BILKENT UNIVERSITY

CS319 – Object Oriented Software

FALL 2015

“RUN FROM SHAPES”

FINAL REPORT

DECEMBER 07,2015

GROUP MEMBERS

Ebubekir Karabuğa

Elif İdiz ARAT

Caner Akça

Çağlar Alkış

SECTION-GROUP NO: 3-1

Table of Contents

[1. Introduction 3](#_Toc437100188)

[2. Requirement Analysis 3](#_Toc437100189)

[2.1. Game Overview 3](#_Toc437100190)

[2.1.1 List of Bonusses 4](#_Toc437100191)

[2.2. Functional Requirements 4](#_Toc437100192)

[2.3. Nonfunctional Requirements 5](#_Toc437100193)

[2.3.1. User Friendly Interface 5](#_Toc437100194)

[2.3.2. Variety 5](#_Toc437100195)

[2.3.3. Game Speed 5](#_Toc437100196)

[2.3.4. Animations 5](#_Toc437100197)

[2.4. Constraints 6](#_Toc437100198)

[2.5. Scenarios 6](#_Toc437100199)

[2.5.1. Use Case: Play Game 6](#_Toc437100200)

[2.5.2. Use Case: Get Help 8](#_Toc437100201)

[2.5.3. Use Case: Change Difficulty 9](#_Toc437100202)

[2.5.4. Use Case: Change Color 10](#_Toc437100203)

[2.5.5. Use Case: Exit 11](#_Toc437100204)

[2.6. Use Case Model 12](#_Toc437100205)

[2.7. User Interface 13](#_Toc437100206)

[2.7.1. Navigational Paths 13](#_Toc437100207)

[2.7.2. Screen Mock-Ups 14](#_Toc437100208)

[3. Analysis 21](#_Toc437100209)

[3.1. Object Model 21](#_Toc437100210)

[3.1.1. Domain Lexicon 21](#_Toc437100211)

[3.1.2. Class Diagram 22](#_Toc437100212)

[3.2. Dynamic Models 23](#_Toc437100213)

[3.2.1. Statechart Diagram 23](#_Toc437100214)

[3.2.2. Sequence Diagrams 27](#_Toc437100215)

[3.2.3. Activity Diagram 32](#_Toc437100216)

[4.Design 33](#_Toc437100217)

[4.1 Design Goals 33](#_Toc437100218)

[4.2 Sub-System Decomposition 35](#_Toc437100219)

[4.2.1 Detailed Sub-System Decomposition 37](#_Toc437100220)

[4.3 Architectural Patterns 38](#_Toc437100221)

[4.4 Hardware/Software Mapping 38](#_Toc437100222)

[4.5 Addressing Key Concerns 38](#_Toc437100223)

[4.5.1 Persistent Data Management 39](#_Toc437100224)

[4.5.2 Access Control Security 39](#_Toc437100225)

[4.5.3 Global Software Control 39](#_Toc437100226)

[4.5.4 Boundary Conditions 40](#_Toc437100227)

[5. Object Design 40](#_Toc437100228)

[5.1 Design Patterns 40](#_Toc437100229)

[5.1.1 Observer Pattern 41](#_Toc437100230)

[5.1.2 Façade Pattern 41](#_Toc437100231)

[5.2 Class Interfaces 41](#_Toc437100232)

[5.2.1 Main Menu Class 41](#_Toc437100233)

[5.2.2. Game Engine Class 42](#_Toc437100234)

[5.2.3 Shape Class 43](#_Toc437100235)

[5.2.4 Circle Class 44](#_Toc437100236)

[5.2.5 Star Class 44](#_Toc437100237)

[5.2.6 Point Class 45](#_Toc437100238)

[5.2.7 Vector Class 45](#_Toc437100239)

[5.2.8 ActionListener 46](#_Toc437100240)

[5.2.9 JPanel 46](#_Toc437100241)

[5.2.10 MouseAdapter 46](#_Toc437100242)

[5.2.11 TimerTask 46](#_Toc437100243)

[5.3. Specifying Contracts Using OCL 46](#_Toc437100244)

[6.Conclusion 49](#_Toc437100245)

[7. References 50](#_Toc437100246)

# Introduction

We decided to design and implement a game called *Run from Shapes*. In our game, aim of the player is escape from the falling down shapes. The game will be a desktop application and will be controled by mouse. This report contains an overview of the game, describes the basic gameplay elements and rules of the game. Then it describes functional requirements, non-functional requirements, use-case’s models including scenarios and use-case diagrams.

# 2. Requirement Analysis

## 2.1. Game Overview

Purpose of this game is stay alive while shapes are falling down from the top of the platform. In our game, our charachter icon is a circle and it will be controlled by mouse move.There will be restricted platform that during the game, it can not be extended or shrinked.Shapes will fall down from the top of the platform and they can pass beyond the platform but user must remain in the boundaries of platform. Shapes will be created randomly which means that shapes can be any polygons and while those shapes falling down from the top of the platform, they will be spin around theirselves which makes harder to play the game for the player. Score of the player will depend on survival time of the player. Score will be increase linearly while player keep avoid from falling down shapes.Whenever score of the player reachs multiples of hundred(ie. 100,200,300,..) level of the player will increase by one. Maximum level of the game will be five and after a player level five, speed of the game remain constant regadless of the player score. Also there will be special bonuses which is represented by special shape which is a star, and if player can collect these stars, player gain bonus abilities or features which makes game easier for the player. If player crush any shape that fall down from above(except shape of star), player lose one life point and if player has used all of his lives, than game will be over and the player can see his/her total score in the pop-up screen. There will be multiple color option for charachter icon and player can easily choose his/her charachter icon color. Also for player who prefer challenge, there will be a difficulty option in our game. There will be three difficulty settings in our game which are easy medium and hard. Whenever player choose one of those options , his/her icon can be shrinked or extended, its depends on difficulty choise of the player. In addition to those features, there will be help button in our game which helps player to understand to game play, level up system and bonuses.

### 2.1.1 List of Bonusses

* **Additional life:** It increases the lives of the player by one.
* **Smaller icon:** Whenever player collect this bonus, his/her charachter icon gets shrinked.
* **Bonus score:** Players score increase by 10.

**Important Note\*\*:** When player hits any object that falling down from the top of the platform, all of the bonuses will be canceled.

## 2.2. Functional Requirements

* User will be able to control the character icon by mouse move.
* User can change the settings of the game. Available settings are ;

**1.Difficulty:** This option allows the player to choose size of the circle that represent the player.

**2.Color:** By clicking this button, user can select the order of the circle.

* Our game has a score system which is based on time that player survive, Score of the player lineary increase while player avoid from falling down polygons.
* Game will offer level system which is depend on player score. If player has achieve multiples of hundred points(100,200,300,400,500) level of the game will increase by one which means that object will fall down faster than before. 500 points is equals to level five which is the maximum level of the game and after that game speed remain constant regadless score of the player.

## 2.3. Nonfunctional Requirements

### 2.3.1. User Friendly Interface

Game will be consist of user-friendly interface, user can easily interact with the game and its easy to play the game. As a group, we care player desires which is feel comfortable while playing the game and for achieving that we developed understandable interface to players.

### 2.3.2. Variety

Since we developing a game that based on shapes, we are planning to use some smooth dynamic animations in order to show objects movement clearly.

### 2.3.3. Game Speed

Game speed changes from 10 fps to 60 fps respectively.

### 2.3.4. Animations

Vertex coordinate of the shapes is calculated according to mathematichal(sin and cosine functions) calculations in order to achieve smooth rotations of shapes.

## 2.4. Constraints

The game will be implemented in Java. We used few libraries from Java for some classes and methods. For example, in Shape class for updatingCoordinate method we benefited from “java.lang.Math”. Also, we use “javax.swing” library for processing graphics. Also, we use “java.awt” and “java.awt.event” for mouse listener. For time calculations, we used “java.util.Timer” and “java.util.TimerTask” .Resolution is 1000x800 in game which gives a huge space for player.

## 2.5. Scenarios

### 2.5.1. Use Case: Play Game

|  |  |
| --- | --- |
| **Use Case Name** | Play Game |
| **Primary Actor** | Player |
| **StakeHolders and Interests** | None |
| **Pre-Condition** | Game should be activated. |
| **Post-Condition** | After players has lost all of his/her lives in any level, his/her score will be shown by the system in a pop-up screen. |
| **Entry Condition** | Player selects “Play Game” button from Main Menu. |
| **Exit Condition** | Player has lost all of his/her lives in any level. |
| **Flow Of Events** | 1. Game is started after the ***“Play*** ***Game”*** button is clicked. 2. Player starts from the first level which is minimum speed of the gam and player score is 0 in the begginning. 3. Shapes will start to fall down from the above of the platform. 4. After shapes start to fall down from above of the platform, player will start to move the mouse in order to escape from the shapes that are coming through the charachter icon of the player. 5. Score of the player will increase automaticly and linearly while he/she avoid from crush any falling down polygons. 6. Shape which is created newly will enter the platform zone. 7. If player succesfully escape from shapes and if player’s score reach next levels threshold point then level of the game will increase by one.. 8. Player will move next level. 9. Player plays until he/she consumes all of lives of the circle. |

### 2.5.2. Use Case: Get Help

|  |  |
| --- | --- |
| **Use Case Name** | Get Help |
| **Primary Actor** | Player |
| **StakeHolders and Interests** | Player aims to get help from the system in order to learn how to play game. |
| **Pre-Condition** | Player should be in main menu. |
| **Post-Condition** | None |
| **Entry Condition** | Player selects “Help” button from main menu |
| **Exit Condition** | Player should press the “OK” button from the pop-up screen. |
| **Flow Of Events** | 1. Player selcets “Help” from the main menu. 2. The system displays the help documentation about the game. 3. After reading the documentation, player returns to the main menu. |

### 2.5.3. Use Case: Change Difficulty

|  |  |
| --- | --- |
| **Use Case Name** | Change Difficulty |
| **Primary Actor** | Player |
| **StakeHolders and Interests** | Player desires to change the game difficulties. System updates the settings which are changed by the player. |
| **Pre-Condition** | For first running, difficulty will be set as default. Player should be in the main menu in order to change game difficulty. If Player changes difficulty, adjusted settings will be saved and used by system. |
| **Post-Condition** | Difficulty of the game updated. |
| **Entry Condition** | Player selects “Difficulty” button from menu. |
| **Exit Condition** | None |
| **Flow Of Events** | 1. Player choses to difficulty from main menu. 2. The system displays different difficulties on the gameplay. 3. Player configures the settings according to his/her comfort. 4. The system saves corresponding changes in order to player play with desired difficulty. 5. Player returns to main menu. |

