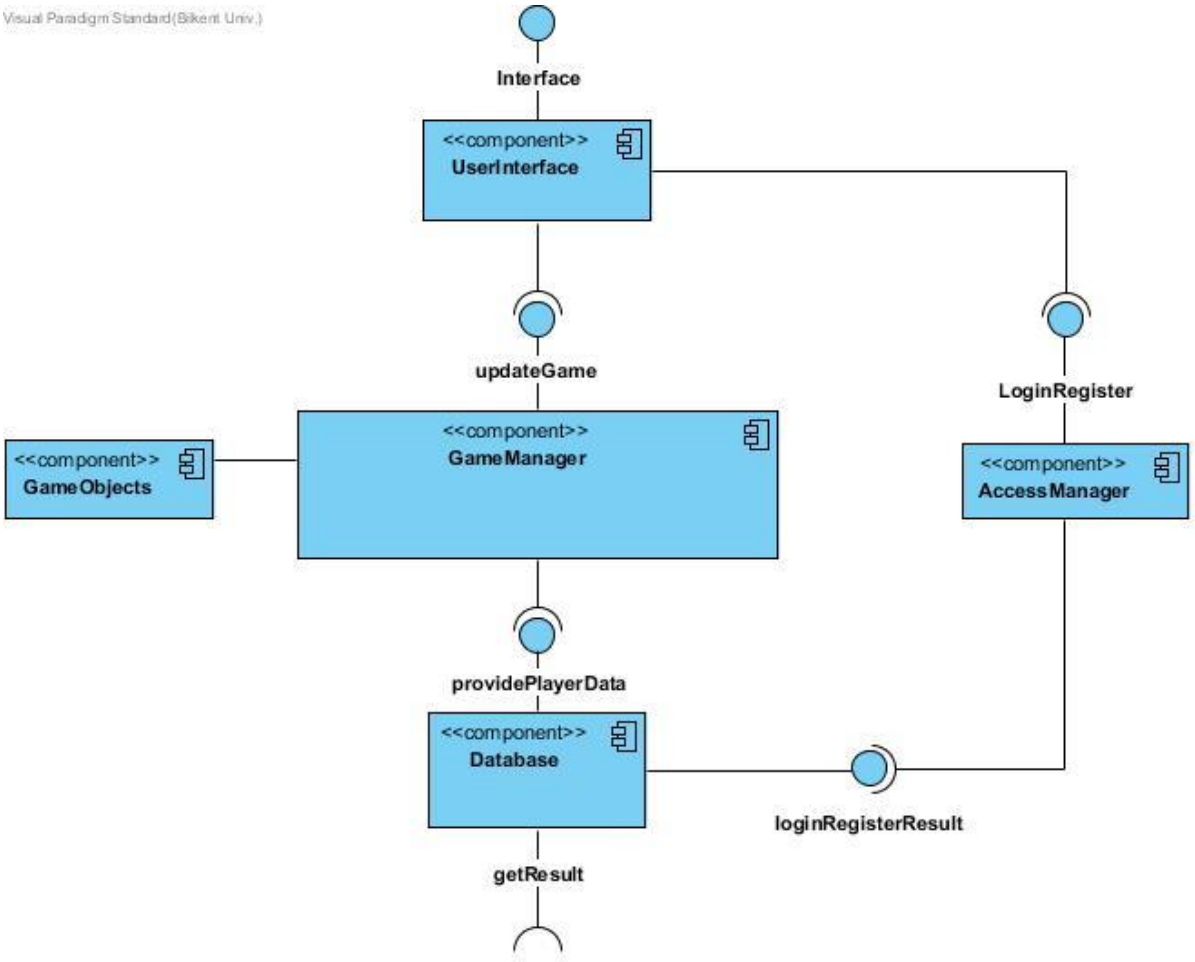


Figure 1- High-Level Representation of Subsystem Decomposition

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, AccessManager and GameObjects subsystems. The GameManager subsystem is responsible for initializing the control , 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. The GameObjects subsystem is responsible for the initialization of all game objects. All 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.

In difference from the first iteration design report we decided to split the GameManager in two subsystems, by creating a new subsystem for game object. In this way it will be easier to make changes in our classes of the game objects.

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, the Interface layer which 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 GameManager subsystem, the GameObjects subsystem and the AccessManager 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 online 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 , 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.

In difference from the first iteration design report, we have added a new package in our system. In this way we tried to reduce the complexity of our game so that it would be easier for implementing the game.

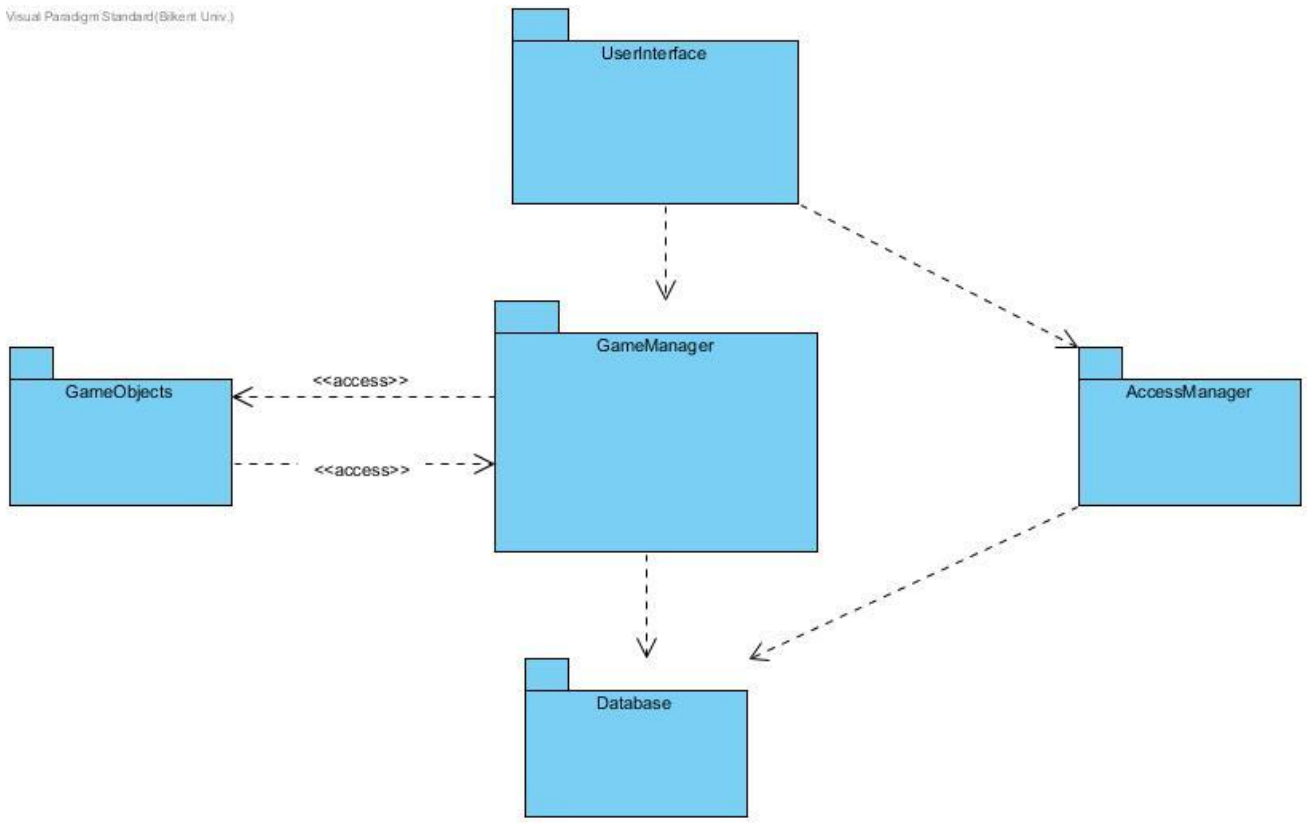


Figure 2- Three-Tier Architectural Style

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

In order to make this game more enjoyable, in difference from the design report in the first iteration we decided to use an online database instead of an offline one. In this way the user will also be able to observe the highest scores of other players, hence it would increase competition. However, 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. These data will be stored in the local storage since fetching them from the online server would drastically decrease the performance of the game. Even though this will increase the memory capacity of our game, we will try to compress these images in photoshop without reducing the quality so that the memory consumption will not be very high.

