OYUN DİYARI

SOFTWARE DESIGN DESCRIPTION

CEMAL AKER

OĞUL CAN ERYÜKSEL

OĞUZHAN TAŞTAN

ANIL GENÇ

Preface

This document contains the system design information about Oyun Diyarı.

“IEEE Standard for Information Technology System Design – Software Design Descriptions – IEEE STD 1016 – 2009” is referenced for preparation of this document.

This Software Design Documentation provides a complete description of all the system design and views of the project. The first section of this document includes project identification, audience identification and stakeholders’ identification. The following sections include design viewpoints of the system.

Table of Contents

1 Introduction

This document is Software Design Description document of Oyun Diyarı. Brief overview about SDD document of Oyun Diyarı is provided in this section. Since, it is intended to be a clarification and a guide for the reader. Purpose and scope of the document, definitions and overview of the document are explained in this section.

1.1 Scope

In this document, the detailed structure of the components of Oyun Diyarı is described along with the precise implementation details required to satisfy the requirements which have been specified in the SRS document. It is assumed that the reader has read the SRS document of Oyun Diyarı, since that document also defines the implementation details of the desired behavior given as requirements within it. The document shows the design views using the several diagrams in order to guide the developers and test engineers with an understanding of the overall system architecture and design.

1.2 Purpose

This document aims to describe the whole architecture of Oyun Diyarı from several perspectives by using 10 design viewpoints which are context, composition, logical, dependency, information, patterns, interface, structure, interaction, and state dynamics viewpoints. All relationships of modules and interfaces of Oyun Diyarı are also main concerns of this document.

1.3 Intended Audience

All developers who will build Oyun Diyarı and test it are the intended audience of the SDD document. Moreover teaching assistants, faculty members of METU CENG Department and any other users, who interested in this project, also include target audience of the SDD document.

1.4 Design Stakeholders and Concerns

The members of the design team and development team, which are the group members of VisionImpossible team, are the design stakeholders of the system. Our main concerns are very high usability of the user interface, code reusability and quality. Hence, throughout the design stage they are extensively considered. Each of the models provided in the following sections are based on these concerns.

1.5 Design Elements and Relationships

With the help of related diagrams in each design viewpoint, following sections describes the corresponding design elements and their relationships. They are also explained in detail with the help of description tables.

1.6 Design Rationale

In the OyunDiyari system, the programming language that used is C++. In one hand, since C++ is an object oriented language, it is possible and easy to implement the design and architectural patterns. On the other hand, many computer vision and point cloud libraries supports C++ like OpenCV and Point Cloud Library(PCL) and C++ runs on any platform, not only Windows.

The main architecture pattern of the system is EDA. EDA has three main advantages for our system. The first advantage of it is that it supports business demand for better services, since there is no batching and it provides less waiting. In EDA, there is also no point-to-point integration. It is based on fire-and-forget idea. In each layer of the EDA, the event is pushed to the next layer, thereafter there is no link with the layer and the event. Therefore, it is easy to add one more event engine or event generator and thanks to this the architecture become scalable. The last advantage of EDA is that its components are loosely coupled which means that each of its components has, or makes use of, little or no knowledge of the definitions of other separate components . This makes the system more agile because a change in one component does not affect the others.

One of the design patterns used in the system is producer-consumer design pattern. One advantage of this pattern is that it is simple to understand and implement. Moreover, the producer and the consumer parts can be implemented separately and concurrently, they just need to know shared objects in the shared queue. Therefore, the code is clean, readable and manageable. Another advantage of it is that producer does not need to know about whom the consumer is and how many consumers there are. The same is valid for the consumer. Thus, the pattern has high flexibility. Yet another advantage of it is that the consumer and the producer can work at different speed. By monitoring the consumers' speed, another consumer can be introduced for better utilization.

Another design patterns used in the system is singleton pattern. The singletons in the system prevent other objects from instantiating their own copies of the singleton object, ensuring that all objects access the single instance. This is exactly what is needed for DatabaseManager Class and OyunDiyari Class. Since the singleton classes control the instantiation process, the singleton classes have the flexibility to change the instantiation process. The advantage of singleton pattern over global variables is that you are absolutely sure of the number of instances and you can change your mind and manage any number of instances.

1.7 Design Language and Tools

Unified Modeling Language (UML) is used for this document as design language, since it has good advantages. It breaks the complex system into discrete pieces that can be understood easily by all intended audiences. Moreover to this, UML is not a system or platform specific.

In Oyun Diyarı, Enterprise Architect, which is built by Sparx System, is used as a design tool. This Enterprise Architect provides us many advantages such as flexibility, fast operations, extensible. It can also export our diagrams many XML format or image format.

For the mockup screen images of Oyun Diyarı, the program named as Balsamiq mockups, is used. It has benefits that thanks to its drag and drop feature, screen images are created very fast; therefore it saves our time.

1.8 Definitions, Acronyms, and Abbreviations

Term	Definition or Abbreviation
CENG	Computer Engineering
DBMS	Database Management System
ER Diagram	Entity-Relationship Diagram
IEEE	The Institute of Electrical and Electronics Engineers
Linux	An open source operating system

METU	Middle East Technical University
SRS	Software Requirement Specification
TCP/IP	A communication protocol for the internet and similar networks
UML	Unified Modeling Language
Kinect	3D sensor camera that that produced by Microsoft company.
PCL	Point Cloud Library
OpengGL	Open Graphic Library
SDD	Software Design Description
EDA	Event Driven Architecture
SDK	Software Development Kit
API	Application Programming Interface
3D	Three Dimensions

1.9 Overview

This document is a software design description for Oyun Diyarı as mentioned in section 1. It is prepared according to IEEE STD 1016 – 2009. Section 2 includes related references about the document. The design from different perspectives is described in section 3 by using related diagrams and tables.

2 References

- IEEE Std 1016 – 2009
- For mockup tool (Balsamiq Mockup) http://balsamiq.com/products/mockups/
- For all diagrams (Enterprise Architect) http://www.sparxsystems.com.au/

3 Design Viewpoints

The fundamental purpose of this section describes all viewpoints of Oyun Diyarı in detail. By this way, this section increases understanding of all stakeholders from all perspectives of Oyun Diyarı.

3.1 Context Viewpoint

The overall context of Oyun Diyarı, which includes all the relationships, interactions and dependencies between Oyun Diyarı and its environment, is explained in context viewpoint. In other words, context viewpoint specifies that the interactions between design objects and actor that include stakeholders and users. Context and use case diagrams with refinements from SRS of Oyun Diyarı help to show these interactions more clearly.

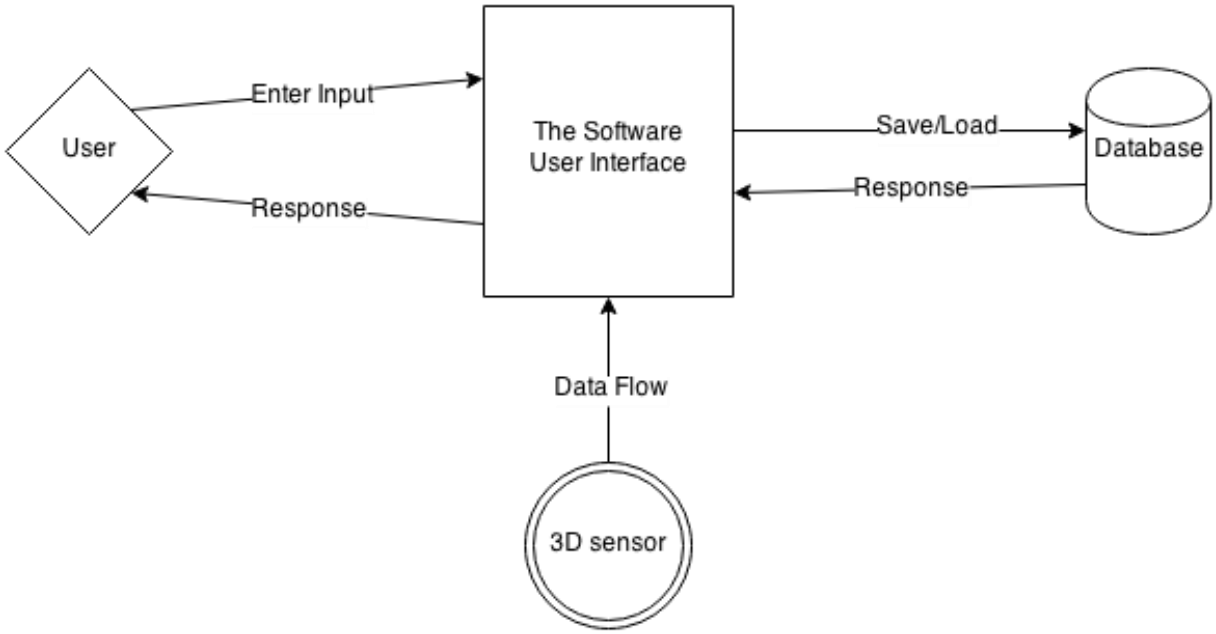

Figure 1 – Context Diagram

Figure 1 shows the context diagram for the Oyun Diyarı. This diagram explains the system parts in an abstract way. User interface of the software is responsible for interacting with the user, 3D sensor and the database as a bridge. It takes inputs from the user and processes it according to data which come from 3D sensor. User interface also generates responses to the user and communicate with the database. The database holds the system data and provides it according to queries made by user interface of Oyun Diyarı.

3.1.1 Use Cases

This section describes the offered services and the actors of the system with the help of the use case diagrams and the scenarios related to them.

3.1.1.1 Use Cases for Actor Parent

All the services provided to the actor parent are shown in Figure 2. Following scenarios explain the use cases.

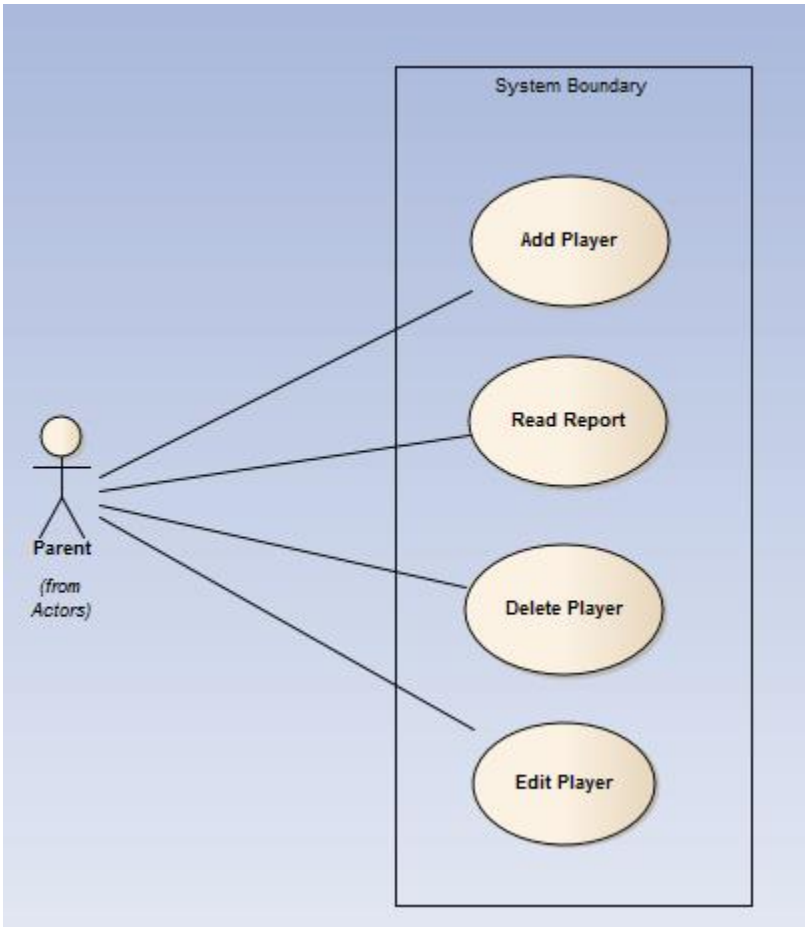

Figure 2 – Use Case Diagram for Actor Parent

Use Case ID	UC1
Use Case Name	Add Player
Description	This use case describes the event in which the parent who wants to add a player.
Actors	The parent
Preconditions	The parent should be in “Profile Manager Screen”.
Trigger	The parent clicks to add player button.
Basic Flow	1. The parent moves cursor by hand movement to add player button. 2. The parent clicks to add player button by grabbing gesture. 3. The add player screen appears on the screen. 4. The parents fill the needed attributes to add a player. 5. The parent moves cursor by hand movement to OK.

	6. The parent clicks to OK by grabbing gesture . 7. The pop up indicating the operation is successful appears
Alternate Flow	-
Exception Flow	-
Post Conditions	The player should be inserted into the database.

Use Case ID	UC2
Use Case Name	Edit Player
Description	This use case describes the event in which the parent who wants to edit a player.
Actors	The parent
Preconditions	The parent should be in “Profile Manager Screen”.
Trigger	The parent clicks to edit player button.
Basic Flow	1. The parent moves cursor by hand movement to edit player button. 2. The parent clicks to edit player button by grabbing gesture. 3. The edit player screen appears on the screen. 4. The parents fill the needed attributes to edit a player. 5. The parent moves cursor by hand movement to OK. 6. The parent clicks to OK by grabbing gesture. 7. The pop up indicating the operation is successful appears
Alternate Flow	-
Exception Flow	-
Post Conditions	The player in the database should be updated.

