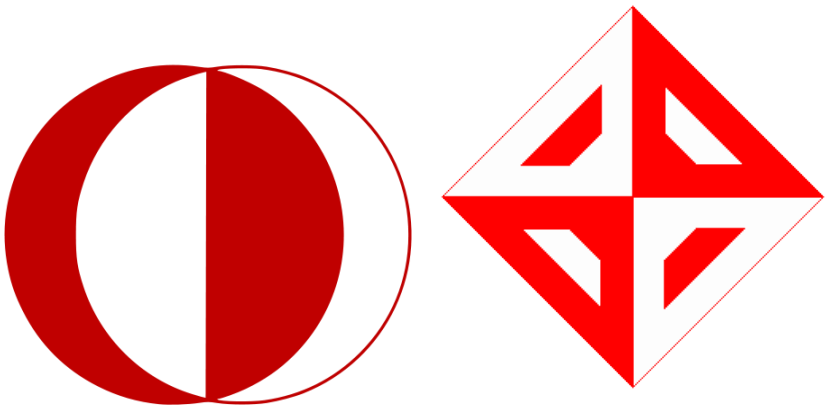

DynaDraw

Software Design Description

(In accordance with IEEE 1016-2009)

v1.0

By DynaDraw

Bünyamin SARIGÜL	**1881440**
Barış ÖZÇELİK	**1881804**
Turan SOYUER	**1881481**
Cemil Kocaman	**1881333**

June 3, 2016

Preface

This document contains the software design description for the “ DynaDraw” project. The document is prepared according to the “IEEE Standard for Information Technology – Systems Design – Software Design Descriptions – IEEE, 1016 – 2009”.

In this Software Design Description Document, a complete description of all the system design and views of the “DynaDraw” project can be found.

While the first section of this document includes Project Identification, the other sections gives information about document purpose and design viewpoints of the system.

Table of Contents

List of Figures

1. Introduction

This document is a Software Design Description for “DynaDraw ”. Features, functionalities, specifications and explanations about the project is included in this document. This project is a design project for the Computer Engineering Design Course of Computer Engineering, Middle East Technical University.

1.1. Scope

This document gives information about the structural overview of all modules, interfaces and data in order to support design and development process. During the implementation, this document serves as a guide for developers.

1.2. Purpose

This document is prepared to describe the basic architecture of DynaDraw. Also, this document aims to identify the software system which meets the requirements of the System Requirements Specification document.

1.3. Intended Audience

The intended audience for this document is the development team of the project. This document should be used as a guide for implementation by the team.

2. System Overview

In this part, the information about the system overview of DynaDraw can be found. The deployment diagram of DynaDraw is in the next page:

