CS319

Object Oriented Software Engineering

System Design Report

Group 2

Furkan Hüseyin
Berfu Anıl
Islomiddin Soqidov

1. Introduction

1.1 Purpose of the system

1.1.1 General information

The game takes place in an area in the space near a station which is built by the company the character is working for. The main purpose of the game is to kill all the enemies on the map. The game graphics will not look very complex. It is more focused on being catchy rather than being complex. The game mechanics are aimed to be main fun part of the game. In the game the maneuverability of the enemy is better than the player. Hence player will strugle to avoid missiles from enemies. This will force player to develop his or her skills to beat the game in a fun way.

1.1.2 How the game works?

According to Bartle's Taxonomy, the game aims to fulfill the expectations of achievers, killers and explorers. In the game, there will be a spaceship which is brought to a corrupted area to complete a task. The game ends when the player kills all the enemies on the map. If player kills an enemy another one wil be generated after 2 minutes which can change later on at implmenetation. Hence the player has 2 minutes to kill all the enemies. However player does not have that much power at the beginning of the game. Player needs to kill a lot of enemies or achieve quests to level up and improve the ship.

In the area where the player plays, there is a station which gives the player its missions and explaining the story behind the game. Story is as follows. The unknown spaceship model appears in the space and everyone thinks that this is for a secret mission. These spaceships start to gather at an area in the space and starts to attack other ships passing near that area. A company builds a station near that area to kill these ships and collect all the money while discovering who is producing these spaceships. This company produces the same spaceship to be seen as an ally in the area to take down other spaceships. The player plays that spaceship produced by the company.

If the player dies in the game player will start from station with half of its life points. Near the station, player will be healing automatically. Player will be seen as an ally to enemies until player attacks one of them. If player achieves to be far enough to an attacking enemy, the attacking enemy will stop following the player. Quests will be easy and less. Quests will be for introducing the map and the game mechanics. Quests will be there as a tutorial and narrator. After the player gets the idea which is killing all the enemies is the crux of the game, player will feel the progress and being able to kill more enemies in the given time before another enemy is generated. The number of remaining enemies will be shown on the screen. Player will be untouchable near station by enemies.

1.2 Design Goals

Our design goal includes using necessary patterns to avoid unnecessary work, a frame-based and viewport approach to better display each game component, and create a fun game by satisfying following goals.

1.2.1 End User Criteria

1.2.1.1 Ease of Use

Although the player will struggle the mechanics of the game and will be forced to improve his or her skills to beat the game, the player will learn the game easily. The only struggle is to develop skills not understanding the game. This design goal is satisfied with ease of the game. The game will provide a tutorial for the keyboard and mouse usage. The player will not struggle to get in which condition he or she can beat the game.

1.2.1.2 Ease of Learning

As stated above, the game will be easy to learn with a good tutorial and quests. The only thingh left to user is to discover the story and finish the game. Other parts will be easy to get.

1.2.2 Maintenance Criteria

1.2.2.1 Extendibility

Since the Engine package is designed to create a game not just single space game, this class can be used to create other games or other features or levels can be easily added to the project.

1.2.2.2 Portability

Although Java Virtual Machine creates the platform independency, the logic behind the Engine package, frame-based approach of the screening and viewport approach to the game with an empty pawn feature can be used to implement the engine and whole game as well in other platforms.

1.2.2.3 Modifiability

While cohesion in a package is aimed to be increased, coupling between classes are aimed to be decreased. Even between classes where most of the coupling occurs, modifying them is easy because the logic of the engine is stable.

1.2.3 Performance Criteria

Since the system will have the frame-based approach and the game is not complex, objects on the screen will not have a performance problem with a normal computer. All entities and models will be optimized and refactored not to exploit this frame-based approach.

1.2.4 Trade Offs

The system can have event driven programming all over its entities. However since it uses frame-based game engine, it cannot have that property easily. For example, the player has a life bar and a life attribute. Life bar can be called only when the attribute changes however the frame-based approach might update the lifebar each frame. These little problems can be optimized. Without this approach, we would have to use thread approach which might not be suitable for 100 objects on the screen.