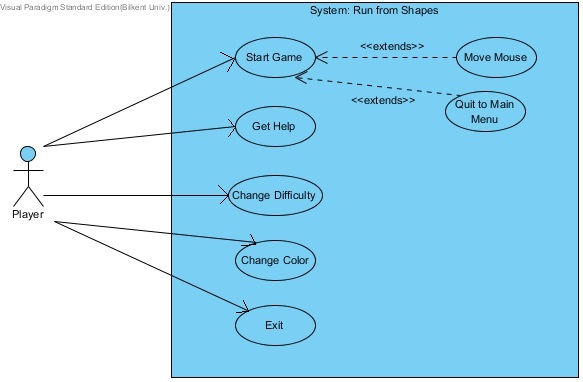
### 2.5.4. Use Case: Change Color

|  |  |
| --- | --- |
| **Use Case Name** | Change Color |
| **Primary Actor** | Player |
| **StakeHolders and Interests** | Player desires to change in game color of his/her charachter icon.System updates the settings which are changed by the player. |
| **Pre-Condition** | User must be in main menu. |
| **Post-Condition** | Color of the his/her shape updated. |
| **Entry Condition** | Player should select desired color from main menu by click the left button of the mouse. |
| **Exit Condition** | None |
| **Flow Of Events** | 1. Player choses to desired color by using color button. 2. The system saves corresponding changes in order to player play with desired color. |

### 2.5.5. Use Case: Exit

|  |  |
| --- | --- |
| **Use Case Name** | Exit |
| **Primary Actor** | Player |
| **StakeHolders and Interests** | Player desires to exit from game. |
| **Pre-Condition** | None |
| **Post-Condition** | Game will be closed by the system. |
| **Entry Condition** | Player selects “Exit” button from main menu or player clicks default close operation which is located in the northeast corner of the game . |
| **Exit Condition** | None |
| **Flow Of Events** | 1. Player selects “Exit” button from main menu. 2. System will close the game. |
| **Alternative Flow Of Event** | 1. If player request exit from the game while keeping play the game,    1. Player clicks the default close operation which is located in the northeast corner of the game .    2. System will close the game. |

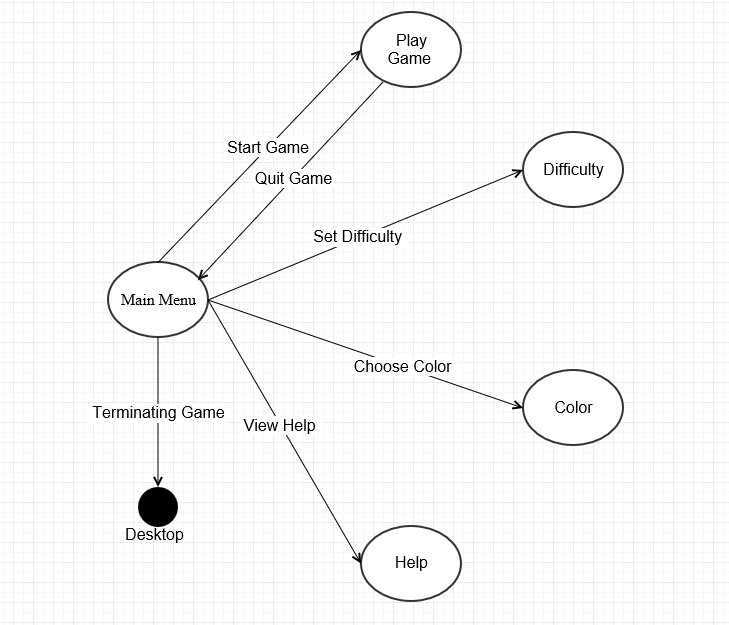
## 2.6. Use Case Model



**Figure 2.6- Shows use case model of the Run from shapes)[1]**

## 2.7. User Interface

### 2.7.1. Navigational Paths

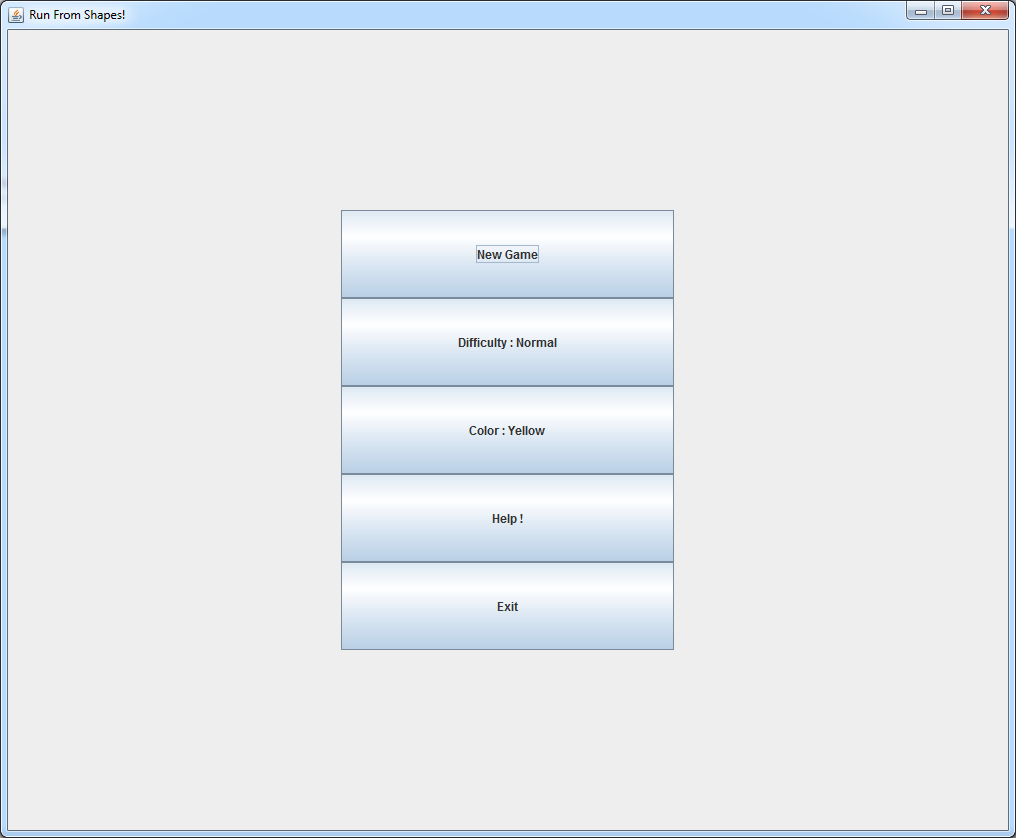


**(Figure 2.7.1 Shows Navigational Paths)[1]**

### 2.7.2. Screen Mock-Ups

#### 2.7.2.1. Main Menu

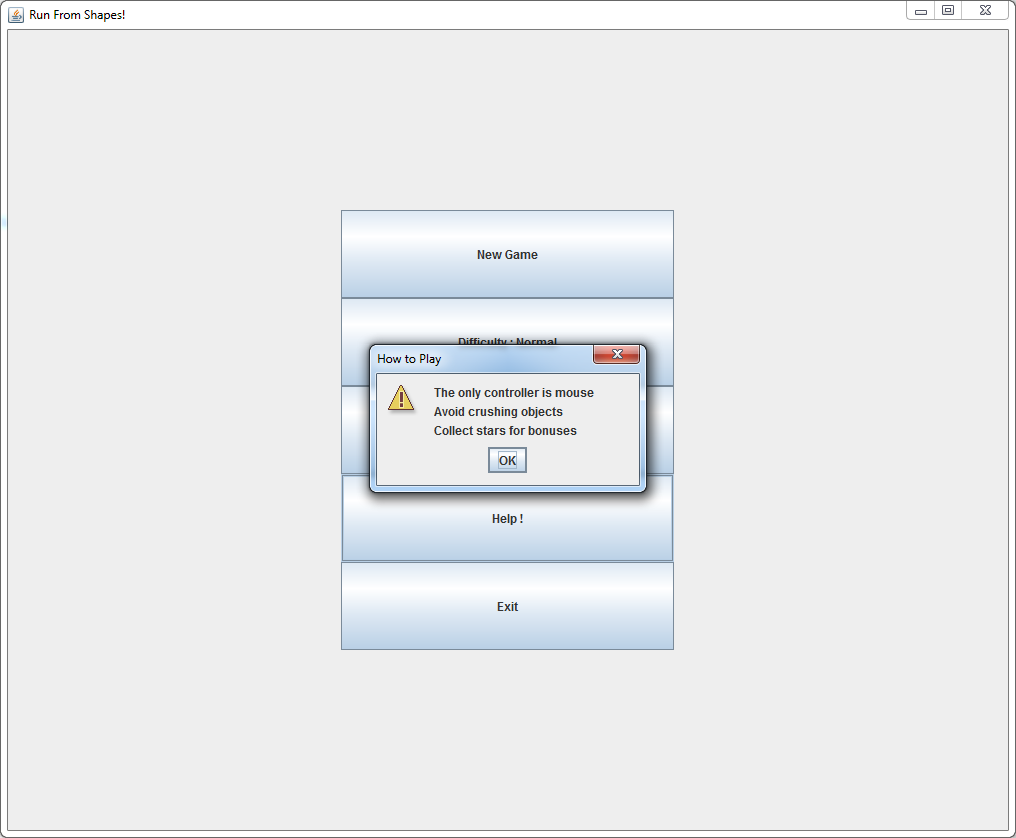
When application begins to run, Player will see main menu screen. Main menu screen shows five options to Player which are New Game, Difficulty, Color, Help, and Exit . (Figure 2.7.2.1)



**(Figure 2.7.2.1 – Shows Main Menu Interface)**

#### 2.7.2.2. getHelp

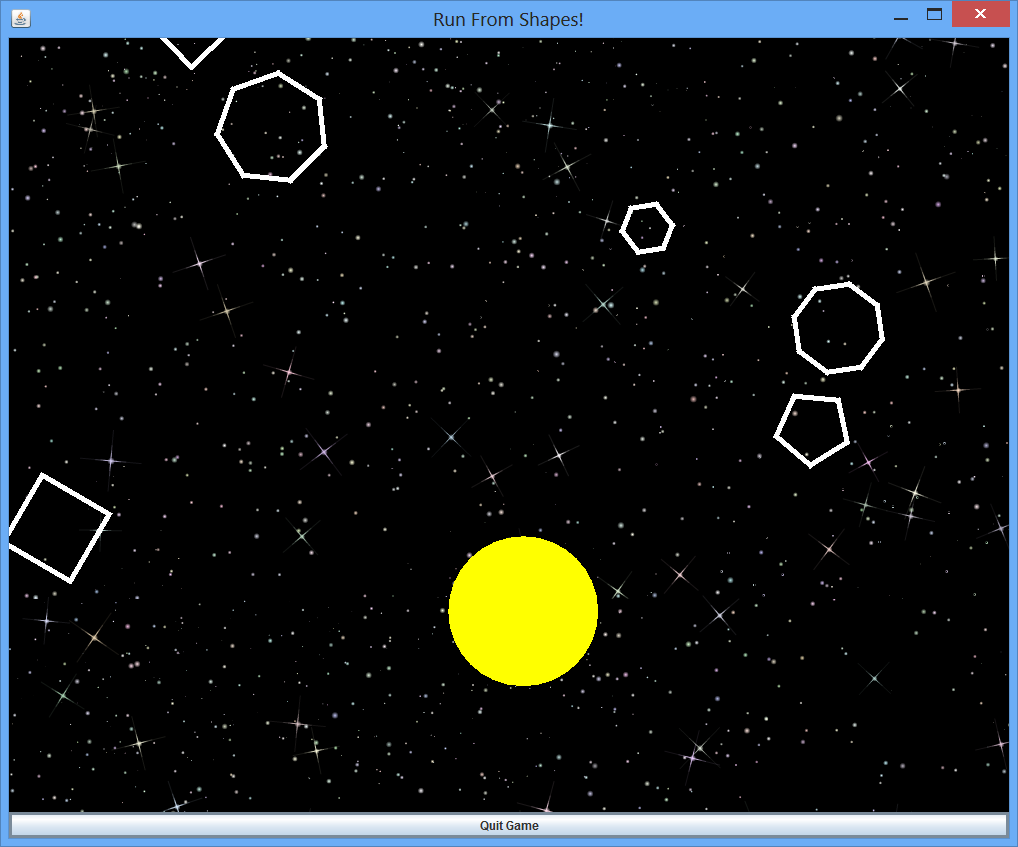
When player clicks help from main menu, new screen will appear and in this screen, player can learn how the play this game. After player finish reading the gameplay than he clicks the “OK” button from the pop-up screen in order to return to main menu.(Figure 2.7.2.2)



