CS 319 - Object-Oriented Software Engineering
Design Report

Group 2E:

Ferhat Serdar Atalay
Aylin Çakal
Ali Bulut
İsmail Serdar Taşkafa

Submission: 10.03.18

1. Introduction

1.1. Purpose of the System

Planet Trip is a 2-D arcade game similar to DX-Ball. User tries to destroy all meteors in current level of the map. To do this user only controls a paddle for giving new direction to the ball. When the ball destroys a meteor, a game bonus can appear. If user wants to get this bonus it has to collect that bonus by moving paddle. The user's aim is to finish all levels successfully. We designed the game in such a way that at each level the speed of ball increases and olsa levels became more difficult. Also to increase challenge, a enemy paddle could appear. Planet Trip aims to improve quick response ability to a sudden changes and increase hand-eye coordination and also good time to spend.

1.2. Design Goals

1.2.1. End User Criteria

1.2.1.1. Ease of Use

From the welcoming screen, every part of the game experience should be self explaining. For this purpose we find it useful to use the terms that players are familiar with in the gaming culture. The main purpose is to help the user find their way through without further explanations needed.

1.2.1.2. Good Documentation

In a quiz game, there should be no questions which will lead to several answers. We need to prevent any subjective topics or ambiguity that will confuse players mind or put them in a dilemma. All and each of the questions must be understood clearly, and have singular answers.

1.2.2. Maintenance Criteria

1.2.2.1. Extendibility:

Extendibility is the ease of adding new features to the software. It is an important criteria in development of any software system as new requirements can be added to the system as result of feedback from users or the product owner. What can we add?

1.2.2.2. Adaptability:

Usage of object oriented design principles implies each component of the software is independent of each other. Because of that, changes in one part does not affect the functioning of other parts. This makes making modifications on our software and testing it more straightforward.

1.2.2.3. Portability:

Portability is the ease of porting the software to another platform. We are currently developing our game as a Java based desktop application. However, there is an ever increasing popularity of mobile apps. Portability will enable us to port the game to mobile platforms in a simpler way.

1.2.3. Performance Criteria

1.2.3.1. Minimum Number of Errors

Minimum number of error is important for an application. Because, user wants to use a program without any trouble. While playing a game, any crash can cause with loss of progress which is unwanted situation for users. Because of that, minimizing the number of error is important for performance.

1.2.3.2. Reusability

We designed our classes in a fashion that they will be able to reused again for further upgrades in project. Our aim is to have the least amount of need for modification on our code when another part is changing.

1.2.4. Trade-Offs

1.2.4.1. Functionality vs. Usability

Since we want to user friendly game, it should be easy to use. In other words, users should not encounter any usage difficulties. Planet Trip game will provide user-friendly interface to user. Because of that, we give priority to usability rather than functionality.

1.2.4.2. Rapid Development vs. Functionality

For the purpose of rapid development, the game will not cover high functionality. Because the project has to be completed in a limited time, implementing functional things will not be important for us.

1.2.4.3. Portability vs. Efficiency

Our application will have portable environment so it will work on various devices that has different specs. Since we can not cover all devices, the game will not be running efficiently on all devices.

1.2.4.4. Memory vs. Performance

Main purpose of the Planet Trip is challenging and entertaining user. To do that the game should run smoothly. To do that we need to keep the response time as low as possible. Because of that we keep some same instances at various places.

2. Software Architecture

2.1. Subsystem Decomposition

In order to clearly show how our system works, we divided it into pieces. To provide a good organisation, we categorised the subsystems by their missions. This method enables us to choose between different architectural styles, and we decided to go on with Model View Controller since it seem the most suitable for our project.

In Figure 2.1, three main subsystems of our project are shown. Those User Interface Management, Game Management, and Game Entities. They have been separated since they are operating on different tasks. We aimed to decrease coupling and reduce dependency in that design. The connections between subclasses are shown in the next figure.