Use Case ID	UC3
Use Case Name	Delete Player
Description	This use case describes the event in which the parent who wants to delete a

	player.
Actors	The parent
Preconditions	The parent should be in “Profile Manager Screen”.
Trigger	The parent clicks to delete player button.
Basic Flow	1. The parent moves cursor by hand movement to the player who will be deleted. 2. The parent clicks to delete player button by grabbing gesture. 3. The pop up indicating the operation is successful appears. 4. A dialog box for confirming the operation pops up. 5. The player clicks on the “OK” button. 6. The player is deleted.
Alternate Flow	-
Exception Flow	3. The player clicks on the “No” button. 4. The player is not deleted.
Post Conditions	The player should be deleted from the database.

Use Case ID	UC4
Use Case Name	Read Report
Description	This use case describes the event in which the parent who wants to read the report.
Actors	The parent
Preconditions	The parent should be in “Profile Manager Screen”.
Trigger	The parent clicks to read report button.
Basic Flow	1. The parent moves cursor by hand movement to read report button. 2. The parent clicks to read report button by grabbing gesture. 3. The report appears on the screen.
Alternate Flow	-

Exception Flow	-
Post Conditions	The report should appear on the screen.

3.1.1.2 Use Cases for Actor Player

All the services provided to the actor player are shown in Figure 3. Following scenarios explain the use cases. The scenario for the "Play Level" use case is excluded since it is explained in the following section.

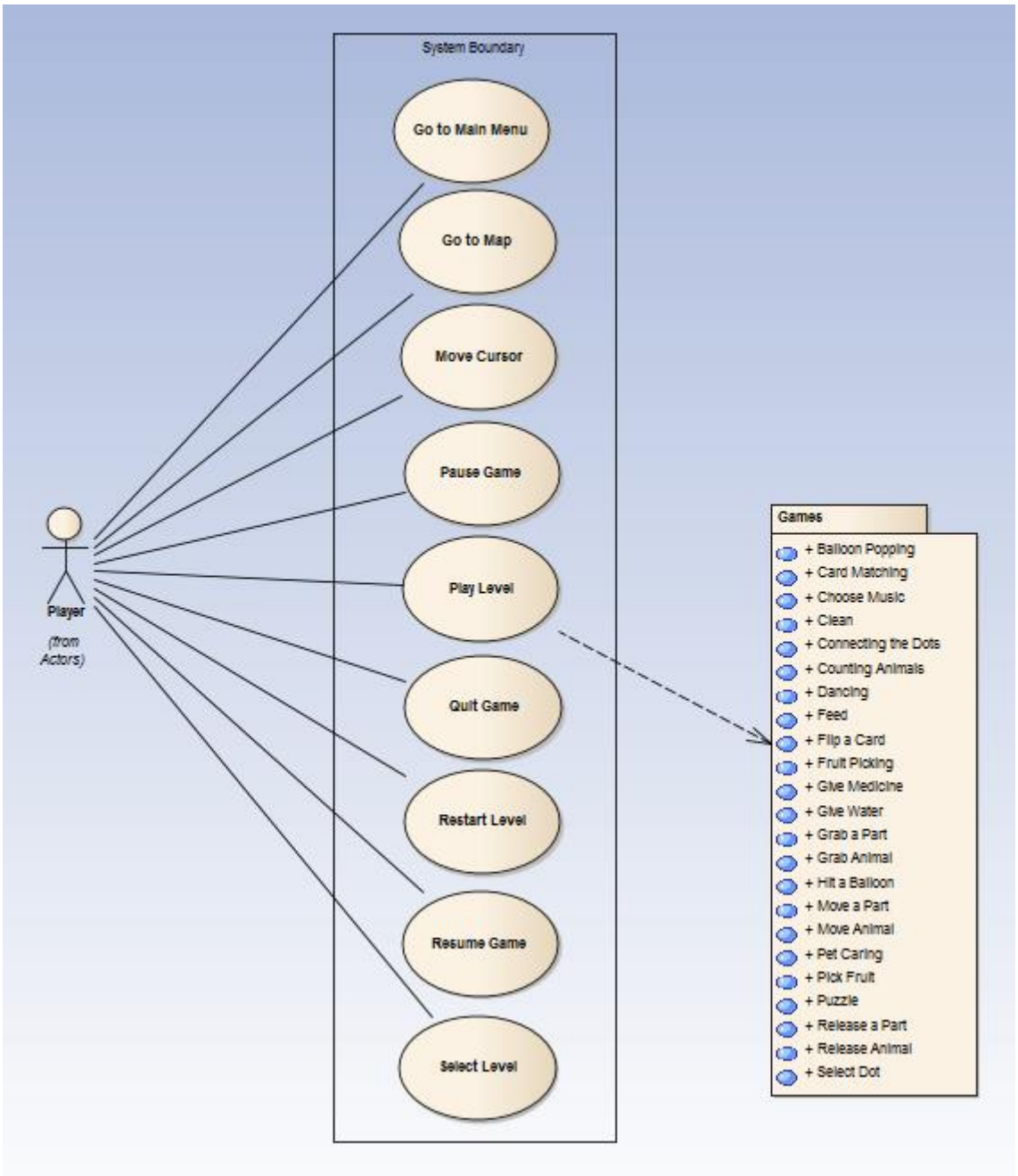

Figure 3 – Use Case Diagram for Actor Player

Use Case ID	UC5
Use Case Name	Go to Main Menu
Description	This use case describes the event in which the player who wants to return back to the main menu.
Actors	The player who wants to go back to main menu.
Preconditions	The player should be in “Profile Manager Screen” or “Map Screen”.
Trigger	Player clicks the “Go to Main Menu”.
Basic Flow	1. The player moves the cursor by hand movement to the “Go To Main Menu”. 2. Player clicks the “Go to Main Menu” by grabbing gesture. 3. The player is directed to the Main Menu.
Alternate Flow	-
Exception Flow	-
Post Conditions	The player should be in Main Menu.

Use Case ID	UC6
Use Case Name	Go to Map
Description	This use case describes the event in which the player who wants to return back to “Map Screen”.
Actors	The player who wants to go back to “Map Screen”.
Preconditions	The player should be playing a level.
Trigger	Player clicks the “Go to Map”.
Basic Flow	1. The player moves the cursor by hand movement to the “Go to Map” 2. Player clicks the “Go to Map” by grabbing gesture. 3. A dialog box for confirming the operation pops up. 4. The player clicks on the “OK” button. 5. The player is directed to the “Map Screen”.
Alternate Flow	

Exception Flow	3. The player clicks on the “No” button. 4. Game is going on.
Post Conditions	The player should be in “Map Screen”.

Use Case ID	UC7
Use Case Name	Move Cursor
Description	This use case describes the event in which the player who wants to move cursor.
Actors	The player
Preconditions	-
Trigger	The player shows his/her palm to the camera.
Basic Flow	1. The player moves his/her hand in any direction. 2. The cursor moves in the direction which is the same as the player's.
Alternate Flow	-
Exception Flow	-
Post Conditions	The cursor must be at the correct position according to its movement.

Use Case ID	UC8
Use Case Name	Pause Game
Description	This use case describes the event in which the player who wants to pause the game
Actors	The player
Preconditions	The player should be playing a level.
Trigger	The player clicks the pause button.
Basic Flow	1. The player moves the cursor by hand movement to the pause button. 2. The player clicks the pause button by grabbing gesture. 3. The game pauses and pause menu is displayed.

Alternate Flow	-
Exception Flow	-
Post Conditions	The game should be paused.

Use Case ID	UC9
Use Case Name	Quit Game
Description	This use case describes the event in which the player who wants to quit the game.
Actors	The player
Preconditions	The player should be in “Profile Manager Screen” or “Map Screen” or “Main Menu”.
Trigger	The player clicks the quit button.
Basic Flow	1. The player moves the cursor by hand movement to the quit button. 2. The player clicks the quit button. 3. A dialog box for confirming the operation pops up. 4. The player clicks on the “OK” button. 5. The game is shut down.
Alternate Flow	-
Exception Flow	3. The player clicks on the “No” button. 4. The player stays in the current screen.
Post Conditions	-

Use Case ID	UC10
Use Case Name	Restart Level
Description	This use case describes the event in which the player who wants to restart the level currently played.
Actors	The player
Preconditions	The player should be playing a level.

Trigger	The player clicks the pause button.
Basic Flow	1. The player moves the cursor by hand movement to the pause button. 2. The player clicks the pause button by grabbing gesture. 3. The player clicks the restart level button in the pause menu. 4. A dialog box for confirming the operation pops up. 5. The player clicks on the “OK” button. 6. The level is restarted.
Alternate Flow	-
Exception Flow	3. The player clicks on the “No” button. 4. Game is going on.
Post Conditions	The level should be restated.

Use Case ID	UC11
Use Case Name	Resume Game
Description	This use case describes the event in which the player who wants to resume the paused game.
Actors	The player
Preconditions	The player should be paused the game.
Trigger	The player clicks resume button.
Basic Flow	1. The player moves the cursor to resume button by hand movement. 2. The player clicks resume button by grabbing gesture. 3. The paused game is started from where it stays.
Alternate Flow	-
Exception Flow	-
Post Conditions	The player continues the level where it stays.

Use Case ID	UC12
Use Case Name	Select Level
Description	This use case describes the event in which the player who wants to play a level.
Actors	The player
Preconditions	The player should be in Map Screen.
Trigger	The player clicks to a level.
Basic Flow	1. The player moves cursor by hand movement to the level which s/he wants to play. 2. The player clicks to the level by grabbing gesture. 3. The corresponding level starts.
Alternate Flow	-
Exception Flow	-
Post Conditions	The player should be playing the level s/he has chosen.

3.1.1.4 Use Cases Extending Play Level Use Case for Actor Player

All the use cases extending “Play Level” use case for the actor player are shown in Figure 4. Following scenarios explain the use cases.

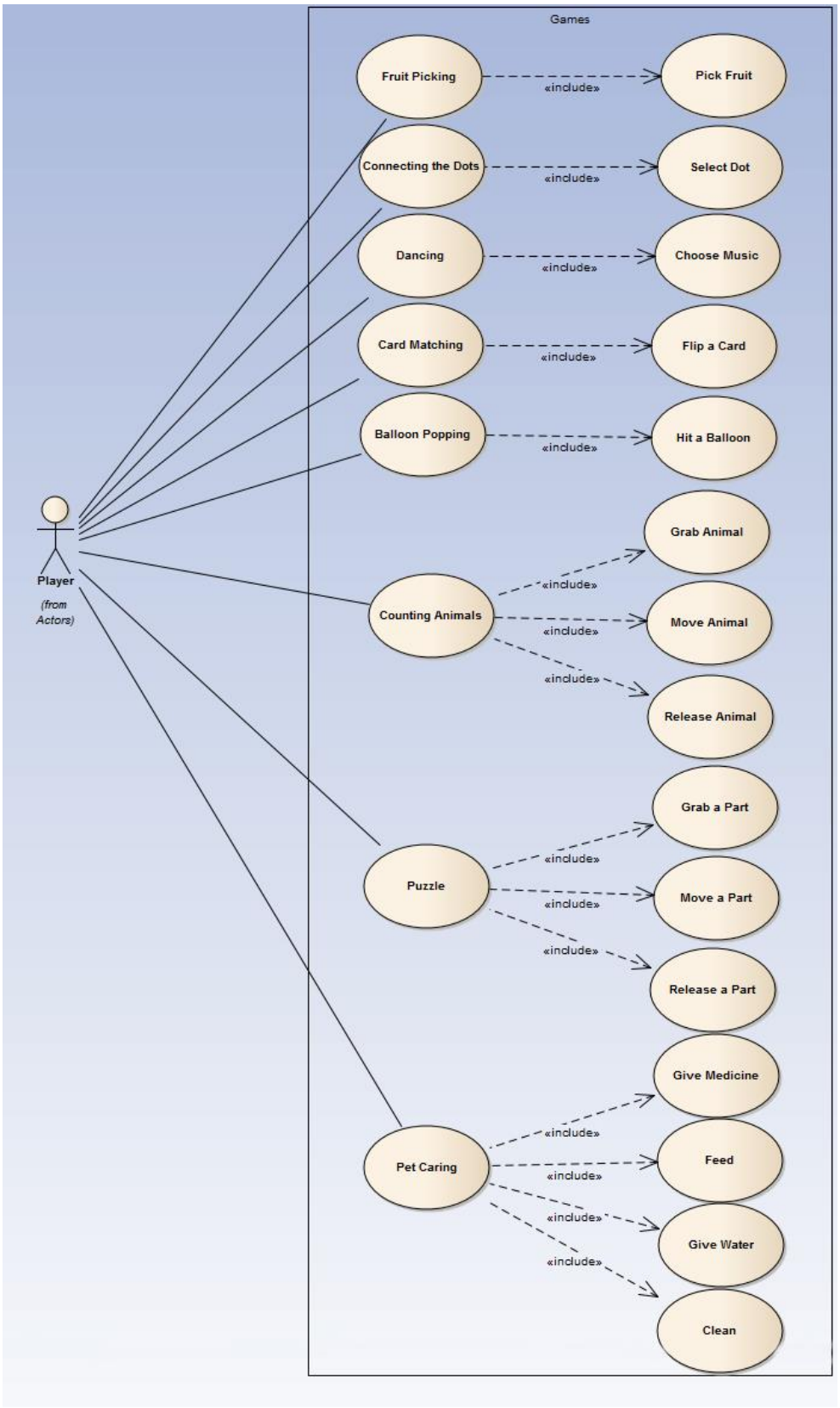

Figure 4 – Use Case Diagram for the Use Cases Extending the Play Level Use Case for Actor Player

Use Case ID	UC13
Use Case Name	Fruit Picking
Description	This use case describes the event in which the player plays a level which is designed as a Fruit Picking game.
Actors	Player who wants to play a level which is a Fruit Picking game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. A tree and fruits on it appear on the screen. 5. Player plays the level by picking fruits.
Alternate Flow	-
Exception Flow	-
Post Conditions	Player should be able to play the level by picking fruits.