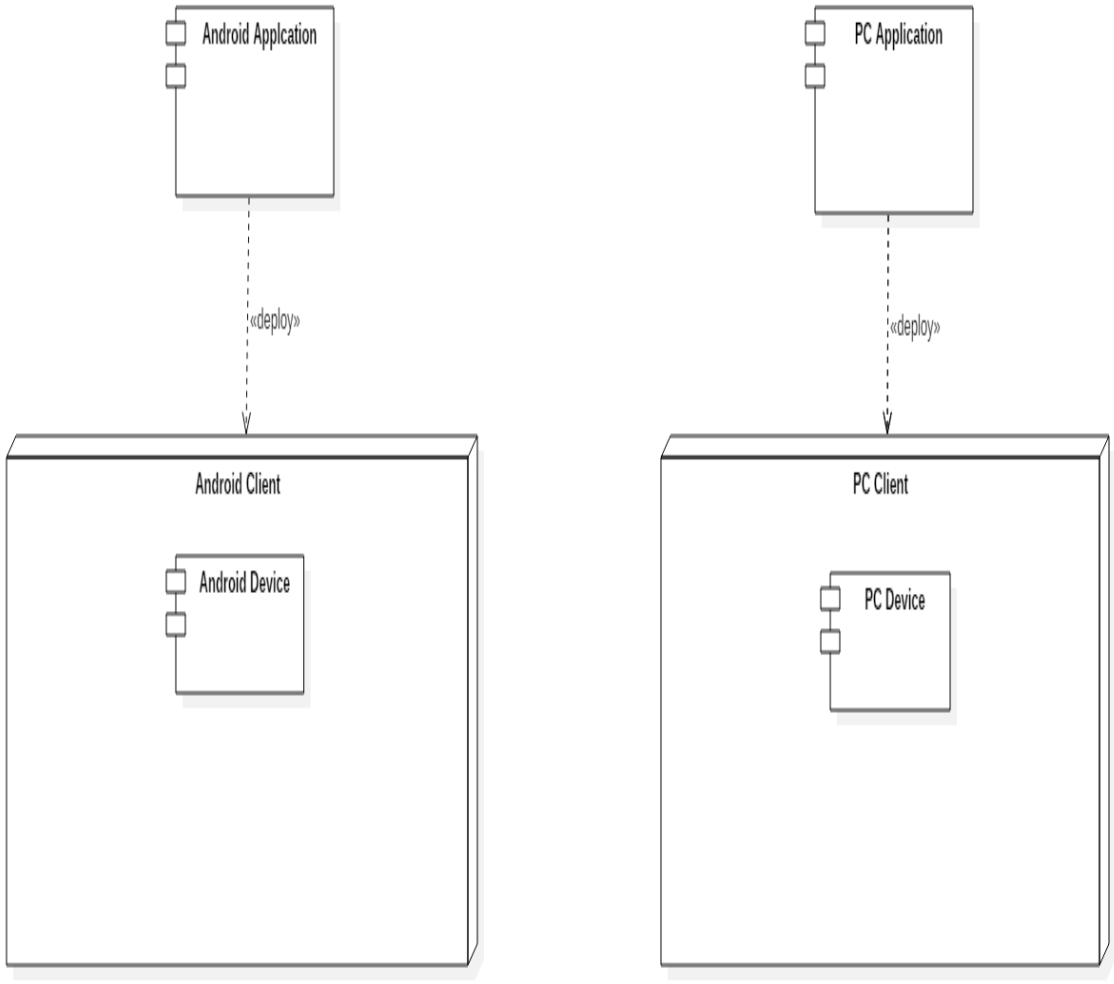

Figure 1: Deployment Diagram

Android Client: DynaDraw application will be installed into an Android Device, and users will be able to use DynaDraw via an Android device.

PC Client: DynaDraw will also run with a PC , and users will be able to use DynaDraw via PC.

3. System Architecture

In this part, the information about the architectural view of DynaDraw can be found. The component diagram of DynaDraw is below:

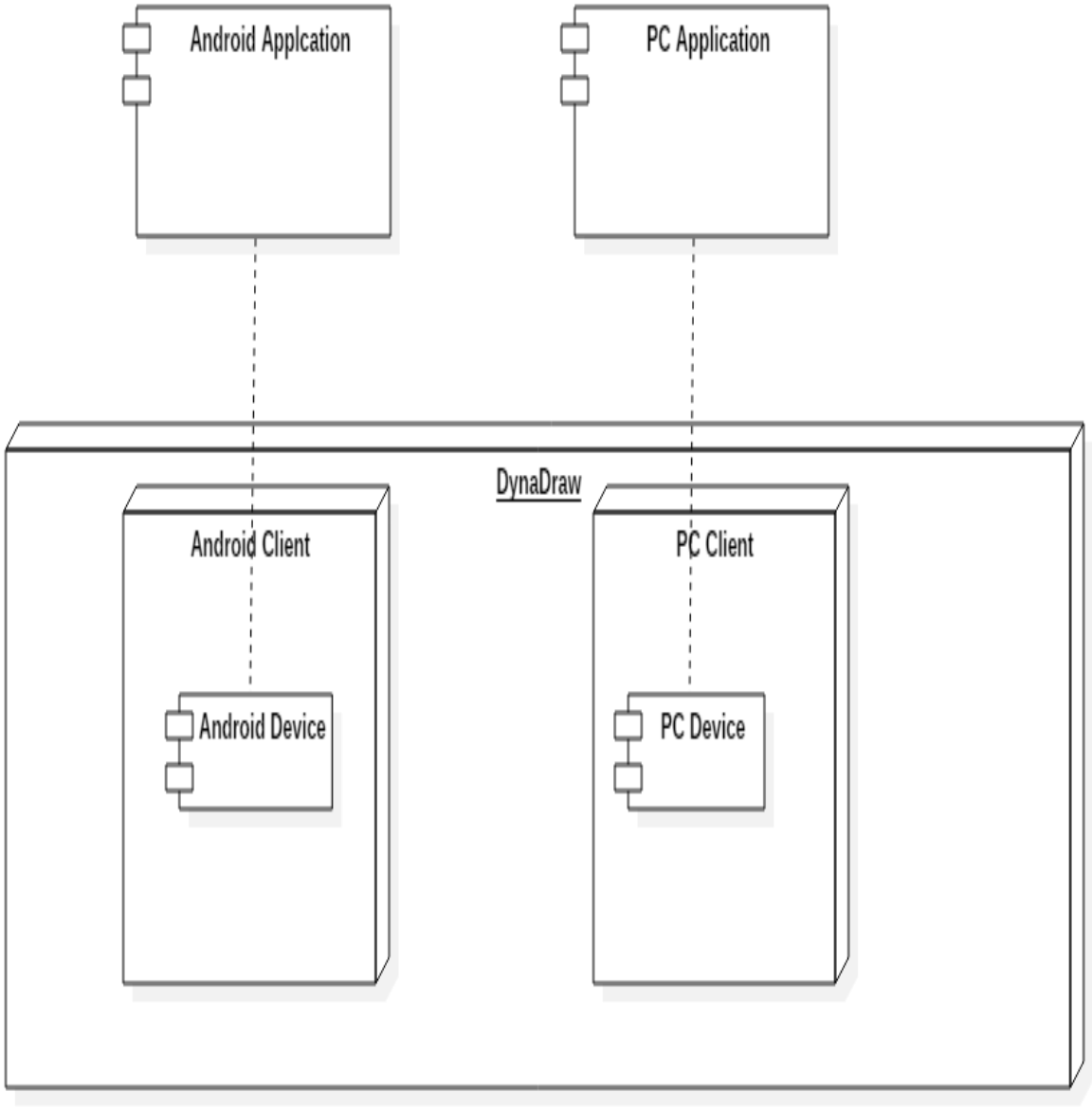

Figure 2: Component Diagram

DynaDraw runs both on Android devices and PC as mentioned in section 3.

4. Design Description Information Content

4.1. SDD Identification

Design specification included in this document will be used in architectural design and system implementation. This SDD report is in accordance with IEEE 1016-2009 standards. Diagram drawing tool is StarUML.

4.2. Design Stakeholders and Their Concerns

Design stakeholders are the development team of this project, supervisor of the development team(Serdar ÇİFTÇİ) and advisor of the development team(Sibel TARI) . Stakeholder concerns are:

- The system should work without any errors.

- Implementation should have maintainability and reusability.

- User interface should be easy to use and attractive since children will use DynaDraw.

4.3. Design Viewpoints

Different viewpoints of DynaDraw will be explained with related diagrams in the following sections

4.4. Design Elements

- The application will be implemented in C#.

- Unity will be used as game engine.

- Visual Studio will be used as IDE.

4.5. Design Raionale

Delivering performance, reliability and robustness is the main concern of the development team while making design decisions.

4.6. Design Languages

Unified Modeling Language (UML) is used for the design viewpoint specification.

5. Design Viewpoints

In this section, viewpoints of the DynaDraw are clearly explained with related UML diagrams.

5.1. Context Viewpoint

There will be only one kind of user type in DynaDraw system.

User can do the followings:

- Create Character: User can create a character by drawing.

- Start New Game: User can start a new game after created a new character.

- Restart Game: User can restart the game scene with the same character

- Add New Object By Drawing: User can add game objects to the game scene by drawing.

- Pick Food: User can pick foods in the game scene by walking over them.

- Jump: User can jump(or double jump) in the game scene.

- Move: User can move his/her character in the game scene.

- Slow Time: User can slow the time after tapping the related icon in order to pass hard parts of the game.

- Open options Menu: User can open Options Menu after tapping the related icon inorder to change some settings(Audio,Brightness etc.)