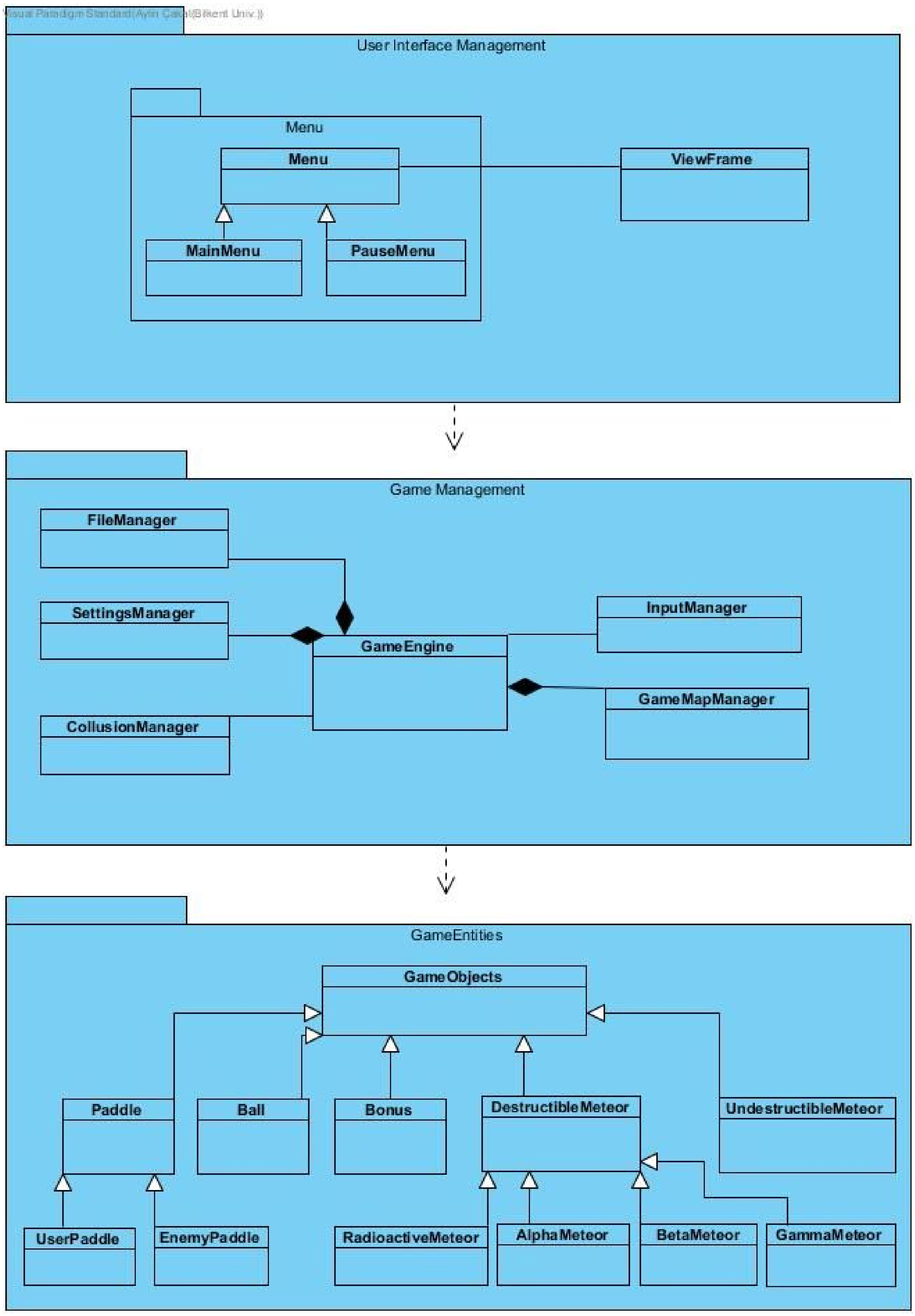


Figure 2.1 : Basic Subsystem Decomposition

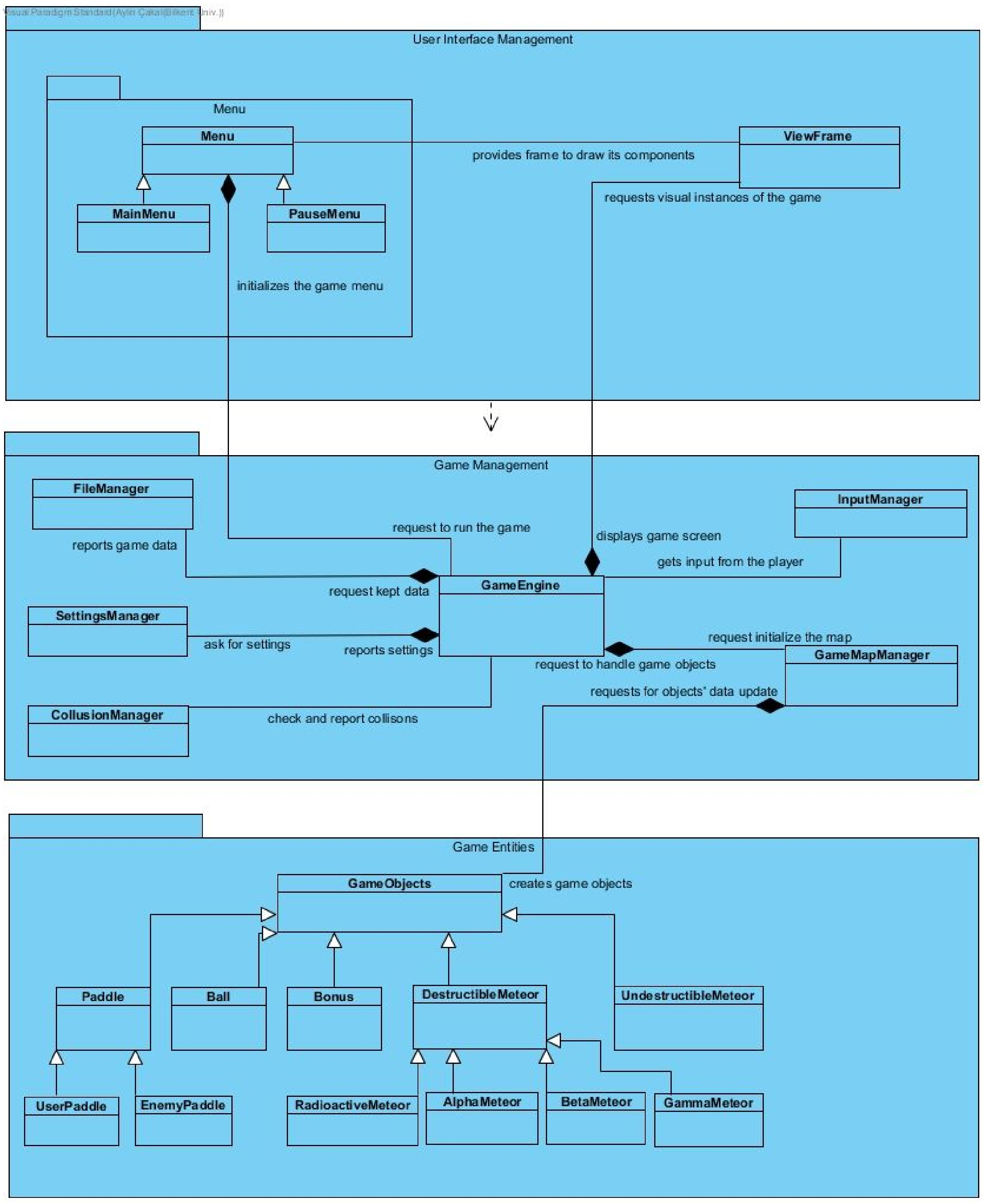


Figure 2.2: Detailed Subsystem Decomposition

2.2. Hardware/Software Mapping

Our game needs a keyboard for the inputs, and a mouse for the settings adjustment. Since we will implement the project with java, a basic computer with java compiler to compile & run the .java file will fulfill the requirements of the game.

2.3. Persistent Data Management

Planet Trip does not need for any databases or data management system like SQL. The game files such as images and text files will be stored in hard drive. So every time the game has started, the software loads needed files from the related file path on the hard drive. Also to find the where data is, we will keep path of a file in the codes. On the other hand, the software has to save options and progress of an user. To do this, the software also override related text files.

2.4. Access Control and Security

Since Planet Trip game has nothing to do with any network connection or database, there is no need to any security protocols. The game also does not need for any personal data and because all game data is stored separately for each device, there is no need to any access control. Any change in a file will only affects that local programme.

2.5. Boundary Conditions

2.5.1. Initialization

Our game will be able to played without installation, since it will come with an executable .jar file.

2.5.2. Termination

User will be able to leave the game through clicking “Exit” button in Main Menu. For those who want to quit during the game play, the “Exit” button also exists in Pause Menu. And since the game will run in a window, it is possible to close the application with “x” button on the upper right.

2.5.3. Failure

If the game cannot load a file, related area for that object will left blank. In that case there is a problem with the file paths or file source.

3. Subsystem Services

3.1. User Interface Subsystem

The user interface subsystem is the component with which user will interact. System will receive inputs and send output via there. Also, transitions between different panels, such as main menu to game screen, will be performed by that. Menu provides template for different types of menus, namely main and pause menu.

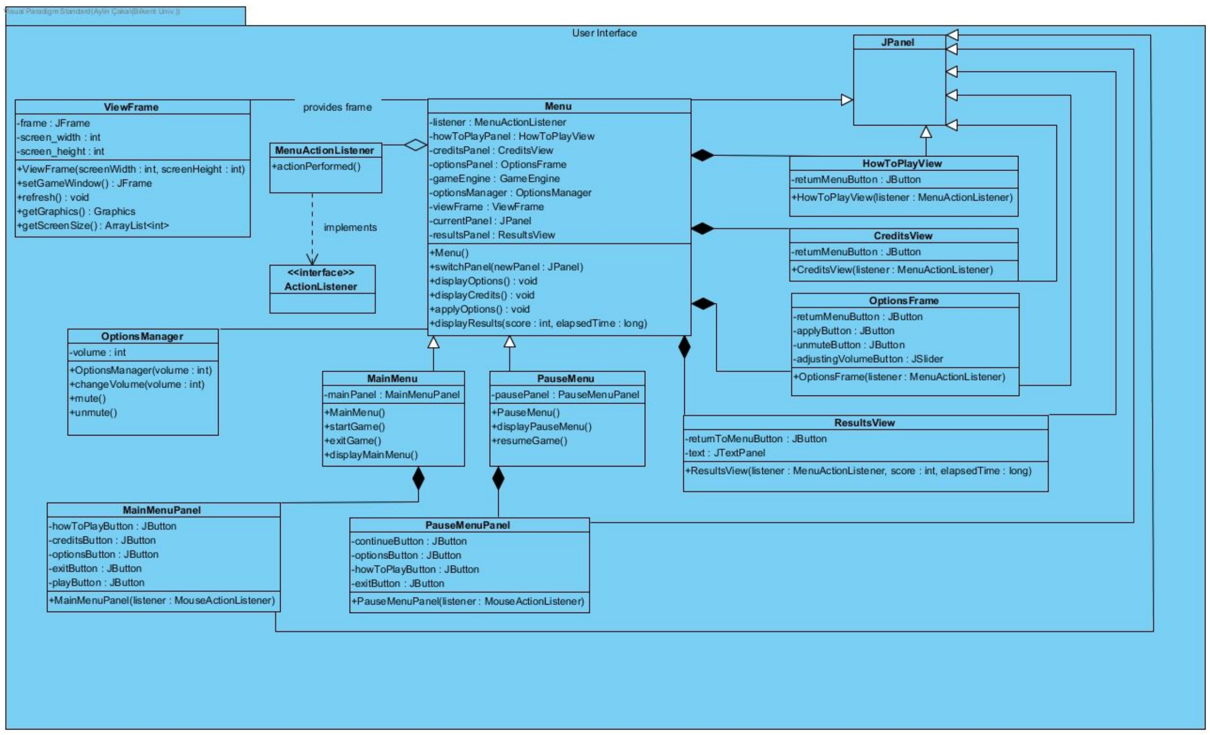
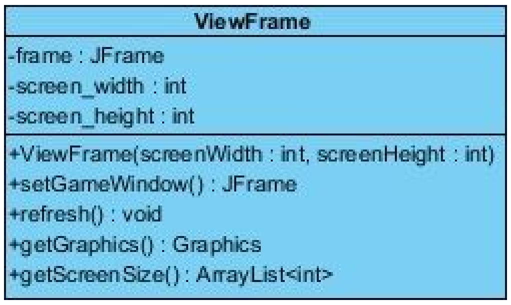

Visual Paradigm Standard(Aylin Çakal(Bilkent Univ.))
User Interface
ViewFrame
-frame : JFrame
-screen_width : int
-screen_height : int
+ViewFrame(screenWidth : int, screenHeight : int)
+setGameWindow() : JFrame
+refresh() : void
+getGraphics() : Graphics
+getScreenSize() : ArrayList<int>
provides frame
MenuActionListener
+actionPerformed()
implements
<<interface>>
ActionListener
Menu
-listener : MenuActionListener
-howToPlayPanel : HowToPlayView
-creditsPanel : CreditsView
-optionsPanel : OptionsFrame
-gameEngine : GameEngine
-optionsManager : OptionsManager
-viewFrame : ViewFrame
-currentPanel : JPanel
-resultsPanel : ResultsView
+Menu()
+switchPanel(newPanel : JPanel)
+displayOptions() : void
+displayCredits() : void
+applyOptions() : void
+displayResults(score : int, elapsedTime : long)
JPanel
HowToPlayView
-returnMenuButton : JButton
+HowToPlayView(listener : MenuActionListener)
CreditsView
-returnMenuButton : JButton
+CreditsView(listener : MenuActionListener)
OptionsFrame
-returnMenuButton : JButton
-applyButton : JButton
-unmuteButton : JButton
-adjustingVolumeButton : JSlider
+OptionsFrame(listener : MenuActionListener)
OptionsManager
-volume : int
+OptionsManager(volume : int)
+changeVolume(volume : int)
+mute()
+unmute()
MainMenu
-mainPanel : MainMenuPanel
+MainMenu()
+startGame()
+exitGame()
+displayMainMenu()
PauseMenu
-pausePanel : PauseMenuPanel
+PauseMenu()
+displayPauseMenu()
+resumeGame()
ResultsView
-returnToMenuButton : JButton
-text : JTextPanel
+ResultsView(listener : MenuActionListener, score : int, elapsedTime : long)
MainMenuPanel
-howToPlayButton : JButton
-creditsButton : JButton
-optionsButton : JButton
-exitButton : JButton
-playButton : JButton
+MainMenuPanel(listener : MouseActionListener)
PauseMenuPanel
-continueButton : JButton
-optionsButton : JButton
-howToPlayButton : JButton
-exitButton : JButton
+PauseMenuPanel(listener : MouseActionListener)

Figure 3.1 : User Interface Subsystem

ViewFrame Class

This class provides frame for Menu class.


ViewFrame
-frame : JFrame
-screen_width : int
-screen_height : int
+ViewFrame(screenWidth : int, screenHeight : int)
+setGameWindow() : JFrame
+refresh() : void
+getGraphics() : Graphics
+getScreenSize() : ArrayList<int>

Attributes:

private JFrame frame: The main frame for our program. All visual content, such as menu or game screen will be displayed through that.

private int screen_width: Width of the frame.

private int screen_height: Height of the frame.