Use Case ID	UC14
Use Case Name	Pick Fruit
Description	This use case describes the event in which the player picks a fruit from the tree in Fruit Picking game.
Actors	Player who wants to pick a fruit in Fruit Picking game.
Preconditions	Player should be playing a level which is a Fruit Picking game.
Trigger	Player moves cursor.
Basic Flow	1. Player starts a level which is a Fruit Picking game. 2. Player moves the cursor on a fruit by hand motion. 3. The fruit is placed into a basket.
Alternate Flow	-
Exception Flow	-
Post Conditions	The fruit disappears on the tree.

Use Case ID	UC15
Use Case Name	Connecting the Dots
Description	This use case describes the event in which the player plays a level which is designed as a Connecting the Dots game.
Actors	Player who wants to play a level which is a Connecting the Dots game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. Dots on the circumference of a shape appear on the screen. 5. Player plays the level by selecting the next dot each time until the shape is completed.
Alternate Flow	-
Exception Flow	-
Post Conditions	Player should be able to play the level by selecting dots.

Use Case ID	UC16
Use Case Name	Select Dot
Description	This use case describes the event in which the player selects the next dot to complete the shape in Connecting the Dots game.
Actors	Player who wants to select the next dot in Connecting the Dots game.
Preconditions	Player should be playing a level which is a Connecting the Dots game.
Trigger	Player moves cursor.
Basic Flow	1. Player starts a level which is a Connecting the Dots game. 2. Player moves the cursor on a dot. 3. Player makes a grabbing gesture to select the dot. 4. The dot is joined by a straight line to the previous one.
Alternate Flow	-
Exception Flow	4. The dot is not joined to the previous one if it is not the correct dot.
Post Conditions	A line between the previous and the newly chosen dots should appear.

Use Case ID	UC17
Use Case Name	Dancing
Description	This use case describes the event in which the player plays a level which is designed as a Dancing game.
Actors	Player who wants to play a level which is a Dancing game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. Player chooses the music s/he wants. 5. Requested move appears on the screen. 6. Player makes the move. 7. Go to 5
Alternate Flow	5. Player dances freely.
Exception Flow	-
Post Conditions	-

Use Case ID	UC18
Use Case Name	Choose Music
Description	This use case describes the event in which the player chooses the music that will be played in Dancing game.
Actors	Player who wants to choose the music in Dancing game.
Preconditions	Player should be playing a level which is a Dancing game.
Trigger	The level starts.
Basic Flow	1. Player starts a level which is a Dancing game. 2. A popup menu appears. 3. Player chooses one of the provided music files by pushing the corresponding button. 4. The popup menu disappears.
Alternate Flow	-
Exception Flow	-
Post Conditions	The music that is chosen should be played.

Use Case ID	UC19
Use Case Name	Card Matching
Description	This use case describes the event in which the player plays a level which is designed as a Card Matching game.
Actors	Player who wants to play a level which is a Card Matching game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. Cards each of which has a pair appear on the screen faced up. 5. Cards are flipped face down. 6. Player flips a card face up. 7. Player flips the pair of that card face up.
Alternate Flow	7. Player flips a card other than the pair face up. 8. Both cards are flipped face down.
Exception Flow	-
Post Conditions	Player should be able to play the level by selecting dots.

Use Case ID	UC20
Use Case Name	Flip a Card
Description	This use case describes the event in which the player flips a card face up in Card Matching game.
Actors	Player who wants to flip a card in Card Matching game.
Preconditions	Player should be playing a level which is a Card Matching game.
Trigger	The level starts.
Basic Flow	1. Player starts a level which is a Card Matching game. 2. Player moves the cursor on a card. 3. Player makes a grabbing gesture. 4. The card flips face up.
Alternate Flow	-
Exception Flow	-
Post Conditions	The card should appear faced up.

Use Case ID	UC21
Use Case Name	Balloon Popping
Description	This use case describes the event in which the player plays a level which is designed as a Balloon Popping game.
Actors	Player who wants to play a level which is a Balloon Popping game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. Balloons moving around appear on the screen. 5. Player hits a balloon. 6. The balloon pops and disappears.
Alternate Flow	-
Exception Flow	-

Post Conditions	The balloon that is hit should disappear.

Use Case ID	UC22
Use Case Name	Hit a Balloon
Description	This use case describes the event in which the player hits a balloon in Balloon Popping game.
Actors	Player who wants to pop a balloon in Balloon Popping game.
Preconditions	Player should be playing a level which is a Balloon Popping game.
Trigger	The level starts.
Basic Flow	1. Player starts a level which is a Balloon Popping game. 2. Player moves the cursor on a balloon. 3. Player makes a gesture to hit. 4. The balloon pops and disappears.
Alternate Flow	-
Exception Flow	2. Player moves the cursor on an empty area. 3. Player makes a gesture to hit. 4. The gesture is ignored.
Post Conditions	

Use Case ID	UC23
Use Case Name	Counting Animals
Description	This use case describes the event in which the player plays a level which is designed as a Counting Animals game.
Actors	Player who wants to play a level which is a Counting Animals game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. Animals behind a fence appear on the screen. 5. Player grabs an animal. 6. Player moves the animal. 7. Player Releases the animal. 8. Go to 5.
Alternate Flow	-
Exception Flow	-
Post Conditions	

Use Case ID	UC24
Use Case Name	Grab Animal
Description	This use case describes the event in which the player grabs an animal in Counting Animals game.
Actors	Player who wants to grab an animal in Counting Animals game.
Preconditions	Player should be playing a level which is a Counting Animals game.
Trigger	The level starts.
Basic Flow	1. Player starts a level which is a Counting Animals game. 2. Player moves the cursor on an animal. 3. Player makes a grabbing gesture.

	4. The animal is attached to the cursor.
Alternate Flow	-
Exception Flow	2. Player moves the cursor on an empty area. 3. Player makes a grabbing gesture. 4. The gesture is ignored.
Post Conditions	The animal should be attached to the cursor to move together.

Use Case ID	UC25
Use Case Name	Move Animal
Description	This use case describes the event in which the player moves an animal in Counting Animals game.
Actors	Player who wants to move an animal in Counting Animals game.
Preconditions	Player should be playing a level which is a Counting Animals game.
Trigger	Player grabs an animal.
Basic Flow	1. Player starts a level which is a Counting Animals game. 2. Player grabs an animal. 3. Player moves the cursor using hand motion. 4. The animal moves together with the cursor.
Alternate Flow	-
Exception Flow	-
Post Conditions	The animal should be positioned with the same position as the cursor.

Use Case ID	UC26
Use Case Name	Release Animal
Description	This use case describes the event in which the player releases an animal in Counting Animals game.
Actors	Player who wants to release an animal in Counting Animals game.
Preconditions	Player should be playing a level which is a Counting Animals game.
Trigger	Player moves an animal.
Basic Flow	1. Player starts a level which is a Counting Animals game. 2. Player grabs an animal. 3. Player moves an animal. 4. Player opens his/her hand to release the animal. 5. The animal is detached from the cursor.
Alternate Flow	-
Exception Flow	-
Post Conditions	The animal should stay still where it is released.

Use Case ID	UC27
Use Case Name	Puzzle
Description	This use case describes the event in which the player plays a level which is designed as a Puzzle game.
Actors	Player who wants to play a level which is a Puzzle game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts.

	4. Randomly positioned puzzle parts appear on the screen. 5. Player grabs a part. 6. Player moves the part. 7. Player Releases the part. 8. Go to 5.
Alternate Flow	-
Exception Flow	-
Post Conditions	

Use Case ID	UC28
Use Case Name	Grab a Part
Description	This use case describes the event in which the player grabs a part in Puzzle game.
Actors	Player who wants to grab a part in Puzzle game.
Preconditions	Player should be playing a level which is a Puzzle game.
Trigger	The level starts.
Basic Flow	1. Player starts a level which is a Puzzle game. 2. Player moves the cursor on a part. 3. Player makes a grabbing gesture. 4. The part is attached to the cursor.
Alternate Flow	-
Exception Flow	2. Player moves the cursor on an empty area. 3. Player makes a grabbing gesture. 4. The gesture is ignored.
Post Conditions	The part should be attached to the cursor to move together.

Use Case ID	UC29
Use Case Name	Move a Part
Description	This use case describes the event in which the player moves a part in Puzzle game.
Actors	Player who wants to move a part in Puzzle game.
Preconditions	Player should be playing a level which is a Puzzle game.
Trigger	Player grabs a part.
Basic Flow	1. Player starts a level which is a Puzzle game. 2. Player grabs a part. 3. Player moves the cursor using hand motion. 4. The part moves together with the cursor.
Alternate Flow	-
Exception Flow	-
Post Conditions	The part should be positioned with the same position as the cursor.

Use Case ID	UC30
Use Case Name	Release a Part
Description	This use case describes the event in which the player releases a part in Puzzle game.
Actors	Player who wants to release a part in Puzzle game.
Preconditions	Player should be playing a level which is a Puzzle game.
Trigger	Player moves a part.
Basic Flow	1. Player starts a level which is a Puzzle game.

	2. Player grabs a part. 3. Player moves a part. 4. Player opens his/her hand to release the part. 5. The part is detached from the cursor.
Alternate Flow	-
Exception Flow	-
Post Conditions	The part should stay still where it is released.

Use Case ID	UC31
Use Case Name	Pet Caring
Description	This use case describes the event in which the player plays a level which is designed as a Per Caring game.
Actors	Player who wants to play a level which is a Pet Caring game.
Preconditions	Player should load his or her profile.
Trigger	Player chooses the level using corresponding button in the Map Screen.
Basic Flow	1. Player opens the Map Screen. 2. Player pushes the button corresponding to the level. 3. The level starts. 4. The pet appear on the screen. 5. Player cares the pet by giving water, medicine, food or cleaning.
Alternate Flow	-
Exception Flow	-
Post Conditions	Player should be able to play the level by caring the pet.

Use Case ID	UC32
Use Case Name	Give Medicine
Description	This use case describes the event in which the player gives medicine to the pet in Pet Caring game.
Actors	Player who wants to give medicine to the pet in Pet Caring game.
Preconditions	Player should be playing a level which is a Pet Caring game.
Trigger	Player moves cursor.
Basic Flow	1. Player starts a level which is a Pet Caring game. 2. Player moves the cursor on the medicine picture by hand motion.

	3. Player clicks the medicine picture by grabbing gesture. 4. The pet gets medicine.
Alternate Flow	-
Exception Flow	-
Post Conditions	The pet gets medicine and increase its health.

Use Case ID	UC33
Use Case Name	Feed
Description	This use case describes the event in which the player feed the pet in Pet Caring game.
Actors	Player who wants to feed the pet in Pet Caring game.
Preconditions	Player should be playing a level which is a Pet Caring game.
Trigger	Player moves cursor.
Basic Flow	1. Player starts a level which is a Pet Caring game. 2. Player moves the cursor on the food picture by hand motion. 3. Player clicks the food picture by grabbing gesture. 4. The pet eats food.
Alternate Flow	-
Exception Flow	-
Post Conditions	The pet eats food and decrease its hunger.

Use Case ID	UC34
Use Case Name	Give Water
Description	This use case describes the event in which the player give water to the pet in Pet Caring game.
Actors	Player who wants to give water to the pet in Pet Caring game.
Preconditions	Player should be playing a level which is a Pet Caring game.
Trigger	Player moves cursor.

Basic Flow	1. Player starts a level which is a Pet Caring game. 2. Player moves the cursor on the water picture by hand motion. 3. Player clicks the water picture by grabbing gesture. 4. The pet drinks water.
Alternate Flow	-
Exception Flow	-
Post Conditions	The pet drinks water and decrease its thirst.

Use Case ID	UC35
Use Case Name	Clean
Description	This use case describes the event in which the player clean the pet in Pet Caring game.
Actors	Player who wants to clean the pet in Pet Caring game.
Preconditions	Player should be playing a level which is a Pet Caring game.
Trigger	Player moves cursor.
Basic Flow	1. Player starts a level which is a Pet Caring game. 2. Player moves the cursor on the brush picture by hand motion. 3. Player clicks the brush picture by grabbing gesture. 4. The pet is cleaned.
Alternate Flow	-
Exception Flow	-
Post Conditions	The pet is clean and decrease its dirt.