**(Figure 2.7.2.2-Shows Help Screen )**

#### 2.7.2.3. Gameplay

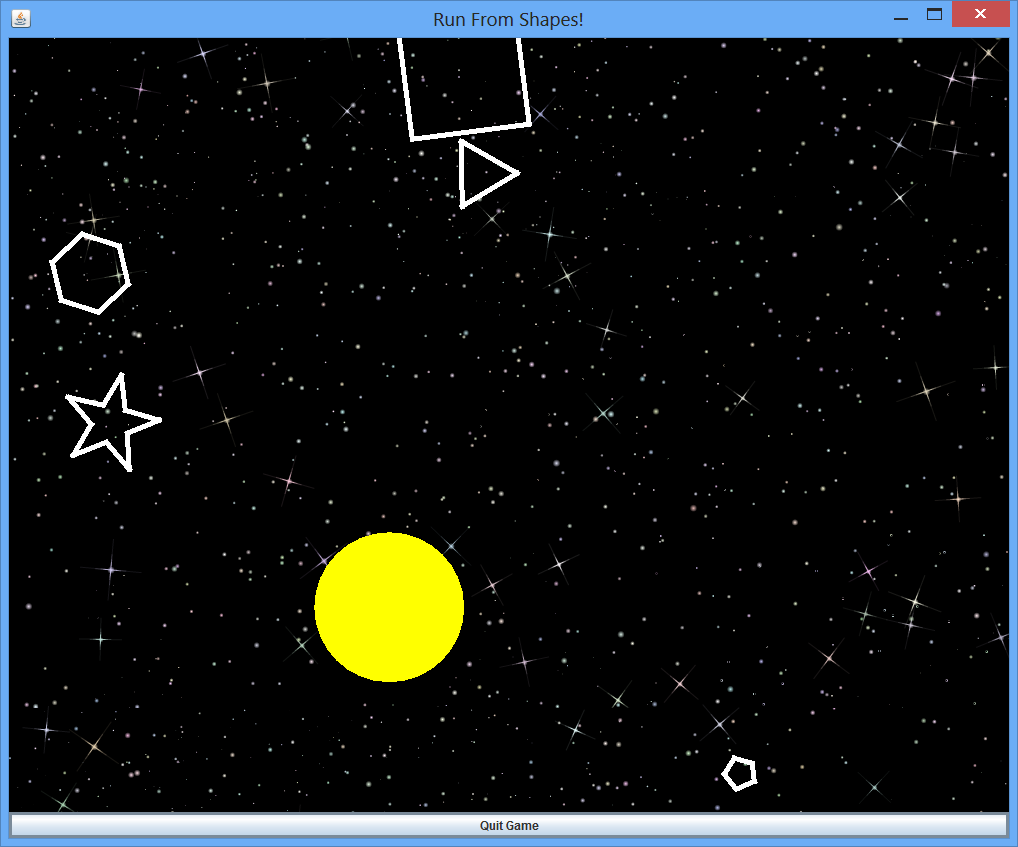
After player clicks new game option from the main menu, game starts to play and shapes will be automatically created and fall down from the top of the platform. In figure 2.7.2.3, yellow circle represents players charachter icon and he/she tries to avoid any collision from shapes above him by using mouse move.

****

**Figure(2.7.2.3- Shows game playing main screen )**

#### 2.7.2.4. Bonusses

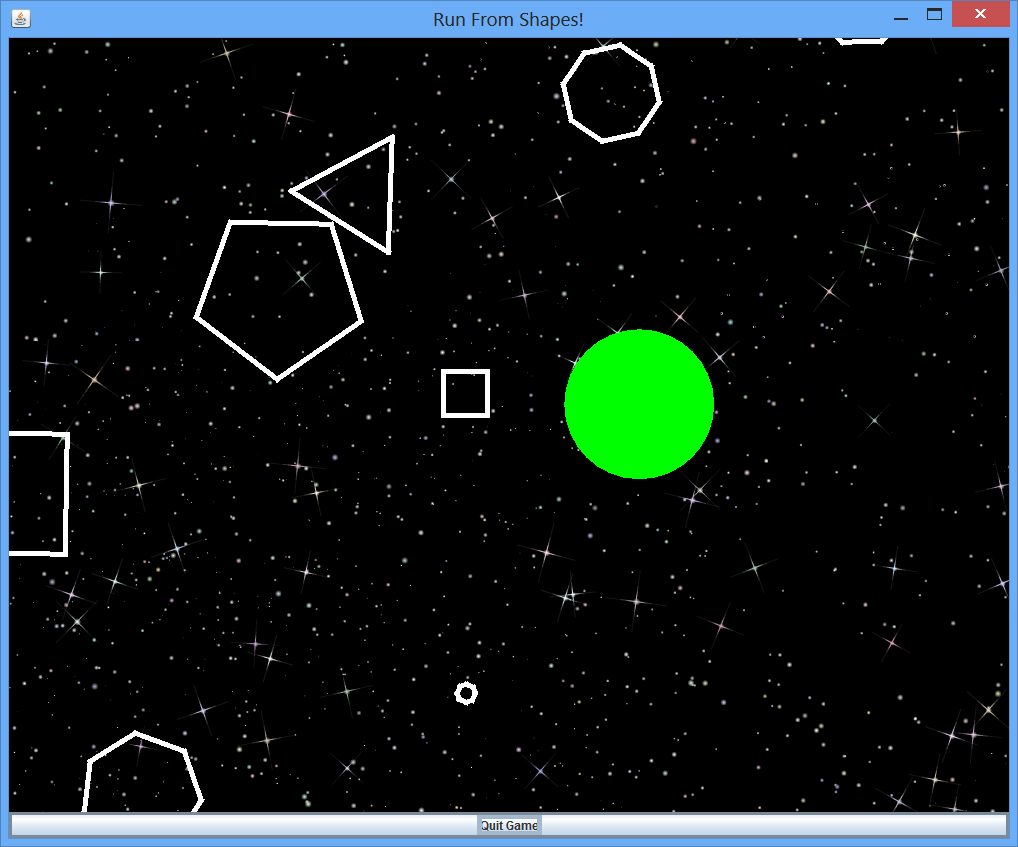
While player keep playing the game a specific shape which is a star falling down from the top of the platform randomly. Since star is also a shape, it created randomly and its creation has same probability with other shapes. Whenever player collects these stars, he/she gains some bonuses and these bonuses might be extra life,extra score or smaller charachter icon. Star is shown in Figure 2.7.2.4



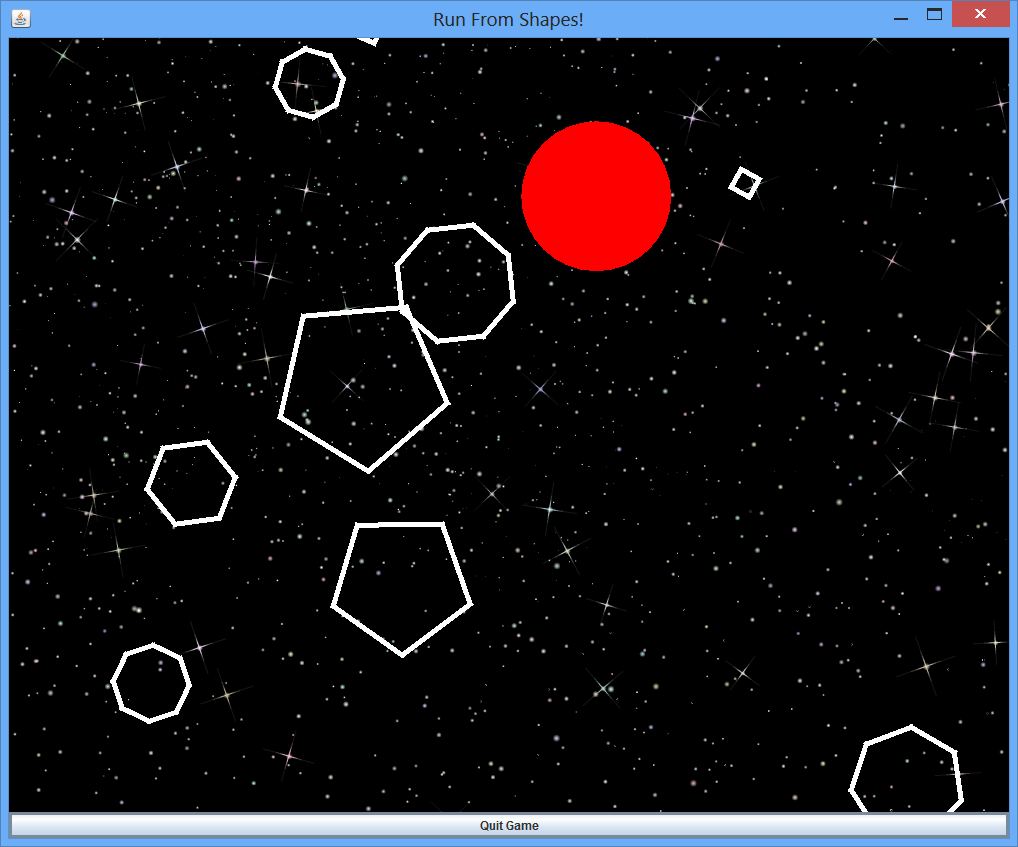
**Figure(2.7.2.4-Shows star bonus)**

#### 2.7.2.5 ChangeColor

Before starting game, if player wants to change color of the main character which is circle in our game, with just clicking Color option from main menu.We have limited color options that are red, green and yellow.



**(Figure 2.7.2.5/1 –Shows green circle option while player keeps playing)**



**(Figure 2.7.2.5/2 –Shows red circle option while player keeps playing)**

# 3. Analysis

## 3.1. Object Model

### 3.1.1. Domain Lexicon

During the design of Run from Shapes more than one comprehensive domain had to be examined : game engine, user interface,content and logic.