1.3 Definitions, acronyms, and abbreviations

MVC: Model View Controller
CP: Composite Pattern
FP: Façade Pattern
JVM: Java Virtual Machine
Frame-based approach: It is an approach where all objects shown on the screen are updated at each frame with a single timer instead of different threads or timers. This creates a syncronization.

1.4. References

[1] https://en.wikipedia.org/wiki/Composite_pattern

[2] Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.

[3] https://www.youtube.com/watch?v=JQJA5YjvHDU

[4] https://www.youtube.com/watch?v=YQ44hVeVdEw

1.5 Overview

The system design contains three different design patterns that are MVC, Façade Pattern and Composite pattern. The system also contains frame-based approach to handle all the objects easily. The aim of the project is to create a game to entertain players with a 2D game. The mechanics will be the main entertainer part of the game.

2. Software Architecture

2.1 Overview

Since the system we are up to implement is a game, view part of the game is an important subject. Hence separating the view from the model is quite important for decomposing the system in a way that one person can

comprehend the complexity of the subsystem. Hence Model View Controller (MVC) pattern is a good choice for the general structure of the project.

In the view part of the game which includes both sound and image viewing, Composite Pattern (CP) is going to be used for the collection of views.

The view and model parts will be controlled via controller object. This control is easy since there are only few keys the player can stroke. Hence controller does not need to increase the coupling between two packages. The view and model will be controlled through one Game Manager class which means it will have the Façade Pattern (FP).

2.2 Subsystem Decomposition

As it is stated above, the whole system will be decomposed into model, view and controller using MVC model. Since view part is too much for just one person to comprehend and handle, view part is also decomposed into other subsystems which are called engine, sound and user interface.

In Figure 1, the overall of all packages are shown. As seen view package is decomposed into three other packages. Engine part handles the view of objects on the game screen. Sound part handles sound mechanics of the game. User interface both shows the game screen to the user and other menu items like play game, help, credits and extra.