3.2 Composition Viewpoint

Relationships between each component of Oyun Diyarı are described in component viewpoint by using component and deployment diagrams. This viewpoint helps the reusing of components and estimating costs for staffing and scheduling of Oyun Diyarı, while managing and creating it. Figure 5 shows the component diagram of the system. The following table explains the components in detail. Figure 6 is the deployment diagram for the system.

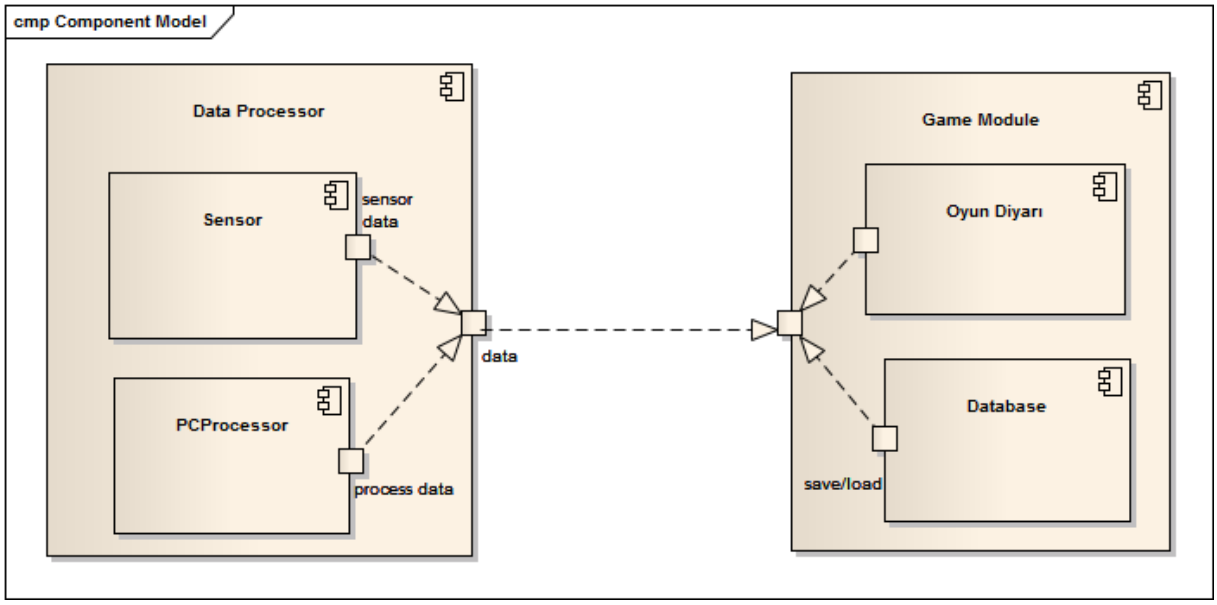

Figure 5 – Component Diagram

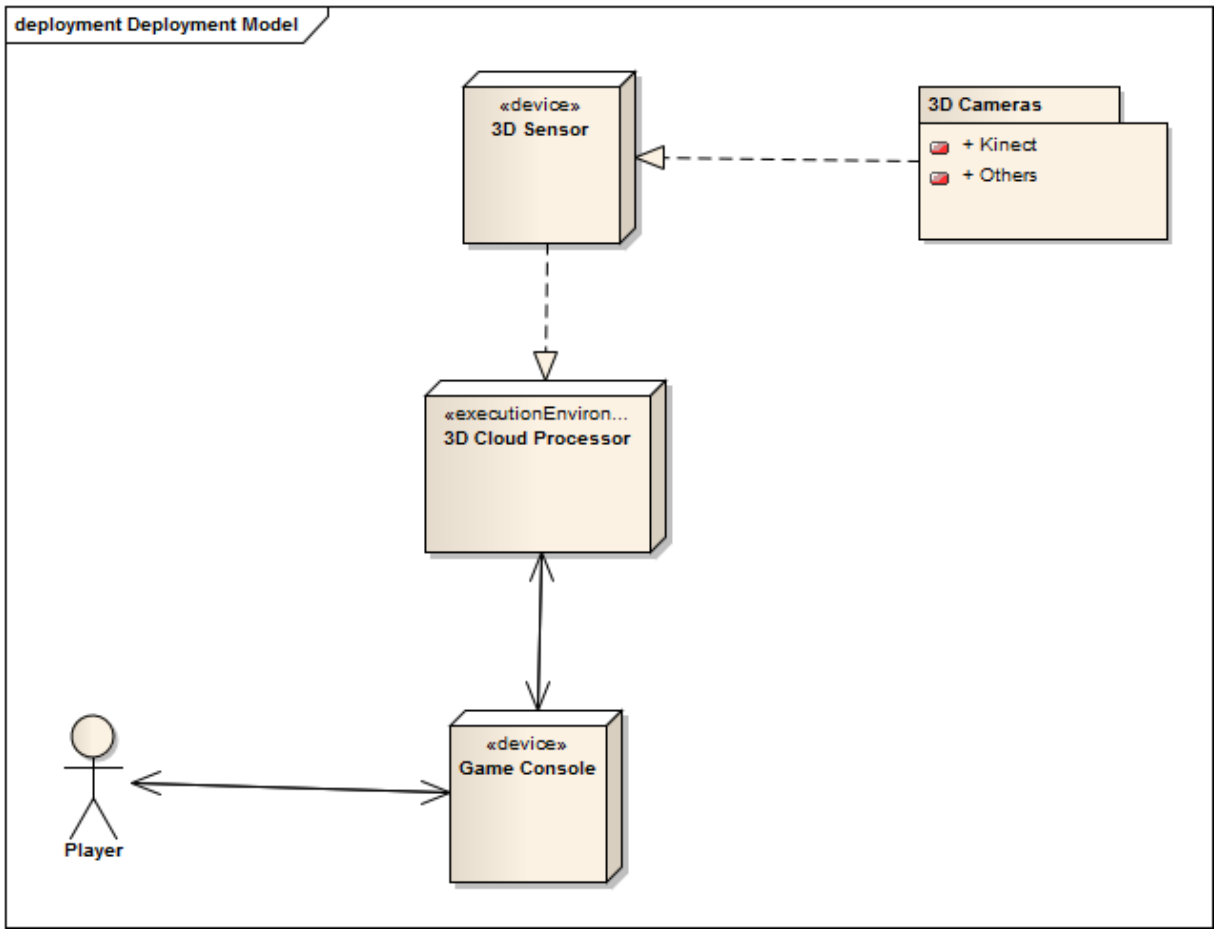

Figure 6 – Deployment Diagram

3.3 Logical Viewpoint

All entities of Oyun Diyarı and their detailed implementations as classes are described in logical viewpoint by using class diagram with refinements from SRS of Oyun Diyarı. Logical viewpoint is also explains static structure of Oyun Diyarı's entities. Except for these main concerns, logical viewpoint

also mentions that the resulting reuse practices and the design pattern adaptations. Figure 7 shows the class diagram describing the objects of the system.

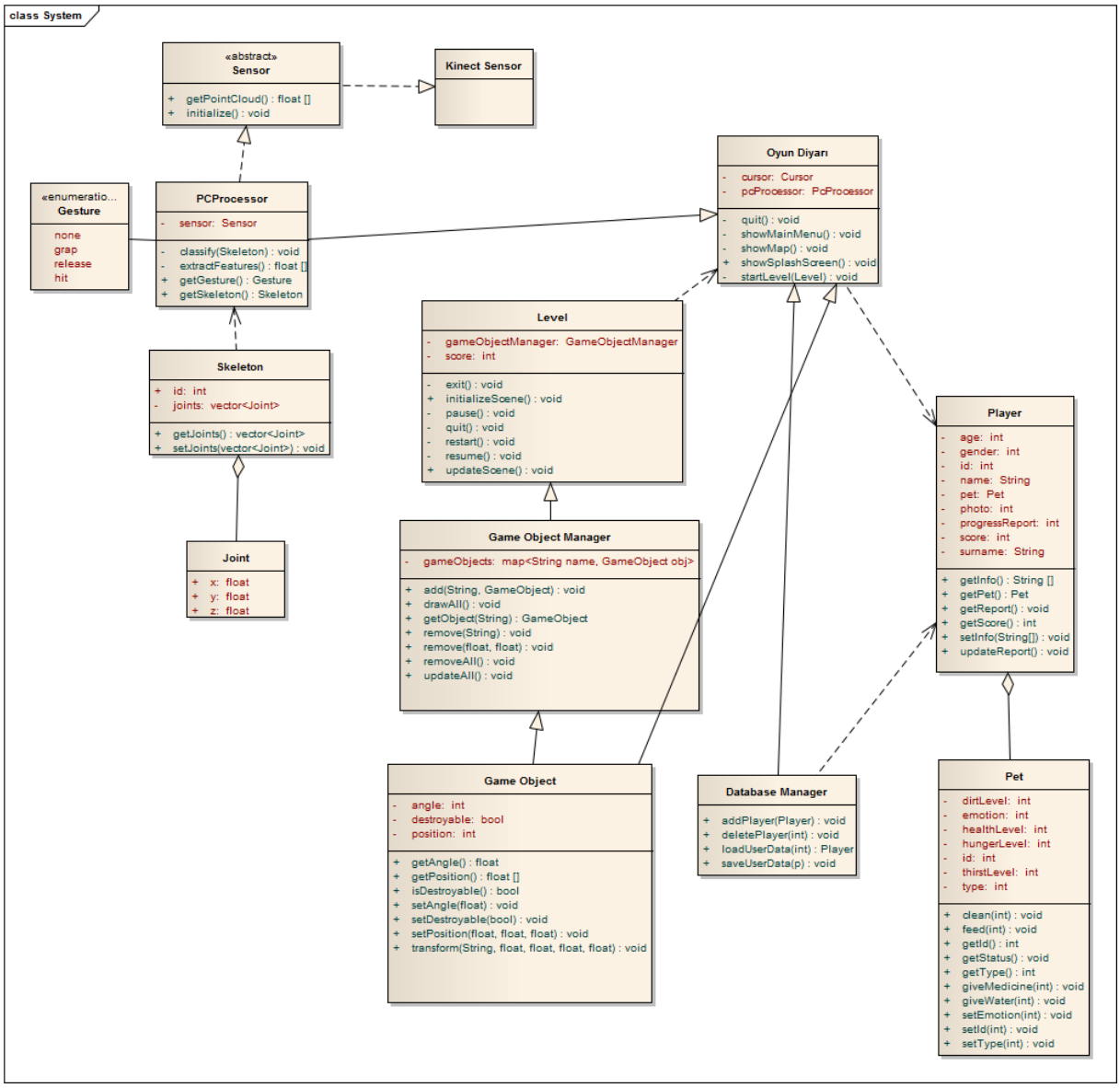

Figure 7 – Class Diagram

3.3.1 Detailed Descriptions of the Classes

3.3.1.1 Level Class

This class is responsible creating every level in the game. These levels can include many different games, fruit picking, connecting dots, puzzle etc. Every game type will inherit from level class. After, it overloads initializeScene() and updateScene() function. Table 1 describes the level class.

Method Signature	Return Type	Description
initializeScene()	void	Creates the game scene with required objects for related game.
updateScene()	void	Updates the whole objects in the scene according to required modifications.
resume()	void	Allows the user to resume the game after the player paused the game.
pause()	void	Pauses the game until player wants to resume again
restart()	void	Restarts current the level.
exit()	void	Exits from the level and returns to the game map.
quit()	void	Quits from the whole game. Terminates the program.

Table 1 – Description of the Level Class

3.3.2 GameObjectManager Class

This class is responsible for managing the objects on the scene. It has a name, gameObject map structure. To find objects with given name fast and efficiently. Table 2 describes the GameObjectManager class.

Method Signature	Return Type	Description
add(String name, GameObject obj)	void	Adds given object with its name to the map structure of the GameObjectManager
remove(String name)	void	Removes the object that has given name from the map structure.
remove(float x, float y)	void	Search in the map structure and removes the object that has given position.
getObject(String name)	GameObject	Returns the game object with given name.
drawAll()	void	Draws all objects to scene.
updateAll()	void	Updates all objects in the manager with required specification.

removeAll()	void	Clears all objects in the manager.

Table 2 – Description of the GameObjectManager Class

3.3.3 GameObject Class

All objects in the game will be inherited from this class. For example, if you want to create a mountain you shall inherit from this class. Then, loadModel() function need to be overloaded with related model. Table 3 describes the GameObject class.

Method Signature	Return Type	Description
transform(String type, float x, float y, float z, float angle)	void	Applies a transformation operation, like scaling, rotation etc., to the object with given parameters.
loadModel()	void	Loads related model to the GameObject.
getPosition()	float[]	Returns the position of the object.
getAngle()	float	Returns the angle of the object.
isDestroyable()	Bool	Returns true if object needs to be destroyed, else return false.
setPosition(float x, float y, float z)	void	Sets the position of the object with given parameters.
setAngle(float angle)	void	Sets the angle of the object with given parameter.
setDestroyable(Bool isDest)	void	Sets 'destroyable' variable of the object with given parameter.