**a)Game Engine**

This part of the system which does not interact with the user.Game engine is about interactive control objects and ultimately changes in other objects, it is completely seperated from game logic and game contents.

* **Collusion :** Collision refers interaction between player’s circle and the other shapes that falls down.
* **Rotation :** A computer graphics term, rotation specifically refers to thev aciton of the rotating a shape or image according to specific origin.
* **X Width :** A graphical term defines the x coordinate of a drawable object in space.
* **Y Length :** A graphical term defines the y coordinate of a drawable object in space.
* **Update :** This refers to applying all changes in the game logic to the game objects which are shapes in the game.

**b)Game Control**

This part of the system deals with user interaction and its effects on the both game engine and game logic.

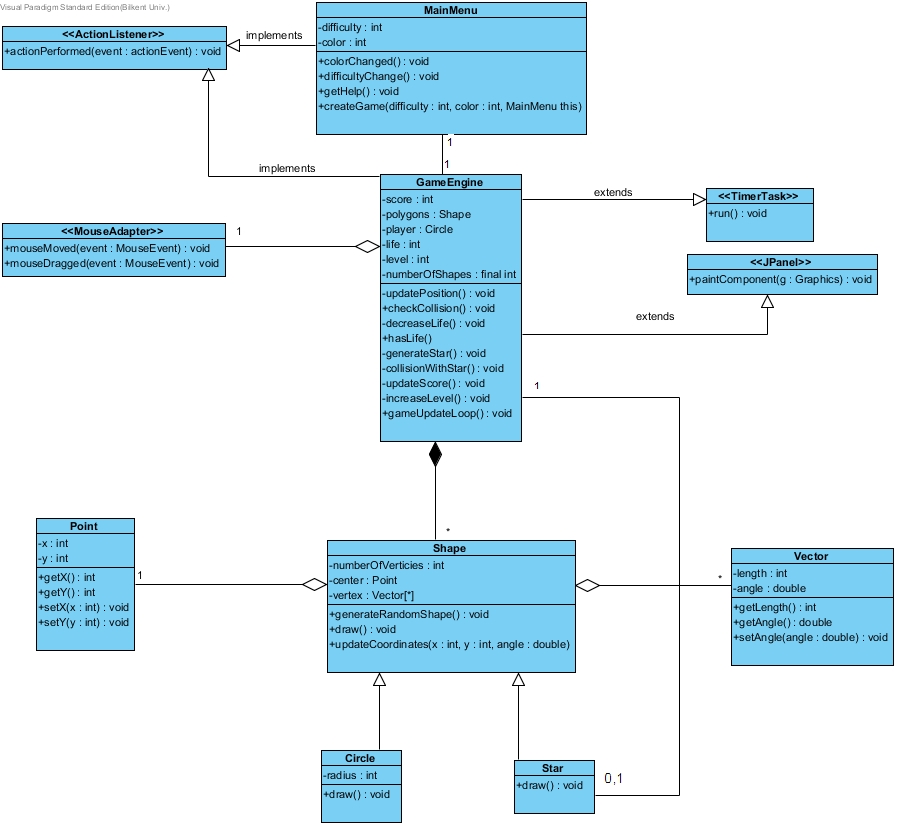
* **Click Area :** It defines areas that can be clicked by user.
* **Main Menü :** The first step that user interact with the game.Most of computer games start with a main menü and allows user to navigate the game with his/her choice.
* **Options :** Allow user to change difficulty of the game and also it gives permission to change color of the main character.

**c)Game Logic**

Game logic is the model of the game. It defines game’s action and interaction between user and game.

* **Bonus:** Specific shape that gives an addition life to the player.
* **Circle:** Specific shape that controlled by player. Circle shrinks when the game level increase.
* **Collision Distance:** Distance between circle and shape. It is the distance of two points: one of them is on the circle’s arc and other one is on the shape’s edge. When this distance become 0, circle and shape collide each other.
* **Game Level:** A game level is a rank that player reachs. Game level increase when player reach a specific score.
* **Life:** It defines the notion of “life”, or the number of “lives” a player has. At the beginning of game, player has specific number of lives and when player lost all his/her lives, game ends.
* **Shape:** Shape objects are actually polygons. Player should try to avoid these shapes in order to increase his/her score.

### 3.1.2. Class Diagram

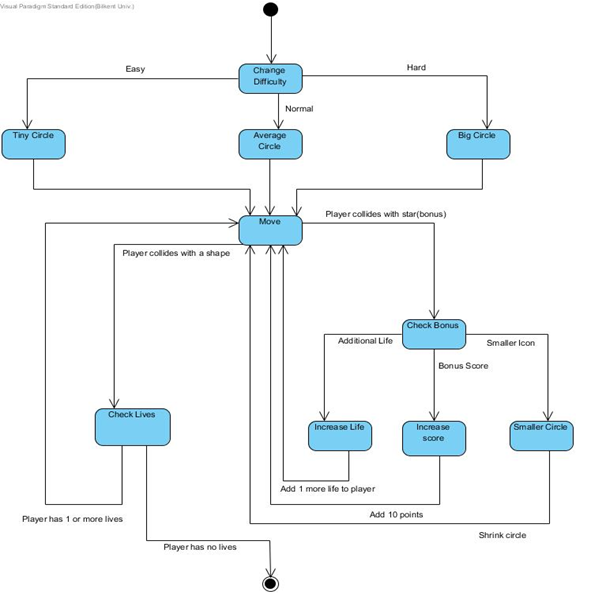


**(Figure 3.1.2 –Shows class diagram)**

## 3.2. Dynamic Models

### 3.2.1. Statechart Diagram

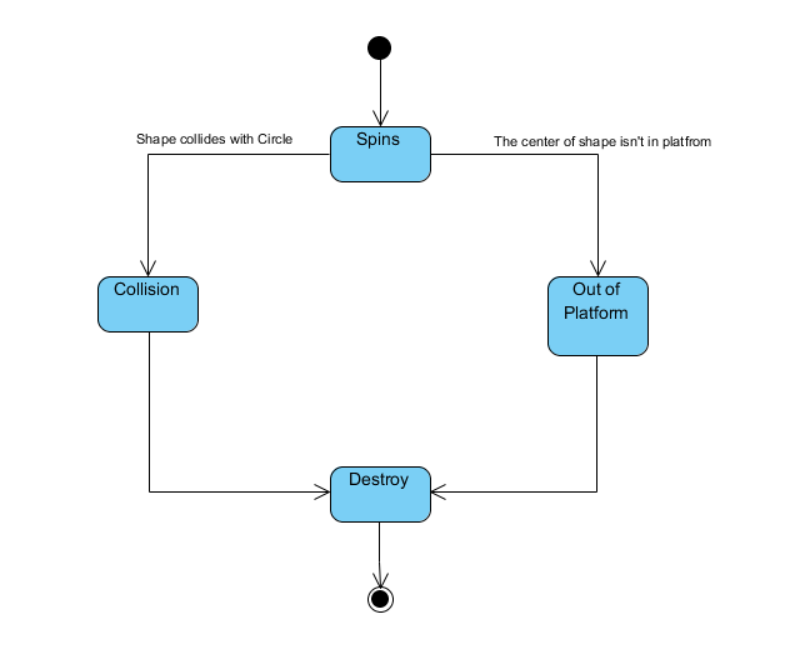
#### 3.2.1.1. Circle State Diagram



**(Figure 3.2.1.2- Shows state diagram of circle which main character)**

Player starts the game with a default diffuculty, which is normal. However, it can be changed depends on the user. Procedure will start with “Difficulty” state and one of the three difficulty choice will be the next state which are “Tiny Circle”, “Average Circle” and “Big Circle”. After the difficulty choice, next state will be “Move” state which define the movement of player on the platform. At this point, circle object either collides with a star or a shape. If user collides with a star, next state will be “Check Bonus” in order to check the type of bonus. There is three type of bonus, one of them is “Additional Life”. It gives an extra life to player. Other one is “Bonus Score” and it gives 10 more points to player. Last type of bonus is “Small Icon”, and it shrinks to Circle of player. After the bonus type state, next state will be the “Move” again and player moves on the platform again. If player lives is 1 or more, next state will be “Move” and player continue to move on platform. However, if player has no life, next state will be end state.

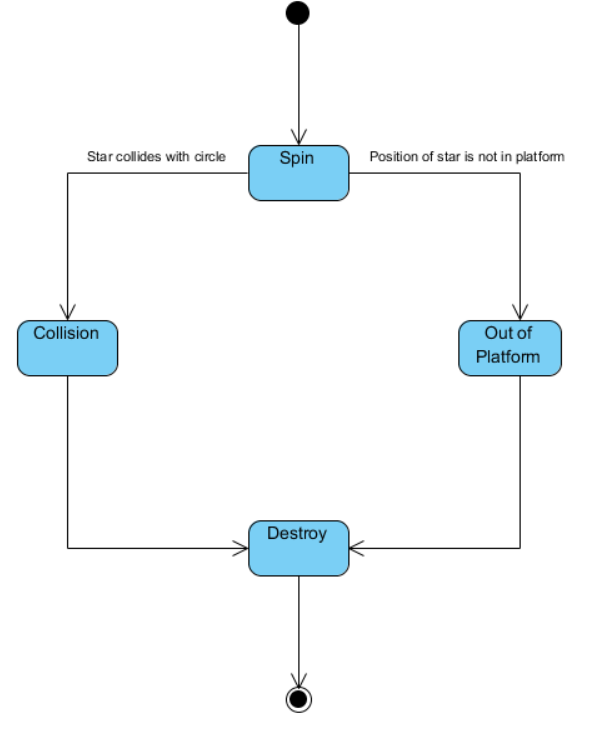
#### 3.2.1.2. Shape State Diagram

****

**(Figure 3.2.1.2 Shows state diagram of shape)**

In the beginning, shape object is in its “Spin” state. It makes state object to spin from top to bottom of the platform. Then, shape object either collides with Circle which belongs to player and goes to “Collision” state, or it will go out of boundaries and goes to “Out of the Platform” state. In both of these states, next state will be the “Destroy” state and object will destroy. After “Destroy” state, end state will be next state and procedure will stop.

#### 3.2.1.3. Star State Diagram



**(Figure 3.2.1.2- Shows state diagram of star shape)**

Star State Diagram is similiary to Shape State Diagram. However, the object in this state diagram will be a star. All states and states direction are same with Star State Diagram.