Constructor:

public ViewFrame(int screenWidth, int screenHeight): It takes width and height and creates a frame with given sizes.

Methods:

public JFrame setGameWindow(): Returns a JFrame object to display the game screen.

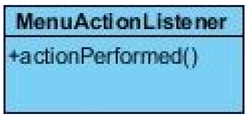
public void refresh(): Refreshes the frame and applies changes after the latest refresh or load to frame.

public Graphics getGraphics(): Returns a graphics context. With that graphics, drawing of off-screen images will be enabled.

public ArrayList<int> getScreenSize(): Size of the frame will be returned as an array list with length two.

MenuActionListener

Listens for any user activity on menu. If user presses one of the buttons on the screen or scrolls the sound level bar on settings menu, it will inform the controller accordingly.

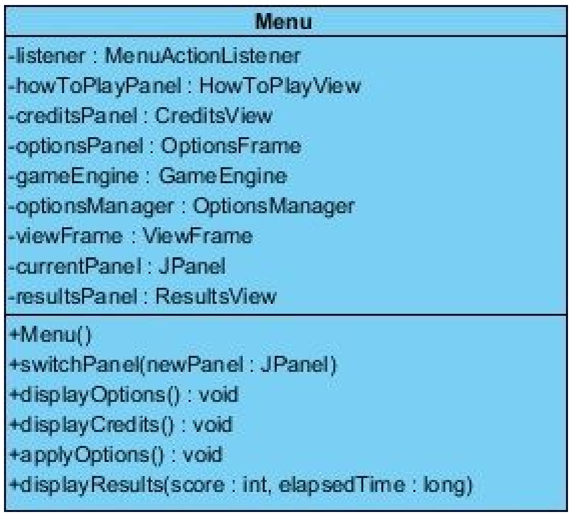


Methods:

public void actionPerformed(): This method is called after user performs any action.

Menu

This is the main class from which all menu classes inherit. Child classes override display methods according to its type. Also, it will contain the frames for all screens.


Menu
-listener : MenuActionListener
-howToPlayPanel : HowToPlayView
-creditsPanel : CreditsView
-optionsPanel : OptionsFrame
-gameEngine : GameEngine
-optionsManager : OptionsManager
-viewFrame : ViewFrame
-currentPanel : JPanel
-resultsPanel : ResultsView
+Menu()
+switchPanel(newPanel : JPanel)
+displayOptions() : void
+displayCredits() : void
+applyOptions() : void
+displayResults(score : int, elapsedTime : long)

Attributes:

private MenuActionListener listener: This listener listens for any activity on menu. In case of an action, it will notify the system accordingly.

private HowToPlayView howToPlayPanel: This is the panel for “How To Play” screen. All objects related to that screen will be added to that panel.

private CreditsView creditsPanel: This is the panel for “Credits” screen. All objects related to that screen will be added to that panel.

private GameEngine gameEngine: That object is the main controller of our game. It controls the game logic and flow of gameplay.

private OptionsFrame optionsManager: This is the panel for “Options” screen. All objects related to that screen will be added to that panel.

private ViewFrame viewFrame: All panels are put on that frame.

private JPanel currentPanel: This variable keeps track of the panel which is currently being displayed.

private ResultsView resultsPanel: This is the panel on which results are shown.

Constructor:

public Menu(): This constructor creates a new menu without any parameter.

Methods:

public void switchPanel(JPanel newPanel): This method changes the current panel by taking new panel as parameter. This method is invoked as result of user action.

public void displayOptions(): This method shows options menu on screen.

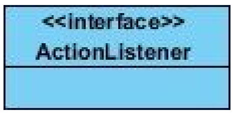
public void displayCredits(): This method shows credits section on screen.

public void applyOptions(): This method applies changes to settings, enabling user to see the difference.

public void displayResults(int score, long elapsedTime): By taking score and elapsed time as parameters, it shows results on screen.

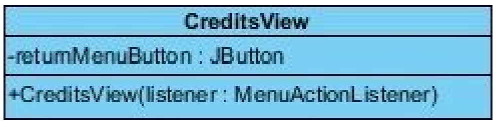
ActionListener

This is the general interface for all action listeners. Menu action listener used implements that interface. It does not have any attributes, constructors or methods.



CreditsView

This is the panel for showing credits section. It also has a button for returning to main menu. This class does not have any methods, apart from a constructor.



CreditsView
-returnMenuButton : JButton
+CreditsView(listener : MenuActionListener)

Attributes:

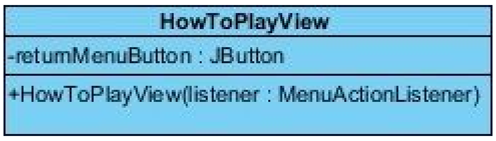
private JButton returnMenuButton: This is the button by which user can return back to main menu.

Constructor:

public CreditsView(MenuActionListener listener): This constructor creates a new credits panel by taking a listener as parameter. That listener will listen if user presses return to main menu button.

HowToPlayView

This is the panel for showing “How to Play” section. It also has a button for returning to main menu. This class does not have any methods, apart from a constructor.



HowToPlayView
-returnMenuButton : JButton
+HowToPlayView(listener : MenuActionListener)

Attributes:

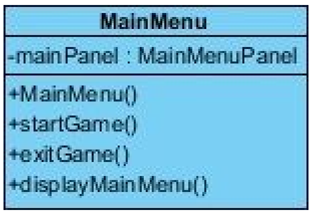
private JButton returnMenuButton: This is the button by which user can return back to main menu.

Constructor:

public HowToPlayView(MenuActionListener listener): This constructor creates a new panel for “How to Play” section by taking a listener as parameter. That listener will listen if user presses return to main menu button.

MainMenu

This is the class for main menu screen. It has controller for main menu and it also contains the panel to display contents of main menu.



Attributes:

private MainMenuPanel mainPanel: This is the panel with which contents of main menu will be displayed. It also has five buttons which user can press.

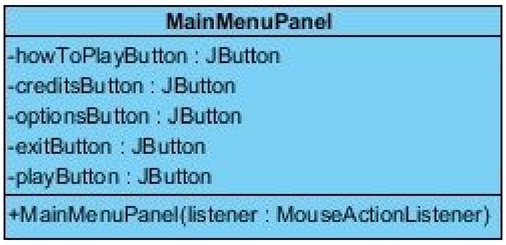
Constructor:

public MainMenu(): It is the constructor for main menu and it has no parameters.

Methods:

MainMenuPanel

This is the panel for main menu. All related components, such as buttons will be put on that panel. Apart from a constructor, it does not have any methods.



Attributes:

private JButton howToPlayButton: If user presses that button, “How To Play” section will be opened.

private JButton creditsButton: If user presses that button, “Credits” section will be opened by setting credits panel as the current panel.

private JButton optionsButton: If user presses that button, “Options” section will be opened by setting options panel as the current panel.

private JButton exitButton: If user presses that button, the program will terminate.

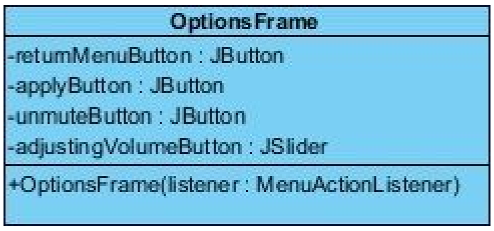
private JButton playButton: If user presses that button, a game is started.

Constructor:

public MainMenuPanel(MouseActionListener listener): This listener will keep track of mouse movements of user.

OptionsFrame

This is the frame for options menu.


OptionsFrame
-returnMenuButton : JButton
-applyButton : JButton
-unmuteButton : JButton
-adjustingVolumeButton : JSlider
+OptionsFrame(listener : MenuActionListener)

Attributes:

private JButton returnMenuButton: With this button, user can return to main menu.

private JButton applyButton: User can apply changes to settings by pressing that button.

private JButton unmuteButton: User can unmute the cound by pressing that button.

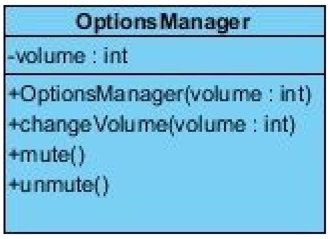
private JSlider adjustingVolumeButton: User can adjust volume by sliding it.

Constructor:

public OptionsFrame(MenuActionListener listener): This constructor creates an options frame by taking the listener as parameter.

OptionsManager

This class is the controller for options but at this point, only volume can be changed. According to user inputs, this class adjusts settings.



Attributes:

private int volume: This variable represents the sound level in application.

Constructor:

public OptionsManager(int volume): This constructor creates an options manager by taking initial sound value as parameter.

Methods:

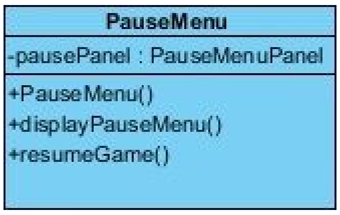
public void changeVolume(int volume): This method sets volume level to a specified integer level.

public void mute(): This method mutes the sound.

public void unmute(): This method unmutes the sound.

PauseMenu

This class contains both controller and view for pause menu. For view part, it has a panel.



Attributes:

private PauseMenuPanel pausePanel: This panel comprises the view part for pause menu.

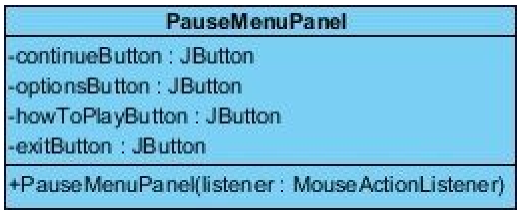
Constructor:

public PauseMenu(): This constructor creates a pause menu by taking no parameter.