Table 3 – Descriptions of the Game Object Class

3.3.4 Player Class

This class keeps all the information about a player. Table 4 describes the Player class.

Method Signature	Return Type	Description
getScore()	int	Returns the current score of the player.
getPet()	Pet	Returns the pet information of the player.

getInfo()	String[]	Returns all information about player.
setInfo(String[] info)	void	Sets the information of player with given parameters.

Table 4 – Description of the Player Class

3.3.5 Pet Class

This class keeps information about the player's pet. Table 5 describes the Pet class.

Method Signature	Return Type	Description
clean(int amount)	void	Reduces the dirt level of pet.
feed(int amount)	void	Reduces the food need of the pet.
giveMedicine(int amount)	void	Increases the health level of the pet.
giveWater(int amount)	void	Reduces the thirst level of the pet.
getStatus()	int[]	Returns the all-current status levels of pet. (hunger, health, emotion, etc.)
getId()	int	Returns the id of the pet.
getType()	int	Returns the type of the pet.
setId(int pid)	void	Sets the id of the pet.
setType(int t)	void	Sets the type of the pet.
setEmotion(int emo)	void	Sets the emotion of the pet.

Table 5 – Description of the Pet Class

3.3.6 DatabaseManager Class

This class provides an interface between database and the game. Table 6 describes the DatabaseManager class.

Method Signature	Return Type	Description
saveUserData(Player p)	void	Updates given the player info from the database.
loadUserData(int pid)	Player	Returns the pet information of the player.
addPlayer(Player p)	void	Adds the given player to the database.
deletePlayer(int pid)	void	Deletes the player from the database.

Table 6 – Description of the DatabaseManager Class

3.3.7 PCProcessor Class

PCProcessor is the class that is responsible for finding the skeleton from the point cloud by using classification algorithms. Table 7 describes the PCProcessor Class.

Method Signature	Return Type	Description
extractFeatures()	float[]	Extracts the current features of the point cloud.
classify(Skeleton skel)	void	Classifies the parts of the skeleton by using extracted features and writes the result to to given skeleton parameter.
getSkeleton(p)	Skeleton	Returns the classified skeleton.
getGesture(int pid)	Gesture	Recognize the gesture from the classified skeleton and returns it.

Table 7 – Description of the PCProcessor Class

3.3.8 Skeleton Class

Skeleton is the class which stores the joints of the skeleton found in the point cloud. This class is a singleton class. Table 8 describes the Skeleton class.

Method Signature	Return Type	Description
setJoints(Joint[] joints)	void	Sets the joints according to the given vector of joints
getJoints()	Joint[]	Returns the current vector of joints.

Table 8 – Description of the Skeleton Class

3.3.9 OyunDiyari Class

This class is responsible for displaying the games' user interface and handles the interactions between player and the game. Table 9 describes the OyunDiyari Class.

Method Signature	Return Type	Description
showMainMenu()	void	Shows the main menu of the game
showMap()	void	Display to the map of the game so that the player can selects a level
showSplashScreen()	void	Displays the splash screen of the game.
startLevel(Level levc)	void	Constructs and displays the selected level and also handles the interactions
quit()	void	Allows the player to quit the game.

Table 9 – Description of the OyunDiyari Class

3.3.10 Sensor Class

This class is an abstract class. Every camera that we used must be inherit from this class and define its functions. Table 10 describes the Sensor class.

Method Signature	Return Type	Description
initialize()	void	Initializes the camera sensor by using camera API.
getPointCloud()	float[]	Returns the point cloud that comes from camera.

Table 10 – Description of the Sensor Class

3.4 Dependency Viewpoint

Interconnections and interfaces between Oyun Diyarı entities are explained in dependency viewpoint. The parameterizations of interfaces between entities are found in the component diagram of the system in Figure 5. Since the order of executions will be explained in Section 3.9, which is interaction viewpoint, this section does not mention the order of executions.

3.5 Information Viewpoint

Information viewpoint describes the data of the system in the database tables by using ER diagram with refinements from SRS of Oyun Diyarı. This viewpoint also contains management and manipulation information of database structure of Oyun Diyarı. Figure 8 shows the entity relationship diagram of Oyun Diyarı. Under the figure, each entity is described in terms of its fields and their descriptions see Table 11 and Table 12.

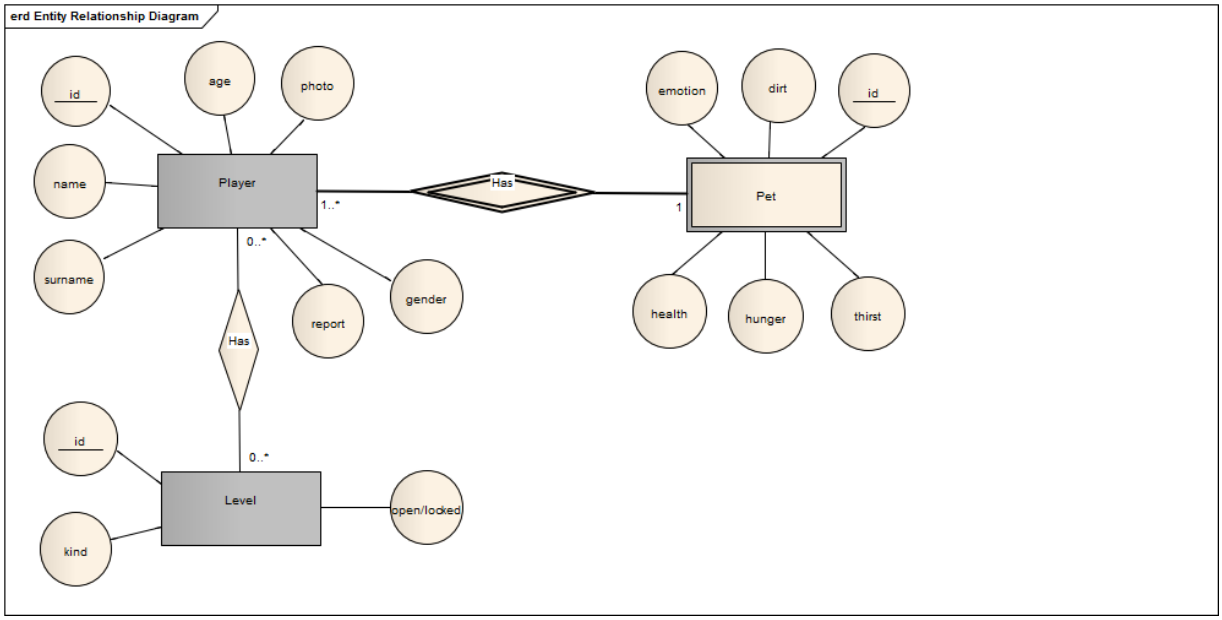

Figure 8 – Entity Relationship Diagram

Entity	Description	Relations
Player	Holds the players' data that added the system as player.	Each player can have more than one level. However each player must have exactly one pet.
Level	Holds all information of game levels about players.	Each level can be had by more than one player.
Pet	Holds all information of pet that is hold by a player.	Each pet must had by exactly one player. In other words, pets do not exist without player.

Table 11 – Entity Descriptions and Relations

Entity	Field Name	Type	Description
Player	id	int	It is the unique id which each player has its own, and the primary key of the table.
	name	varchar	It is name of the player.
	surname	varchar	It is surname of the player.
	age	int	It is age of the player.
	gender	int	It is gender of the player.
	photo	jpg	It holds profile photo of the player.
	report	varchar	It holds parental report of the player.
Level	id	int	It is the unique id which each level has its own, and the primary key of the table.
	kind	int	It is the kind of the level.
	open/locked	int	It holds the information about the level that can be played by user or not.
Pet	id	int	It is the unique id which each pet has its own, and the primary key of the table.
	dirt	int	It holds dirt level of the pet.
	thirst	int	It holds thirst level of the pet.
	hunger	int	It holds hunger level of the pet.
	health	int	It holds health level of the pet.

	emotion	int	It holds emotion of the pet

Table 12 – Descriptions of the Fields of the Entities

3.6 Patterns Viewpoint

Design pattern is a general usable solution template to a commonly occurring problem. Design ideas as collaboration patterns, which involve abstracted roles and connectors, are described in patterns viewpoint.

A design pattern is a general usable solution template to a commonly occurring problem. In this viewpoint, design ideas as collaboration patterns, which involve abstracted roles and connectors, are described. In Oyun Diyarı system, the Event-Driven Architecture (EDA) is used as architectural pattern and singleton and producer-consumer are used as design patterns.

EDA is an architectural pattern which has four main event flow layers. The first layer is event generator which is the starting point of the flow. The event generator senses a fact and represents the fact into an event. In Oyun Diyarı system, the 3D sensor of the camera is the event generator and the fact is the point cloud.

The second layer is the event channel which is a mechanism whereby the information from an event generator is transferred to the event engine (will be explained later). The events processed asynchronously, the events are stored in a queue, waiting to be processed later by the event processing engine. In Oyun Diyarı system, Sensor Class represents the event channel by obtaining the point cloud from the 3D sensor and put it into the queue to be processed by the Point Cloud Processor Class.

The third layer is the event processing engine where the event is identified, and the appropriate reaction is selected and executed. Our event identification is based on recognition of the skeleton. In Oyun Diyarı system, Point Cloud Processor Class represents the event processing engine by processing the point cloud in the queue sequentially.

The last layer is event-driven activity where the consequences of the event are shown. The event-driven activity is represented by Oyun Diyarı Class in our system by displaying the graphical user interface and handling the interactions between game and the user.

The Producer – Consumer is a design pattern which is based on the idea of separating work that needs to be done from the execution of that work. This means that the separation of the work that needs doing from the execution of that work is achieved by the Producer placing items of work on the queue for later processing instead of dealing with them the moment they are identified (see Figure 9). The Consumer is then free to remove the work item from the queue for processing at any time in the future. This decoupling means that Producers don't care how each item of work will be processed, how many consumers will be processing it, or how many other producers there are.

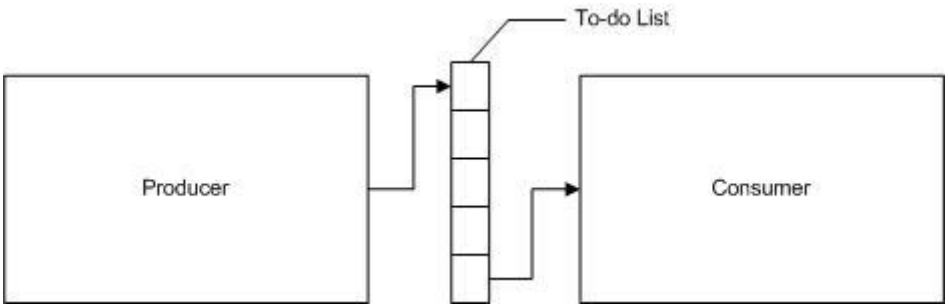

Figure 9 – The producer – Consumer Pattern

In Oyun Diyarı system, the Producer part of the pattern is Point Cloud Processor Class which produces the singleton by processing the point cloud and the Consumer part of the pattern is Oyun Diyarı which makes use of the skeleton produced by Point Cloud Processor Class.

The singleton pattern is a design pattern that restricts the instantiation of a class to one object. This is useful when exactly one object is needed to coordinate actions across the system. The concept is sometimes generalized to systems that operate more efficiently when only one object exists, or that restrict the instantiation to a certain number of objects. In OyunDiyari system, there are two singleton classes: OyunDiyari and DatabaseManager.

3.7 Interface Viewpoint

Interface viewpoint provides detailed explanations for external and internal interfaces of Oyun Diyarı for designers, programmers and testers. In other words, it is also a base source for users about how to use interfaces of Oyun Diyarı correctly.

3.7.1 Overview of User Interface

When the player enters Oyun Diyarı, s/he encounters the welcome screen. In this screen, player can go to main menu, profile manager screen or exit from the game. In main menu screen, player can go to game map, go back to the welcome screen, can show his/her profile page or can exit from the game. If player goes to the map screen, s/he can go back to the main menu, can exit from the game or can select a level that s/he wants to play. On the other hand if the player goes to profile manager screen, s/he can add a new user, delete an existing user, edit an existing user or read the progress report of the user. On this page, if the player goes add new user screen, s/he can see some blanks to fill with new user's information.

3.7.2 Screen Images

This section provides detailed descriptions of the user interface with the help of screen images. Although all screens haven't been created yet, main samples will be provided. The screenshots of this section are created by using Balsamiq Mockup software up to now. All of the functionalities in this section user will control a cursor with his/her hands.

3.7.2.1 Welcome Screen

This screen will be showed when the game started. It is our opening page. How it will see is shown in Figure 10. In the welcome screen player can go to the main menu, can go to the profile manager screen and can exit from the game.