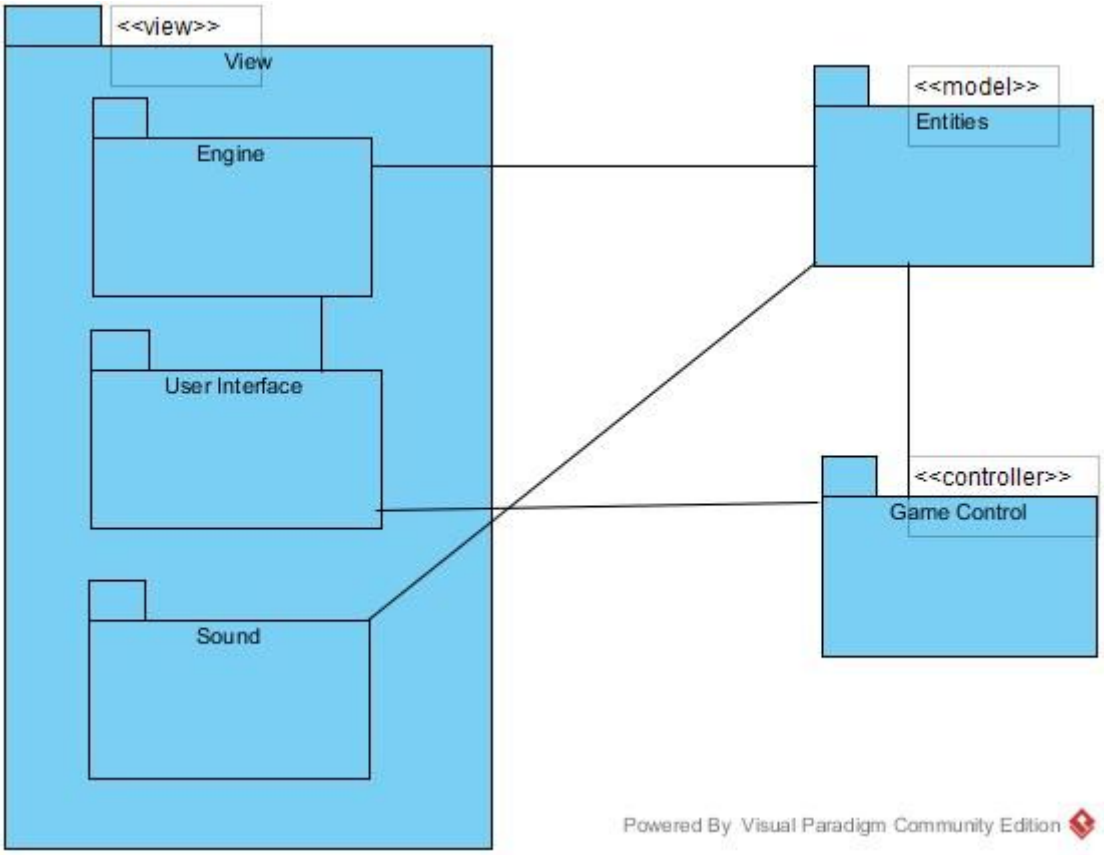

Figure 1

To reduce coupling between classes, Game Control is connected to model via one Façade Class. Model class is extending classes from Engine package to create its Actors and ActorCollections for the game and implement the mechanics and logic upon them. This is where most of the coupling between two packages is happening.

2.3 Architectural Structure

2.3.1 Layers

There are three layers in the system. The first layer is the Actor level. Actor level includes Actor and ActorCollection which are Drawable GameObjects. The second layer consist of entities which are extending from Actor and ActorCollection to implement general behaviour of Actors. Third layer has GameScreen, User Interface and Game Control to interact with user.

2.3.2 Model View Controller

As stated above, the system is going to be mainly structured using the MVC pattern. The model will not have a whole different class implementation but model classes will extend the view classes to implement the model upon view. Instead of nofiying the view from model classes, view classes will have a frame based structure in which controller class will call each view object's tick() method in each frame. View classes will have the data from model classes and update themselves at each frame.

2.4 Hardware/Software Mapping

The game does not require too much quality in hardware since it is a simple game. The system will be rendering only what should be seen on the screen. Hence this will optimize the rendering. The game also destroys unnecessary objects depending on the garbage collector of Java. The game requires keyboard and a mouse to play the game.

Game uses images to render the graphics. Hence this creates some delays at the first opening. However after the map is loaded or rendered, there is no delay on other objects.

2.5 Persistent Data Management

Since the game does not have too complex data to handle, the project will not include any database. The level of the player and enemies and quests are going to be stored in the memory directly. The images are going to be stored in the Images file and sounds are going to be stored in Sounds file.

2.6 Access Control and Security

This part is not applicable since the game does not reuire any log in and does not require any network connection. Security problem does not exist since the game does not apply any licensing and network connection.

2.7 Boundary Conditions

2.7.1 Initialization

The game does not have any installation. Hence the program will be executed via a .jar file.

2.7.2 Termination

The game can be closed with the “x” button on the game window. However the menu will stay if the game is closed. The program will be closed if the “x” button or “Quit” button on the menu frame is clicked.

2.7.3 Error

In an error situation, images and sounds may not be loaded. The mechanics of the game might not be working well. If the performance issues occurs, the game progress will be lost.

3. Subsystem Services

3.1 Design Patterns

3.1.1 Façade Design Pattern

Since the game does not have too complex controls which means that player controls whole game using only up, left, right, space, escape and mouse buttons, the control class will control everything on the model via one Façade Class which is GameManager.

This design pattern helps us to decrease coupling between model and control and also gives us the ease of model control. This design pattern obviously descreases coupling. On the other hand, this design pattern helps to make model control easy in the following way. Since the game composed of levels (entrance screen is also regarded as a level), each level reacts differently to keyboard strokes. For example, in the entrance screen while mouse strokes does not do anything, all the keyboard strokes mean the same thing which means

that skip the entrance and go to the game. On the other hand, at the game level, the space and enter strokes mean different things. Hence Façade Pattern helps to handle this differentiation between levels.

