Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Version** | **Description** | **Author** |
| 22.03.2014 | 1.11 | Initial version of SDD | İsmetcan Hergünşen, Mehmet Kağan Kayaalp, Nazlı Karalar, Gamze Küçükçolak, Erdi Koç |
| 23.03.2014 | 1.12 | Introduction (References, abbreviations etc.) | Nazlı Karalar, Kağan Kayaalp |
| 24.03.2014 | 1.13 | Component 1 | Gamze Küçükçolak, Erdi Koç |
| 24.03.2014 | 1.13 | Component 2 | Mehmet Kağan Kayaalp, Nazlı Karalar, |
| 24.03.2014 | 1.13 | Component 3 | İsmetcan Hergünşen |
| 26.03.2014 | 1.15 | Review (Introduction) | İsmetcan Hergünşen, Mehmet Kağan Kayaalp, Nazlı Karalar, Gamze Küçükçolak, Erdi Koç |
| 27.03.2014 | 1.16 | Review (Component 1) | Gamze Küçükçolak, Erdi Koç |
| 28.03.2014 | 1.17 | Review (Component 2) | Nazlı Karalar, Kağan Kayaalp |
| 29.03.2014 | 1.18 | Review (Component 3) | İsmetcan Hergünşen |
| 30.03.2014 | 1.19 | Review (Components) | Nazlı Karalar, Gamze Küçükçolak, Erdi Koç |
| 31.03.2014 | 1.20 | Review | İsmetcan Hergünşen, Mehmet Kağan Kayaalp |
| 01.04.2014 | 1.21 | Review | İsmetcan Hergünşen, Mehmet Kağan Kayaalp, Nazlı Karalar, Gamze Küçükçolak, Erdi Koç |

**TABLE OF CONTENTS**

Revision History 1

1 Introduction 3

1.1 References 3

1.1.1 Project References 3

1.1.2 Abbreviations 3

2 Software Architecture overview 4

2.1 Game Screen – Game Logic – Game Controller Model 4

3 Software design description 4

3.1 Component 1 – Game Logic 5

3.1.1 Component interfaces 5

3.1.2 Component design description 5

3.1.3 Workflows and algorithms 8

3.1.4 Software requirements mapping 8

3.2 Component 2 – Game Screen 8

3.2.1 Component interfaces 8

3.2.2 Component design description 8

3.2.3 Workflows and algorithms 9

3.2.4 Software requirements mapping 11

3.3 Component 3 – Game Controller 11

3.3.1 Component interfaces 11

3.3.2 Component design description 11

3.3.3 Workflows and algorithms 12

3.3.4 Software requirements mapping 14

# Introduction

SDD is a representation of the software system design such as software components and interfaces. It also shows how the software system will be structured to satisfy the requirements. The purpose of SDD is to define the detailed design for all components of CCB software system.

The SDD document is organized as follows. In the first part of SDD, it describes the project references and abbreviations and in component part, it shows the design of the system with UML diagrams.

## References

### Project References

| **#** | **Document Identifier** | **Document Title** |
| --- | --- | --- |
| DOC#SRS V1.0 | [1] | Software Requirement Specification |
| DOC#STP  V1.0 | [2] | Software Test Plan |

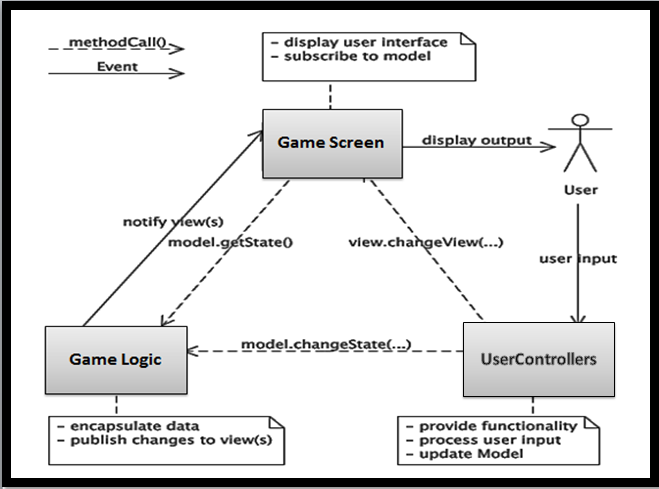
### Abbreviations

|  |  |
| --- | --- |
| **Term** | **Description** |
| CCB Project | Crazy Copter Battle Project Game |
| DOC #SDDv.1.0.x | Document version 1.0.x |
| JDA | Java Desktop Application |
| MVC | Model View Controller |
| UI | User Interface |
| GUI | Graphical User Interface |
| SDP | Software Development Plan |
| SRS | Software Requirements Document |
| UML | Unified Modeling Language |
| STP | Software Test Plan |
| JDK | Java Development Kit |
| SRS-CCB-XXX.X | Software Requirement Specification – CCB – XXX.X |
| STR | Software Test Report |