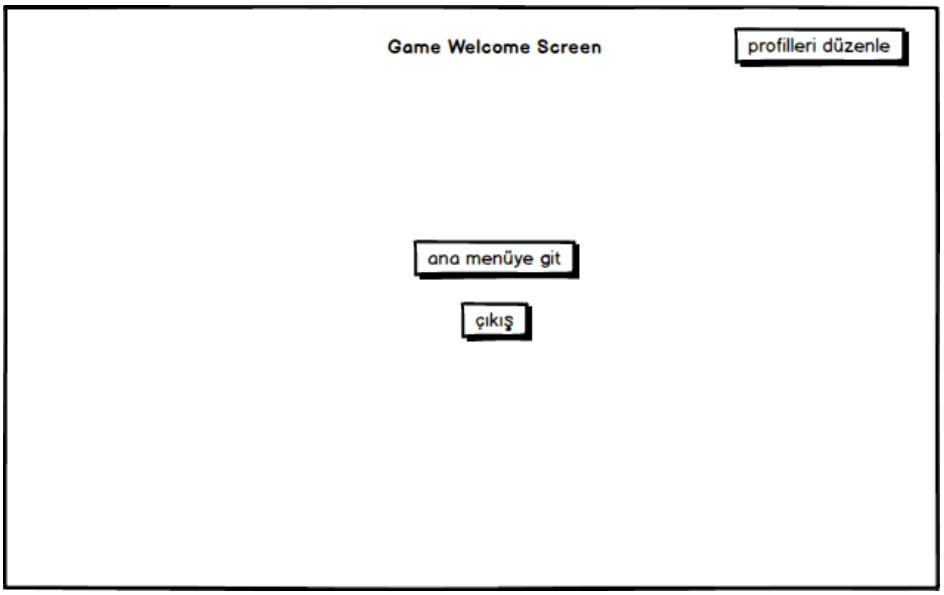

Figure 10 - Game Welcome Screen

3.7.2.2 Main Menu Screen

In this screen user can go to game map, go back to the welcome screen, can show his/her profile page and can exit from the game. This screen can be seen in Figure 11.

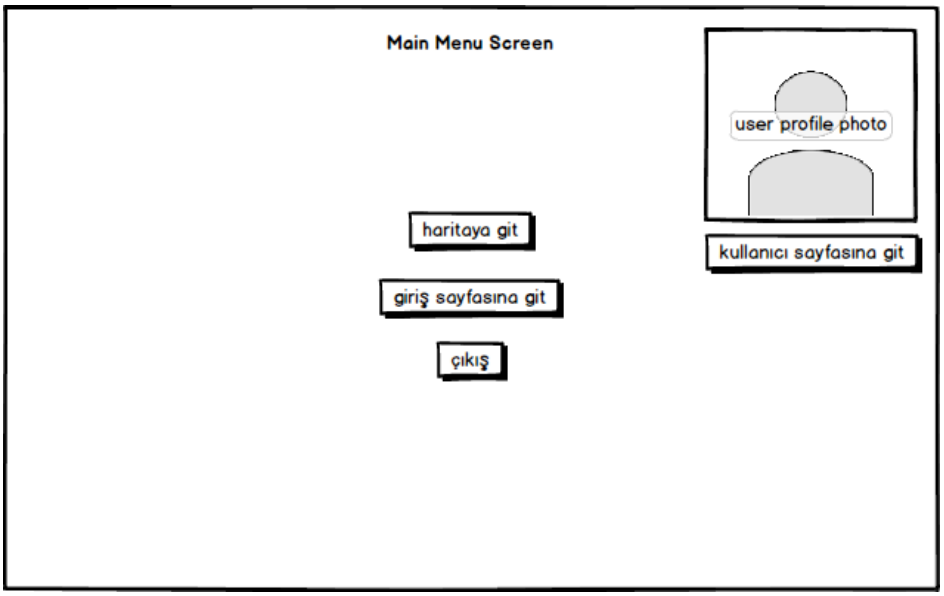

Figure 11 - Main Menu Screen

3.7.2.3 Profile Manager Screen

In this screen user will see current user list in his/her game. The user can add a new user, delete an existing user, edit an existing user, or read the progress report of the user in this screen.

This screen can be seen from Figure 12. When the user click to the add user, the “add new user pop-up” will be seen on the screen. The pop-up screen of this functionality is shown at Figure 13. In the add user pop-up screen user shall fill the user’s name, last name, age, and gender. Also, s/he must take the photo of the user from the existing camera. After giving all the information, user clicks the save button and adding operation has been finished.

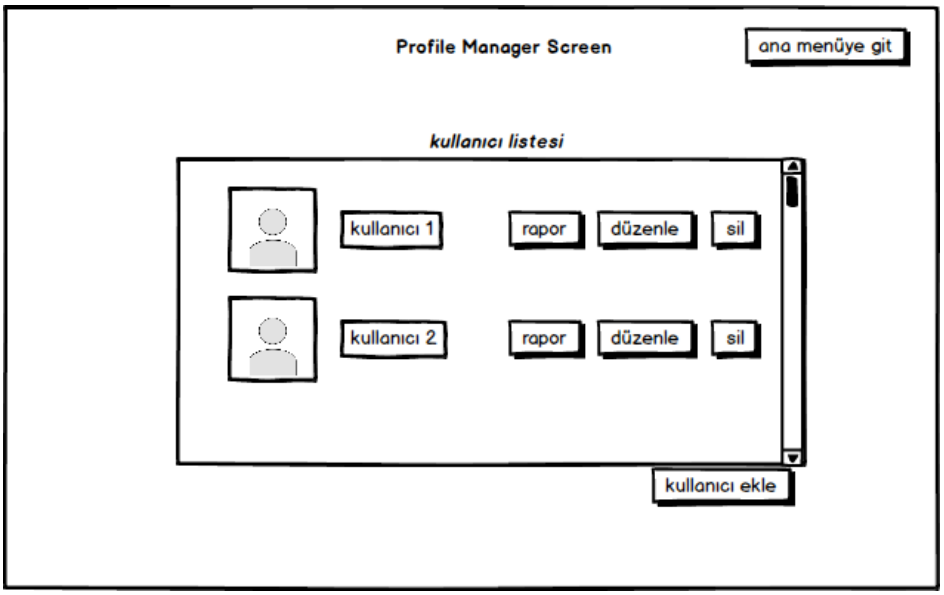

Figure 12 - Profile Manager Screen

Moreover, user can edit an existing user on the profile manager screen. Since the screen will almost look like “add new user pop-up screen” we will refer to this pop-up screen. Editing operation is almost same as adding operation. The only difference this time pop-screen will be shown with pre-filled with chosen user’s info. Also, user can delete an existing user from user list. When the user is deleted, it shall disappear from the list.

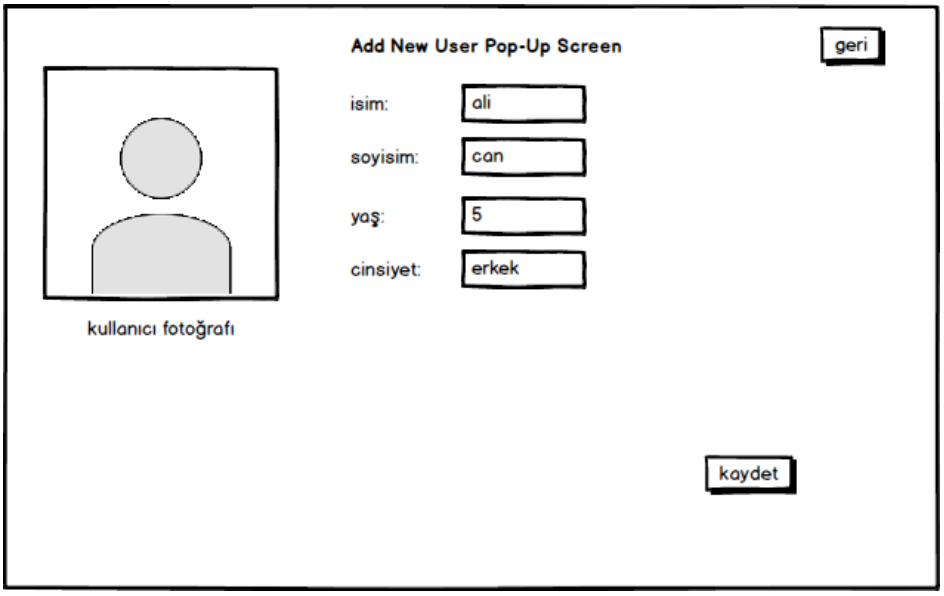

Figure 13 - Add New User Pop-Up Screen

Read report functionality of the game will be created with the domain experts. What it includes is not certain for now

3.7.2.4 Map Screen

In this screen user will see our game levels as a map. Mock up organization of this screen is shown at Figure 14. In this screen, the user can go back to the main menu, can exit from the game, or can select a level that s/he wants to play. The organization of these levels will be arranged by

consulting the child experts. The hardness of levels and which level includes which educational purpose will be also done by consulting the experts.

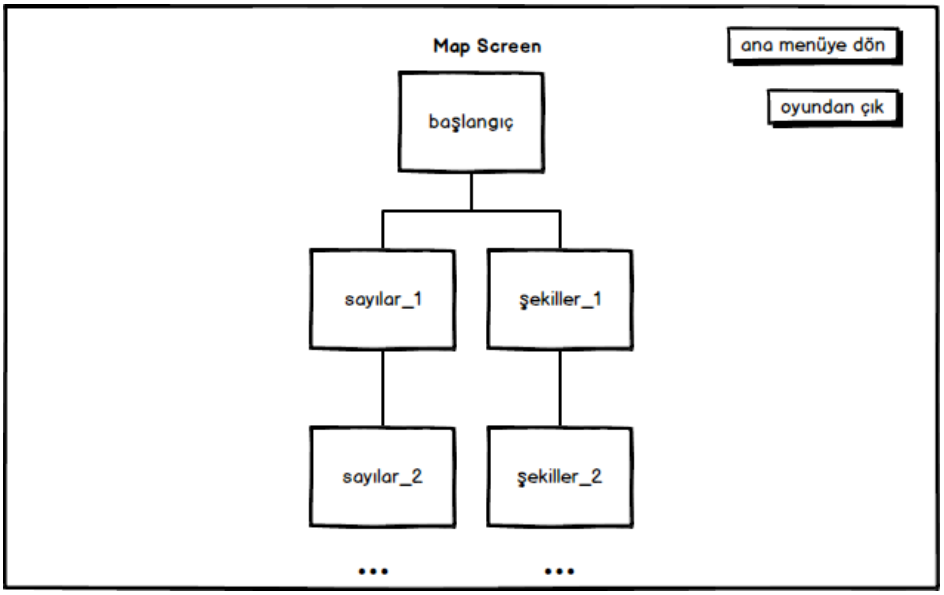

Figure 14 - Map Screen

3.7.3 Screen Objects and Actions

Page	Button	Action
Welcome Screen	Edit profiles	Directs to Profile Manager Screen.
	Go to main menu	Directs to main menu.
	Exit	Exit from the game.
Main Menu Screen	Go to Map	Directs to game map.
	Go to welcome screen	Directs to welcome screen.
	Go to profile page	Directs to profile page.
	Exit	Exit from the game.
Profile Manager Screen	Go to main menu	Directs to main menu.
	Report	Open parental report of selected user.
	Edit	Edit the selected user.
	Delete	Delete the selected user.
	Add user	Pops up add new user screen.
Add New User Screen(Pop - Up)	Back	Directs to profile manager screen.
	Save	Save the user information and create new user.
Map Screen	Go to main menu	Directs to main menu.
	Exit	Exit from the game.
	Play	**Starts the selected game**

Table 13 – Description of the Screen Actions

3.8 Structure Viewpoint

The structural view of the software is important as much as the architecture of the system, therefore it is important to explain the structure viewpoint. Since the OyunDiyari system uses EDA architectural pattern, it depends on EDA characteristics and structure. EDA has four main components: the event generator, the event channel, the event processing engine and the event activity. The event generator, which is the 3D sensor in our case, pushes the event data to the event processor engine to process the event data via event channel. After the process, event processor engine sends the results to the event activity which is responsible for outputting the results. The Figure 15 shows the structure diagram of the OyunDiyari system.