3.1.2 Composite Design Pattern

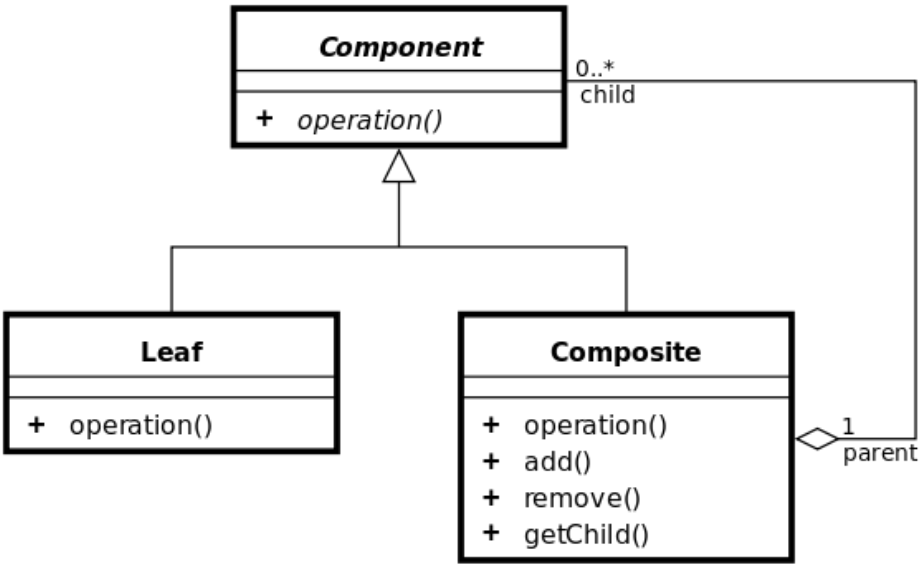

Figure 2

In the Composite Pattern, there is a Component and a Leaf which is also a Component and there is a Composite which has many components and which is also a component itself.

In the game, the design started as a generalization in the following way. Everything displayed on the screen is an Actor. Bullets, spaceship, background, information windows… However player and enemy have more than one Actor. A player or an enemy have a life bar, a spaceship and a name. This Actor and Leaf pattern will generate the following problem. If there are only actors on the screen, the control would be very hard because each time the object is moved to another point in the screen other objects belongs to the same model have to be moved respectively. If there was a Composite which is composed of many Components, the movement applied to Composite would solve this problem.