2.6 Access Control and Security

Since we now have an online database we will have some security issues. We know that if "Catch The Deadline" will be decompiled, the username and password of the database connection can be accessed and hence provide the possibility for remote code execution in the database. However we will do our best to hide this while implementing our game. Beside this issue our database will be secured. After the player has been granted access, the only access that wil have to other player's data will be their name and their highest scores.

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.

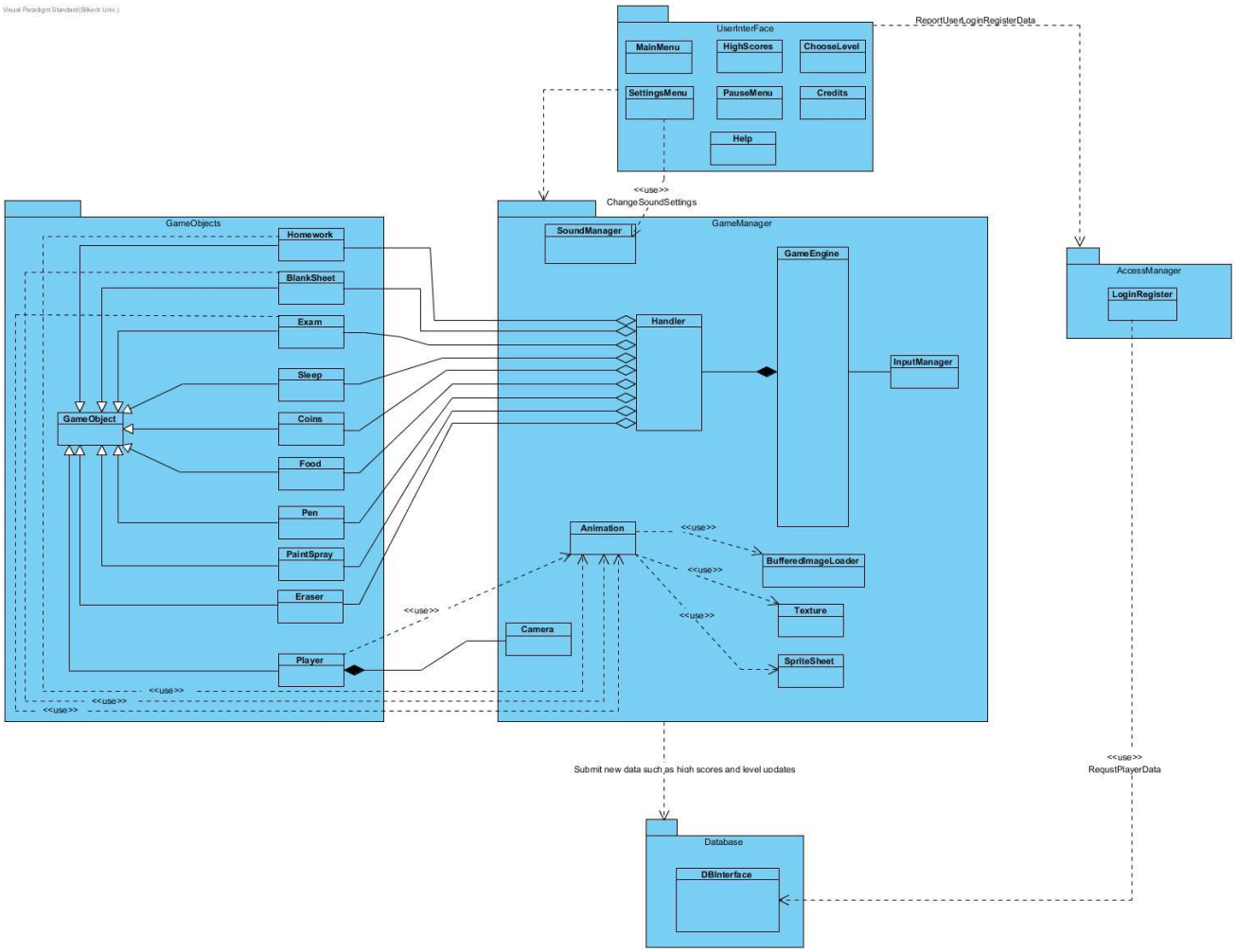


Figure 3- Detailed System Design

3.2 Final Object Design

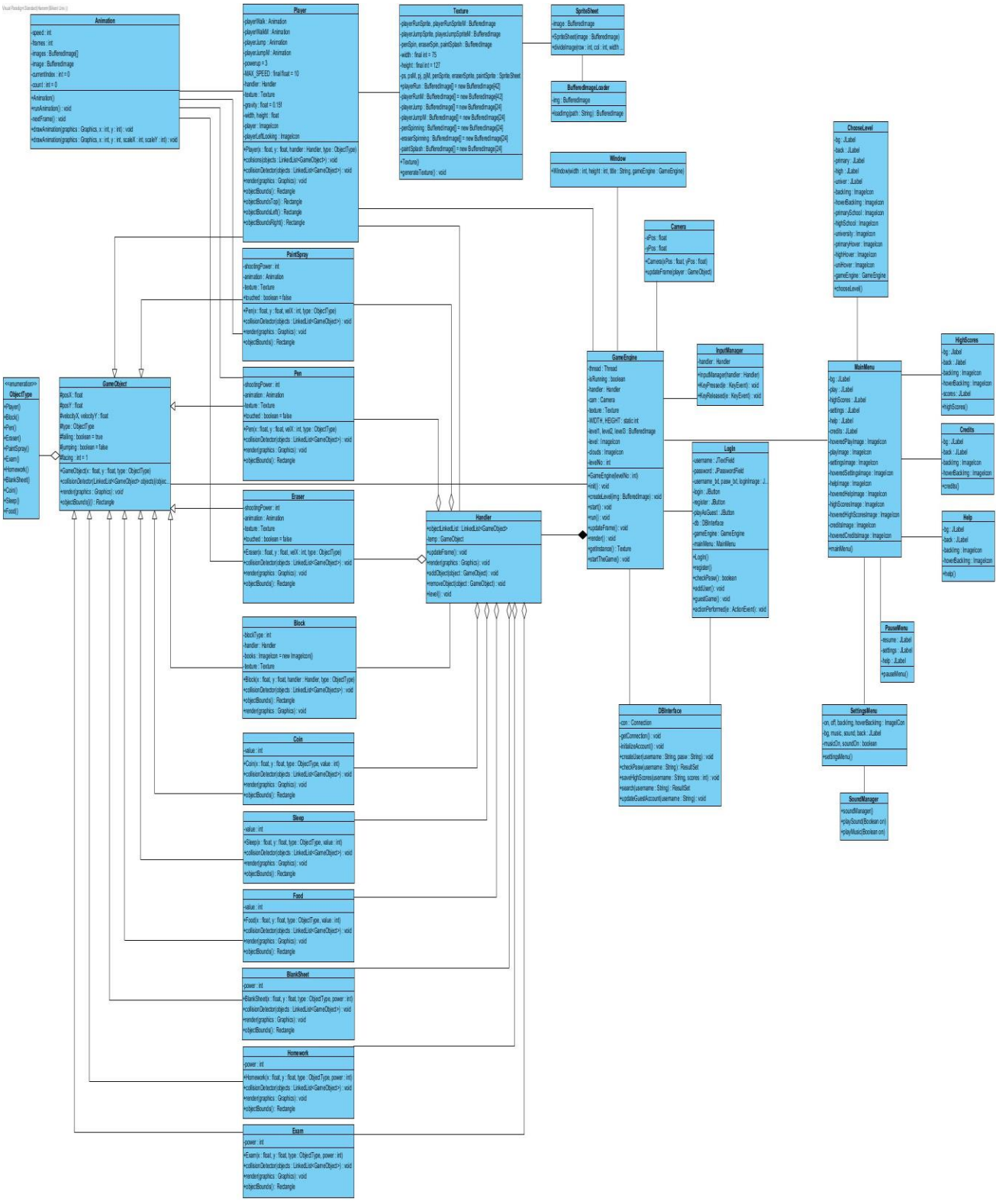


Figure 4 Final Object Design

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, PauseMenu, HighScores, ChooseLevel, Credits and Help 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, the choose level menu to choose the level of the game, help menu to help user navigate through the game and learn how to play, the high scores menu to check the highest scores and the credits menu to check about the developers of the game.

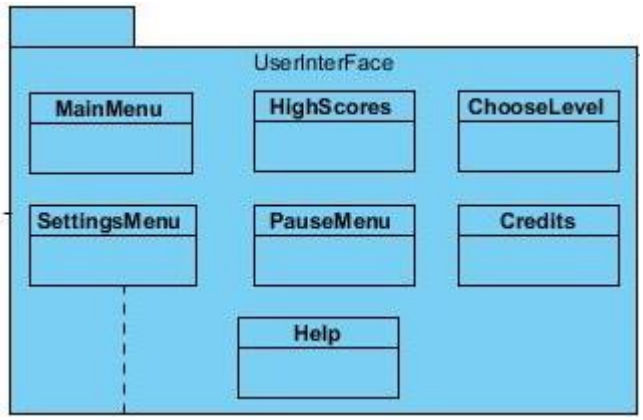


Figure 5- User Interface Subsystem

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 show up after the user has been granted access in the game.

Attributes

private JLabel bg: It is a JLabel property, which is used for displaying the background of this menu. All other labels will be children of this label. Hence the other labels will be added to the “bg” label.

private JLabel play: It is a JLabel property, which is used for displaying the “Play” label. On click event will triger the initialization of the “ChooseLevel” menu.

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

On click event will triger the game to quit.

private JLabel settings: It is a JLabel property, which is used for displaying the “Settings Menu” screen. On click event will triger the initialization of the “SettingsMenu”.

private JLabel help: It is a JLabel property, which is used for displaying the “Help Menu” screen. On click event will triger the initialization of the “Help” menu.

private JLabel highScores: It is a JLabel property, which is used for displaying the “High Scores Menu” screen. On click event will triger the initialization of the “HighScores” menu.

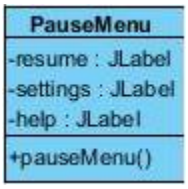
private JLabel credits: It is a JLabel property, which is used for displaying the “Credits Menu” screen. On click event will triger the initialization of the “Credits” menu.

private ImageIcon playImage, hoveredPlayImage, settingsImage, hoveredSettingsImage, helpImage, hoveredHelpImage, highScoresImage, hoveredHighScoresImage, creditsImage, hoveredCreditsImage: These images are all used for the labels to make the UI look more enhanced instead of using simple JButtons.

Methods

In this class there are no methods since all JLabel attributes will have their own mouse listener which will trigger the events.

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 , settings and help JLabels.

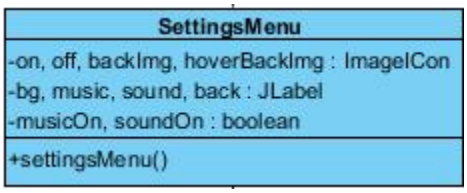
Attributes

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.