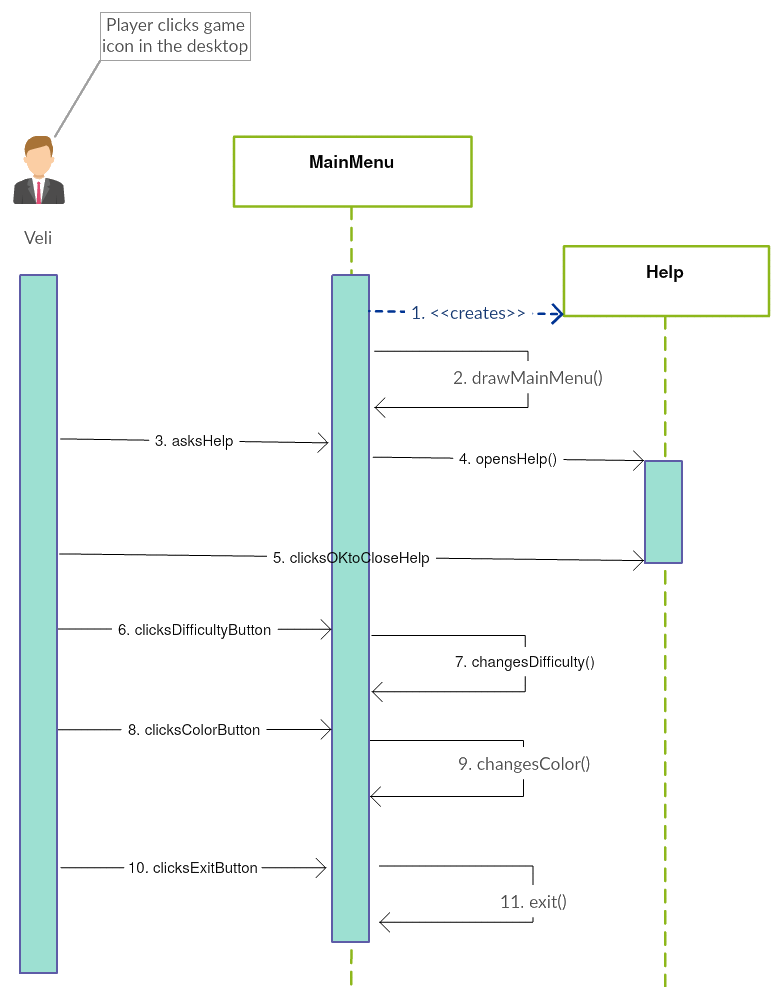
### 3.2.2. Sequence Diagrams

#### 3.2.2.1. Execute Game

Following sequence diagram illustrates the scenario explained below:

**Scenario Name:** Execute Game

**Scenario:** Veli double clicks the game icon in the desktop. Then, he sees the main menu and chooses the “Help” button. Veli reads the instruction which was written there and after understand how to play this game, he select “Ok” buttun from pop-up screen and return to the main menu. Afterward, he chooses “Difficulty” option from main menu. Veli chooses one of the shown difficulty according to his desire and then he returns to the main menu. After Veli adjust difficulty settings, he decided to change to color of his charachter icon. He choose one of the color from main menu and than he decided to exit from the game and choose to “Exit Game” button from the main menu.



**(Figure 3.2.2.1. Shows the sequence diagram which explains execution of game)[2]**

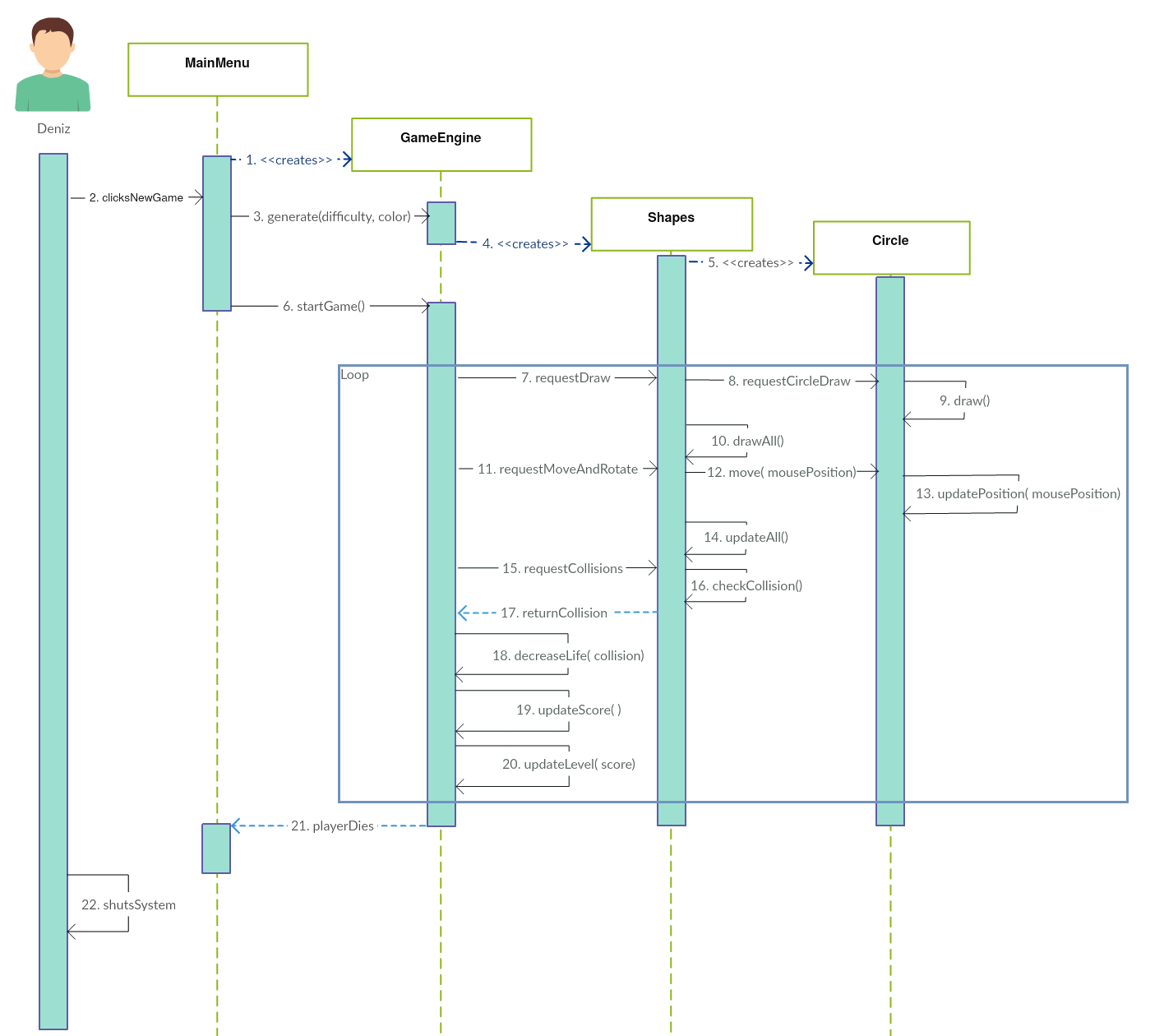
**Description:**After player executes game, player can select help, choose color,difficulty,new game or exit options. In this scenario player choose help,difficulty,color and exit options. While player clicks help button from main menu, new screen will be open and MainMenu creates Helps life cycle. After player reads neccessary information from the help screen, he clicks to “Ok” button from pop-up screen and return to the main menu.

#### 3.2.2.2. Play Game

Following sequence diagram illustrates the scenario explained below:

**Scenario Name:** Play Game

**Scenario:**Deniz decided to choose play game from the main menu. For this scenario we assume that Deniz did same steps with Execute Game scenario but in the end he clicks “Play Game” button rather than “Exit Game” button. After Deniz starts to play game, shapes will start to fall down drom the top of the platform and Deniz avoid from crush this shapes by using mouse. After a while, Deniz hits a shape while he has only one life left and after this crush he died. Than Deniz returns main menu automatically.

**

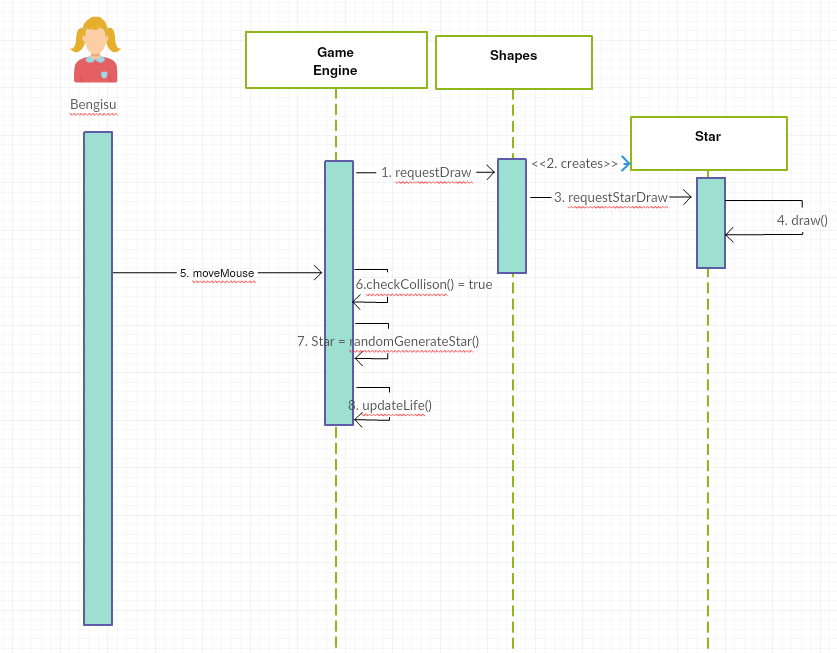
**(Figure 3.2.2.1. Shows the sequence diagram which explains play game case)[2]**

**Description:** After player selects the new game button, the main game loop starts to execute. In every cycle, game loop updates charachter icon and shapes position then checks if any collision happens between charachter icon and falling down shapes. Also in the loop every shapes starts to rotate while they are falling down from the top of the platform. Also in the loop,while a shape is go beyond the boundaries of the platform new shape will be randomly drawn and it starts to fall down from the top of the platform. In addition to these, GameEngine will keep the recordof the current score of the player and if the score bigger than threshhold point, then GameEngine update the level of the game and the shapes starts the fall down faster than before. After game is over, player returns to the main menu.

#### 3.2.2.3. Acquiring Bonusses

**Scenario Name:** Acquiring Bonuses

**Scenario:** Bengisu has already started playing the game. Than suddenly a star shape falling down from the top of the platform and Bengisu captured it. Then Bengisu realise that this is a life bonus and her total life increase by one.