Methods:

public void displayPauseMenu(): This method shows the pause menu by setting pause menu panel as current panel.

PauseMenuPanel


PauseMenuPanel
-continueButton : JButton
-optionsButton : JButton
-howToPlayButton : JButton
-exitButton : JButton
+PauseMenuPanel(listener : MouseActionListener)

Attributes:

private JButton continueButton: By pressing that button, user can continue playing the game by removing pause panel from the frame.

private JButton optionButton: By pressing that button, user can go to settings section by putting settings panel on the frame.

private JButton howToPlayButton: “How to Play” section is opened if user presses that button.

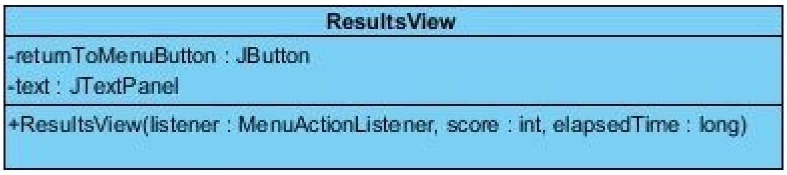
private JButton exitButton: Pressing this button will open main menu by removing pause panel and setting main menu panel on frame.

Constructor:

public PauseMenuListener(MouseActionListener listener): This constructor creates a panel for pause menu by taking a mouse listener. That will keep track of user's mouse actions.

ResultsView

This class has a text field and it shows score for current game on that. Score is shown only after game is over. It also has a button for going back to main menu.


ResultsView
-returnToMenuButton : JButton
-text : JTextPanel
+ResultsView(listener : MenuActionListener, score : int, elapsedTime : long)

Attributes:

private JButton returnToMenuButton: By pressing that button, user can return to main menu as main menu panel will be set to panel.
private JTextPanel text: This text field shows total score and elapsed time from initializing the game to endgame.

Constructor:

public ResultsView(MenuActionListener listener, long elapsedTime): This constructor creates a results view by taking a listener for the button and the time passed from the start of game as parameters.

3.2. Game Management Subsystem

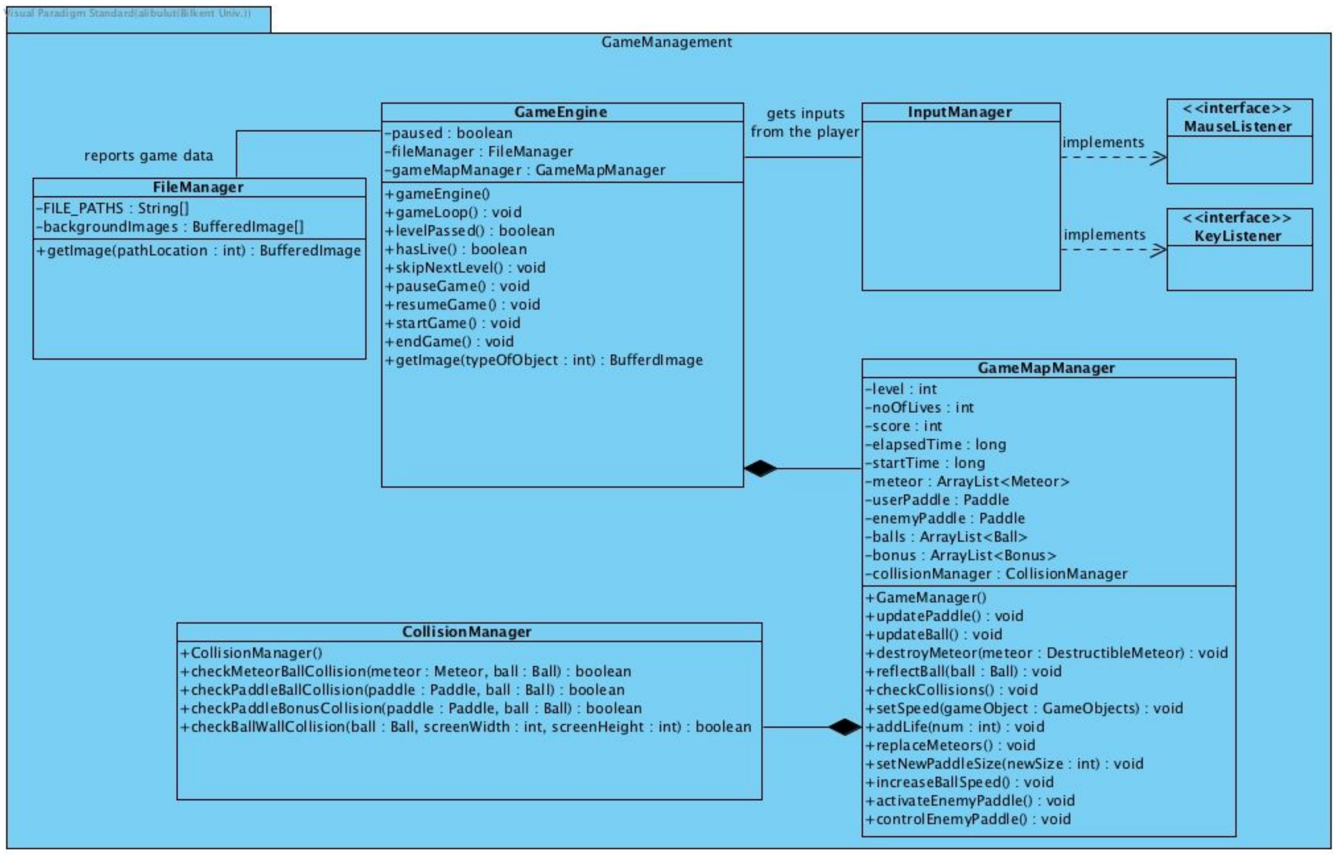
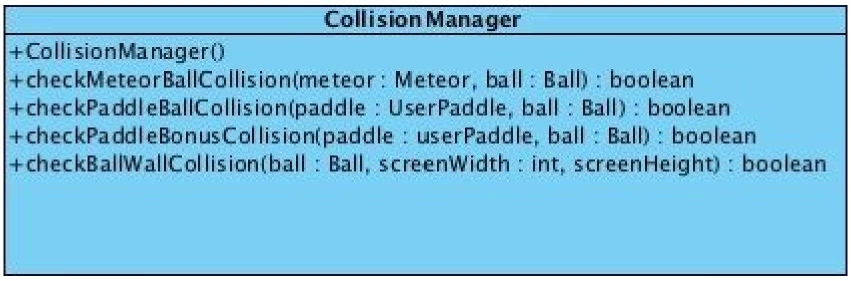

Visual Paradigm Standard(alibulut(Bilkent Univ.))
GameManagement
reports game data
GameEngine
-paused : boolean
-fileManager : FileManager
-gameMapManager : GameMapManager
+gameEngine()
+gameLoop() : void
+levelPassed() : boolean
+hasLive() : boolean
+skipNextLevel() : void
+pauseGame() : void
+resumeGame() : void
+startGame() : void
+endGame() : void
+getImage(typeOfObject : int) : BufferdImage
gets inputs from the player
InputManager
implements
<<interface>> MauseListener
implements
<<interface>> KeyListener
FileManager
-FILE_PATHS : String[]
-backgroundImages : BufferedImage[]
+getImage(pathLocation : int) : BufferedImage
GameMapManager
-level : int
-noOfLives : int
-score : int
-elapsedTime : long
-startTime : long
-meteor : ArrayList<Meteor>
-userPaddle : Paddle
-enemyPaddle : Paddle
-balls : ArrayList<Ball>
-bonus : ArrayList<Bonus>
-collisionManager : CollisionManager
+GameManager()
+updatePaddle() : void
+updateBall() : void
+destroyMeteor(meteor : DestructibleMeteor) : void
+reflectBall(ball : Ball) : void
+checkCollisions() : void
+setSpeed(gameObject : GameObjects) : void
+addLife(num : int) : void
+replaceMeteors() : void
+setNewPaddleSize(newSize : int) : void
+increaseBallSpeed() : void
+activateEnemyPaddle() : void
+controlEnemyPaddle() : void
CollisionManager
+CollisionManager()
+checkMeteorBallCollision(meteor : Meteor, ball : Ball) : boolean
+checkPaddleBallCollision(paddle : Paddle, ball : Ball) : boolean
+checkPaddleBonusCollision(paddle : Paddle, ball : Ball) : boolean
+checkBallWallCollision(ball : Ball, screenWidth : int, screenHeight : int) : boolean

Figure 3.2 : Game Management Subsystem

CollisionManager

This class will provide functions that does maths of the several type of collision between the objects.



```
CollisionManager
+CollisionManager()
+checkMeteorBallCollision(meteor : Meteor, ball : Ball) : boolean
+checkPaddleBallCollision(paddle : UserPaddle, ball : Ball) : boolean
+checkPaddleBonusCollision(paddle : userPaddle, ball : Ball) : boolean
+checkBallWallCollision(ball : Ball, screenWidth : int, screenHeight) : boolean
```

Constructor:

public CollisionManager(): It is the constructor for CollisionManager class and has no parameters.

Methods:

public boolean checkBallMeteorCollision(meteor : Meteor, ball : Ball): This method checks whether there is a collision between the meteor and the ball. If there is a collision this method returns true, otherwise returns false as a response.