private JLabel settings: It is a JLabel property, which is used for displaying the settings menu.

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 ImageIcon on, off, backImg, hoverBackImg: Are image icons which are used to show the status of the sound (on, off) or to go to the previous menu (back).

private JLabel bg: This attribute is used to display the background image.

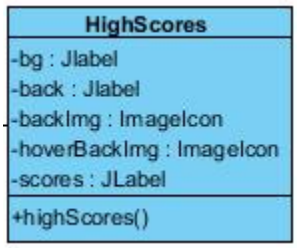
private JLabel music: It is a JLabel property, which is used for displaying the button of music. On click event this label will trigger to change the status of the music (on or off).

private JLabel sound: It is a JLabel property, which is used for displaying the button of sound. On click event this label will trigger to change the status of the sound (on or off).

private JLabel back: It is a JLabel property, which is used for displaying the back button. On click event this label will trigger to go back to the previous menu.

private boolean musicOn, soundOn: These flags will be used to check the status of music and sound. According to their value, the button images will be changed to on or off.

HighScores class



This is the user interface class of the High Scores Menu. It has JLabel properties to create the

High Scores Menu screen.

Constructor **public HighScores():** This constructor used for initializing the bg, scores and back JLabels.

Attributes

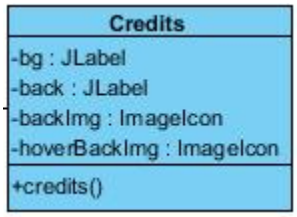
private ImageIcon backImg, hoverBackImg: Are image icons which are used to show the back button which helps the user go to the previous menu (back).

private JLabel bg: This attribute is used to display the background image.

private JLabel back: It is a JLabel property, which is used for displaying the back button. On click event this label will trigger to go back to the previous menu.

private Jlabel scores: This JLabel will display all the highest scores which will be fetched by the database.

Credits class



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

Constructor **public Credits():** This constructor used for initializing the bg and back JLabels.

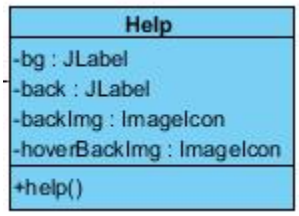
Attributes

private ImageIcon backImg, hoverBackImg: Are image icons which are used to show the back button which helps the user go to the previous menu (back).

private JLabel bg: This attribute is used to display the background image. The names of the developers will also be shown in the background image.

private JLabel back: It is a JLabel property, which is used for displaying the back button. On click event this label will trigger to go back to the previous menu.

Help class



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

Constructor **public Help():** This constructor used for initializing the bg and back Jlabels.

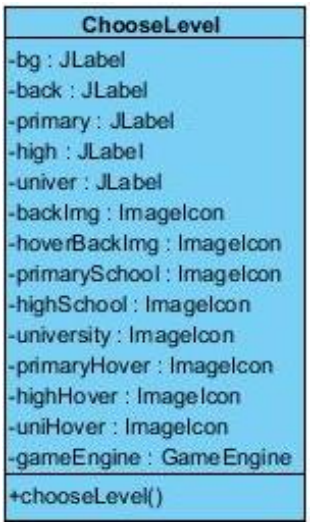
Attributes

private ImageIcon backImg, hoverBackImg: Are image icons which are used to show the back button which helps the user go to the previous menu (back).

private JLabel bg: This attribute is used to display the background image.

private JLabel back: It is a JLabel property, which is used for displaying the back button. On click event this label will trigger to go back to the previous menu.