# Software Architecture overview

## Game Screen – Game Logic – Game Controller Model

This diagram shows basic operations of Game Screen – Game Logic – Game Controller model that will be used in CCB software system.



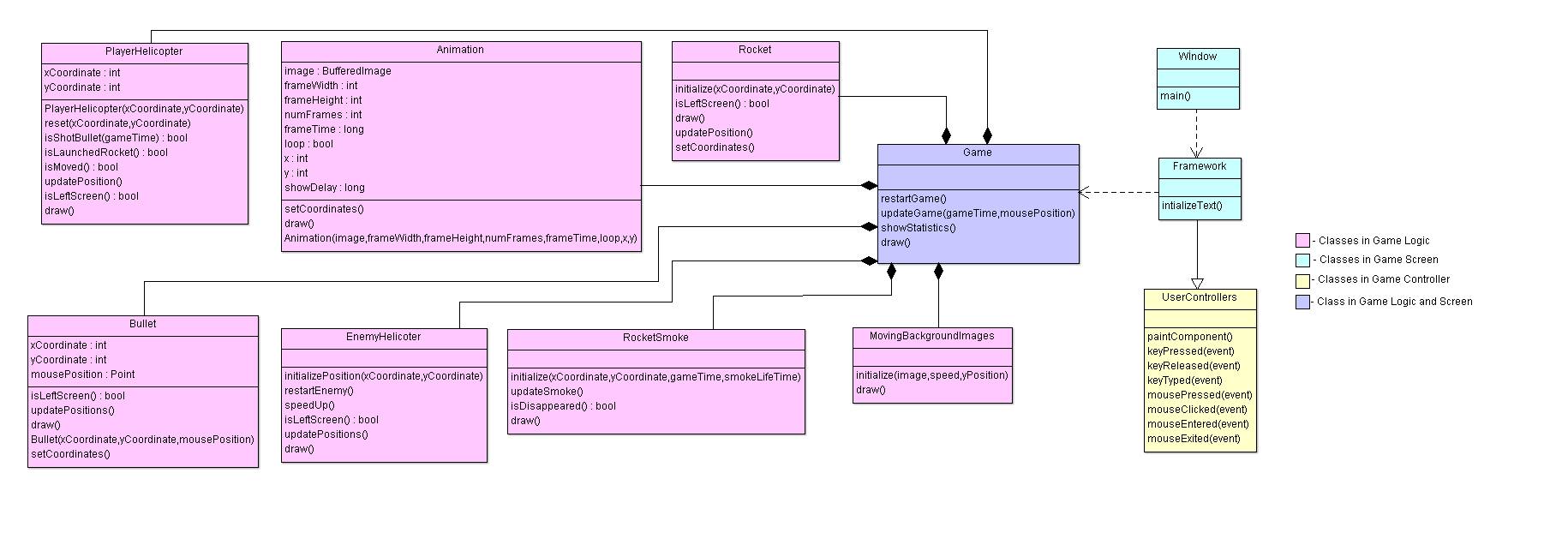
# Software design description

CCB software system will have one package and in content of this package, there will be 11 classes whose operations are divided depending on MVC model. Therefore, the components of CCB are Game Logic, Game Screen and Game Controller.

Game Logic notifies association between views and controllers when there is a change in its state. This notification allows views to update their presentation, and controllers to change the available set of instructions/commands (like pause).

Game Screen is told by the Game Controller all the information it needs for generating an output representation to the user. It will also provide generic mechanisms to inform the controller of the user input.

Game Controller is the link between a user and the system. It sends commands to the Game Logic to update the model’s state. Also, Game controller receives such user output, translates it into messages and passes them on to Game Screen.