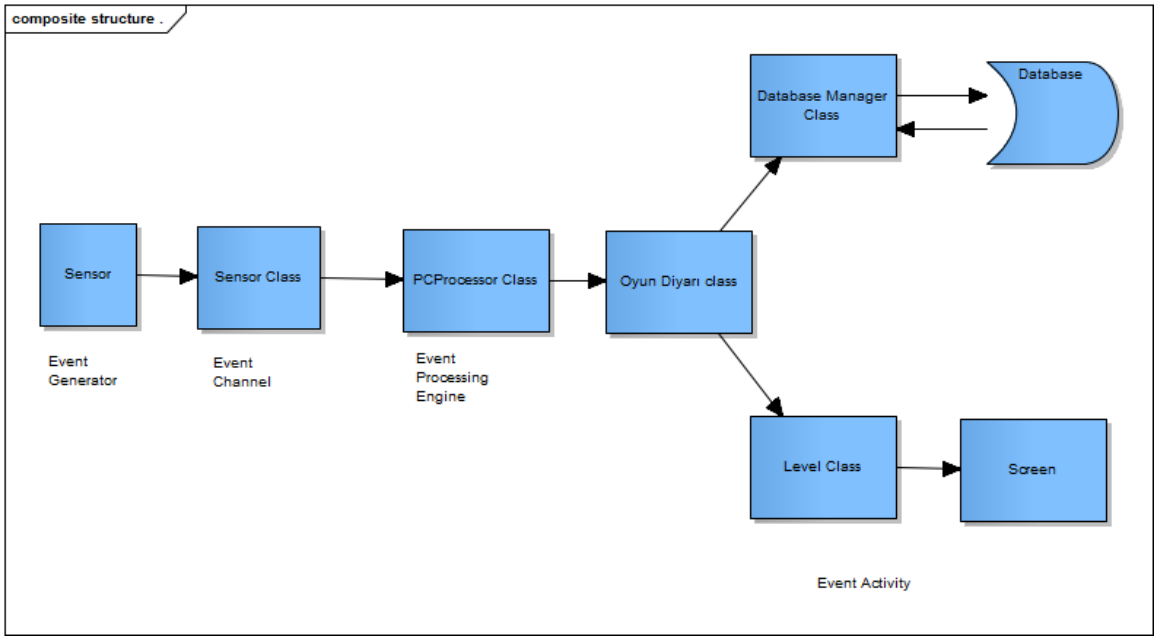

Figure 15 - Structure Diagram

3.9 Interaction Viewpoint

Interactions between users and objects of Oyun Diyarı and data flow are described in interaction viewpoint by using sequence diagrams. Following sections provides sequence diagrams the use cases described in 3.1.

3.9.1 Gesture Recognition Interaction

This section describes the interactions between the player and the objects of the system while recognizing the gestures of the player. The gesture recognition is included in all the use cases since the all interactions are provided by it. All the levels are played with gestures, and the buttons and the cursor are controlled by gestures. In order not to repeat in each sequence diagram, it is explained only in Figure 16.

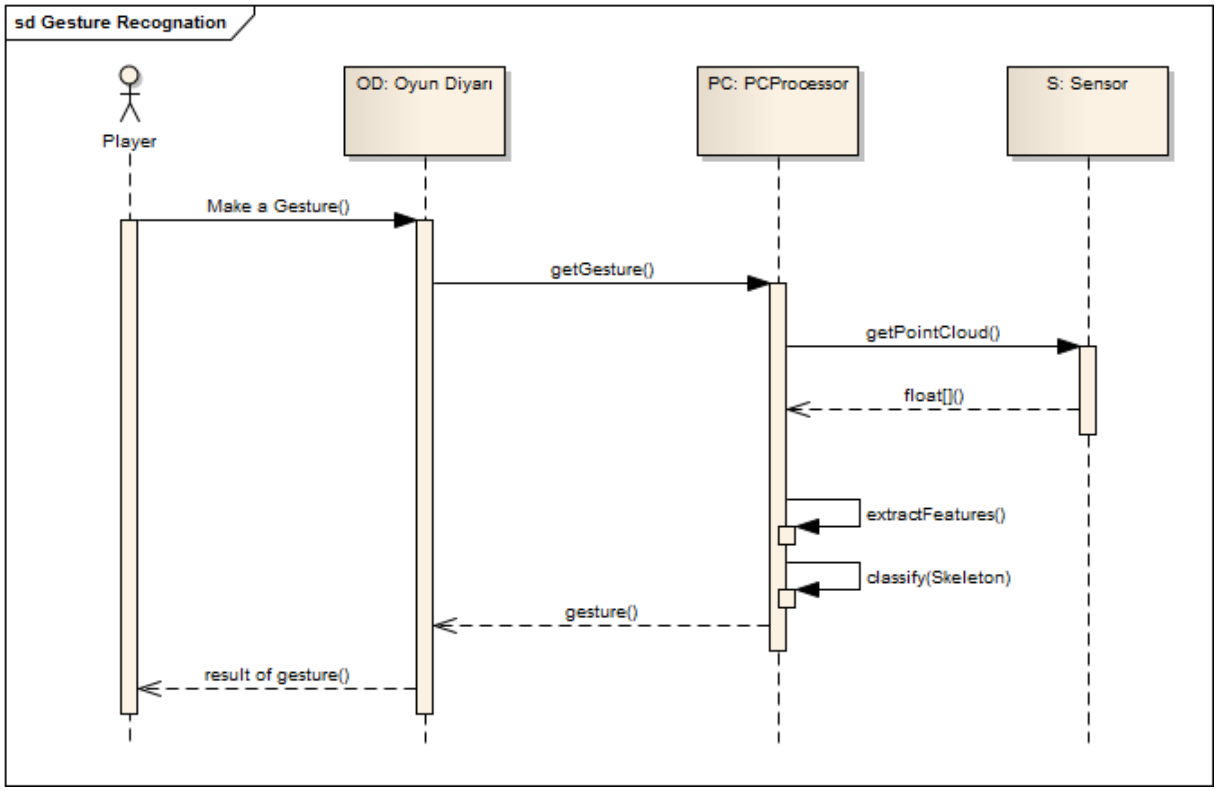

Figure 16 – Gesture Recognition Sequence Diagram

3.9.2 Interactions between the System and the Actor Parent

This section describes the sequence of interactions and messages passed through them between the actor parent and objects of the system. Figure 17, Figure 18, Figure 19, and Figure 20 shows the sequence diagrams for the use cases UC1, UC2, UC3, and UC4 respectively.

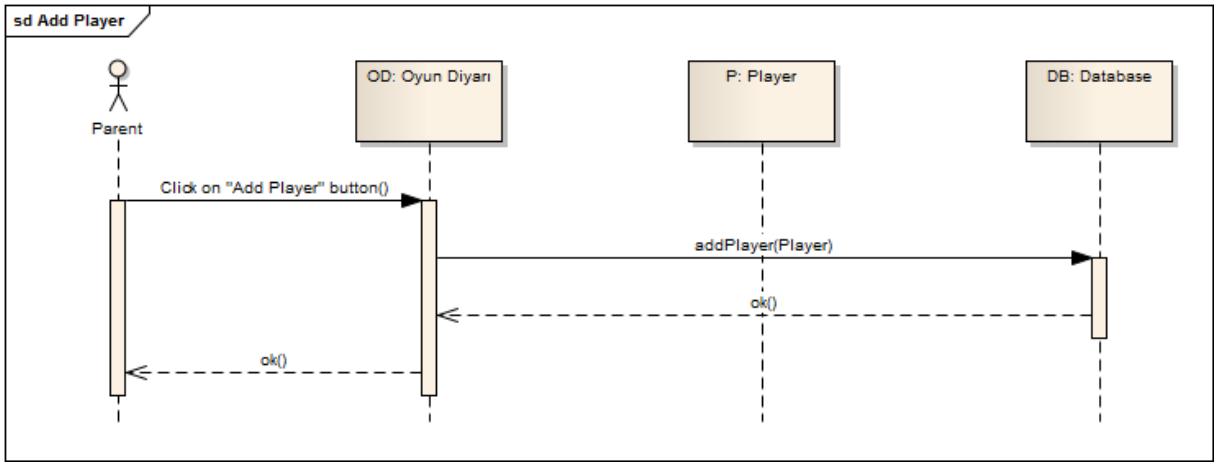

Figure 17 – Add Player Sequence Diagram

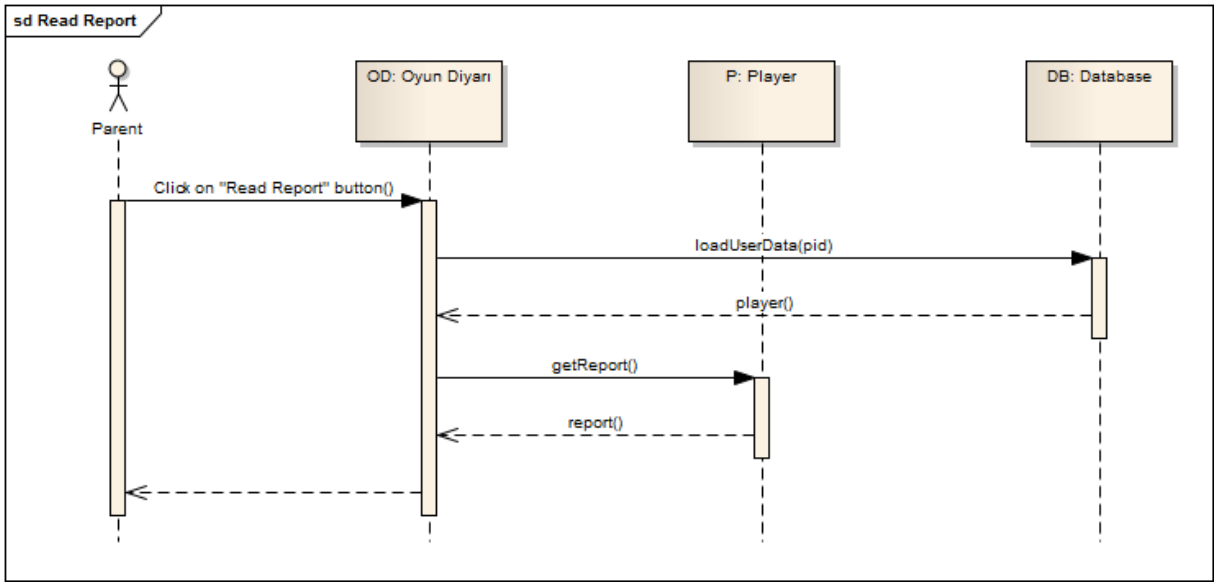

Figure 18 – Read Report Sequence Diagram

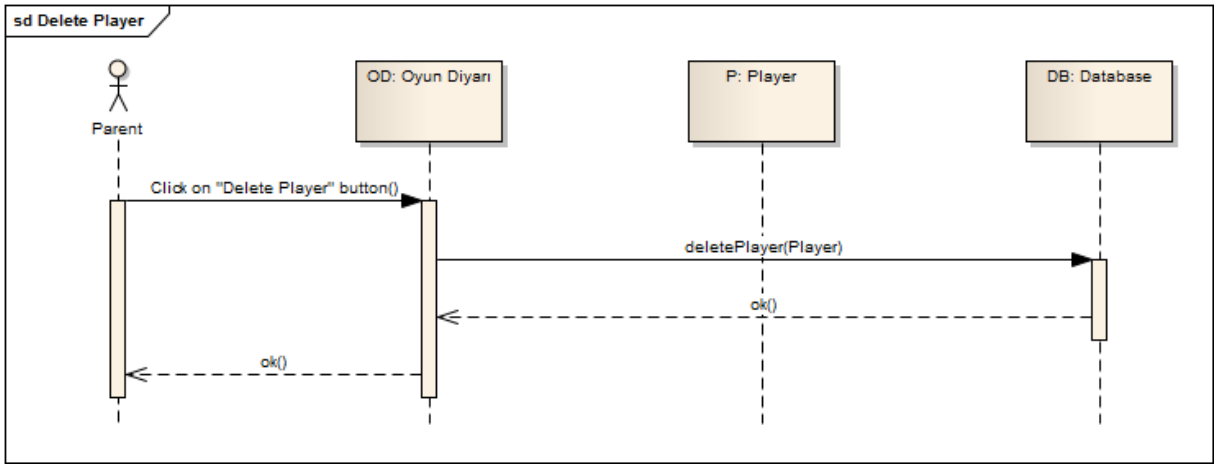

Figure 19 – Delete Player Sequence Diagram

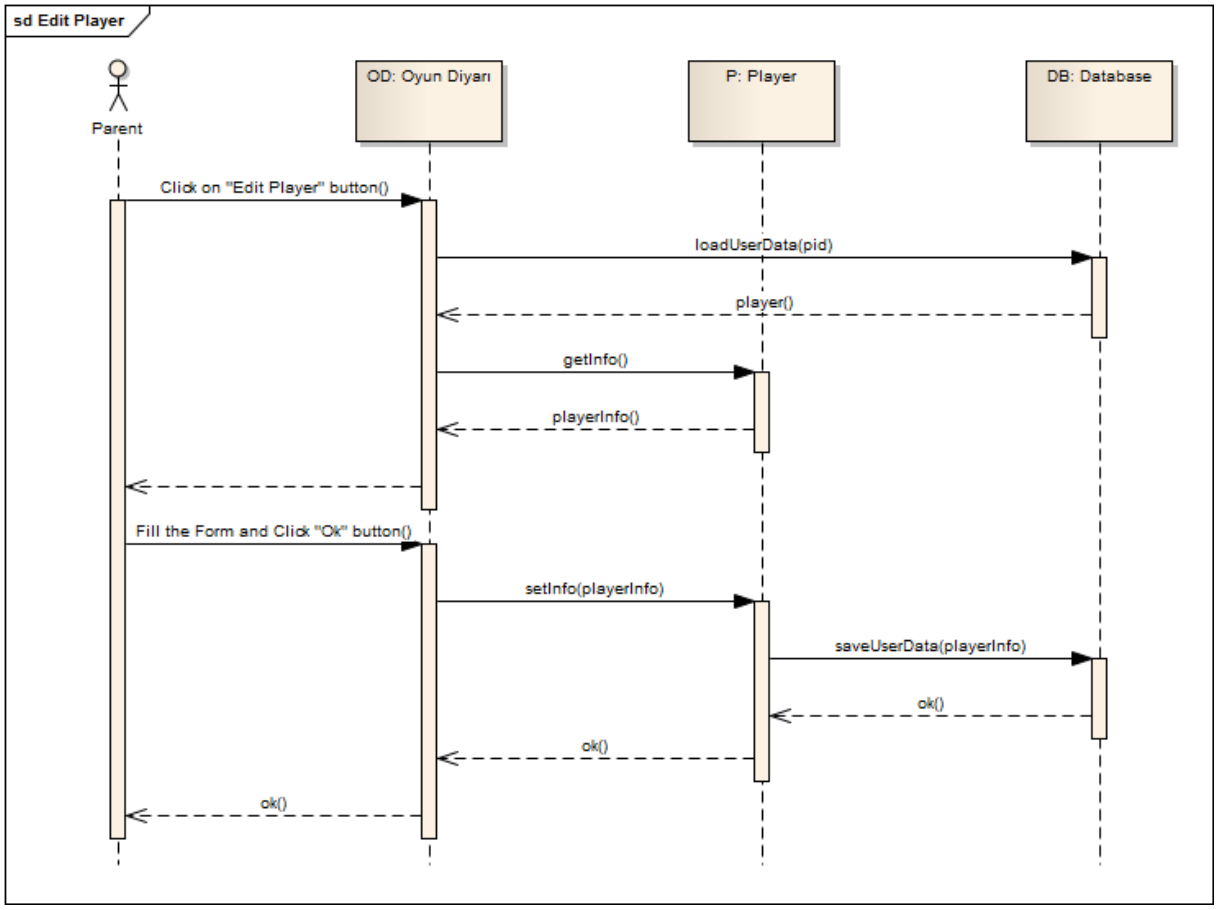

Figure 20 – Edit Player Sequence Diagram

3.9.3 Interactions between the Actor Player and the System while Starting a Level

This section describes the sequence of events when a player wants to start a level to play. It includes the use cases UC6 and UC12. Figure 21 shows the sequence diagram that explains how a level is started.