ChooseLevel class



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

Constructor **public ChooseLevel():** This constructor used for initializing the bg, primary, high, univer and back Jlabels.

Attributes

private ImageIcon backImg, hoverBackImg, primaySchool, primaryHover, highSchool, highHover, university, uniHover: Are image icons which are used to show the different levels as buttons and also to provide the “on hover” effect when the mouse is over a JLabel. The back button helps the user go to the previous menu (back).

private JLabel bg: This attribute is used to display the background image.

private JLabel back: It is a JLabel property, which is used for displaying the back button. On click event this label will trigger to go back to the previous menu.

private JLabel primary: It is a JLabel property which is used to display the primary school button (first level). On click event this label will trigger to initialize the game with the first level.

private JLabel high: It is a JLabel property which is used to display the high school button (second level). On click event this label will trigger to initialize the game with the second level.

private JLabel univer: It is a JLabel property which is used to display the university button (last level). On click event this label will trigger to initialize the game with the last level.

private GameEngine gameEngine: This atribute will create an instance of the game engine with the level id provided according to the level chosen.

3.4 Game Management Subsystem

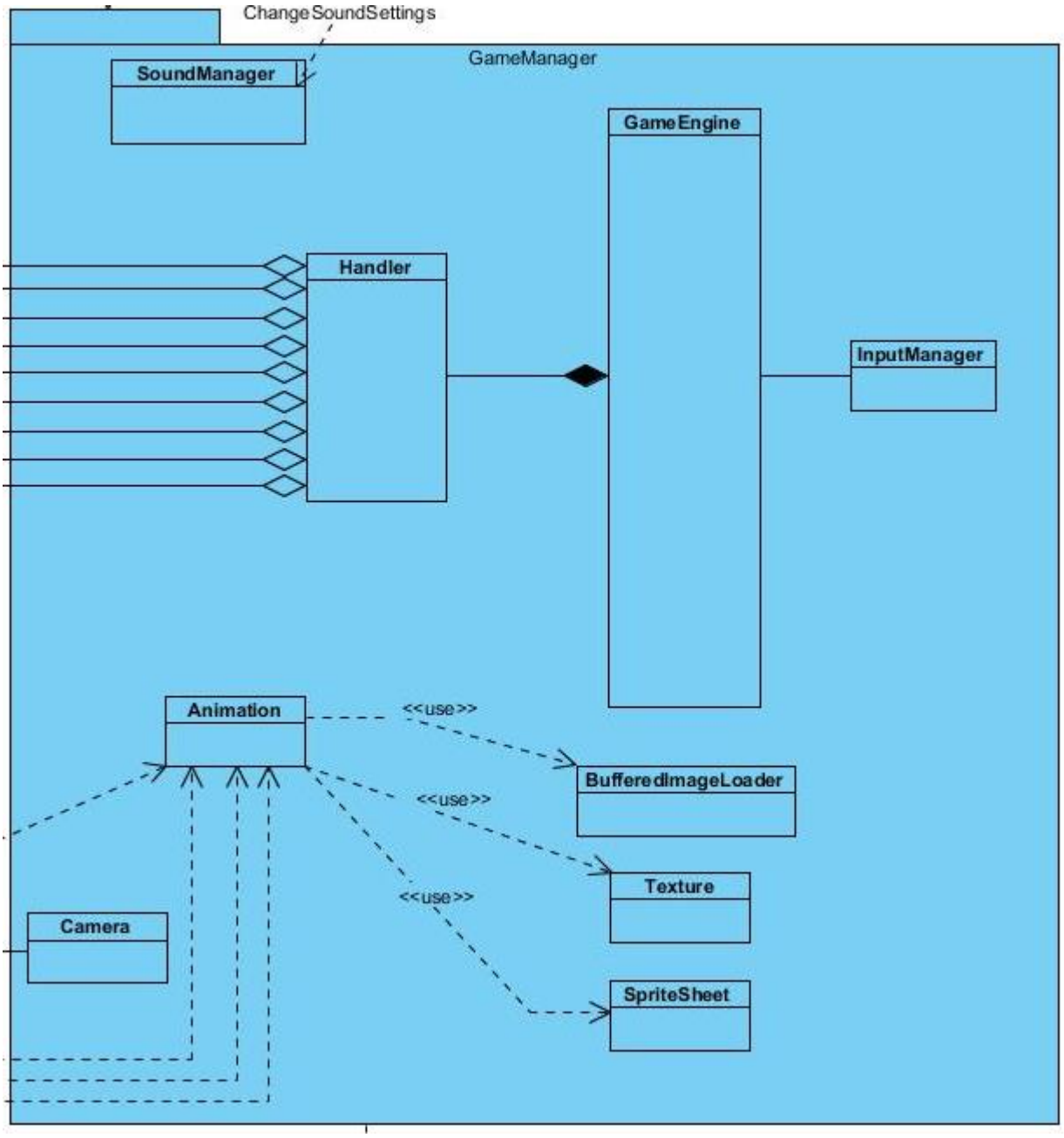
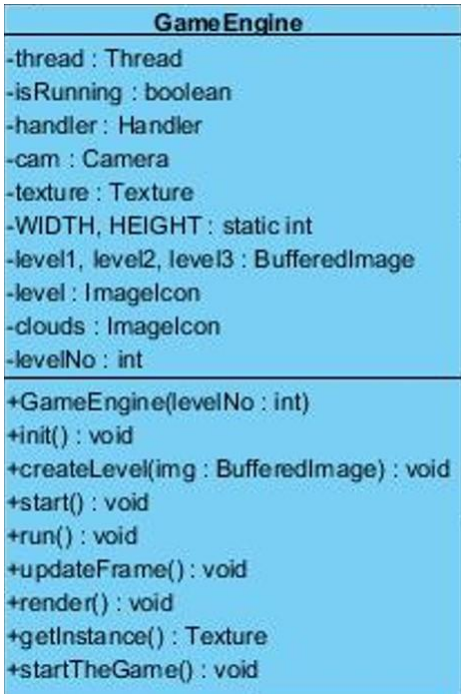


Figure 6- GameManager Subsystem

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(int levelNo): This constructor is responsible for initializing the level.

Attributes

private Thread thread : this attribute holds the thread object for the game.

private boolean isRunning : this attribute holds the boolean value of player if he is running or not.

private Handler handler: this attribute holds the handler object for event handling.

private Camera cam: this attribute holds the camera object of the game

static Texture texture : this attribute holds the Texture object.

public static int WIDTH, HEIGHT: these attributes holds the width and height of the image of player.

private BufferedImage level1, level2, level3: This attributes holds the background images of their levels.

private ImageIcon level: holds the image for the Level of the game

private ImageIcon clouds: This attribute holds the images of clouds.

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

Methods

public void init(): This method is for initializing all the entity and control objects of the game and for setting up display for the game play according to the level. it loads the images for each of the levels, then initializes texture, handler and camera objects. Then displays the image according to the level selected by the player. Then sends the handler object to the inputManager instance and adds that to the keyListener.

public void createLevel(BufferedImage img): this methods reads a picture pixel by pixel and adds an object to the frame in place of the pixel. The color of the pixel represents what kind of object it is, for example: player, block, enemy or powerUp.

public synchronized void start(): This method is responsible for creating a new thread and starting the thread, if there isnt one running already.

public void run(): This method calls updateframe and render methods to update the frame when the player is running.