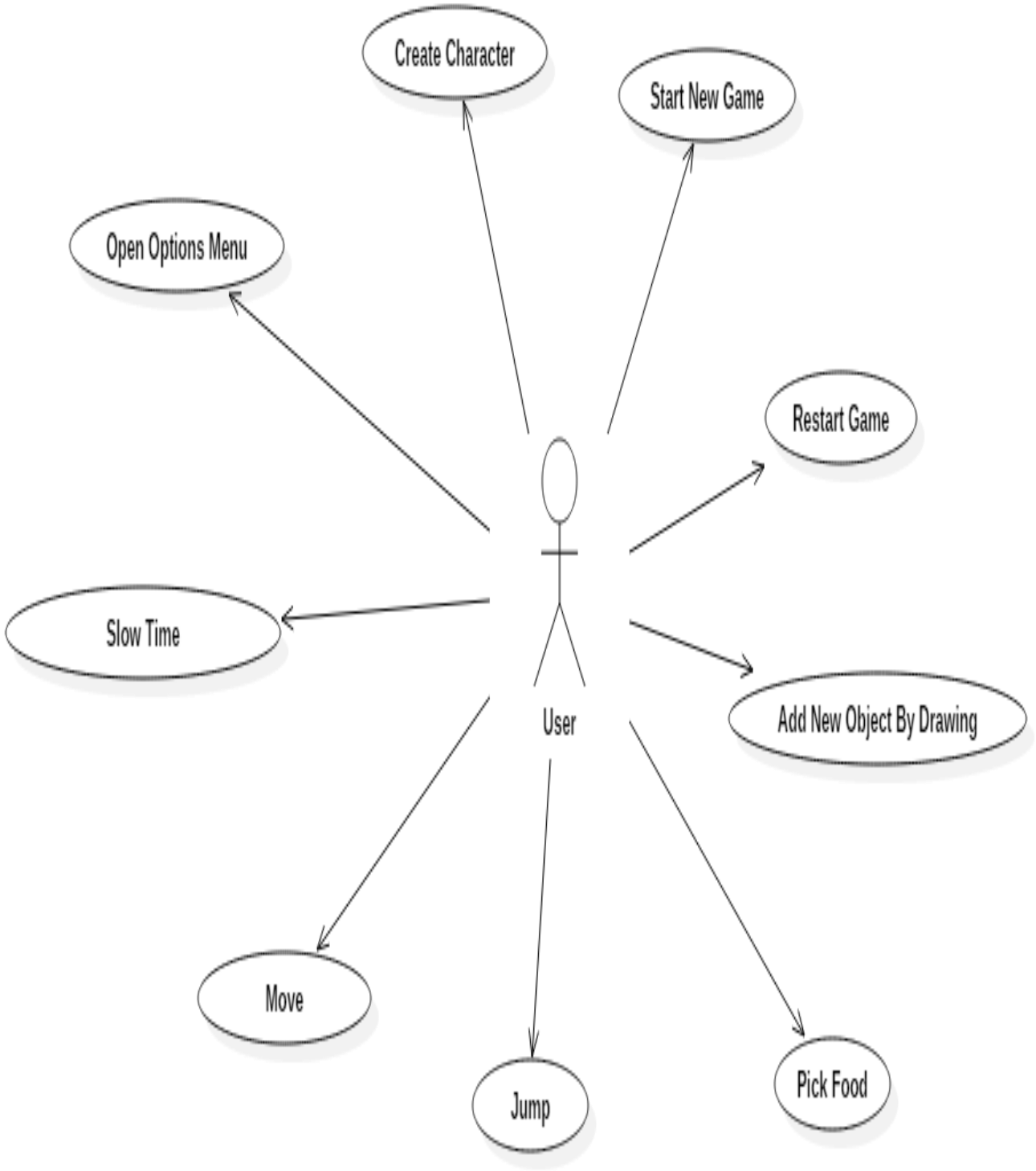

Figure 3: Use Case Diagram

5.2. Logical Viewpoint

Logical viewpoint is mainly related with static structure of the system, and this section will explain all classes and relations between them.