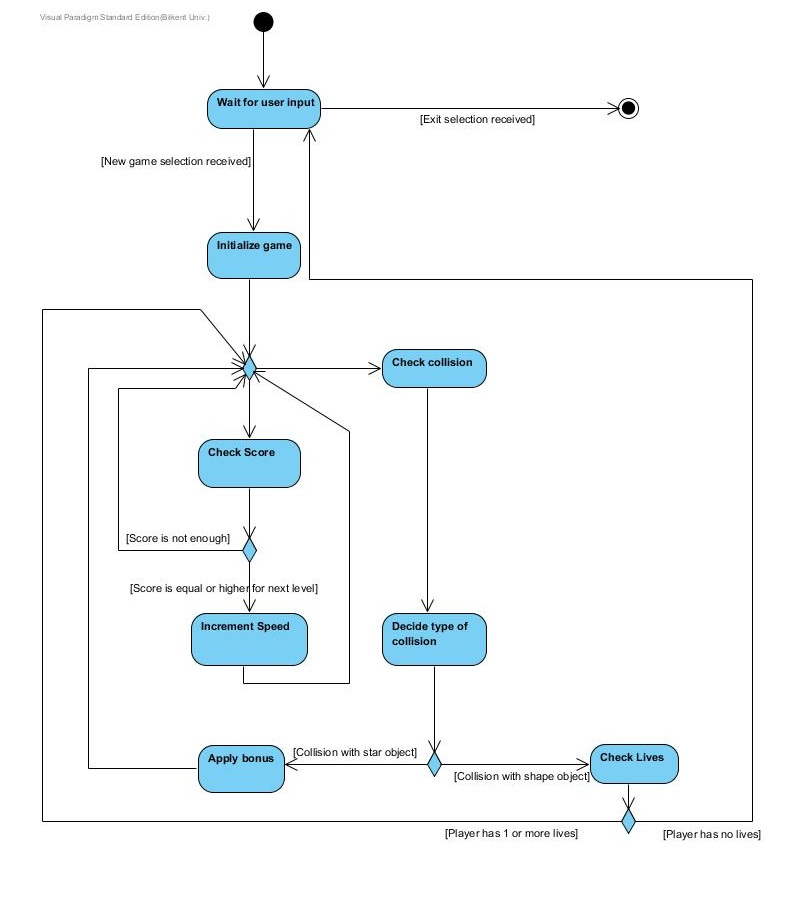
****

**(Figure 3.2.2.1. Shows the sequence diagram which explains getting bonuses)[2]**

**Description:**Shapes will create random polygons and star shape. Star shape can be created same possiblity as other shapes and whenever star shape created from Shapes, a star shape starts to fall down from the top of the platform and if player can catch this star shape than he/she get random bonus which is created by GameEngine. After player gets his bonus, he/she keep playing the game with this extra bonus feature.

### 3.2.3. Activity Diagram

This diagram shows how system maintains gameplay:

****

**(Figure 3.2.3.- Shows activity diagram of the gameplay)**

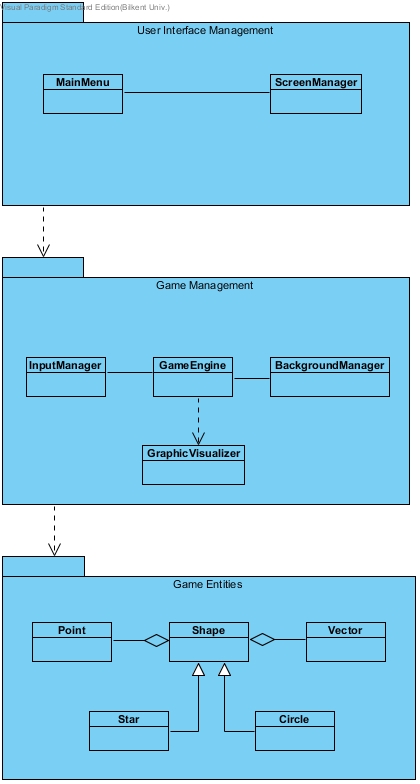
When player selected New Game selection, system initialize game by instantiating game objects and creating game map. When this preparation process is finished, system waits user to interact with the mouse in order to start the game. When shapes start to fall down from above firstly system checks are there any collision between circle and the other shapes. If there is any, system checks type of collision. If type is collision with the star object system increases score of player. After increment of score, system checks score of player in order to decide level up or not .If player’s score is enough system opens the next level and increases speed of game. If score is not enough, player will keep going the same level. Then system checks again to see is there any new collision or not. If system detects a new collision system will tries to categorize type of collision. If type is collision with the other shapes that fall down randomly, system checks life of player that he/she has. If player has life than system allows continuation of the game, otherwise if player has no life, system ends the game.

# 4.Design

## 4.1 Design Goals

* ***Portability* :** Java is flexible language that accepts codes without many changes.When the operating system is changed java is not affected by this kind of changes.Also java is suitable language that convert to codes through android platform with minor changes.
* ***Understanability:***We have aimed that create a game which is easily understandable by the user.Our game is easy to play , for accomplish understandability criterion we created user friendly menu. Difficulty and color settings can be changed only using one button click. Also we put an “Help” button in main menu, if player doesn’t understand the game he/she can get general information about the game-play.
* ***Reliability :*** “Run From Shapes”will be bug-free and non-crashing program because there will not be so many inputs to it and it will not use too much space in stack memory.
* ***Modifiability :*** Since “Run From Shapes” is an object-oriented programmed game, it is always possible to modify existing functions and add new features (new level,new bonuses) to it by simply creating new classes. We aimed well designed subsystem decomposition in order to make changes easily.Each subsystem has relation with at most one level down layer .

## 4.2 Sub-System Decomposition



**( Figure 4.2 –Shows decomposition of the game )**

**Subsystem Decomposition**

We used three main subsystem in our game: User Interface, Game Management and Game Entities.

* **User Interface Management**

User Interface Management subsystem holds necessary elements for user-interface in our game. Game Management consists of graphical objects(panels, frames, buttons) for interacting with user and it also coordinates the transitions between different panels.

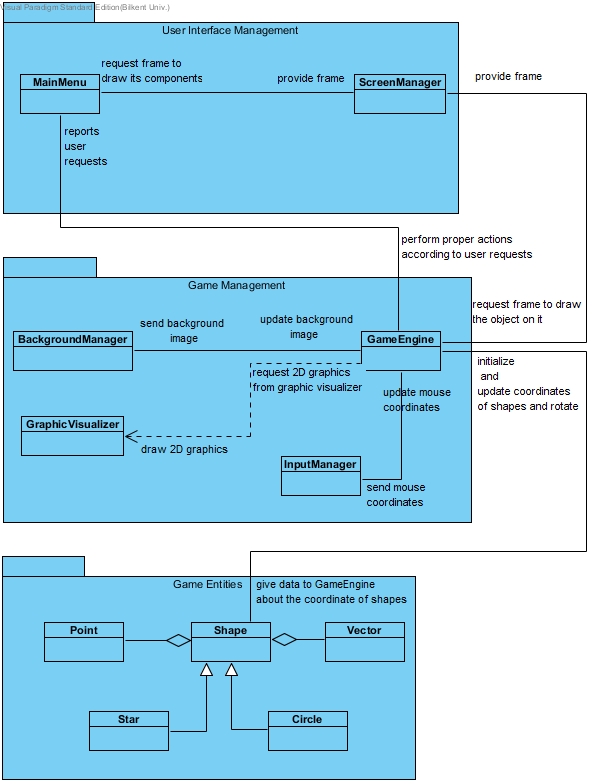
* **Game Management**

Game Management subsystem is responsible from game logic. Also, it is responsible from physics(collision between two shapes, rotation and animation) level/score coordination, input management and map coordination(how many shapes in game, how many bonus etc.).

* **Game Entities**

All entity objects’ classes(Shape, Star, Circle etc.) are holding in Game Entities subsystem. They are created and used by Game Management subsystem.

### 4.2.1 Detailed Sub-System Decomposition



**(Figure 4.2.1- Shows detail of the sub-system decomposition)**

## 4.3 Architectural Patterns

We prefered to use two main design patterns that are Layered Pattern and MVC(model-view-controller) in our game. The relationship between subsystems and MVC architecture is following: User Interface Management is “View”, Game Management is “Controller” and Game Entitites is “Model” part of MVC architecture. We grouped our domain objects into game entities which constitues the model of our system. The domain objects are only controlled and accessed by manager classes that constitues the controller part. Also, we grouped the classes which are responsible for interaction between user and system into User Interface which constitues the view of our system. We chose MVC because it is good a design for games because changes on the interface doesn’t lead to change the model of the system.

In our system decomposition, we seperated our system into three layers: User Interface Management, Game Management and Game Entities. Our top layer is User Interface Management which is responsible for interaction between user and system and it has highest hierarchy because it is not used by any other layer above. The following layer is Game Management which is responsible for game logic. Our bottom layer is Game Entities. Our layers are closed which means a layer can access only to layer below it.

## 4.4 Hardware/Software Mapping

Our game runs on one PC at a time i.e a stand alone system.As ıt can be seen below , there is a deployment diagram of the stand alone system. The programming language that we will use for implementing the core design of our game project is JAVA. We will implement the game helping with Java programming language .Furthermore, Java has large library that helps implementer, for practical usage. Also, Java answers our design goals that are reliable functionality and high performance since its compilers, memory management and multithreading make programs more reliable and faster.

## 4.5 Addressing Key Concerns

### 4.5.1 Persistent Data Management

Game data will be stored in the client hard disk drive, we will not use any database since the data we use in the game needs to be accessed in real-time. Therefore, we will load all the necessary files on to the memory and access those files when the game logic or the rendering system requires. The background images will be stored unencrypted to encourage modifying background images for their personal preference.

### 4.5.2 Access Control Security

Our game will not implement any user authentication system so we will not suppose to have database that stores user type. For the security of our program, the only access to the file system is given to the "GameEngine" which is the main game logic class. Also, we made all critical decision variables in the program, because of that fact it is not supposed to change at any time, constant to assure the security of the information flow. We also decentralized the game logic by delegating basic tasks such as input management. It did not only reduce the complexity of the "GameEngine" but also increase the validity of the code.

### 4.5.3 Global Software Control

In our project, we consider to use event-driven control flow mechanism. In event-driven mechanism, flow of the system is determined by events. There is an event detection part for detecting the changes which occurs in program while it is runing and a part for event handling which will decide what will be done when an event occurs. we decided to use Model-View-Controller, so event-driven mechanism will be the best choice for us because it has separated parts that are control, view and logic parts. Thus, when an event occurs, detection will be made by a part and handling will be made by another part. Furthermore, our project should have decentralized design because we plan to have more than one control object. Decentralized design stands for the distribution of dynamic behavior to objects. Our system will have physics engine because of the collisions and rules engine as control objects, so system should be decentralized to be able to spread responsibility.

### 4.5.4 Boundary Conditions

Game will return to the main menu if all the lives of the player are gone. In case of death, score of the player will be updated and showhed before returning main menu. The game has 5 number of levels and if the user achieves all five levels in the game,he/she will continue to play with maximum speed until he/she will die. If the user quits from the program when he/she returns again score will be set 0 we dont have database system for recording score.As you can see at the below, general boundary conditions have been seperated subtitles for clarifying.

**Initialization**

* User starts the game by opening game file.
* When the game file is opened,main menu shows up by the system.In the main menu there 5 different options which are “Start”, “Help”,”Difficulty”,”Color” and “Exit” .
* If user selects “Start” option, system orients the user to the game directly.
* If user selects “Help” option, pop-up information screen is shown.
* If user selects “Difficulty” option, he/she can set new difficulty for the new game.After setting new difficulty,user can choose “Start” option and than game will be started.

**Termination**

* User has a chance to exit program any time he/she wants, without returning the main menu
* The information about the system that are kept in memory is cleaned up so that memory

should be emptied and so the system could start from the beginning next time.

* If there is any high score in incomplete game, they can not be saved before termination. In this case,after re-opening the game, user will start the game from the zero.