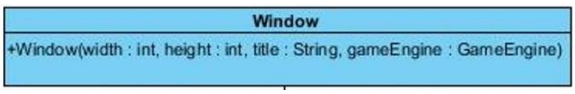
public void render(): This method will perform the rendering of the animations through the BufferedImage so that we increase the performance of the game. this method draws the graphics onto the screen by calling drawImage method.

private void updateFrame(): This method is for updating the frame by calling the hnadler's updateFrame method.

public static Texture getInstance(): This method returns the texture object.

public void startTheGame(): This method initiates the game play and load the display. It is executed after the player has pressed the "play game" button. This method creates a new window for the gameplay.

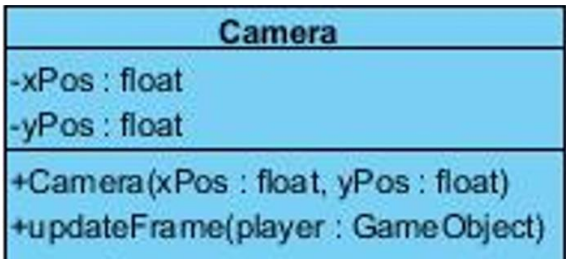
Window class



Constructor

public Window (int width, int height, String title, GameEngine gameEngine): This constructor sets the size, frames and images in a window. Its sets a window with the parameters width and height with a title of the parameter title and starts the gameEngine.

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 (float xPos, float yPos): This constructor initializes the horizontal and vertical positions of the frame.

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.

Methods

public void updateFrame(GameObject player): This method is updating the frame by getting the player's positions and adding that to half the window's width, so that the player always remains at the centre of the screen.

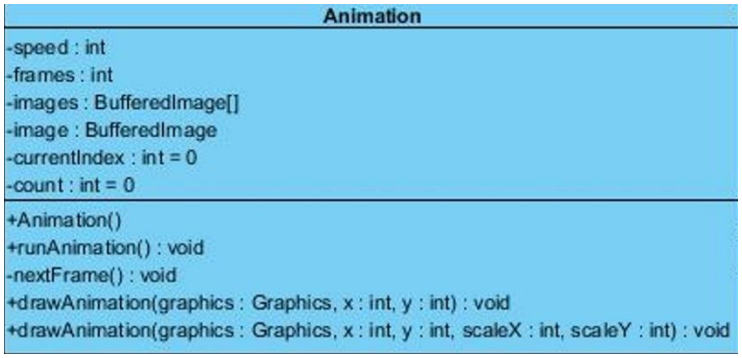
public float getXPos(): This method returns the float variable xPos.

public void setXPos(float xPos): This method sets the xPos variable to the parameter xPos.

public float getYPos(): This method returns the float variable yPos.

public void setYPos(float yPos): This method sets the yPos variable to the parameter yPos.

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(int speed, BufferedImage... args): This constructor initializes the Animation object by drawing all the objects for the first time and it saves each image to the array to be displayed for the movement of objects.

Attributes

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.

private BufferedImage image: : This attribute holds the image icon.

private int currentIndex : This attribute holds the current index and initialize to zero.

private int count : this is a counter to load as many frames as their are images.

Methods

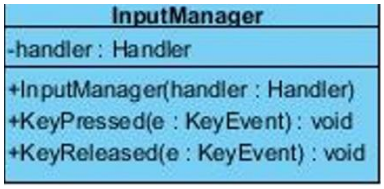
public void runAnimation(): This method keeps on incrementing the currentIndex until it becomes greater than the speed of this animation, after which it sets the currentIndex to zero and calls the nextFrame method, which displays the next frame. So basically this method is responsible for displaying the image icons according to the given speed.

public void drawAnimation(Graphics graphics, int x, int y): This method basically draws the graphics onto the screen at x and y position by calling the drawImage method.

public void drawAnimation(Graphics graphics, int x, int y, int scaleX, int scaleY): this is the overloaded version of the above method, the only difference is that it also takes the width and height of the graphics.

private void nextFrame(): This method loads each image from the image array and keeps there to display, meaning it loads the next frame of the animation, and keeps on incrementing the count until it becomes greater than the number of frames of this animation, after which it sets the count to zero, meaning the animation starts all over again.

InputManager class



This class is for directing the player according to the keyboard inputs.

Constructor

public InputManager(Handler handler): This constructor will initialize the handler variable.

Attributes

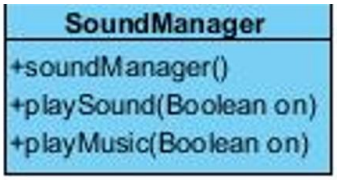
Handler handler: this attribute is responsible for storing the handler instance of the game.

Methods

public void keyPressed(KeyEvent event): This is a method for managing the pressed input and creating an event accordingly to that. This method takes event variable as a parameter and gets the key code of event. If the key is equal to the space bar or arrow keys the method creates an event accordingly. For each time a key is pressed, the player object in handler's objectLinkedList, is directed to do something, for example: UP or space = jump, DOWN = move down fastly if the player is in the air, LEFT = move east, RIGHT = move west, ESCAPE = quit game.

public void keyReleased(KeyEvent event): This is a method for managing the released input and creating an event according to that. This method takes event variable as a parameter and gets the key code of event. If the key is equal to the space bar or arrow keys the method creates an event accordingly. For example, if the key A is released then a pen is thrown.

SoundManager class



SoundManager class is for managing the sounds and music.

Constructor **public SoundManager():** This constructor will set the music and sounds to 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.

BufferedImageLoader Class

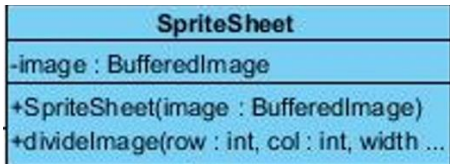
Attributes

private BufferedImage img: This attribute holds the image icons for the objects.

Methods

public BufferedImage loadImg (String path): This method takes string path as a parameter loads the image in the path with the help of calling getResource method and returns that image.

SpriteSheet Class



Constructor

public SpriteSheet(BufferedImage image): This constructor initializes the image variable.

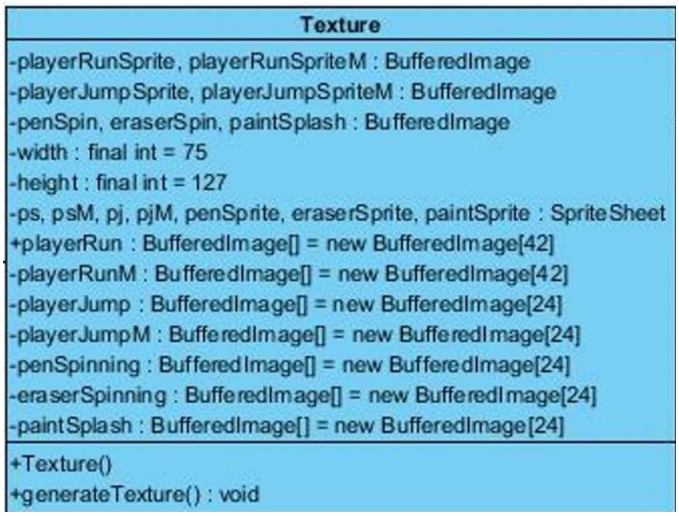
Attributes

private BufferedImage img: This attribute holds the image of the sprite sheet that contains all the icons for each object's animation

Methods

public BufferedImage divideImage(int row, int col, int width, int height): this method divides the sprite sheet into parts and gets each single image for the animation of an object. It uses the getSubImage method to get the image at (x,y) coordinate with the given width and height and returns that image.