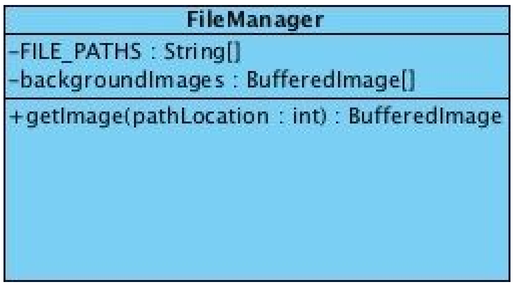
public boolean checkPaddleBallCollision(paddle : Paddle, ball : Ball): This method checks whether there is a collision between the paddle and the ball. If there is a collision this method returns true, otherwise returns false as a response.

public boolean checkPaddleBonusCollision(paddle : Paddle, ball : Ball): This method checks whether there is a collision between the paddle and the ball. If there is a collision this method returns true, otherwise returns false as a response.

public boolean checkWallBallCollision(ball: Ball, screenWidth: int, screenHeight: int): This method checks whether the ball reached to map boundaries. If the ball reaches the map boundaries this method returns true, otherwise returns false as a response.

FileManager

This class loads related files from directories. It also keeps background image of the game levels.


FileManager
-FILE_PATHS : String[]
-backgroundImages : BufferedImage[]
+getImage(pathLocation : int) : BufferedImage

Attributes:

private static final string[] FILE_PATHS: This attribute keeps file paths of the object. At each place of string array there is a file path. This array is preloaded.

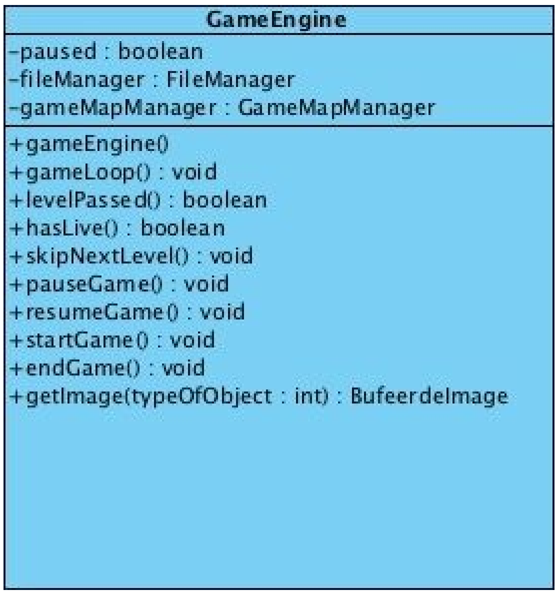
private BufferedImage[] backgroundImages: This attribute keeps related background images of level as an array of BufferedImage.

Methods:

public BufferedImage getImage(pathLocation: int): This method returns related type of objects of image. It finds them from path location.

GameEngine

This class's task is the managing status of the game. It performs actions according to the user requests.



Attributes:

private boolean paused: This attribute keeps status of the game whether the game has paused or not.

private FileManager fileManager: This attribute is defined for using FileManager class's functions.

private GameMapManager gameMapManager: This attribute is for creating GameMapManager object.

Constructor:

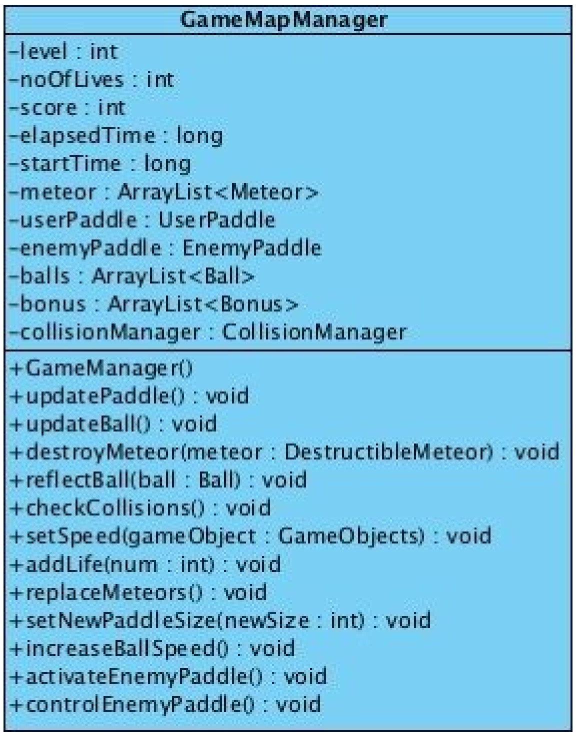
public GameEngine(): This constructor initialize the attributes of the GameEngine class for the first run.

Methods:

public void gameLoop(): This method runs a loop for the continuity of the game and calls needed functions for next actions.
public boolean levelPassed(): This method checks the current level condition and decides whether the level has finished or not.
public boolean hasLive(): This method checks if the user has still life.
public void skipNextLevel(): This method changes current level to next level and calls needed methods for next level.
public void pauseGame(): This method pauses the game and stops the loop till the next user instruction.
public void resumeGame(): This method resumes the game.
public void startGame(): This method runs only at the start of the game.
public void endGame(): This method finishes the game and quit from the game.
public BufferedImage getImage(typeOfObject: int): This method returns images that is requested by the object.

GameMapManager

GameMapManager class keeps objects that will be used in the game. It provides functions for game object modification and performs needed actions. This class also keeps progress informations of the game.


GameMapManager
-level : int
-noOfLives : int
-score : int
-elapsedTime : long
-startTime : long
-meteor : ArrayList<Meteor>
-userPaddle : UserPaddle
-enemyPaddle : EnemyPaddle
-balls : ArrayList<Ball>
-bonus : ArrayList<Bonus>
-collisionManager : CollisionManager
+GameManager()
+updatePaddle() : void
+updateBall() : void
+destroyMeteor(meteor : DestructibleMeteor) : void
+reflectBall(ball : Ball) : void
+checkCollisions() : void
+setSpeed(gameObject : GameObjects) : void
+addLife(num : int) : void
+replaceMeteors() : void
+setNewPaddleSize(newSize : int) : void
+increaseBallSpeed() : void
+activateEnemyPaddle() : void
+controlEnemyPaddle() : void

Attributes:

private int level: This attribute keeps current level of the game.
private int noOfLives: This attribute keeps remaining number of lives.
private int score: This attribute is for keeping the user score.
private long elapsedTime: This attribute keeps total elapsed time while user playing the game. When game paused elapsed time does not continue to count.
private long startTime: This attribute holds starting time of the level. It is updated when a level starts or user returns to the game from the pause condition.
private ArrayList<Meteor>: This method keeps the meteors that are currently on the map. It does not keep destroyed meteors.
private UserPaddle Paddle: This is an object for user. User controls this paddle.
private EnemyPaddle Paddle: This is an object for enemy paddle. This paddle is controlled by the programme algorithm.
private ArrayList<Ball> balls: This arraylist keeps ball objects on the map.
private Arraylist<Bonus> bonus: This arrylist attribute keeps bonuses on the map.
private CollisionManager collisionManager: This collisionManager object will provide related collision functions.

Constructor:

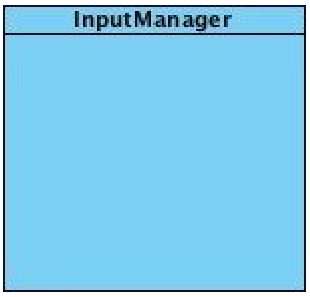
public GameMapManager(): This constructor initialize needed objects for the begging of the map.

Methods:

public void updatePaddle(): This method updates speed, length and position of a paddle.
public void updateBall(): This method updates speed and position of a ball.
public void destroyMeteor(meteor: DestructibleMeteor): This method destroys a meteor and removes it from the map.
public void reflectBall(ball: Ball): This method does new speed direction calculation of a ball when ball hits an obstacle.
public void checkCollisions(): This method test all possible kind of collisions.
public void setSpeed(gameObject: GameObjects): This method set speed of an given object.
public void addLife(num: int): This method increase or decrease remaining life respect to taken bonus.
public void replaceMeteors(): This method replaces all destroyed meteors. It sets meteors to initial state of the level.
public void setNewPaddleSize(newSize: int): This method changes size of a paddle/
public void increaseBallSpeed(): This method increase speed of a ball.
public void activateEnemyPaddle(): When a related bonus collected, this method activates enemy paddle.
public void controlEnemyPaddle(): This method gives command to enemy paddle to give direction to a ball.

InputManager

This class receives user inputs from keyboard and mouse, and passes them to game engine. It will then act according to these inputs. This class implements KeyListener and MouseListener interfaces.



KeyListener

It is a general interface to receive inputs from keyboard and it can be implemented by different classes with separate implementations. In this case, input manager class will provide these implementations.

MouseListener

It is a general interface to receive inputs from mouse and it can be implemented by different classes with separate implementations. In this case, input manager class will provide these implementations.