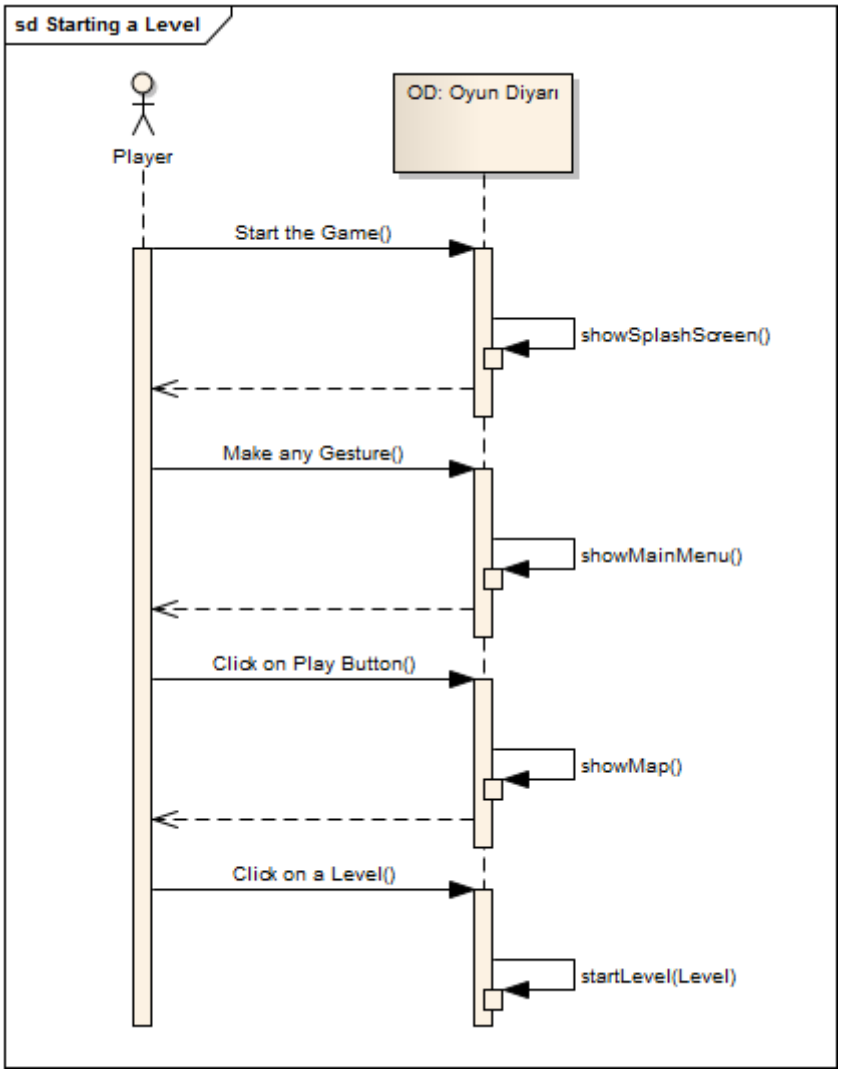

Figure 21 – Starting a Level Sequence Diagram

3.9.4 Interactions between the Actor Player and the System while Playing a Level

This section describes the main game loop. Since it is same for each level except the game objects and the way they are updated each frame, the playing level interaction is shown as a single sequence diagram in Figure 22. It is responsible for the use cases UC6, UC8, UC10, UC11, and UC13 to UC35.

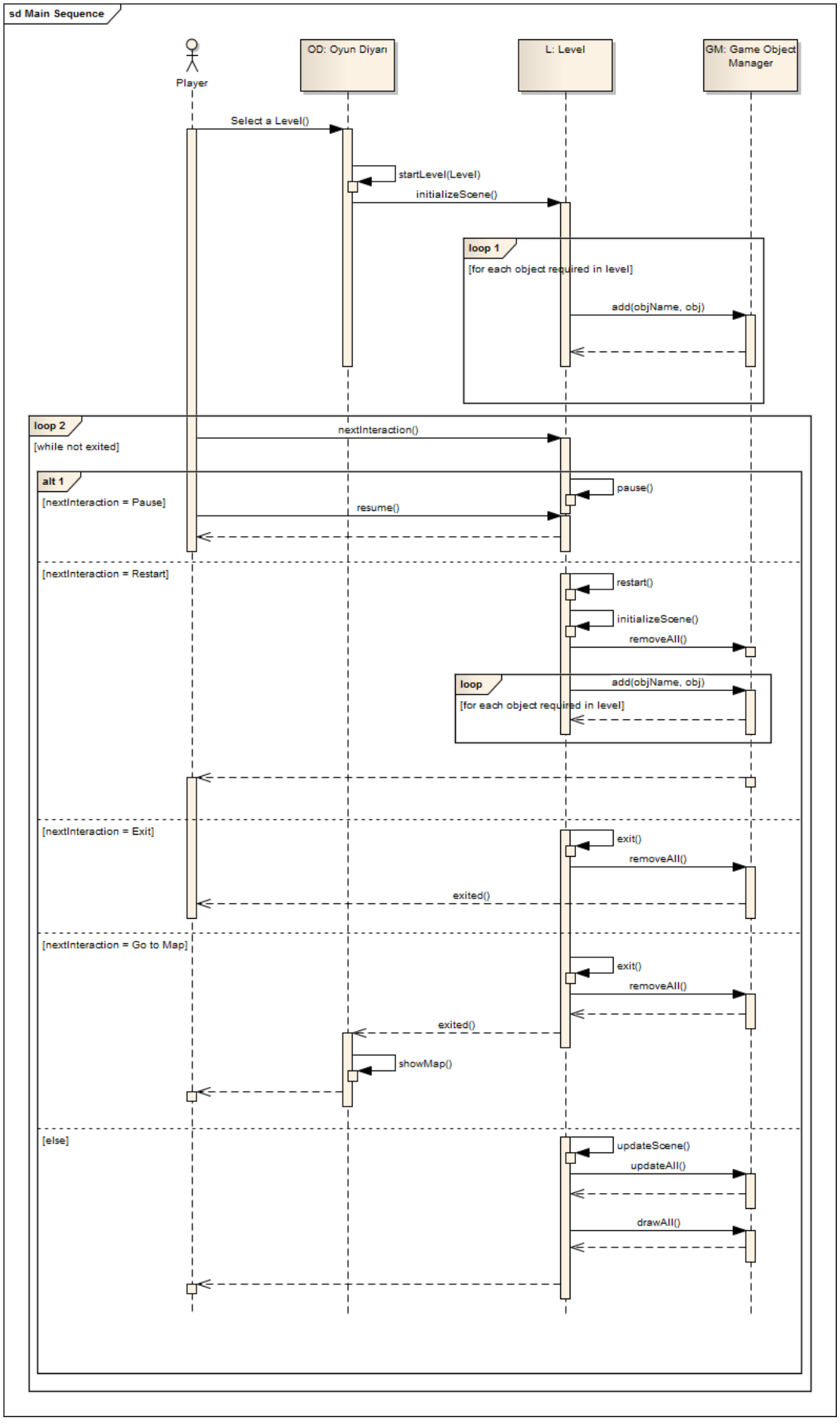

Figure 22 – Game Loop Sequence Diagram

3.10 State Dynamics Viewpoint

The states of Oyun Diyarı, transitions between them and the responses to the user events are the main concern of state dynamics viewpoint. Figure 23 shows the state diagram of Oyun Diyarı which includes all states of Oyun Diyarı and transitions between them with the causing events.

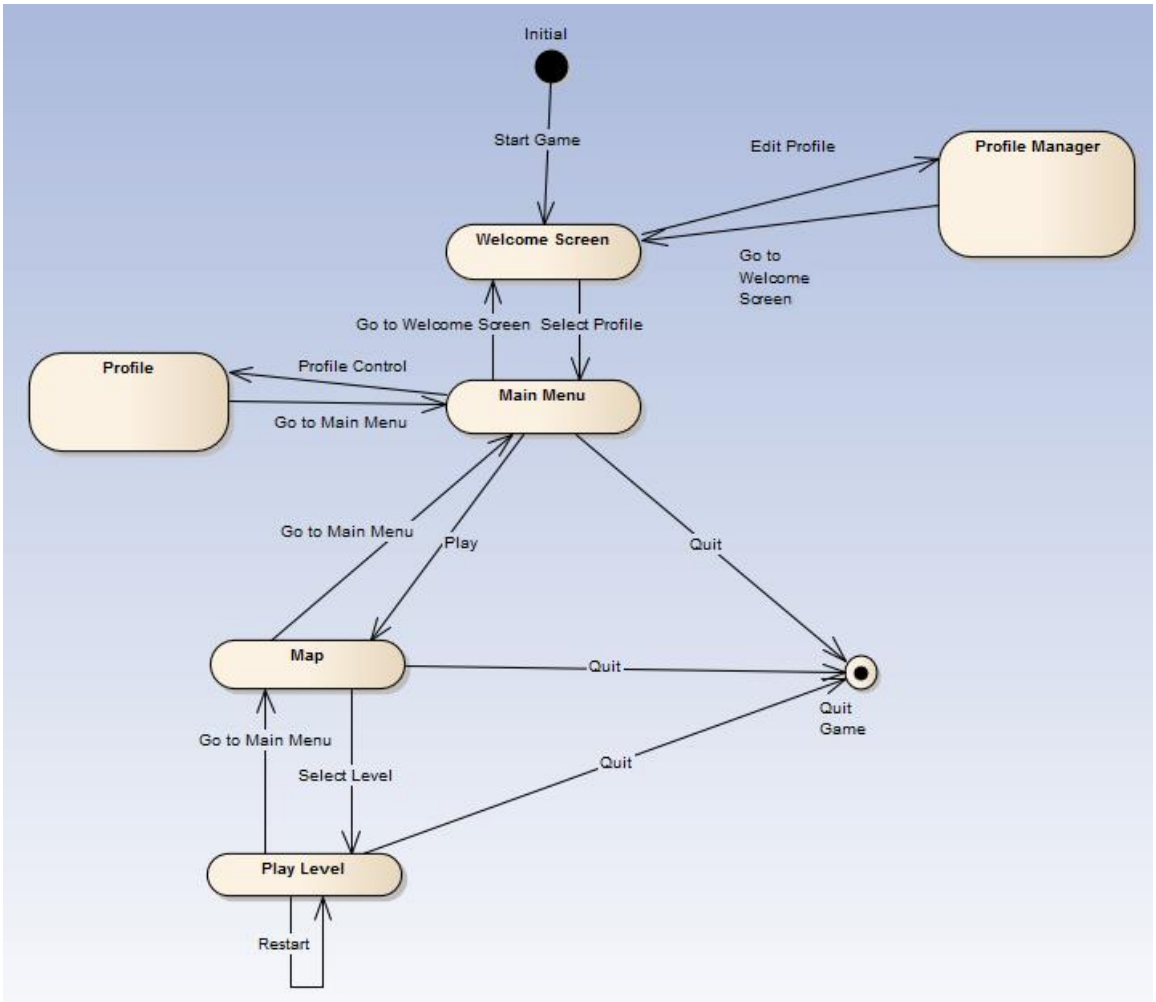

Figure 23 - State Diagram

4 Appendices

4.1 Table of Figures

4.2 Table of Tables