Texture Class



Constructor

public Texture(): this constructor initializes the running, jumping and weapon throwing

animations of the player.

Attributes

private BufferedImage playerRunSprite, playerRunSpriteM, playerJumpSprite: these attributes save the sprite sheets for the running and jumping animation.

public BufferedImage[] playerRun : this attribute holds each image for the running animation towards right.

public BufferedImage[] playerRunM : this attribute holds each image for the running animation towards left.

public BufferedImage[] playerJump: this attribute holds each image for the running animation towards right.

public BufferedImage[] playerJumpM : this attribute holds each image for the running animation towards left.

public BufferedImage[] penSpinning : this atttribute holds each image for the animation of pen spinning.

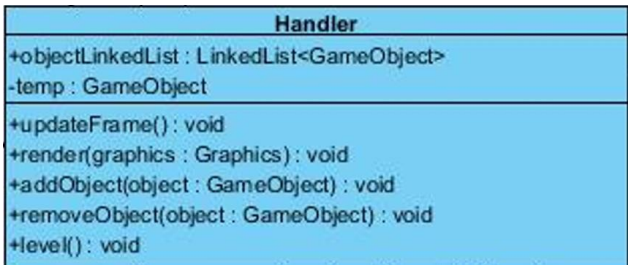
private final int width, height: these hold the height and width of the player images that we use.

private SpriteSheet ps, psM, pj, pjM, penSprite: these hold the spritesheets for running, jumping and weapon throwing animations.

Methods

public void generateTextures(): this method calls the divideImage method of the SpriteSheet class and gets each single image from the spriteSheets and stores them into the respective arrays for animations.

Handler Class



Attributes

public LinkedList<GameObject> objectLinkedList: holds all the GameObjects that have been created during the since the start of the game.

private GameObject temp: a temporary GameObject variable to store any gameObject.

Methods

public void updateFrame(): this method calls the collsionDetector for each object in the objectLinkedList, which is responsible for calling the runAnimation method of that object. So basically whenever this method is called the frames of the object are updated, by calling nextFrame method in runAnimation.

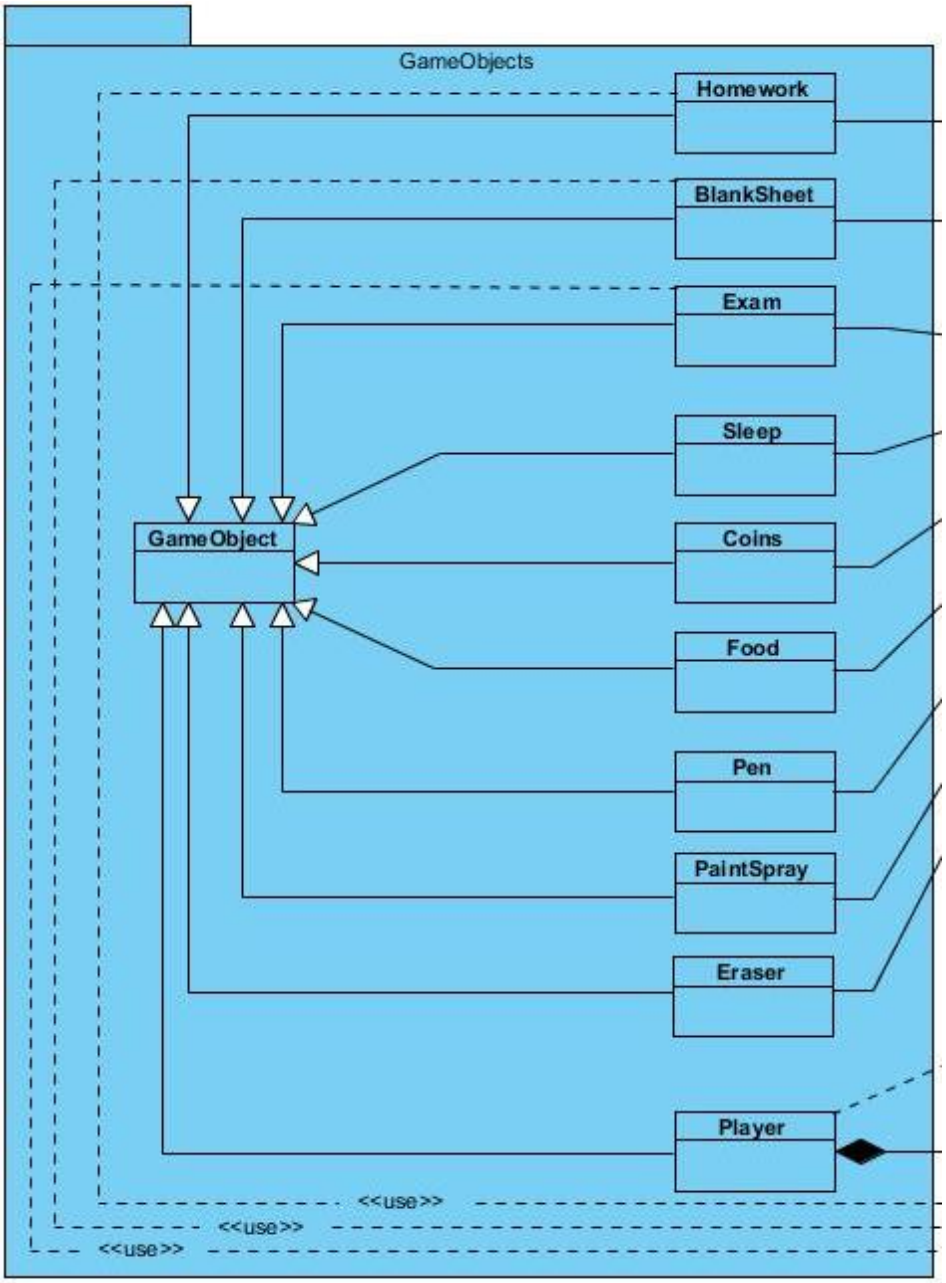
public void render(Graphics graphics): this method calls the render method for each object in the objectLinkedList, which is responsible for calling the drawImage method of that object, so that the graphics are drawn and updated.

public void addObject (GameObject object): this method adds a new object to the objectLinkedList of this class by calling the add method of the linkedList class.

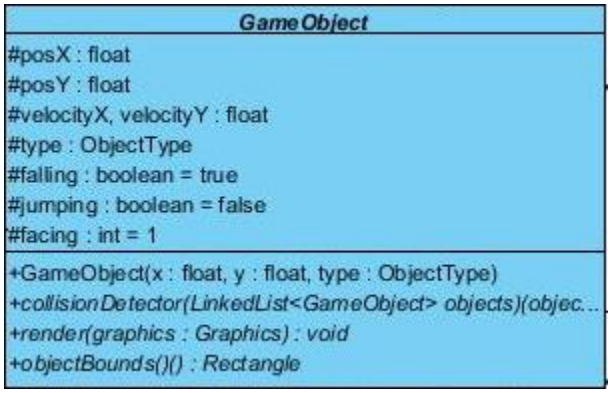
public void removeObject (GameObject object): this method removes an object from the objectLinkedList of this class by calling the remove method of the linkedList class.

public void level(): this method is just a test method for testing a sample level on the sceen by adding new block objects to the objectLinkedList.

3.5 Game Objects Subsystem



GameObject class



The GameObject is the parent object for all the object classes of the game. 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. Also it determines whether an object is falling or jumping and also their velocity.

Constructor

public GameObject (float x, float y, ObjectType type): This constructor initializes the objects for GameObject class. The first two parameters define the horizontal and vertical positions while the type is an object ID defined according to the enumerator ObjectType.