3.3. Game Entities Subsystem Interface

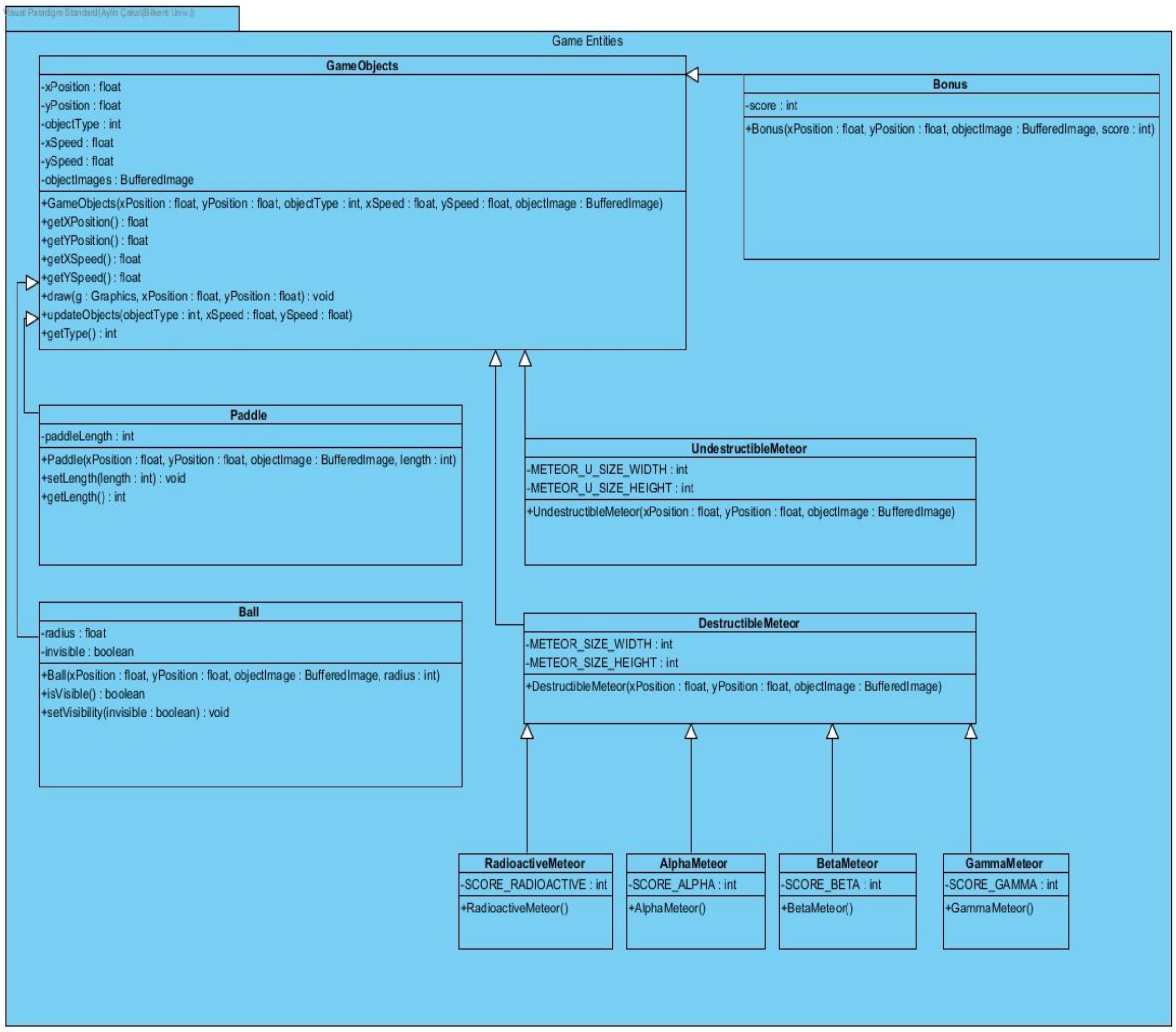
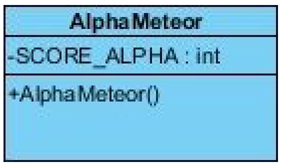

Visual Paradigm Standard(Aylin Çakal(Bilkent Univ.))
Game Entities
GameObjects
-xPosition : float
-yPosition : float
-objectType : int
-xSpeed : float
-ySpeed : float
-objectImages : BufferedImage
+GameObjects(xPosition : float, yPosition : float, objectType : int, xSpeed : float, ySpeed : float, objectImage : BufferedImage)
+getXPosition() : float
+getYPosition() : float
+getXSpeed() : float
+getYSpeed() : float
+draw(g : Graphics, xPosition : float, yPosition : float) : void
+updateObjects(objectType : int, xSpeed : float, ySpeed : float)
+getType() : int
Bonus
-score : int
+Bonus(xPosition : float, yPosition : float, objectImage : BufferedImage, score : int)
Paddle
-paddleLength : int
+Paddle(xPosition : float, yPosition : float, objectImage : BufferedImage, length : int)
+setLength(length : int) : void
+getLength() : int
UndestructibleMeteor
-METEOR_U_SIZE_WIDTH : int
-METEOR_U_SIZE_HEIGHT : int
+UndestructibleMeteor(xPosition : float, yPosition : float, objectImage : BufferedImage)
Ball
-radius : float
-invisible : boolean
+Ball(xPosition : float, yPosition : float, objectImage : BufferedImage, radius : int)
+isVisible() : boolean
+setVisibility(invisible : boolean) : void
DestructibleMeteor
-METEOR_SIZE_WIDTH : int
-METEOR_SIZE_HEIGHT : int
+DestructibleMeteor(xPosition : float, yPosition : float, objectImage : BufferedImage)
RadioactiveMeteor
-SCORE_RADIOACTIVE : int
+RadioactiveMeteor()
AlphaMeteor
-SCORE_ALPHA : int
+AlphaMeteor()
BetaMeteor
-SCORE_BETA : int
+BetaMeteor()
GammaMeteor
-SCORE_GAMMA : int
+GammaMeteor()

Figure 3.3: Game Entities Subsystem

AlphaMeteor

This class inherits from destructible meteors. This type of meteors can be destroyed after single hit. After it is destroyed, user will gain a specified amount of score.



Attributes:

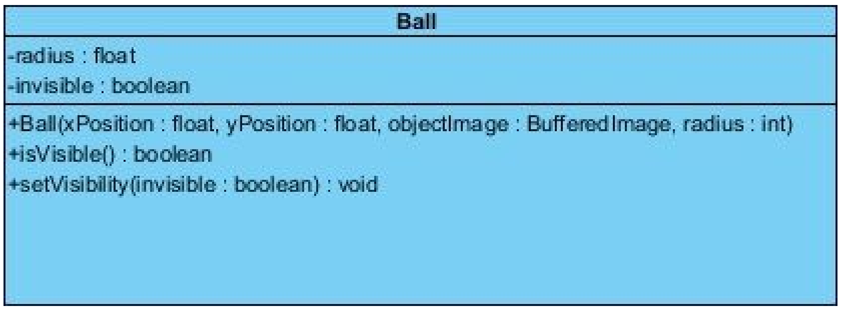
private static final int SCORE_ALPHA: Constant integer holds value for the amount of score player gets when an alpha type brick is destroyed.

Constructor:

public AlphaMeteor(): Empty constructor class initialises Alpha meteors without taking any parameters.

Ball

It is an object with which user can hit and destroy meteors. In order not to fail the game, user should not let it fall through the void.


Ball
-radius : float
-invisible : boolean
+Ball(xPosition : float, yPosition : float, objectImage : BufferedImage, radius : int)
+isVisible() : boolean
+setVisibility(invisible : boolean) : void

Attributes:

private float radius: It is a size parameter, representing ball’s radius.
private boolean invisible: It represents whether ball is visible or not.

Constructor:

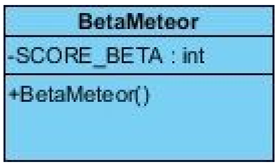
public Ball(xPosition: float, yPosition: float, objectImage: BufferedImage, radius: int): This constructor creates a new ball and takes ball’s initial location, size and its image as parameters. Initially, a ball is set visible.

Methods:

public boolean isVisible(): It returns whether the ball is visible or not.
public void setVisibility(invisible: boolean): This method is used to change visibility status of ball.

BetaMeteor

This class inherits from destructible meteors. This type of meteors can be destroyed after two hits. After it is destroyed, user will gain a specified amount of score.



Attributes:

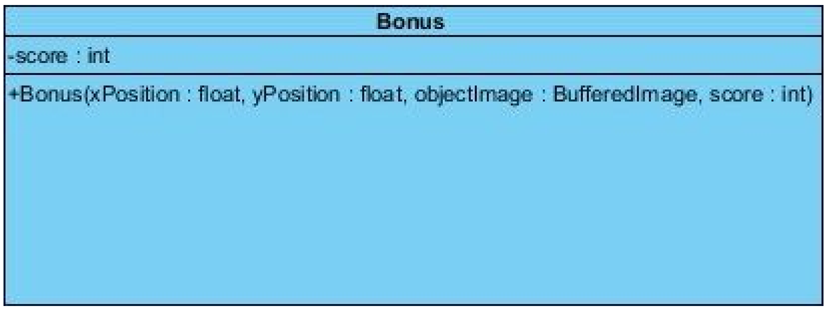
private static final int SCORE_BETA: Constant integer holds value for the amount of score player gets when a beta type brick is destroyed.

Constructor:

public BetaMeteor(): Empty constructor initialises Beta meteors without taking any parameters.

Bonus

This class inherits from game objects. This type of objects keep some amount of score and if it gets hit, user will receive that score. After being hit, it will be destroyed.


Bonus
-score : int
+Bonus(xPosition : float, yPosition : float, objectImage : BufferedImage, score : int)

Attributes:

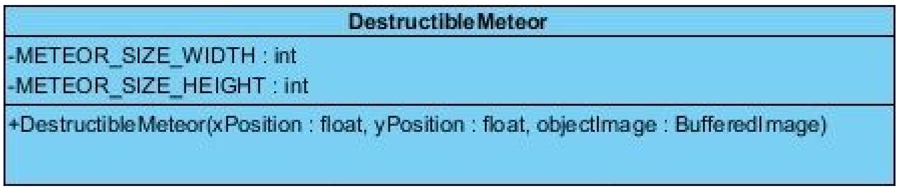
private int score: This variable stores the amount of “reward” will user receive after hitting the bonus object.

Constructor:

public Bonus(xPosition: float, yPosition: float, objectImage: BufferedImage, score: int): This constructor creates a bonus instance with given initial location, image and score it will contain. These values are received as parameters to constructor.

DestructibleMeteor

This is the parent class for all meteor types which are destructible. Specific types of meteors inherits from this type.