# 5. Object Design

## 5.1 Design Patterns

### 5.1.1 Observer Pattern

Observer pattern is used for observing the changes in an object’s state. It doesn’t know anything except the changes in the state of objects it’s tracking. We use Observer pattern to track the coordinates of the Circle and shapes that falling from the top. If there is a change in one of these objects, Observer notifies the view to update its state. Also, Score and level will be updated and the Observer pattern gives us this opportunity.

### 5.1.2 Façade Pattern

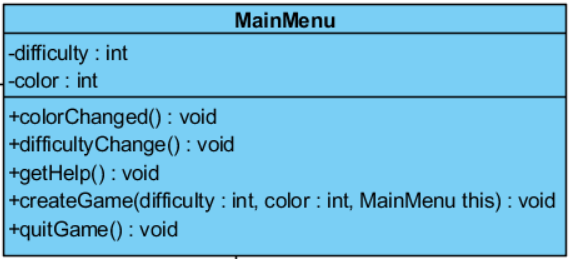
Façade pattern encapsulates a complex subsystem within a single interface object. Thus, it reducescomplexity by grouping common properties of classes in one interface. In GameEngine class of our project Run from Shapes, we use this pattern to simplify subsystem connections. We group all game objects such as Shape, Circle and Star in Game Entities subsystem in order to apply Game Engine class as Façade class.

### 5.1.3 Adapter Pattern

Adapter pattern works as a bridge between two incompatible interfaces. This type of design pattern comes under structural pattern as this pattern combines the capability of two independent interfaces. In our project, we applied this pattern on MouseAdapter and we constructed a bridge between two incompatible interfaces.

## 5.2 Class Interfaces

### 5.2.1 Main Menu Class

****

Main Menu(it’s a public class) is the first class that will be instantiated when the game is opened by user. In this menu, there are 5 buttons and those are “New Game”, “Difficulty”, “Color”, “Help”, and “Exit” buttons.

**-public void difficultyChange() :** This method will change difficulty of the game depends on user demands.

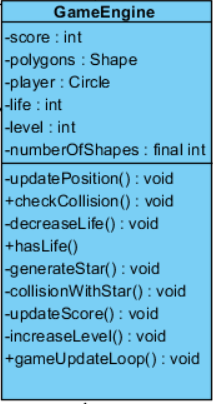
**-public void colorChange()** : This method will change the color of main charachter icon’s color.

**-public void getHelp()** : This method will be executed for the calling help panel.

**-public void createGame(difficulty: int, color: int, MainMenu this):** This method will create default game for user.

**-public void quitGame():** This method will be executed when user clicks “Exit” button from main menu and it stops execution of the game.

### 5.2.2. Game Engine Class



Game Engine Class(it’s a public class) is the façade class of the entire system; therefore it handles almost everything about game logic via using methods. Moreover, main loop of the game starts in Game Engine class. Game Engine class responsible for update several important in-game futures and those are increase the level of the game , check whether player has left any life or not, update score of the player,generate bonuses and collect those bonuses, terminate the game with click on default exit icon and lastly detect whether any falling down shapes collide with main charachter icon or not.

**-public void updatePosition()** : This method is responsible for update positions of all shapes and main charachter icon which is circle.

**-public void checkCollision()** : This method checks is there any collision between main charachter icon and falling down shapes.

**-public void decreaseLife()** : This method is used for decrease the life of player by one whenever a collision happens.

**-public boolean hasLife()** : This method is responsible for checking whether player has left any life or not.

**-public void generateStar()** : This method is used for generate random bonus and this randomly created bonus represented with a star shape. There are multiple bonuses in our game which are extra life,extra score and smaller charachter icon size and this method randomly create one of these bonuses.

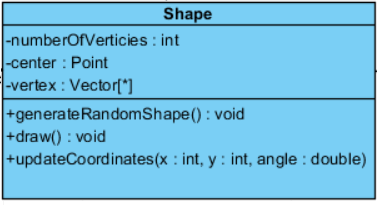
**-public void collisionWithStar()** : This method responsible for checking collision with star and main charachter icon. If there is a collision between star and main charachter icon, than it apply changes according to that specific bonus.

**-public void updateScore()** : This method updates score of the player according to player’s survival time.

**-public void increaseLevel()** : This method checks whether player passed threshold score for that level or not and if player passed threshold score than increase the level of the game by one.

-**public void gameUpdateLoop()** : This method runs main loop of the game and this loop ends whenever hasLive() method returns a false boolean expression and if player has left no life, than it terminates the main loop and game returns to the main menu.

### 5.2.3 Shape Class



Shape class(it’s a public class) is responsible for drawing any shapes(circle, star and convex polygons).This class holds vertices of shapes and their centers as instances in order to draw shapes.

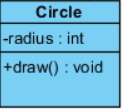
**-public void draw():** This method is responsible for drawing individual shapes.

**–public void generateRandomShape()** : This method responsible for generate different polygons by changing the number of vertices of a polygon.

**-public void updateCoordinates()** : This method responsible for update coordinates of polygons(falling down from the top of the platform).

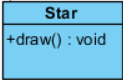
Also, Shape Class has setter and getter methods in order to change and get the center of the polygons and length of a vertices in a polygon. Those setter and getter methods helps us to create and update polygons.

### 5.2.4 Circle Class



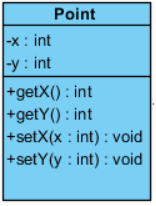
Circle Class(it’s a public class) is responsible for drawing circle which is a special shape for our game. Circle represents our main charachter icon in the game and Circle Class holds radius instances in order to draw the circle.

### 5.2.5 Star Class



Star Class(it’s a public class) is responsible for drawing star shape. It extends draw method from Shape class and by using draw() method, Shape Class able to draw star shape. Since we have multiple bonuses, we need to get random bonuses and we achieve this by using getTypeOfBonus() method. getTypeOfBonus() method generates a random bonus which might be additional score, additional life and smaller charachter icon and one of these bonuses are falling down from the top of the platform in random time with a star shape.

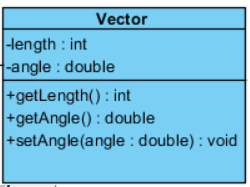
### 5.2.6 Point Class

****

Point Class(it’s a public class) is responsible for determine the coordinates of all shapes including stars and circle. Shape class include this Point class in order to draw polygons and other shapes. Point class has integer x and y instances and has getter and setter methods which makes possible to determine coordinates of a shape for us. Point Class create a point for our shapes and with

length, vertices and angle instances, we will be able to create any polygons.

### 5.2.7 Vector Class



Vector Class(it’s a public class) is responsible for determining whole edges of shapes by using mathematical sine and cosine functions. In order to draw polygons these edges are essentials and Shape class include Vector class for drawing all polygons.

**-public void getLength():** Thismethod gets magnitude of a vector.

**-public void getAngle()** : This method holds the angle between edges in a polygon.

Also, these methods will also used for checking collision ( checkCollision() ) method ,in the GameEngine Class.

### 5.2.8 ActionListener

This interface is necessary for button clicks in Main Panel and Game Panel.

****

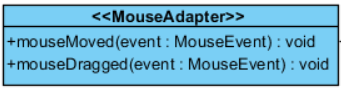
### 5.2.9 JPanel

This interface is necessary for making 2D graphical drawings.

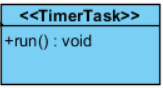
****

### 5.2.10 MouseAdapter

This is the only interaction between user and the game except display.

****

### 5.2.11 TimerTask

****

**-public void run() :** This method initiates draw() and generateRandomShape() methods from Shape class in order to create necessary objects.

## 5.3. Specifying Contracts Using OCL

1. context GameView:: setShapes (Circle : Shape, Bonus : Shape, Shapes : Shape[]) :

post : self.circle = Circle, self.Bonus = Bonus, self.Shapes = Shapes

2. context GameView

inv: difficulty <= 2 && difficulty >= 0

3. context GameView::setDifficulty( ) int

post : self.difficulty = (self.difficulty + 1 )%3

4. context GameView

inv: color <= 4 && color >= 0

5. context GameView::setColor( ) int

post : self.color = (self.color + 1 )%5

6. context GameView

inv : life >= 0

7. context GameView::decreaseLife() int

post : self.life = self.life - 1

8. context GameView::increaseLife() int

post : self.life = self.life + 1

9. context GameView

inv : score >= 0

10. context GameView::increaseScore() int

post : self.score = self.score + 1

11. context Circle

inv: x >= 0 && x <= 1000 && y >= 0 && y<=800

12. context Circle

inv: beginningX = 500 && beginningY = 600

13. context Circle

inv: radius = GameView.difficulty\*25 + 50

14. context Circle::setPoint(x:int, y:int)

post : self.x = x && self.y = y

15. context Circle::getX() : int

post : result = x

16. context Circle::setX(X : int)

post : self.x = x

17. context Circle::getY() : int

post : result = y

18. context Circle::setY(y : int)

post : self.y = y

19. context Circle::getRadious() : int

post : result = radius

20. context Circle::setRadius(radius : int)

post : self.radius = radius

21. context Shape

inv: length <= 90 && length >= 10

22. context Shape::getLength() : int

post : result = length

23. context Shape::setLength(length : int) : int

post : self.length = length

24. context Shape

inv: Vector -> Angle() <= 360 && Vector -> Angle() >= 0

25. context Vector::getAngle() : double

post : result = angle

26. context Vector::setAngle(angle : double) : double

post : self.angle = angle

27. context Shape

inv: Point -> X() <= 1000 && Point -> X() >= 0 && Point -> Y() <= 800 && Point -> Y() >= 0

28. context Point::getX() : int

post : result = x

29. context Point::setX(x : int)

post : self.x = x

30. context Point::getY() : int

post : result = y

31. context Point::setY(y : int)

post : self.y = y

# 6.Conclusion

Main purpose of all reports,analysis,design and final, is trying to explain our project step by step, what it does, functionalities that project has, design of the project and some further details about the implementation of the project. Game features, functional and non-functional requirements had helphed us for creating use cases and their scenarios. The analysis report was like a draft plan in order to constitute basics of the design part of the game . The scenarios and use cases in this report led us to pass to the next step which is design process. In design report, we used what we discussed in analysis and revised some parts according to feedbacks that we have received. In design report, we saw more concrete image of the project. By considering further about project in design report, we saw some of our mistakes and corrected them.

In design part, we have tried to show subsystem of our game in order to make it more clear and understandable . Moreover,design report inclueded more techincal details about the project than the analysis part. Also, in design report we show all relationships between subsystems. In final report , we analyzed details of all classes by looking their attributes and their methods. In addition to that, we explained all methods in detail. Also we determined the patterns that we have used, those pattern helped to coordinates each classes and their relationships with each others.

To sum up, this whole report part helped us to understand how to deal with a complex object oriented projects with using UML. We learn to interact with different classes and different subsystems in this projects and by the help of UML patterns, we can manage complexity and hierarchy in this project. Also, by analysis and design stage of the project we become familiar with abstraction in a project and we learn to manage this abstraction in analysis and design stage. Run from shapes, is a project that we associate theoretical knowledge with practical knowledge and by composition of theoretical and practical knowledge, we conclude our game.