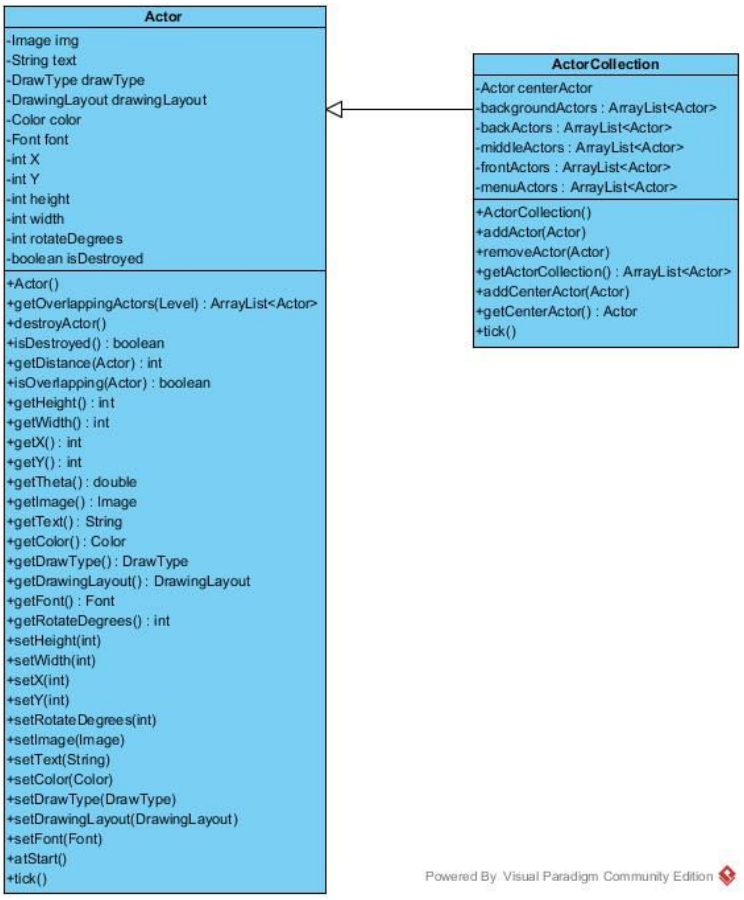

Figure 3

In Figure 3, the Composite Design applied to the Actor case is depicted. Entities which are not ActorCollection are all Leaf objects such as Bullet. However some Actors are just the collection of other actors and these are called ActorCollection in the project which stands for Composite Class. When the game design team extends an Actor class to model entities, they set x, y coordinates and rotation degrees for the actor to display the actor. If the game design team extends an ActorCollection and fills it with several actors, they set x and y coordinates of inner actors relative to the actor collection that they are into. This makes the design and control very much easy.

3.2 User Interface Subsystem Interface

User Interface Subsystem is a part of view subsystem. This subsystem contains four classes extending JFrame. The first one is the entrance of the program which has buttons which are Play Game, How to play and Information. Each button creates a new JFrame. Play Game button creates the Game Screen to play the game in it. This is the simplest package among subsystems.

3.3 Sound Subsystem Interface

This is the second subsystem in view susbsystem. Sound subsystem is planned to have one static class which generates sounds. Other classes calls its static methods to create the sound that game needs. This is separeted to avoid confusion with other view classes because it has a quite different structure than other view classes.