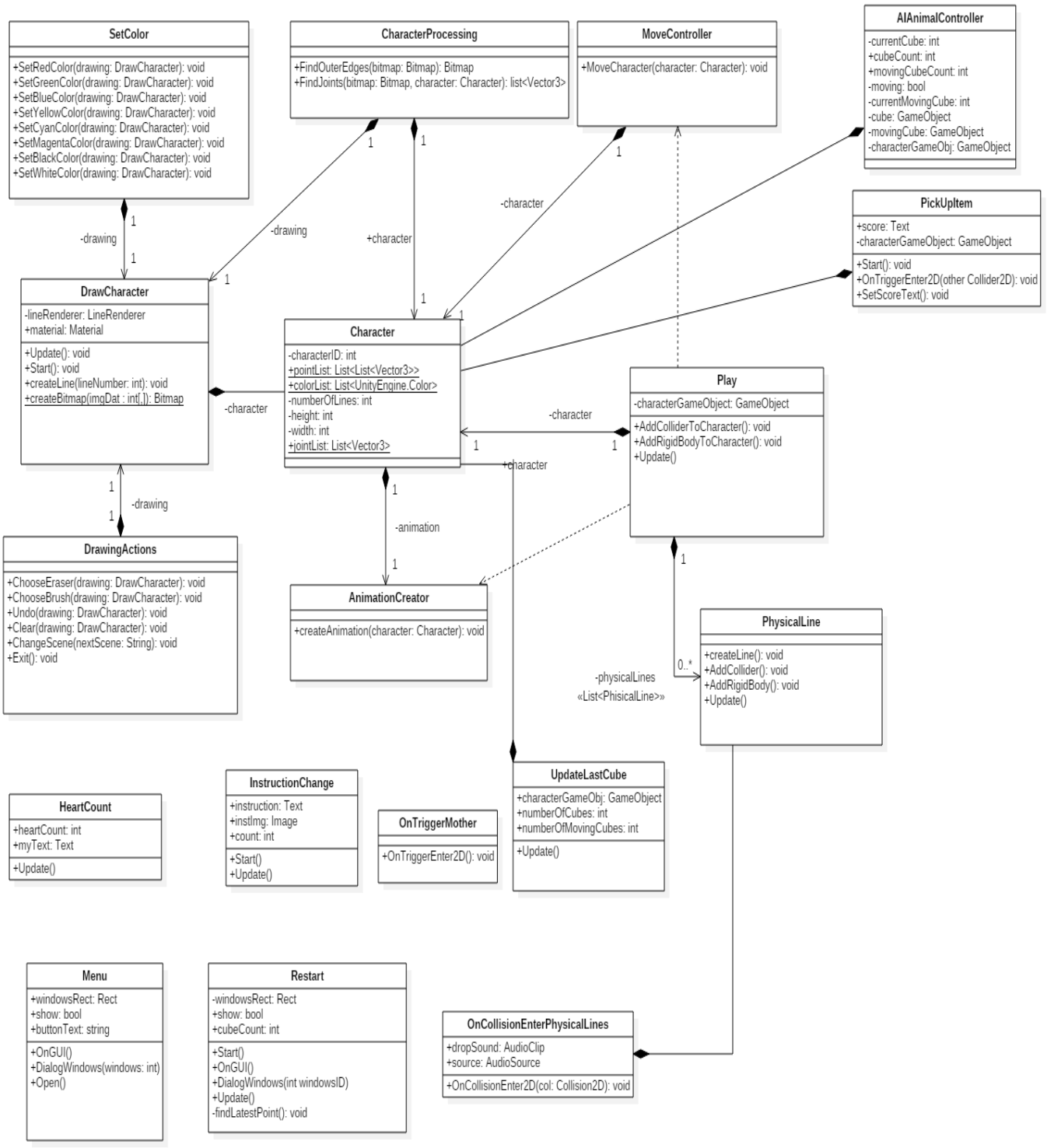

Figure 4: Class Diagram

Class Name	Attiributes & Methods
General Explanation	✓ **Start():** This method exists some classes which is required to make initializition. ✓ **Update():** This method exists in every class and is called once per frame.
DrawCharacter	Attributes: ✓ **lineRenderer:** It is a game object component in Unity and is used to draw lines into sceene. ✓ **material:** We use unlit/color material to determine color of the line renderer. Methods: ✓ **createLine(int lineNumber):** It creates a game object for the given line number and attach a line renderer component to it. ✓ **createBitmap(int [,] imgData):** It creates bitmap image according to character points and colors.
SetColor	Methods: The methods in this class changes the color of brush. ✓ **SetRedColor(DrawCharacter drawing)** ✓ **SetGreenColor(DrawCharacter drawing)** ✓ **SetBlueColor(DrawCharacter drawing)** ✓ **SetYellowColor(DrawCharacter drawing)** ✓ **SetCyanColor(DrawCharacter drawing)** ✓ **SetMagentaColor(DrawCharacter drawing)** ✓ **SetBlackColor(DrawCharacter drawing)** ✓ **SetWhiteColor(DrawCharacter drawing)**
DrawingActions	Methods: ✓ **ChooseEraser(DrawCharacter drawing):** It allows the user select eraser while drawing character. ✓ **ChooseBrush(DrawCharacter drawing):** It allows the user to select brush and to continue draw lines while drawing character. ✓ **Undo(DrawCharacter drawing):** It deletes the last drawn line. ✓ **Clear(DrawCharacter drawing):** It deletes the all lines in the drawing scene. ✓ **ChangeScene(String nextScene):** It chages the current scene to next scene given as argument. ✓ **Exit ():** It exists from the game when pressing Esc on computer or back button on the phone .
Character	Attributes: ✓ **characterID:** This is the unique character id for character. ✓ **pointList:** It stores all points for each line renderer which form the character. ✓ **colorList:** It stores color of each line renderer. ✓ **numberOfLines:** It is the count of line renderers. ✓ **height:** The height of the character. ✓ **width:** The width of the character. ✓ **jointList:** It stores the points which are the joints of the character.

CharacterProcessing	Methods: ✓ **FindOuterEdges(Bitmap bitmap):** It finds the outer edges of the character by using bitmap and return a bitmap. ✓ **FindJoints(Bittmap bitmap, Character character):** It finds the joint points of the character and return a list of Vector3 for each joint.
AnimationCreator	Methods: ✓ **createAnimation(Character character):** While the character is moving, this method is change the points of character lines to give an animation to it.
Play	Attributes: ✓ **characterGameObject:** It is a game object. All game objects which have line renderer componets are child of this game object. Methods: ✓ **AddColliderToCharacter():** It adds box collider 2D to all lines of the character. By doing this,the character has the ability of making physical interactions. ✓ **AddRigidBodyToCharacter():** It adds the physical aspects to the character such as mass, gravity.
PhysicalLine	Methods: ✓ **createLine():** It creates a game object and attach a line renderer to it when the user draw a line to play sceen. ✓ **AddCollider():** It adds polygonal collider 2D to drawn lines in game sceen. By doing this, the lines have the ability of making physical interactions. ✓ **AddRigidBody():** Iı adds physical aspects to the character such as mass, gravity.
Menu	Attributes: ✓ **windowsRect:** Rectangle to form Menu Dialog Box. ✓ **show:** Bool to check if Menu Dialog Box should be shown. ✓ **buttonText:** Text to show in the Menu Dialog Box. Methods: ✓ **OnGUI():** It shows the created Dialog Box on the screen. ✓ **DialogWindows(int windows):** It forms the Dialog Box with the given ID. ✓ **Open():** Check if Dialog Box should be shown and then change the show boolean to true.
Restart	Attributes: ✓ **windowsRect:** Rectangle to form Dialog Box. ✓ **show:** Bool to check if Restart Dialog Box should be shown. ✓ **cubeCount:** Number of cubes that indicates where the character fell down. Methods: ✓ **OnGUI():** It shows the created Dialog Box on the screen. ✓ **DialogWindows(int windows):** It forms the Dialog Box with the given ID. ✓ **findLatestPoint():** Keeps track of the falling position of the character.
InstructionChange	Attributes:

	✓ **instruction:** Instruction text. ✓ **instImg:** Image of the active instruction. ✓ **count:** Number to specify that which instruction text will be shown.
HeartCount	Attributes: ✓ **heartCount:** Count of current lives. ✓ **myText:** Text to show current lives.
OnTriggerMother	Methods: ✓ **OnTriggerEnter2D():** Specifies the actions to take when the trigger event occured.
OnCollisionEnterPhysicalLines	Attributes: ✓ **dropSound:** Sound to be played when the line dropped. ✓ **source:** AudioSource to play the drop sound. Methods: ✓ **OnCollisionEnter2D(Collision2D col):** Specifies the actions to take when the col object collided with another object.
MoveController	Methods: ✓ **MoveCharacter(Character character):** Moves the character by changing its position with respect to the touch input.
UpdateLastCube	Attributes: ✓ **characterGameObj:** Game Object of the main character. ✓ **numberOfCubes:** Number of cubes in the scene. ✓ **numberOfMovingCubes:** Number of moving cubes in the scene.
AIAnimalController	Attributes: ✓ **currentCube:** ID of the current cube that animal stands atop. ✓ **cubeCount:** The total number of cubes in the scene. ✓ **movingCubeCount:** The total number moving cubes in the scene. ✓ **moving:** Bool to check if animal is moving. ✓ **currentMovingCube:** The last moving cube that animal stands atop most recently. ✓ **cube:** The cube that animal will try to reach. ✓ **movingCube:** The moving cube that animal will try to reach. ✓ **characterGameObj:** Game Object of the main character.
PickUpItem	Attributes: ✓ **score:** Current score of the player. ✓ **characterGameObject:** Game Object of the main character. Methods: ✓ **OnTriggerEnter2D(Collider2D other):** Pick up the item when the character moved towards to it. (Update the score accordingly.) ✓ **SetScoreText():** Text to show the current score.

5.3. Interaction Viewpoint

In this section, the interactions between the objects are clearly shown by using Sequence Diagrams. Related Sequence Diagrams are below:

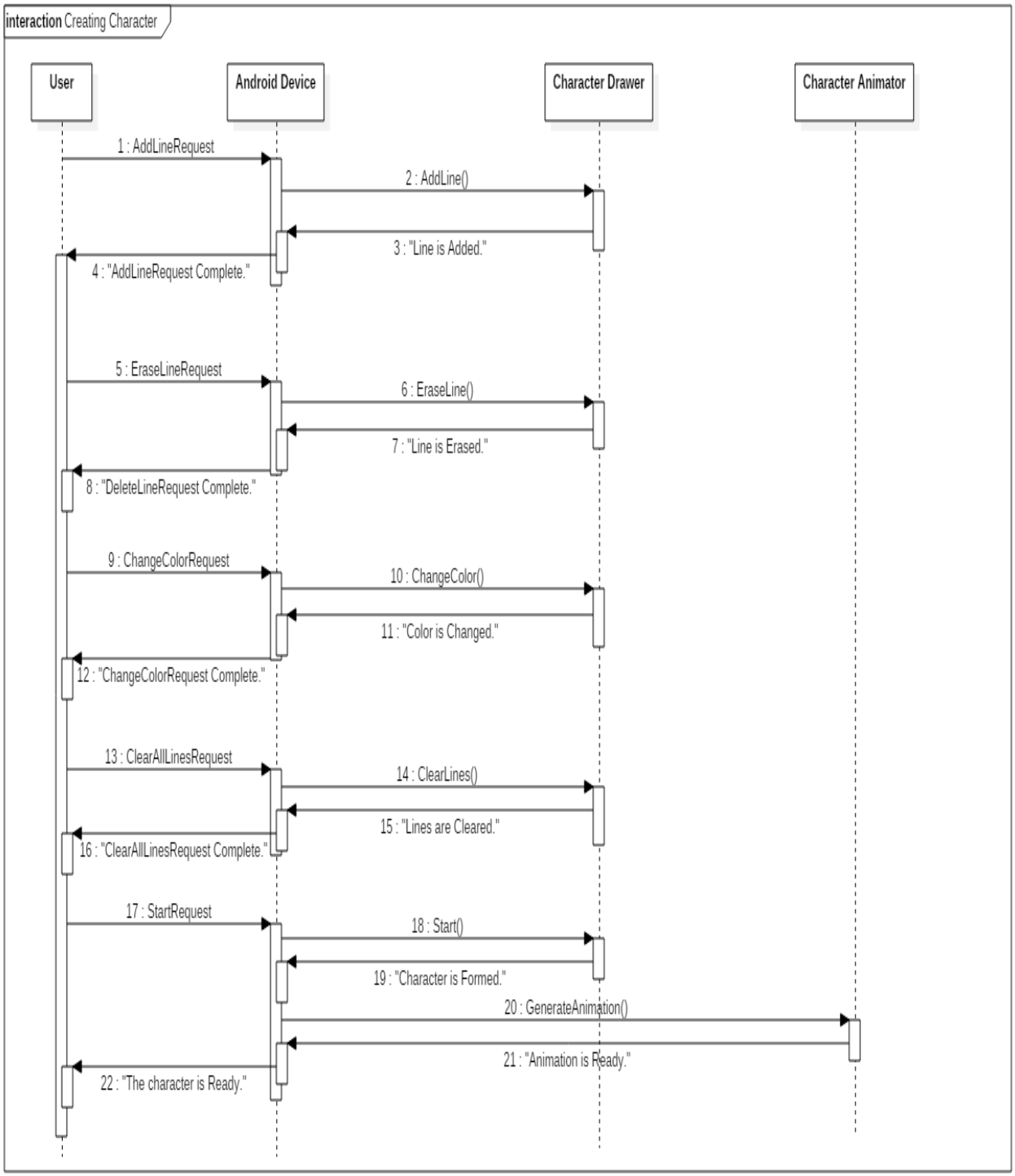

Figure 5: Sequence Diagram for Creating Character

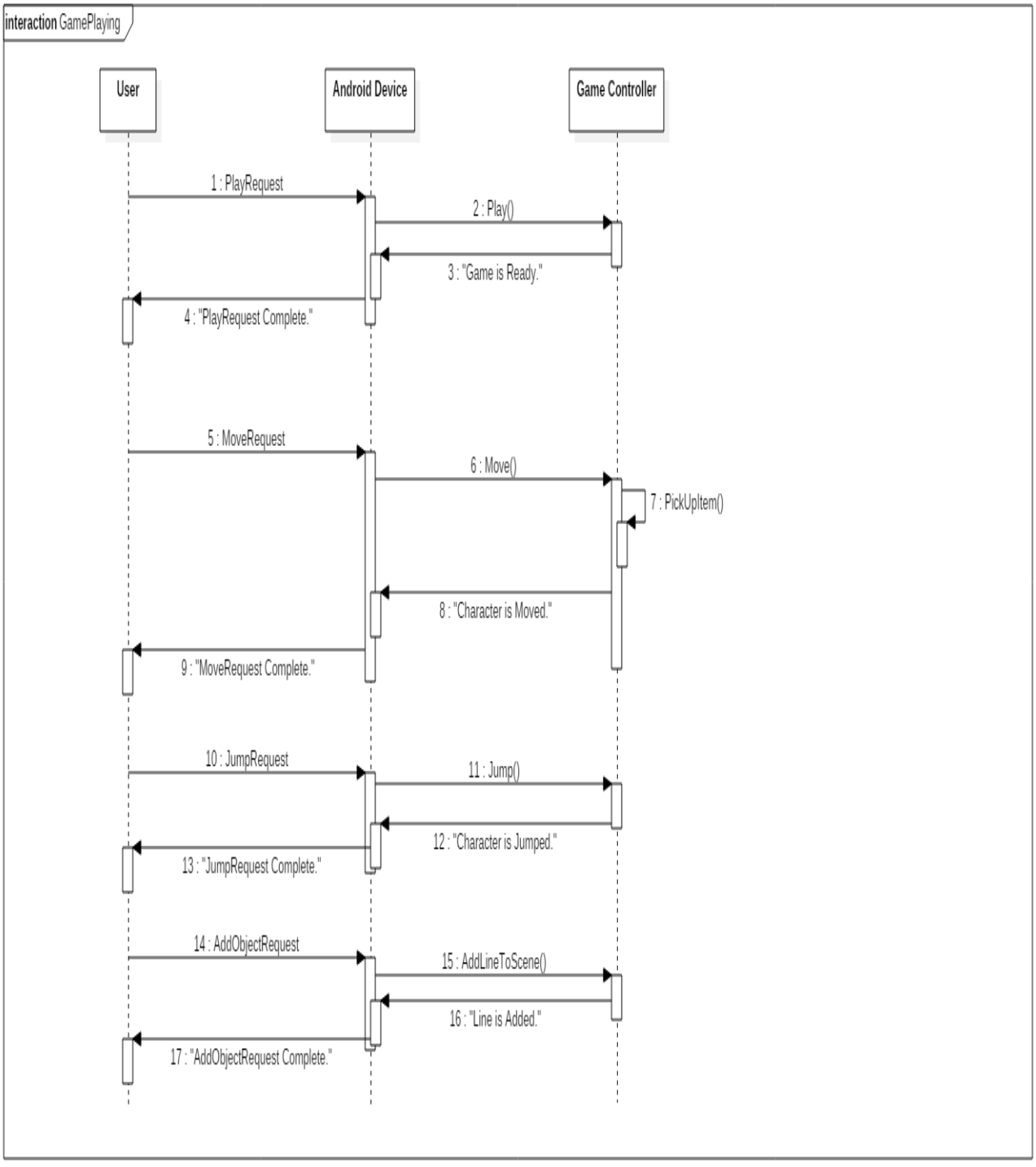

Figure 6: Sequence Diagram for Game Playing

5.4. State Dynamics Viewpoint