# 7. Implementation

## 7.1 MainMenu Class

|  |
| --- |
| class MainMenu extends JPanel{  private JFrame frame;  private MainMenu menuPanel;    private JButton newGameButton;  private JButton difficultyButton;  private JButton colorButton;  private JButton helpButton;  private JButton exitButton;    private JButton clickedButton;    private int difficulty;  private String[] difficulties;  private int color;  private String[] colors;    public MainMenu( JFrame frame)  {  this.frame = frame;  menuPanel = this;    setPreferredSize(new Dimension(1000,800));    newGameButton = new JButton("New Game");  difficulty = 1;  difficultyButton=new JButton("Difficulty : Normal");  color = 1;  colorButton=new JButton("Color : Yellow");  helpButton=new JButton("Help !");  exitButton=new JButton("Exit");    colors = new String[]{"Red", "Yellow", "Green", "Blue", "White"} ;  difficulties = new String[]{"Easy", "Medium", "Hard"} ;    JPanel buttonPanel = new JPanel();    JPanel empty1 = new JPanel();  JPanel empty2 = new JPanel();  JPanel empty3 = new JPanel();  JPanel empty4 = new JPanel();  JPanel empty5 = new JPanel();  JPanel empty6 = new JPanel();    buttonPanel.setLayout(new GridLayout(9,0));  buttonPanel.add(empty1);  buttonPanel.add(empty2);  buttonPanel.add(newGameButton);  buttonPanel.add(difficultyButton);  buttonPanel.add(colorButton);  buttonPanel.add(helpButton);  buttonPanel.add(exitButton);  buttonPanel.add(empty3);  buttonPanel.add(empty4);    setLayout(new GridLayout(0,3));  add(empty5);  add(buttonPanel);  add(empty6);      newGameButton.addActionListener(new ButtonListener());  difficultyButton.addActionListener(new ButtonListener());  colorButton.addActionListener(new ButtonListener());  helpButton.addActionListener(new ButtonListener());  exitButton.addActionListener(new ButtonListener());  }    public class ButtonListener implements ActionListener  {  public void actionPerformed(ActionEvent e)  {  clickedButton = (JButton)e.getSource();    if ( clickedButton == exitButton )  {  System.exit(0);  }  if ( clickedButton == helpButton )  {  JOptionPane.showMessageDialog(frame, "oyun oynanmaz", "naber", JOptionPane.WARNING\_MESSAGE);  return;  }  if ( clickedButton == colorButton )  {  color = (color + 1)%5;  colorButton.setText( "Color : " + colors[color]);  return;  }  if ( clickedButton == difficultyButton )  {  difficulty = (difficulty + 1)%3;  difficultyButton.setText( "Difficulty : " + difficulties[difficulty]);  return;  }  if ( clickedButton == newGameButton )  {  frame.setContentPane( new GameEngine( frame, menuPanel, color, difficulty));  frame.revalidate();  return;  }  }  }    public void paintComponent(Graphics g)  {  super.paintComponent(g);  }  } |

## 7.2 GameEngine Class

|  |
| --- |
| class GameEngine extends JPanel{  private JFrame frame;  private MainMenu menuPanel;  private Shapes allShapes;  private Point mouseLocation;  private Timer timer;  private Image background;  private int backgroundY;    public GameEngine( JFrame frame, MainMenu menuPanel, int color, int difficulty)  {  this.frame = frame;  this.menuPanel = menuPanel;    setPreferredSize(new Dimension(1000,800));  setBackground(Color.white);    allShapes = new Shapes( color, difficulty);    JButton button=new JButton("Quit Game");    setLayout(new BorderLayout());  add(button,BorderLayout.SOUTH);  button.addActionListener(new ButtonListener());  addMouseMotionListener(new mouseListener());    mouseLocation = new Point( 500, 600);    timer = new Timer();  timer.schedule(new timerTask(), 200, 40);      background = Toolkit.getDefaultToolkit().getImage("..\\space.png");  backgroundY = -1024;  }    public class ButtonListener implements ActionListener  {  public void actionPerformed(ActionEvent e)  {  timer.cancel();  frame.setContentPane( menuPanel);  frame.revalidate();  return;  }  }    public class mouseListener extends MouseAdapter  {  public void mouseMoved(MouseEvent e)  {  int x = e.getX();  int y = e.getY();  if ( x<100)  x=100;  if ( x>899)  x=899;  if ( y<100)  y=100;  if ( y>699)  y=699;    mouseLocation.setX( x);  mouseLocation.setY( y);  }  public void mouseDragged(MouseEvent e)  {  int x = e.getX();  int y = e.getY();  if ( x<100)  x=100;  if ( x>899)  x=899;  if ( y<100)  y=100;  if ( y>699)  y=699;    mouseLocation.setX( x);  mouseLocation.setY( y);  }  }    public void paintComponent(Graphics g)  {  super.paintComponent(g);    Graphics2D g2 = (Graphics2D) g;  g2.drawImage( background, 0, backgroundY, this);  backgroundY += 5;  if ( backgroundY > 0)  backgroundY -= 1024;  g2.finalize();  allShapes.drawAll(g);  /\*  g.setColor(Color.black);  g.fillOval( x-25,y-25, 50,50);\*/  }    class timerTask extends TimerTask {  public void run() {  allShapes.updateAll( mouseLocation);  repaint();  }  }  }  class Shapes  {  private final int maxShapes = 20;  private Shape []shapes;  private Circle player;  private Star bonus;  private Color []colors;    public Shapes( int color, int difficulty)  {  colors = new Color[]{ Color.red, Color.yellow, Color.green, Color.blue, Color.white} ;    shapes = new Shape[maxShapes];    for ( int i = 0 ; i < maxShapes ; i ++ )  shapes[i] = new Shape( i\*-100);    player = new Circle( 50 + difficulty \* 25, colors[color], new Point( 500, 600) );  bonus = new Star();  }    public void drawAll(Graphics g)  {  for ( int i = 0 ; i < maxShapes ; i ++ )  shapes[i].draw(g);    player.draw(g);    bonus.draw(g);  }    public void updateAll( Point mouseLocation)  {  for ( int i = 0 ; i < maxShapes ; i ++ )  {  shapes[i].update();  if ( shapes[i].getCenter().getY() > 900 )  shapes[i] = new Shape( shapes[i].getCenter().getY() - maxShapes\*100);  }    player.update( mouseLocation);    bonus.update();    if ( bonus.getCenter().getY() > 900 )  bonus = new Star( );  }  } |

## 7.3 Shape Class

|  |
| --- |
| class Shape  {  private Point center;  private Point speed;  private double rotatingSpeed;  private Vector[] vertices;  private int length, maxVertices;    public Shape( int y)  {  maxVertices = (int)( (Math.random()) \* 6) + 3;  length = (int)( (Math.random()) \* 80) + 10;    vertices = new Vector[maxVertices];    for ( int i = 0 ; i < maxVertices ; i ++)  vertices[i] = new Vector( length, (double)( (360.0\*i) / maxVertices) );    center = new Point( (int)( (Math.random()) \* 800) + 100, y );    speed = new Point( (int)( (Math.random()) \* 11) - 5, -5);    rotatingSpeed = (Math.random()\*40) - 20 ;  }    public Shape( )  {  maxVertices = 10;  speed = new Point( 0, -15);  rotatingSpeed = (Math.random()\*60) - 30 ;  center = new Point( (int)( (Math.random()) \* 800) + 100, ((int)( (Math.random()) \* 3000) + 200) \* -10);    vertices = new Vector[maxVertices];    for ( int i = 0 ; i < maxVertices ; i ++)  vertices[i] = new Vector( (i&1) == 0 ? 50:20 , (double)( (360.0\*i) / maxVertices) );  }    public void setCenter( Point center)  {  this.center = center;  }    public Point getCenter()  {  return center;  }    public void update()  {  for ( int i = 0 ; i < maxVertices ; i ++)  vertices[i].updateAngle( rotatingSpeed);  if ( center.getY() + length > 0 )  center.setX( center.getX() - speed.getX());  center.setY( center.getY() - speed.getY());  }    public int getLength()  {  return length;  }    public void draw( Graphics g1)  {  if ( (center.getY() + length) <= 0 )  return;  Graphics2D g = (Graphics2D) g1;  int x = center.getX(), y = center.getY();    g.setStroke( new BasicStroke(5));  g.setColor( Color.white);    for ( int i = 0 ; i + 1 < maxVertices ; i ++)  g.drawLine( x + (int)(vertices[i].getLength()\*Math.cos(vertices[i].getAngle()\*Math.PI/180.0)),  y + (int)(vertices[i].getLength()\*Math.sin(vertices[i].getAngle()\*Math.PI/180.0)),  x + (int)(vertices[i+1].getLength()\*Math.cos(vertices[i+1].getAngle()\*Math.PI/180.0)),  y + (int)(vertices[i+1].getLength()\*Math.sin(vertices[i+1].getAngle()\*Math.PI/180.0)) );    g.drawLine( x + (int)(vertices[maxVertices-1].getLength()\*Math.cos(vertices[maxVertices-1].getAngle()\*Math.PI/180.0)),  y + (int)(vertices[maxVertices-1].getLength()\*Math.sin(vertices[maxVertices-1].getAngle()\*Math.PI/180.0)),  x + (int)(vertices[0].getLength()\*Math.cos(vertices[0].getAngle()\*Math.PI/180.0)),  y + (int)(vertices[0].getLength()\*Math.sin(vertices[0].getAngle()\*Math.PI/180.0)) );  }  } |

## 7.4 Circle Class

|  |
| --- |
| class Circle extends Shape  {  private int radius;  private Color color;    public Circle( int radius, Color color, Point center)  {  this.radius = radius;  this.color = color;    super.setCenter(center);  }    public void draw( Graphics g)  {  g.setColor( color);  g.fillOval( super.getCenter().getX()-radius, super.getCenter().getY()-radius, radius<<1, radius<<1);  }    public void update( Point center)  {  super.setCenter(center);  }  } |

## 7.5 Star Class

|  |
| --- |
| class Star extends Shape  {  public Star( )  {  super ( );  }  } |

## 7.6 Vector Class

|  |
| --- |
| class Vector  {  private int length;  private double angle;    public Vector( int length, double angle)  {  this.length = length;  this.angle = angle;  }    public int getLength()  {  return length;  }  public double getAngle()  {  return angle;  }    public void updateAngle( double rotatingSpeed)  {  angle = ( angle + rotatingSpeed) % 360;  }  } |

## 7.7 Point Class

|  |
| --- |
| class Point  {  private int x,y;    public Point( int x, int y)  {  this.x = x;  this.y = y;  }    public void setX( int x)  {  this.x = x;  }  public void setY( int y)  {  this.y = y;  }    public int getX()  {  return x;  }  public int getY()  {  return y;  }  } |

# 8. References

[1] <https://www.gliffy.com/go/> . 14.October.15

[2] <https://creately.com/>. 21.October.15