3.4 Engine Subsystem Interface

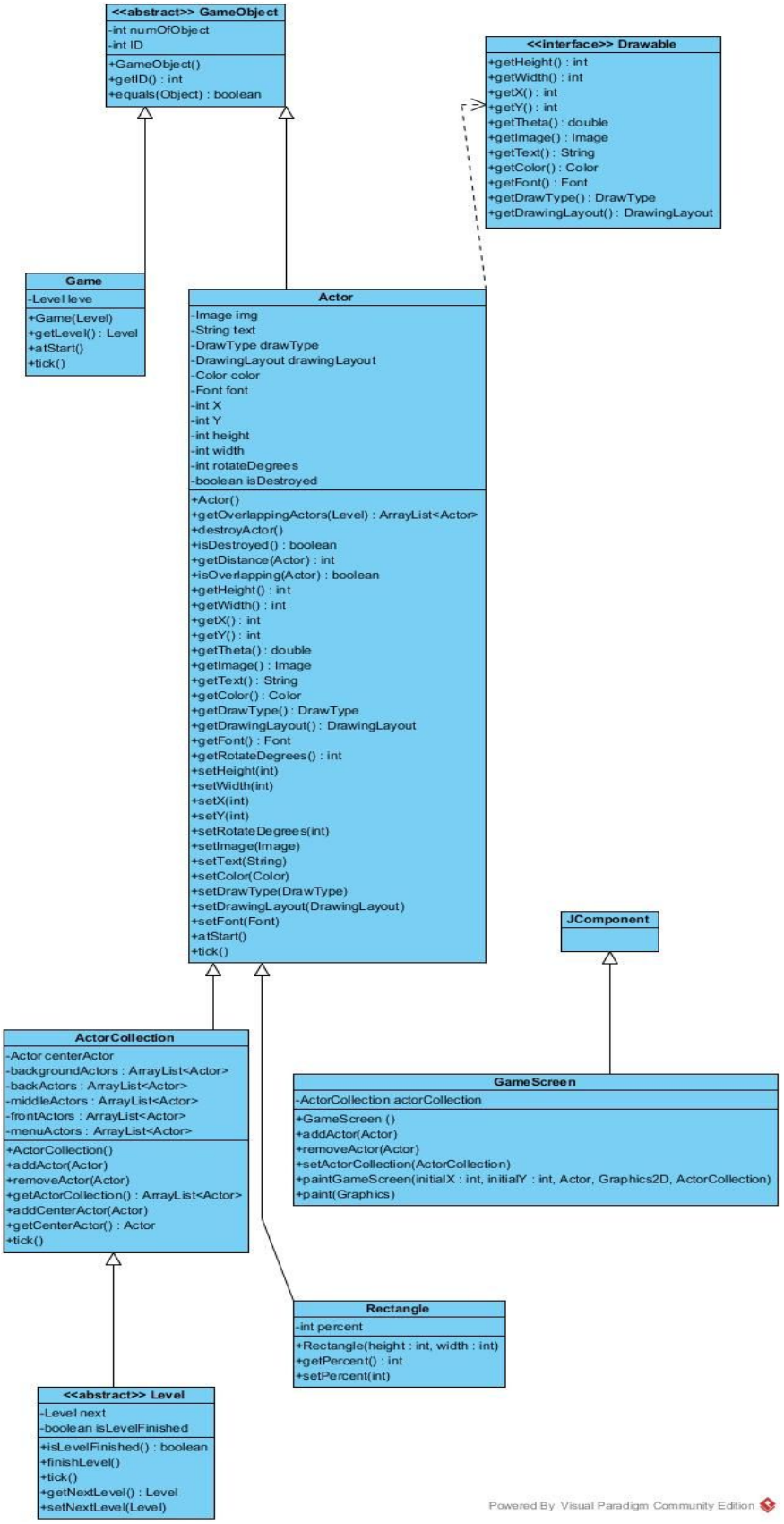

Figure 4

Engine subsystem provides the most important part of the view subsystem. This subsystem is designed using Composite Pattern and has a GameScreen object to display all the objects on an actor collection.

3.4.1 Drawable Interface

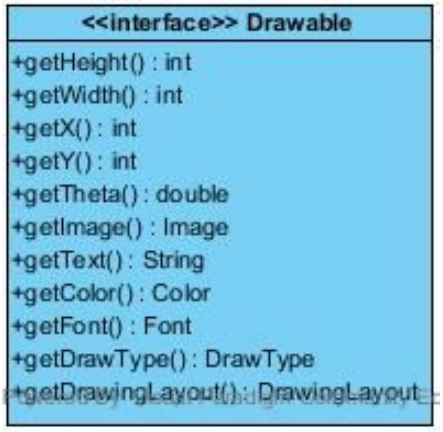

Figure 5

Drawable Interface includes methods to draw an object on the screen. Hence the GameScreen Class ensures that an Actor can be Drawable on the screen.

Attributes:
public enum DrawType{IMAGE, TEXT, RECTANGLE, COLLECTION, EMTY_PAWN} : This tells what is going to be drawn on the screen.

public enum DrawingLayout{BACKGROUND, BACK, MIDDLE, FRONT, MENU} : This tells on what layer the Drawable is going to be drawn.

Constructors: There is no constructor for an interface.

Methods:
public abstract int getHeight
public abstract int getWidth
public abstract int getX
public abstract double getTheta
public abstract Image getImage
public abstract String getText

public abstract Color getColor
public abstract Font getFont
public abstract DrawType getDrawType
public abstract DrawingLayout getDrawingLayout

3.4.2 GameObject Class

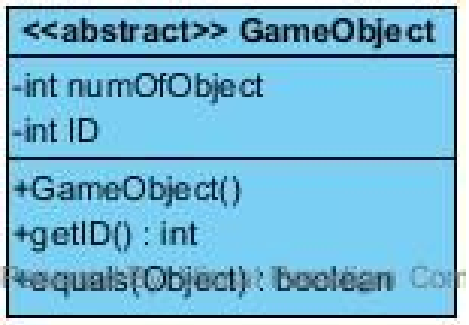

Figure 6

GameObject Class is an abstract class which has abstract tick() and atStart() methods to be overridden. Hence game control can call these methods for each object in each frame. This class has also an ID attribute to compare GameObjects.

Attributes:
private static int numberOfObjects
private int ID

Constructors:
Since this class is abstract, no instances can be constructed.

Methods:
public int getID
public boolean equals: This overrides equals method for Object class.

3.4.3 Actor Class

Figure 7

This class extends GameObject and implements Drawable Interface. It has mutators for every Drawable attributes.