DestructibleMeteor

-METEOR_SIZE_WIDTH : int
-METEOR_SIZE_HEIGHT : int

+DestructibleMeteor(xPosition : float, yPosition : float, objectImage : BufferedImage)

Attributes:

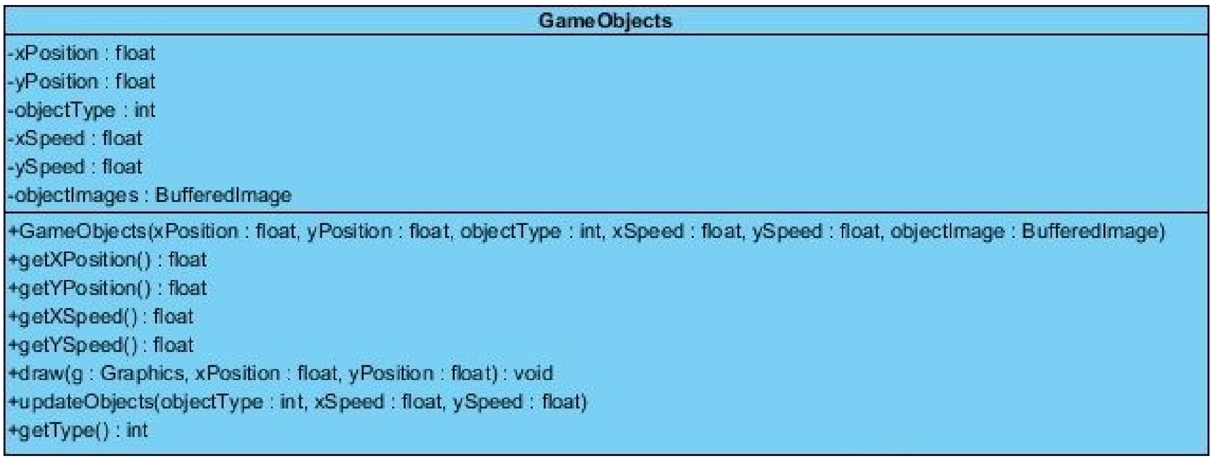
private static final int METEOR_SIZE_WIDTH: Constant width of the destructible meteor objects in integer format.
private static final int METEOR_SIZE_HEIGHT: Constant height of the destructible meteor objects in integer format.

Constructor:

public DestructableMeteor(xPosition: float, yPosition: float, objectImage: BufferedImage): Constructor initialises destructible meteors by taking their positions in float variables and image as parameters.

GameObjects

This class provides a template for all types of objects in a game. This type keeps parameters common to all objects, namely location, object type and image. It also has two-dimensional speed but it will not be used by all classes.



GameObjects

-xPosition : float
-yPosition : float
-objectType : int
-xSpeed : float
-ySpeed : float
-objectImages : BufferedImage

+GameObjects(xPosition : float, yPosition : float, objectType : int, xSpeed : float, ySpeed : float, objectImage : BufferedImage)
+getXPosition() : float
+getYPosition() : float
+getXSpeed() : float
+getYSpeed() : float
+draw(g : Graphics, xPosition : float, yPosition : float) : void
+updateObjects(objectType : int, xSpeed : float, ySpeed : float)
+getType() : int

Attributes:

private float xPosition: Float variable for the horizontal position of the game objects.
private float yPosition: Float variable for the vertical position of the game objects.
private int objectType: Integer variable specifying the type of the game object.
private float xSpeed: Float variable of the horizontal speed of a game object in screen.
private float ySpeed: Float variable of the vertical speed of a game object in screen.
private BufferedImage objectImage: The image of the object to represent in the screen.

Constructor:

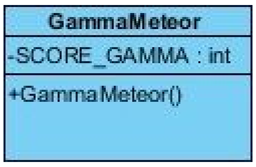
public GameObjects(xPosition: float, yPosition: float, objectType: int, xSpeed: float, ySpeed: float, objectImage: BufferedImage): Constructor initialises each game object by taking their positions and speeds in float format, object type defined in integer, and image as parameters to extract from FileManager.

Methods:

public float getXPosition(): Public method to get the horizontal position of a game object in float.
public float getYPosition(): Public method to get the horizontal position of a game object in float.
public float getXSpeed(): This method returns horizontal speed of a game object in float format.
public float getYSpeed(): This method returns vertical speed of a game object in float format.
public void draw(g: Graphics, xPosition: float, yPosition: float): This method draws the instance on a given graphics context at a given location.
public void updateObjects(objectType: int, xSpeed: float, ySpeed: float): This method updates object type and speed of a game object.
public int getType(): This method returns type of a game object in integer format.

GammaMeteor

This class inherits from destructible meteors. This type of meteors can be destroyed after three hits. After it is destroyed, user will gain a specified amount of score.



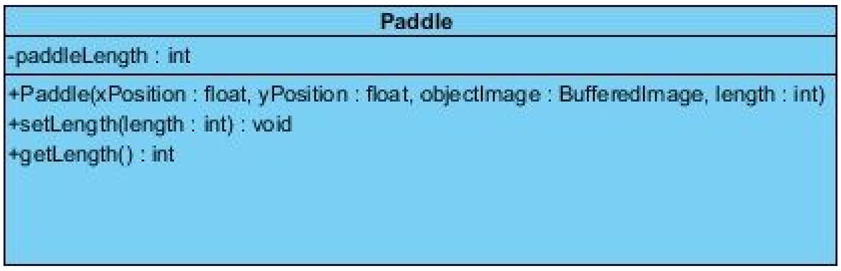
Attributes:

private static final int SCORE_GAMMA: Constant integer holds value for the amount of score player gets when a gamma type brick is destroyed.

Constructor:

public GammaMeteor(): This constructor creates an instance of gamma meteors with no parameters in it.

Paddle


Paddle
-paddleLength : int
+Paddle(xPosition : float, yPosition : float, objectImage : BufferedImage, length : int)
+setLength(length : int) : void
+getLength() : int

Attributes:

private int paddleLength: This is the length of a paddle instance.

Constructor:

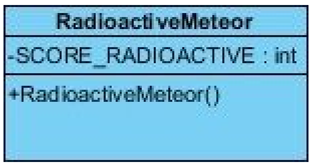
public Paddle(xPosition: float, yPosition: float, objectImage: BufferedImage, length: int): This constructor returns a paddle instance by taking its initial location, image and length as parameters.

Methods:

public void setLength(length: int): This method sets length of the paddle to a specified integer value.
public int getLength(): This method returns length of the paddle instance.

RadioactiveMeteor

These type of meteors inherit from destructible meteors. They will destroy surrounding meteors if one of its instances gets hit.



Attributes:

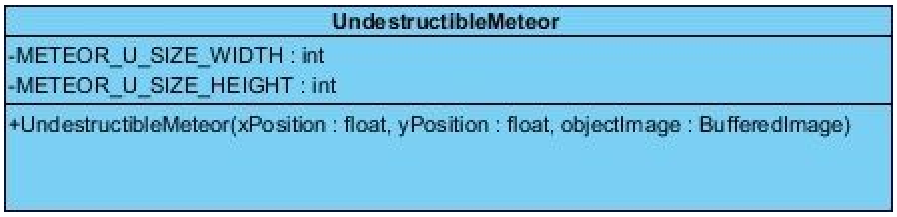
private static final int SCORE_RADIOACTIVE: Constant integer holds value for the amount of score player gets when a radioactive brick is destroyed.

Constructor:

public RadioactiveMeteor(): This constructor does not have any parameters. It creates an instance of radioactive meteors.

UndestructibleMeteor

This class inherits from game objects. This type of meteors cannot be destroyed even if they get hit.


UndestructibleMeteor
-METEOR_U_SIZE_WIDTH : int
-METEOR_U_SIZE_HEIGHT : int
+UndestructibleMeteor(xPosition : float, yPosition : float, objectImage : BufferedImage)

Attributes:

private static final int METEOR_U_SIZE_WIDTH: Constant integer value of the undestructible meteors width.
private static final int METEOR_U_SIZE_HEIGHT: Constant integer value of the undestructible meteor height.

Constructor:

public UndestructableMeteor(xPosition: float, yPosition: float, objectImage: BufferedImage): This constructor creates an undestructible meteor with given initial location and image representing it.

4. Low-Level Design

4.1. Final Object Design

The diagram below demonstrates our project’s final object design.