In this section, the states of the application are clearly shown by using Activity Diagram. Related Activity Diagram is below:

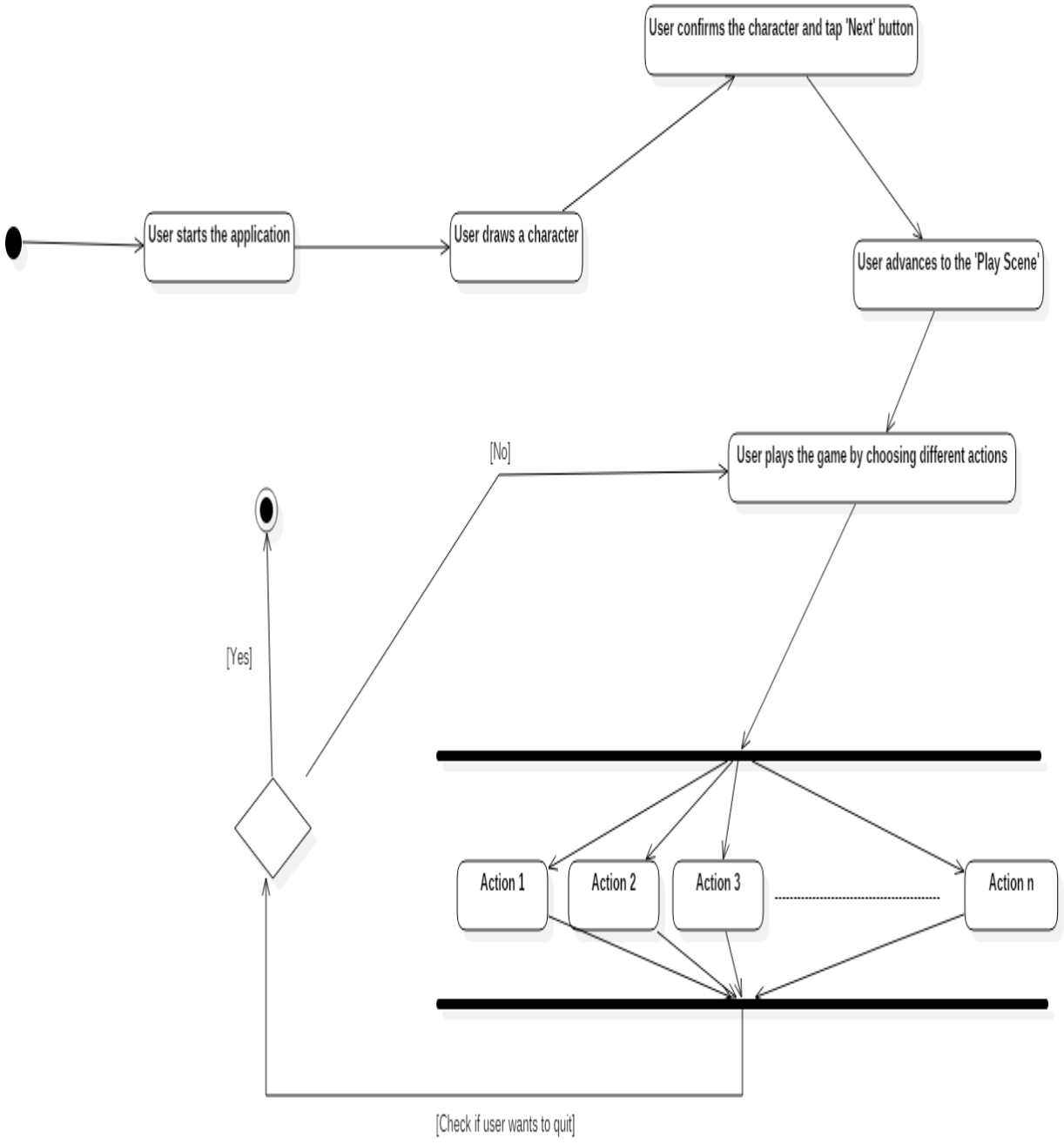

Figure 7: Activity Diagram