Attributes:
private Image image
private String text

private DrawType drawType
private DrawingLayout drawingLayout
private Color color
private Font font

Constructors:
Actor()

Methods:
public ArrayList<Actor> getOverlappingActors(Level level) : This method takes a level and returns the array list of actors which are overlapping with the object which is called for.
public void destroyActor()
public boolean isDestroyed()
public int getDistance(Actor actor) : This gets an actor and returns the distance between this and the actor.
public boolean isOverlapping(Actor actor) : This method gets an actor and returns true if this and actor are overlapping.

3.4.4 ActorCollection Class

Figure 8

This class extends Actor class. Each actor class has only one center actor since there can only be one actor at the center of the screen which is going to be reference to other actors.

Attributes:
private ArrayList<Actor> backgroundActors
private ArrayList<Actor> backActors
private ArrayList<Actor> middleActors

private ArrayList<Actor> frontActors
private ArrayList<Actor> menuActors
private Actor centerActor

Constructors:
ActorCollection()

Methods:
public void addActor(Actor actor)
public void removeActor(Actor removeObject)
public ArrayList<Actor> getActorCollection()
public void addCenterActor(Actor object)
public Actor getCenterActor()

3.4.5 Level Class

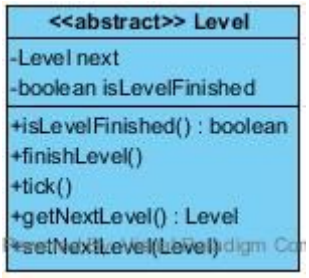

Figure 9

This class is an abstract class to be extended and design a level and it is an ActorCollection to put other level items to put in..

Attributes:
private Level next
private boolean isLevelFinished

Constructors:
There is no constructor for this class.

Methods:
public abstract void upPressed()
public abstract void downPressed()
public abstract void leftPressed()
public abstract void rightPressed()
public abstract void spacePressed()
public abstract void enterPressed()

public abstract void escapePressed()
public abstract void mousePressed(MouseCode code, int x, int y)
public void setNextLevel(Level next)
public Level getNextLevel()
public boolean isLevelFinished()
public void finishLevel()

3.4.7 GameScreen Class

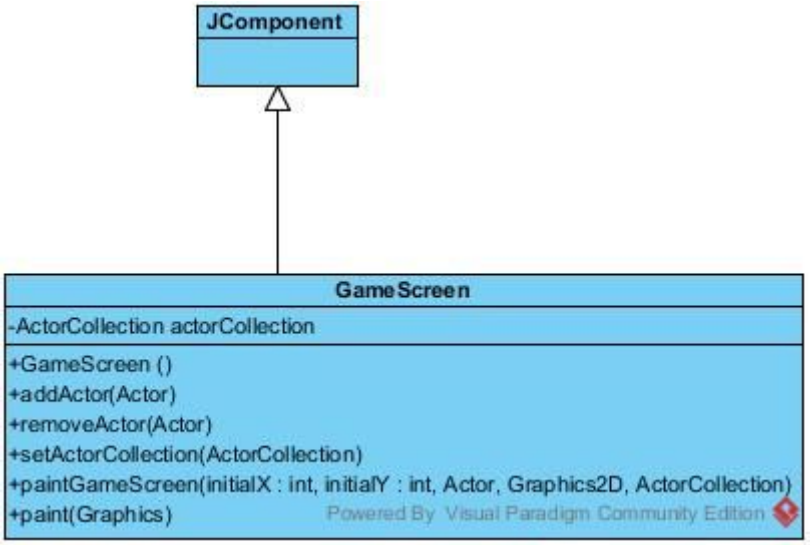

Figure 10

Attributes:
private ActorCollection actorCollection

Constructors:
GameScreen()

Methods:
public void setActorCollection(ActorCollection actorCollection)
public void paintGameScreen(int initialX, int initialY, Actor centerActor, Graphics2D g2, ActorCollection collection)

3.5 Game Control Subsystem ınterface

Game control package has single class which implements MouseListener, KeyListener and MouseMotionListener at the same time to control the game. It has a timer to call GameManager object's tick method at each frame. Game Control package creates the frame ticks.

3.6 Entities/Model Subsystem Interface

This subsystem extends Level from Engine package and designs a level. It extends from Actor or ActorCollection to implement the model of entities. It knows that implementing model will make necessary changes on the view.