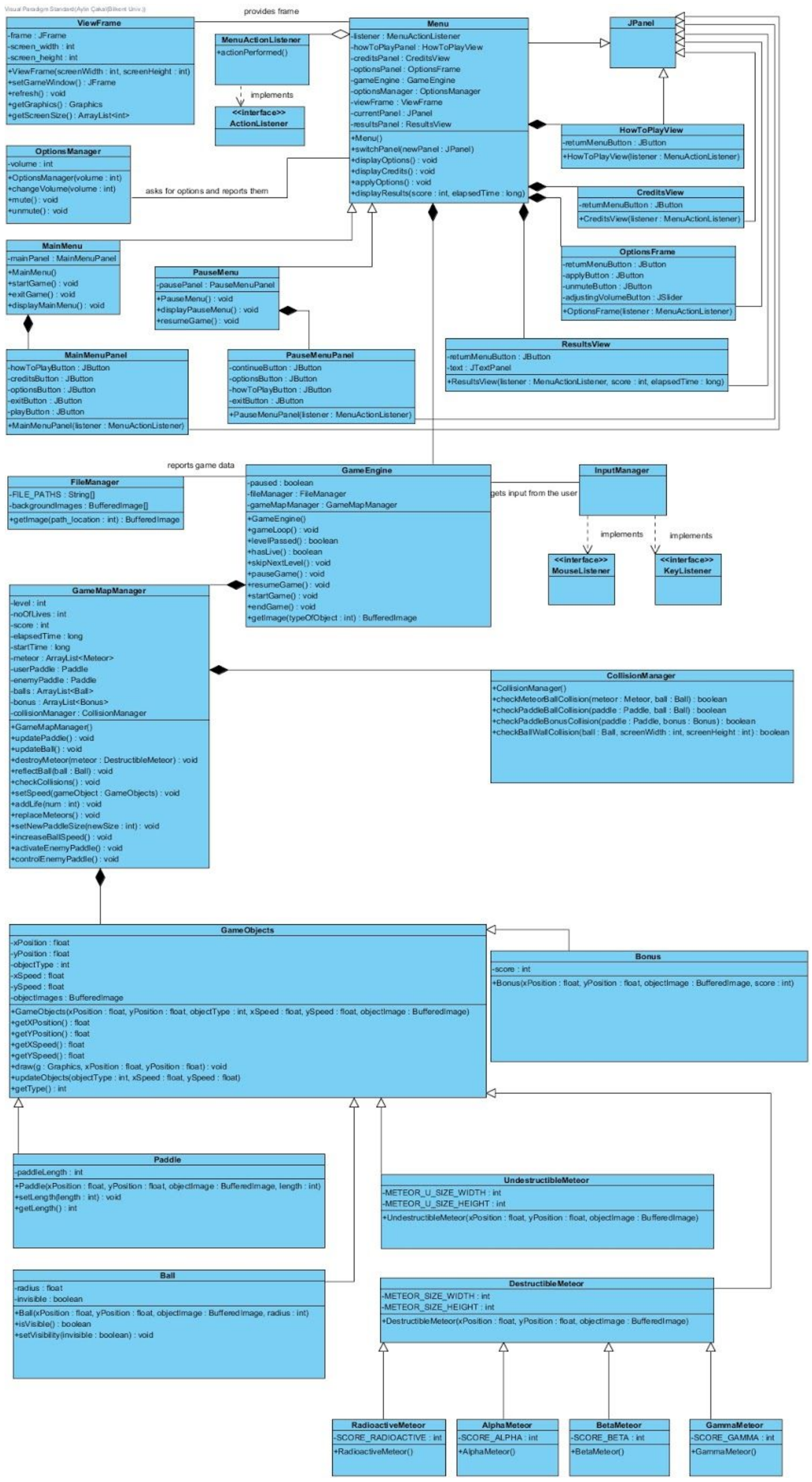

Visual Paradigm Standard(Aylin Çakal(Bilkent Univ.))
provides frame
ViewFrame
-frame : JFrame
-screen_width : int
-screen_height : int
+ViewFrame(screenWidth : int, screenHeight : int)
+setGameWindow() : JFrame
+refresh() : void
+getGraphics() : Graphics
+getScreenSize() : ArrayList<int>
MenuActionListener
+actionPerformed()
implements
<<interface>>
ActionListener
Menu
-listener : MenuActionListener
-howToPlayPanel : HowToPlayView
-creditsPanel : CreditsView
-optionsPanel : OptionsFrame
-gameEngine : GameEngine
-optionsManager : OptionsManager
-viewFrame : ViewFrame
-currentPanel : JPanel
-resultsPanel : ResultsView
+Menu()
+switchPanel(newPanel : JPanel)
+displayOptions() : void
+displayCredits() : void
+applyOptions() : void
+displayResults(score : int, elapsedTime : long)
JPanel
HowToPlayView
-returnMenuButton : JButton
+HowToPlayView(listener : MenuActionListener)
CreditsView
-returnMenuButton : JButton
+CreditsView(listener : MenuActionListener)
OptionsManager
-volume : int
+OptionsManager(volume : int)
+changeVolume(volume : int)
+mute() : void
+unmute() : void
asks for options and reports them
MainMenu
-mainPanel : MainMenuPanel
+MainMenu()
+startGame() : void
+exitGame() : void
+displayMainMenu() : void
PauseMenu
-pausePanel : PauseMenuPanel
+PauseMenu() : void
+displayPauseMenu() : void
+resumeGame() : void
OptionsFrame
-returnMenuButton : JButton
-applyButton : JButton
-unmuteButton : JButton
-adjustingVolumeButton : JSlider
+OptionsFrame(listener : MenuActionListener)
MainMenuPanel
-howToPlayButton : JButton
-creditsButton : JButton
-optionsButton : JButton
-exitButton : JButton
-playButton : JButton
+MainMenuPanel(listener : MenuActionListener)
PauseMenuPanel
-continueButton : JButton
-optionsButton : JButton
-howToPlayButton : JButton
-exitButton : JButton
+PauseMenuPanel(listener : MenuActionListener)
ResultsView
-returnMenuButton : JButton
-text : JTextPanel
+ResultsView(listener : MenuActionListener, score : int, elapsedTime : long)
reports game data
FileManager
-FILE_PATHS : String[]
-backgroundImages : BufferedImage[]
+getImage(path_location : int) : BufferedImage
GameEngine
-paused : boolean
-fileManager : FileManager
-gameMapManager : GameMapManager
+GameEngine()
+gameLoop() : void
+levelPassed() : boolean
+hasLive() : boolean
+skipNextLevel() : void
+pauseGame() : void
+resumeGame() : void
+startGame() : void
+endGame() : void
+getImage(typeOfObject : int) : BufferedImage
gets input from the user
InputManager
implements
implements
<<interface>>
MouseListener
<<interface>>
KeyListener
GameMapManager
-level : int
-noOfLives : int
-score : int
-elapsedTime : long
-startTime : long
-meteor : ArrayList<Meteor>
-userPaddle : Paddle
-enemyPaddle : Paddle
-balls : ArrayList<Ball>
-bonus : ArrayList<Bonus>
-collisionManager : CollisionManager
+GameMapManager()
+updatePaddle() : void
+updateBall() : void
+destroyMeteor(meteor : DestructibleMeteor) : void
+reflectBall(ball : Ball) : void
+checkCollisions() : void
+setSpeed(gameObject : GameObjects) : void
+addLife(num : int) : void
+replaceMeteors() : void
+setNewPaddleSize(newSize : int) : void
+increaseBallSpeed() : void
+activateEnemyPaddle() : void
+controlEnemyPaddle() : void
CollisionManager
+CollisionManager()
+checkMeteorBallCollision(meteor : Meteor, ball : Ball) : boolean
+checkPaddleBallCollision(paddle : Paddle, ball : Ball) : boolean
+checkPaddleBonusCollision(paddle : Paddle, bonus : Bonus) : boolean
+checkBallWallCollision(ball : Ball, screenWidth : int, screenHeight : int) : boolean
GameObjects
-xPosition : float
-yPosition : float
-objectType : int
-xSpeed : float
-ySpeed : float
-objectImages : BufferedImage
+GameObjects(xPosition : float, yPosition : float, objectType : int, xSpeed : float, ySpeed : float, objectImage : BufferedImage)
+getXPosition() : float
+getYPosition() : float
+getXSpeed() : float
+getYSpeed() : float
+draw(g : Graphics, xPosition : float, yPosition : float) : void
+updateObjects(objectType : int, xSpeed : float, ySpeed : float)
+getType() : int
Bonus
-score : int
+Bonus(xPosition : float, yPosition : float, objectImage : BufferedImage, score : int)
Paddle
-paddleLength : int
+Paddle(xPosition : float, yPosition : float, objectImage : BufferedImage, length : int)
+setLength(length : int) : void
+getLength() : int
UndestructibleMeteor
-METEOR_U_SIZE_WIDTH : int
-METEOR_U_SIZE_HEIGHT : int
+UndestructibleMeteor(xPosition : float, yPosition : float, objectImage : BufferedImage)
Ball
-radius : float
-invisible : boolean
+Ball(xPosition : float, yPosition : float, objectImage : BufferedImage, radius : int)
+isVisible() : boolean
+setVisibility(invisible : boolean) : void
DestructibleMeteor
-METEOR_SIZE_WIDTH : int
-METEOR_SIZE_HEIGHT : int
+DestructibleMeteor(xPosition : float, yPosition : float, objectImage : BufferedImage)
RadioactiveMeteor
-SCORE_RADIOACTIVE : int
+RadioactiveMeteor()
AlphaMeteor
-SCORE_ALPHA : int
+AlphaMeteor()
BetaMeteor
-SCORE_BETA : int
+BetaMeteor()
GammaMeteor
-SCORE_GAMMA : int
+GammaMeteor()

Figure 4.1 : Final Object Design

4.2. Packages

4.2.1 Java.util:

We used the Java util library in order to benefit from ArrayLists in project.

4.2.2 javafx.scene.layout:

Scene Layout package is used since it provides classes to support user interface layouts.

4.2.3 java.awt.Graphics:

Awt Graphics application to draw onto the component or on image.

4.2.4 java.lang.System:

We used java.lang.System library for measuring time. The calculation will be based on the system time.

4.3. Class Interfaces

4.3.1 ActionListener:

This interface will be invoked whenever an action occurs with receiving the action events. It is directly related with the Menu Class in order to track the action events throughout the game.

4.3.2 KeyListener:

In the levels of the game, the player uses the keyboard to control the paddle and ball. To implement this feature, the system needs to receive keyboard event by this interface, KeyListener. This interface will be dependent on InputManager Class which inputs to GameEngine Class.

4.3.3 MouseListener:

Throughout the game, the player uses the mouse commands to click buttons. To implement this feature, the system needs to receive mouse event by this interface, MouseListener. This interface will be dependent on InputManager Class which inputs to GameEngine Class.

Glossary & References

https://docs.oracle.com/javase/8/docs/api/