***(Game Screen – Game Logic – Game Controller Detailed Class Diagram)***

## Component 1 – Game Logic

### Component interfaces

Input: Gets and changes state

Output: Notifies view(s)

### Component design description

Class: Animation

Constructor

-Usage: Initializes number of frames and set time between frames

Methods

* **public** **void** setCoordinates()
* **public** **void** draw()

Class: MovingBackgroundImages

Methods

* **public** **void** initialize()
* **public** **void** draw()

Class: Game

Constructor

-Usage: Initializes bullets, rockets, enemy helicopters etc.

Methods

* **public** **void** restartGame()
* **public** **void** updateGame()
* **public** **void** draw()
* **public** **void** showStatistics()

Class: PlayerHelicopter

Constructor

-Usage: Loads parts of the helicopter images and creates animation objects

Methods

* **public** **void** reset()
* **public** **boolean** isShotBullet ()
* **public** **boolean** isLaunchedRocket()
* **public** **boolean** isMoved()
* **public** **void** updatePosition()
* **public** **void** draw()
* **public boolean** isLeftScreen()

Class: EnemyHelicopter

Methods

* **public** **void** initializePosition()
* **public boolean** isLeftScreen()
* **public** **void** restartEnemy()
* **public** **void** speedUp()
* **public** **void** draw()
* **public** **void** updatePosition()

Class: Bullet

Constructor

-Usage: Initializes bullet position and speed.

Methods

* **public boolean** isLeftScreen()
* **public** **void** draw()
* **public** **void** updatePosition()
* **public void** setCoordinates()

Class: Rocket

Constructor

-Usage: Initializes rocket position and its speed.

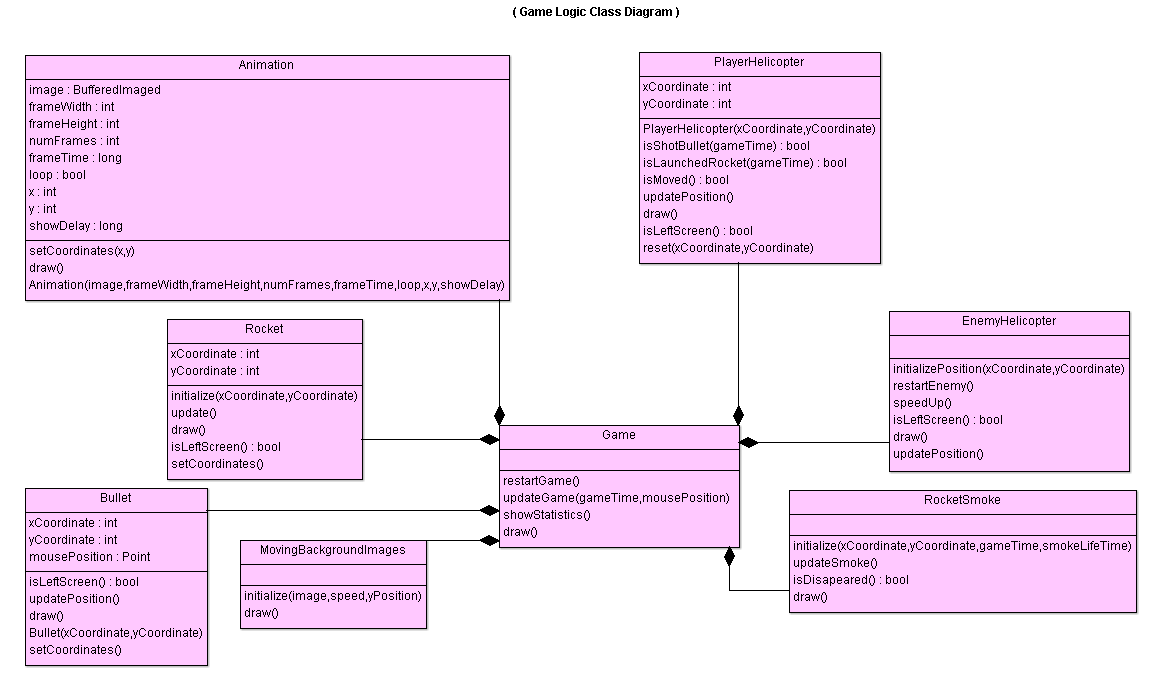
Methods

* **public boolean** isLeftScreen()
* **public** **void** draw()
* **public** **void** updatePosition()
* **public void** setCoordinates()