Attributes

protected float posX : This attribute holds the horizontal position of the object on the frame as a float.

protected float posY: This attribute holds the vertical position of the object on the frame as a float.

protected float velocityX , velocityY : This attributes holds horizontal and vertical velocities of the object on the frame as a floats. These attributes are firstly initialized to zero.

protected ObjectType type: This attributes holds the type of the object. Te type of the object can be Player, Block, Pen, PaintSpray, Eraser, Coins, Food, Sleep, Homework, Exam, BlankSheet.

protected int facing : This attribute is used to determine whether the player is facing left or right. According to the value of this attribute will be determined not only the player images but also the direction of the shooting.

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

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

Methods

public boolean isFalling(): This method returns the value of the falling attribute.
public void setFalling(boolean falling): This method assigns the falling parameter to the falling attribute.
public boolean isJumping(): This metod returns the value of the jumping attribute.
public void setJumping(boolean jumping): This method assigns the jumping parameter to the jumping attribute.
public int getFacing(): This method returns the integer variable facing.
public void setVelocityY (float velY): This method sets the parameter velocityY to the variable velY.
public void setVelocityX (float velX): This method sets the parameter velocityX to the variable velX.
public void setPosY (float y): This method sets the parameter y to the variable posX.
public void setPosX(float x): This method sets the parameter x to the variable posY.
public ObjectType getType(): This method returns the object type.

public float getVelocityY(): This method returns the float variable velocityY.

public float getVelocityX(): This method returns the float variable velocityX.

public float getPosX(): This method returns the float variable posX.

public float getPosY(): This method returns the float variable posY.

public abstract void collisionDetector(LinkedList<GameObject> objects): This abstract method will be overriden in all the game objects which will extend this class in order to detect whether they collide with another object or not.

public abstract void render (Graphics graphics): This abstract method will also ve overriden in the game objects that will extend this class. This method will be used to render the animations of the objects and draw them on the Graphics.

public abstract Rectangle objectBounds(): This abstract method will be overriden in the objects which will extend this class. It will return a rectangle which will define the boundaries of the object.

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 and jumping to the left or right. It also has a Handler attribute which is the same handler created in the GameEngine class. This class uses the Handler to interact with other objects of the game.

Attributes

private Animation playerWalk, playerWalkM, playerJump, playerJumpM: these attributes are instances of the Animation class and they will be used to display the player movements.

private int powerup: This attributes defines the initial value of the powerups with which the player starts the game.

private final float MAX_SPEED: this attribute is a final int which defines the maximum speed by which the player can move.

private Handler handler: the handler attribute is an allias of the same handler created in the GameManager class. This attribute is used to interact with the other game objects of the game.

private Texture texture: This is an instance of the texture class which will be used to get all the images for the object animations.

private float gravity: This attribute defines the speed by which the object will fall down.

private float width, height: These attributes define the size of the player.

private ImageIcon player, playerLookingLeft: These attributes define the images of the player in the Idle state.

Methods **public void collisions(LinkedList<GameObject> objects):** this method is responsible for checking if the player has collided with an enemy or an object(i.e, powerUp or obstacle). According to this collision the player interacts. If it is a block it stands over it. If it is an enemy one life is lost.

public void collisionDetector (LinkedList<GameObject> objects): checks whether the player is in collision with any other object or not. According to that it plays the animations. If the player isn't touching any object then the falling animation is played.

public void render (Graphics graphics): this method renders all the animations and draws them on the graphics parameter provided.

public Rectangle objectBounds(): This method return the lower bounds of the player in a rectangular shape.

public Rectangle objectBoundsTop(): This method return the upper bounds of the player in a rectangular shape.

public Rectangle objectBoundsLeft(): This method return the left bounds of the player in a rectangular shape.

public Rectangle objectBoundsRight(): This method return the right bounds of the player in a rectangular shape.

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(float x, float y, ObjectType type, int velX): The constructor of this class will call the super() function to initialize the x, y, and type variables to parent classes parameters. In addition, this consturctor initialize the velocityX and animation.

Attributes

private Animation animation: This attribute is for showing an animation of Pen.

private Texture texture:

public boolean touched:

Methods

public void collisionDetector(LinkedList<GameObject> objects):

public void render(Graphics graphics): This method overrides the render method in GameObject class. The purpose of this method is to draw the image of pen. Inside of this method graphics object call drawImage method to draw the image of pen.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class. The method creates and returns a Rectangle 35X35 pixels to determine the object Bounds.

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 (float x, float y, ObjectType type, int velX): The constructor of this class will call the super() function to initialize the x, y, and type variables to parent classes parameters. In addition, the image of the eraser weapon will be initialized in the constructor.

Attributes

private Animation animation: This attribute is for showing an animation of eraser.

private Texture texture:

public boolean touched:

Methods

public void collisionDetector(LinkedList<GameObject> objects):

public void render(Graphics graphics): This method overrides the render method in GameObject class. The purpose of this method is to draw the image of eraser. Inside of this method graphics object call drawImage method to draw the image of eraser.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class. The method creates and returns a Rectangle 35X35 pixels to determine the object Bounds.

PaintSpray class

PaintSpray class extend the weapon class. PaintSpray 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 paintspray for a limited time.

Constructor

public paintSpray(float x, float y, ObjectType type, int velX): The constructor of this class will call the super() function to initialize the x, y, and type variables to parent classes parameters. In addition, the image of the paint spray weapon will be initialized in the constructor.

Attributes

private Animation animation:

private Texture texture:

public boolean touched:

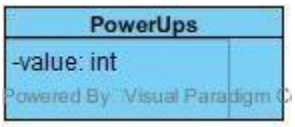
Methods

public void collisionDetector(LinkedList<GameObject> objects):

public void render(Graphics graphics): This method overrides the render method in GameObject class. The purpose of this method is to draw the image of paint spray. Inside of this method graphics object call drawImage method to draw the image of paint spray.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class. The method creates and returns a Rectangle 35X35 pixels to determine the object Bounds.

PowerUp class



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

Constructor

public PowerUps(float x, float y, ObjectType type, int value) The constructor of this class will initialize the value of this power up to 0. In addition with the help of calling super(), the x, y and type variables is equalize to GameObject parameters.

Attributes

private int value: The integer value will show the value of the power-ups. All of the powerups will be initialized in the amount of 0. In the game, as user collect the power-up this value will be increased.

Methods

public void setValue(int value): This method sets the value variable in class to the parameter value.

public int getValue(): This methos returns the integer value.

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

public void render(Graphics graphics): This method overrides the render method in GameObject class.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class.

Food class

Food class extends the PowerUp class. Food has all of the features and methods of PowerUp 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(float x, float y, ObjectType type, int value): The constructor of this class will initialize the value of this power up to 3. In addition with the help of calling super(), the x, y and type variables is equalize to parent classes parameters.

Attributes

private ImageIcon food: This attribute declarate the food ImageIcon to the food image in the Images folder.

private int value: The integer value shows the amount of the food lefted. The variable of the Food value will be privatized to 3 with the help of the constructor. In the game, as user collect the foods this value will be increased hovewer in each fight this value will be decreased.

Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

public void render(Graphics graphics): This method overrides the render method in GameObject class. The purpose of this method is to draw the image of food. Inside of this method graphics object call drawImage method to draw the image of food.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class. The method creates and returns a Rectangle 32X32 pixels to determine the object Bounds.