Class: RocketSmoke

Methods

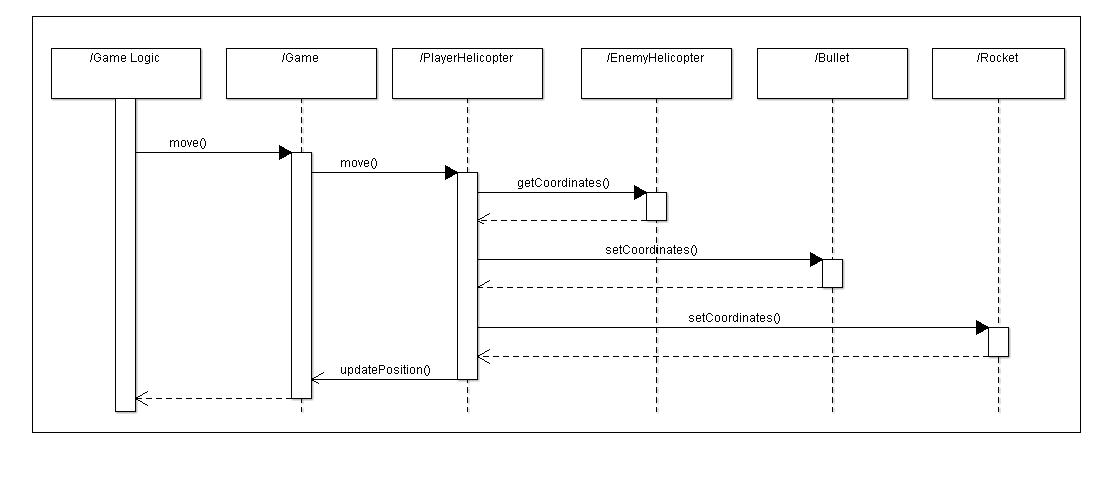
* **public** **void** initialize()
* **public boolean** isDisappeared()
* **public** **void** draw()
* **public** **void** updateSmoke()



***(Class Diagram for Game Logic)***

### Workflows and algorithms

* *Sequence Diagram - 1:* Game logic consists of 5 different classes whose names are Game, PlayerHelicopter, EnemyHelicopter, Bullet and Rocket classes. In Game class, rocket, bullet, player helicopter and enemy helicopter objects are created and are drawn. Also, game class is responsible for resetting enemy helicopters and limited number of bullets and rockets. That’s the reason why Game class should call PlayerHelicopter, EnemyHelicopter, Bullet and Rocket classes in order to create the objects. At each moves of player helicopter, it gives information of updates of its position to Game class. Similarly, PlayerHelicopter class inform EnemyHelicopter about player helicopter’s coordinates. Because of that, enemy helicopters are created with respect to player helicopter’s current coordinate. Identically, Bullet and Rocket classes need to have coordinate of player helicopter as well because when bullet and rocket objects are created they need to use player helicopter’s present coordinate.



***(Sequence Diagram – 1)***

### Software requirements mapping

* SRS-CCB-003.1
* SRS-CCB-004.3
* SRS-CCB-007.1

## Component 2 – Game Screen

### Component interfaces

Input: Notifies and changes view(s)

Output: Displays output

### Component design description

Class: Window

Constructor

-Usage: Creates a new instance of this class

Methods

* Main

**public** **static** **void** main(String[] args)

Class: Framework

Constructor

-Usage: Initializes framework, background images etc.

Methods

* **public** **void** initializeText()
* **public** **void** keyReleased(KeyEvent event)
* **public** **void** mouseClicked(MouseEvent event)

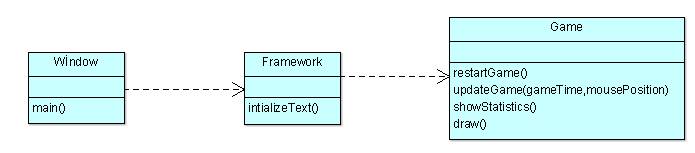
Class: Game

Constructor

-Usage: Initializes all objects of the game such as bullets, helicopter, and create content of the game.

Methods

* **public** **void** restartGame()
* **public** **void** updateGame()
* **public** **void** draw()
* **public** **void** showStatistics()



***(Class diagram for Game Screen)***

### Workflows and algorithms

*Sequence Diagrams for Game Screen:*

* *Sequence Diagram–1:* When user starts the game, Window class which includes the main method of the game will call Framework class and then Framework will create splash screen texts, images. In the main method, window class will be called in order to create the menu size, image, text etc.

**

* *Sequence Diagram-2:* Framework class will create Game object in its one of the method, createGame, and methods of Game class can be reached by Framework.



* *Sequence Diagram-3:* When the user restarts the game, player helicopter will be created and objects such as enemy helicopters, bullets, rockets etc. will be cleared.

**

### Software requirements mapping

* SRS-CCB-001.1
* SRS-CCB-001.2
* SRS-CCB-002.1
* SRS-CCB-004.1
* SRS-CCB-004.2
* SRS-CCB-009.1
* SRS-CCB-009.2

## Component 3 – Game Controller

Game controller component includes all of the commands which user uses to play the game. This part acts on both Game Logic and Game Screen. It maps user actions to Game Logic. Therefore, it controls the data flow and updates the Game Screen part whenever data changes depending on what user does. Also, game controller checks some events whether it happens correctly.

### Component interfaces

Input: Gets user input(s)

Output: Changes state(s) and view(s)

### Component design description

Class: UserControllers

Constructor

-Usage: Creates a mouse cursor and calls key and mouse listeners.

Methods

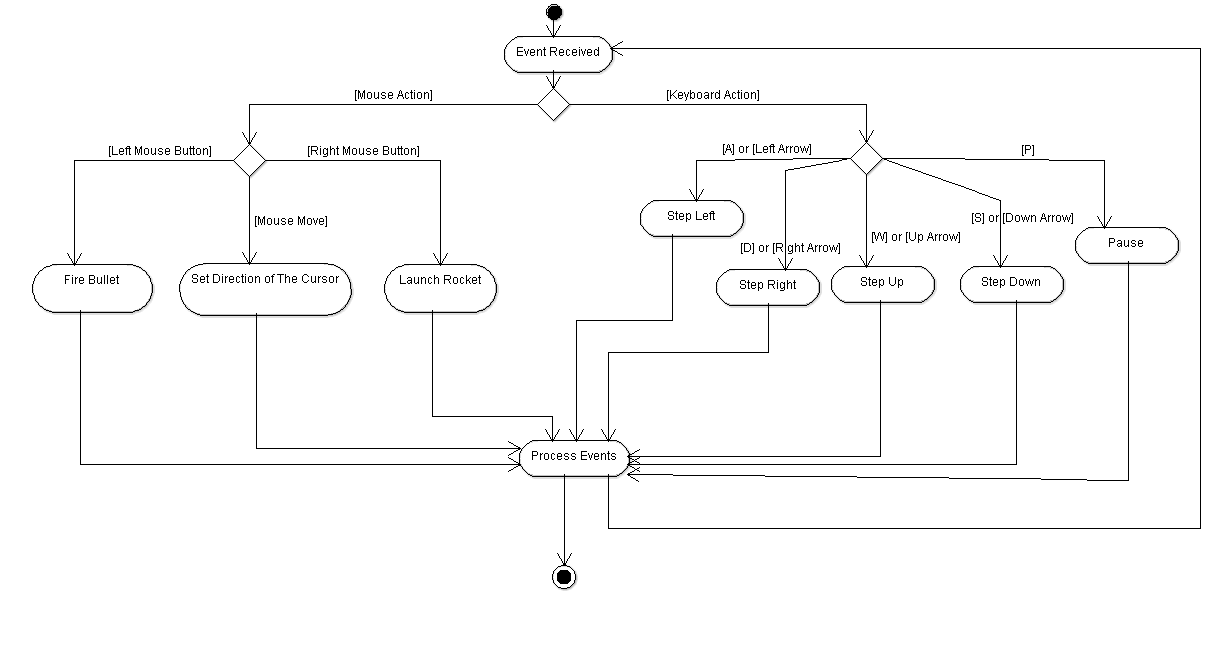
* **public** **void** paintComponent()
* **public** **void** keyPressed(KeyEvent event)
* **public** **void** keyReleased(KeyEvent event)
* **public** **void** keyTyped(KeyEvent event)
* **public** **void** mousePressed(MouseEvent event)
* **public** **void** mouseRelased(MouseEvent event)
* **public** **void** mouseClicked(MouseEvent event)
* **public** **void** mouseEntered(MouseEvent event)
* **public** **void** mouseExited(MouseEvent event)