Sleep class

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

Constructor

public sleep (float x, float y, ObjectType type, int value): The constructor of this class will initialize the value of this power up to 3. In addition with the help of calling super(), the x, y and type variables is equalize to parent classes parameters.

Attributes

private ImageIcon bed: This attribute declarate the bed ImageIcon to the bed image in the Images folder.

private int value: The integer value shows the amount of the sleep in the player. The variable of the Sleep value will be privatized to 3 with the help of the constructor. In the game, as

user collect the bed images this value will be increased hovewer in each fight this value will be decreased.

Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

public void render(Graphics graphics): This method overrides the render method in GameObject class. The purpose of this method is to draw the image of sleep. Inside of this method graphics object call drawImage method to draw the image of sleep.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class. The method creates and returns a Rectangle 40X40 pixels to determine the object Bounds.

Coins class

Coins class extends the PowerUp class. Coins has all of the features and methods of PowerUp 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 weapons with the help of this coins.

Constructor

public coins (float x, float y, ObjectType type, int value): The constructor of this class will call the super() function to initialize the x, y, value and type variables to parent classes parameters. (The initialization value of the Coins will be zero.)

Attributes

private ImageIcon coin: This attribute declarate the bed ImageIcon to the coin image in the Images folder.

private int value: The integer value shows how much Money a player has. The variable of the Coin value will be privatized to 0 with the help of calling the süper constructor. In the game, as user collect the coin images this value will be increased, in the case of fight other than the rest of the powerups the value dont change.

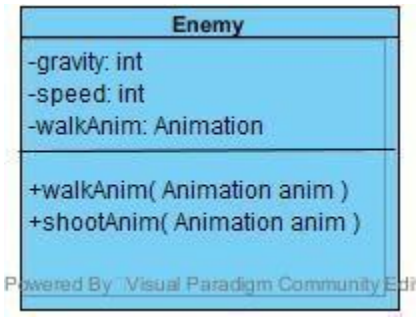
Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

public void render(Graphics graphics): This method overrides the render method in GameObject class. The purpose of this method is to draw the image of coin. Inside of this method graphics object call drawImage method to draw the image of coin.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class. The method creates and returns a Rectangle 28X28 pixels to determine the object Bounds.

Enemy class



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

Constructor

public Enemy(float x, float y, ObjectType type, float gravity): The constructor of this class will call the super() function to initialize the x, y, and type variables to Game object class's parameters. Additionally, the float gravity variable is also initialized.

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.

Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

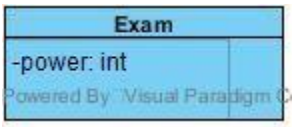
public void render(Graphics graphics): This method overrides the render method in GameObject class.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class.

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 exam (float x, float y, ObjectType type, float gravity , int speed, int power): The constructor of this class will call the super() function to initialize the x, y, gravity, and type variables to Enemy class's parameters. Additionally, speed variable is equalized to 2 and the power variable is equalized to 5.

Attributes

private int power: This attribute shows the amount of the power an exam has. The most powerful enemy is the exam with the value of power 5. In the case of fight, it decrease and when it is zeroed, the enemy will die.

private int speed: This attribute shows the amount of the speed an exam has. The fastest enemy is the exam with the value of speed 2.

Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

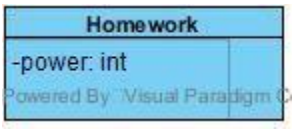
public void render(Graphics graphics): This method overrides the render method in GameObject class.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class.

public void walkAnim(Animation anim) : This method overrides the walkAnim method in enemy class. This method's purpose is to show a walking exam enemy

Public void shootAnim(Animation anim) : This method overrides the shootAnim method in enemy class This method's purpose is to show a shooting exam enemy.

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 homework (float x, float y, ObjectType type, float gravity , int speed, int power): The constructor of this class will call the super() function to initialize the x, y, gravity, and type variables to Enemy class's parameters. Additionally, speed variable is equalized to 1 and the power variable is equalized to 2.

Attributes

private int power: This attribute shows the amount of the power an homework enemy has. The second most powerful enemy is the homework with the value of power 3. In the case of fight, the power value is decreased and when it is zeroed, the enemy will die.

private int speed: This attribute shows the amount of the speed an homework has. The speed of homework enemy is 1.

Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

public void render(Graphics graphics): This method overrides the render method in GameObject class.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class.

public void walkAnim(Animation anim) : This method overrides the walkAnim method in enemy class. This method's purpose is to show a walking homework enemy

public void shootAnim(Animation anim) : This method overrides the shootAnim method in enemy class This method's purpose is to show a shooting homework enemy

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 blankSheet(float x, float y, ObjectType type, float gravity , int speed, int power): The constructor of this class will call the super() function to initialize the x, y, gravity, and type variables

to Enemy class's parameters. Additionally, speed variable is equalized to 1 and the power variable is equalized to 1.

Attributes

private int power: This attribute shows the amount of the power an blanksheet enemy has. The least powerful enemy is the blanksheet with the value of power 1. In the case of fight, the power value is decreased and when it is zeroed, the enemy will die.

private int speed: This attribute shows the amount of the speed an blanksheet has. The speed of homework enemy is 1.

Methods

public void collisionDetector(LinkedList<GameObject> objects): This method overrides the collisionDetector method in GameObject class.

public void render(Graphics graphics): This method overrides the render method in GameObject class.

public Rectangle objectBounds(): This method overrides the objectBounds method in GameObject class.

public void walkAnim(Animation anim) : This method overrides the walkAnim method in enemy class. This method's purpose is to show a walking blanksheet enemy

public void shootAnim(Animation anim) : This method overrides the shootAnim method in enemy class. This method's purpose is to show a shooting blanksheet enemy

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.

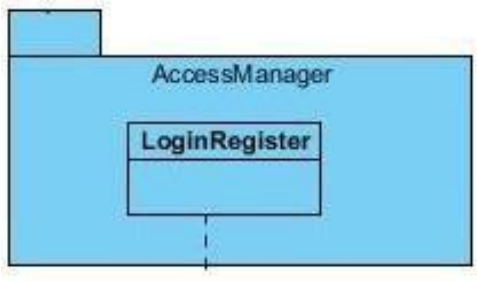
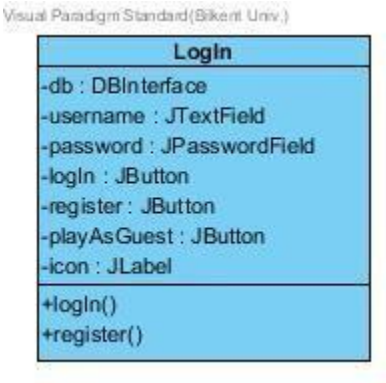


Figure 7- AccessManager Subsystem

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

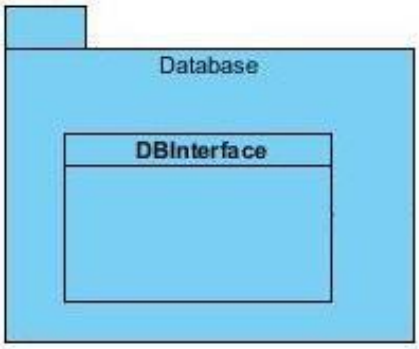
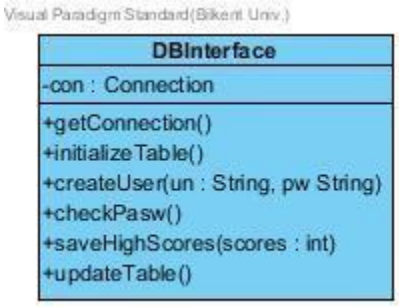


Figure 8- Database Subsystem

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



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.