### Workflows and algorithms

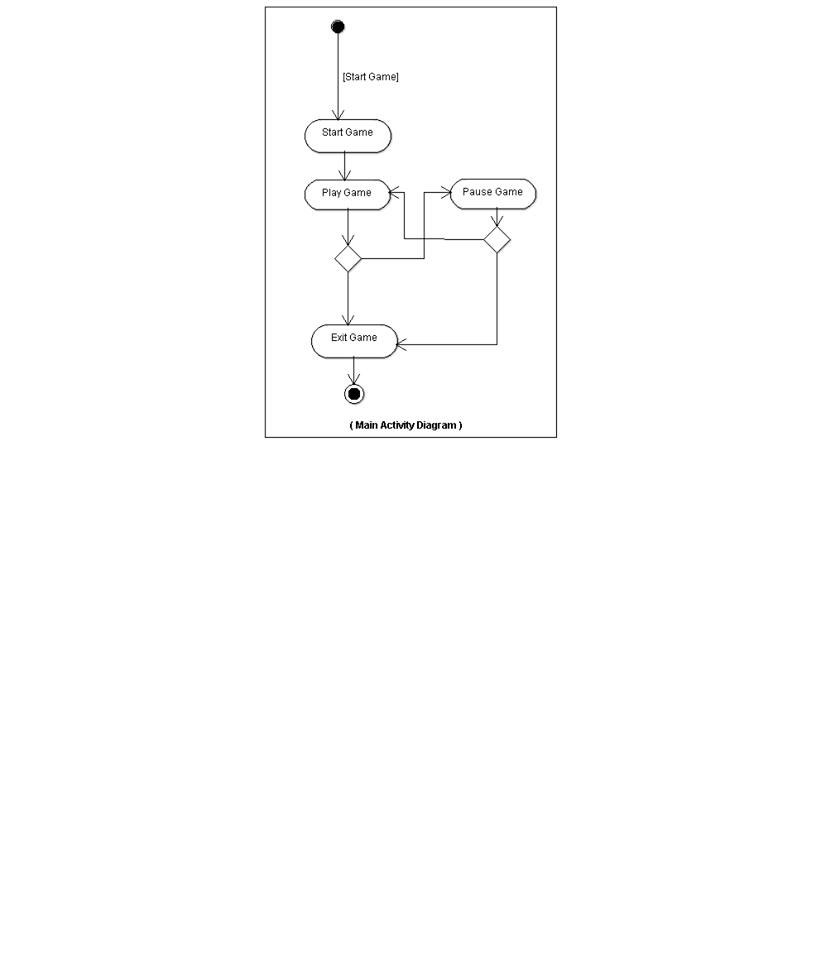
*Activity Diagram – 1:* When game starts, the user will play the game both mouse and keyboard. S/he can set direction of the cursor by moving mouse up and down. Also, mouse left and right clicks will be functional for the game. When user holds on mouse button either left or right, player helicopter will fire bullets or bombs repetitiously concerning which button the user presses. Pressing one left click enables player helicopter to fire one bullet. Similarly, by pressing one right clicks, player helicopter launches a rocket. When the user presses both left and right at the same time, bullets and rockets will be fired together.

On the other hand, the player helicopter moves with respect to what the user presses on the keyboard. Pressing either A or left arrow makes the player helicopter to go left while pressing either D or right arrow makes it to go right. Identically, pressing either W or up arrow enables it to go up while pressing either S or down arrow enables it to go down. In addition to these, pressing P button pause the game.



***(Activity Diagram – 1)***

* *Activity Diagram – 2:* After pressing the start game button, user of the game starts to play the game. While the user starts or plays the game, s/he can pause it by pressing the correct key and after that at his/her pleasure, the user can exit the game. In this way, the game is over. Besides, while playing the game, the user can pause it and the player can resume the game.



***(Activity Diagram – 2)***

### Software requirements mapping

* SRS-CCB-001.3
* SRS-CCB-001.4
* SRS-CCB-005.1
* SRS-CCB-006.1
* SRS-CCB-006.2
* SRS-CCB-008.1