**REVISION HISTORY**

| **Date** | **Version** | **Description** | **Author** |
| --- | --- | --- | --- |
| 17.11.2024 | 1.1 | Added 3.3 Application Layer and all of its subparts (3.3.1, 3.3.2, 3.3.3 and 3.3.4) | D. Yağmur Uğut  Poyraz Köroğlu |
| 17.11.2024 | 1.2 | Added 3.1 Presentation Layer and all of its subparts | Doruk Esen |
| 17.11.2024 | 1.3 | Added 3.2.2 diagram. | Eda Nur Yılmaz |
| 17.11.2024 | 1.4 | Added 3.2.1 Component Interfaces | Ömer Emre Bozkurt |
| 17.11.2024 | 1.5 | Added 3.2.3 Workflows and algorithms and 3.2.4 Software requirements mapping | Ömer Emre Bozkurt,  Eda Nur Yılmaz |
| 18.11.2024 | 1.6 | Updated Contents | Ömer Emre Bozkurt |
| 24.11.2024 | 1.7 | Updated 3.3.2 | Poyraz Köroğlu |
| 24.11.2024 | 1.8 | Updated 3.1.2 and 3.1.3 | Doruk Esen |
| 24.11.2024 | 1.9 | Updated 3.2.1, 3.2.2 and 3.2.3 | Eda Nur Yılmaz, Ömer Emre Bozkurt |

**TABLE OF CONTENTS**

[**1 Introduction 3**](#_heading=)

[**1.1 References 3**](#_heading=)

[1.1.1 Project References 3](#_heading=h.3znysh7)

[**2 Software Architecture Overview 3**](#_heading=h.2et92p0)

[**3 Software design description 4**](#_heading=)

[**3.1 GUI (Presentation Layer) 4**](#_heading=)

[3.1.1 Component interfaces 4](#_heading=h.1t3h5sf)

[3.1.2 Component design description 4](#_heading=)

[3.1.3 Workflows and algorithms 5](#_heading=h.h1vqm37lw77o)

[3.1.4 Software Requirements Mapping 5](#_heading=)

[**3.2 Model (Domain Layer) 5**](#_heading=)

[3.2.1 Component interfaces 5](#_heading=)

[3.2.2 Component design description 6](#_heading=)

[3.2.3 Workflows and algorithms 6](#_heading=h.35nkun2)

[3.2.4 Software requirements mapping 7](#_heading=)

[**3.3 Player Controls (Application Layer) 7**](#_heading=h.o66vfklim0r4)

[3.3.1 Component interfaces 7](#_heading=)

[3.3.2 Component design description 8](#_heading=)

[3.3.3 Workflows and algorithms 10](#_heading=h.fo2hd0j6z7ut)

[3.3.4 Software requirements mapping 10](#_heading=)

[**4 COTS Identification 10**](#_heading=)

# **Introduction**

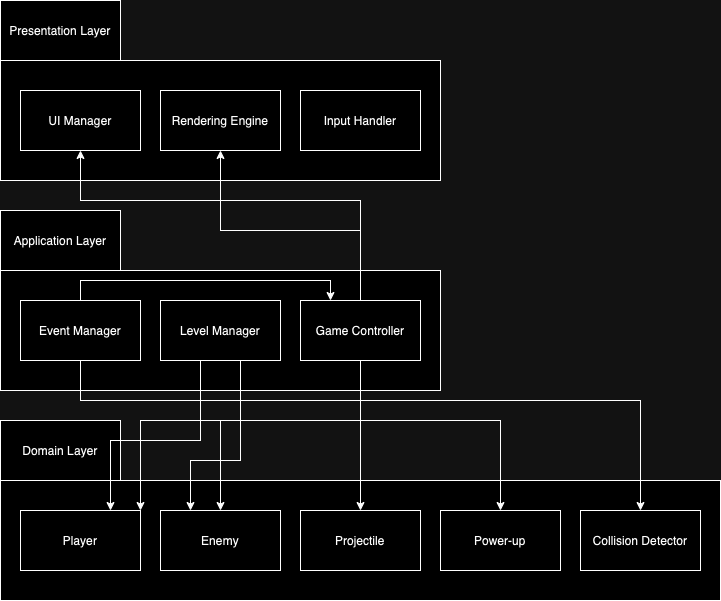
This document describes the design of the SB software system.

## ***References***

### **Project References**

| # | Document Identifier | Document Title |
| --- | --- | --- |
| [SRS] | SB-SRS-1 | SB Software Requirements Specifications |

# **Software Architecture** Overview



# **Software design description**

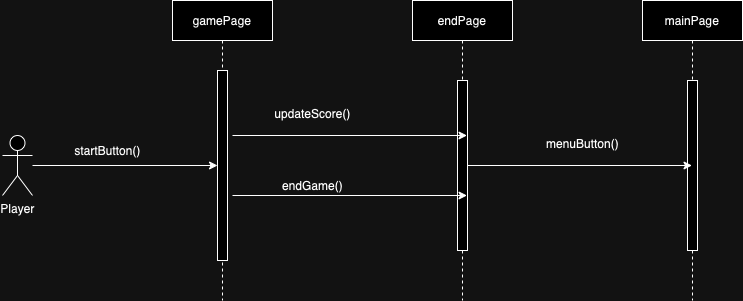
## GUI (Presentation Layer)

### **Component interfaces**

Once the software has started the user sees the “mainPage” containing clickable boxes titled as Start, Volume, and Exit and a Non-clickable box titled High-Scores. If the player hits the “Volume” button a window will appear with a slide bar. On this bar, the user will be able to adjust the volume. The game will terminate if the player clicks on the “Exit” button. Under the box titled “High-Scores” the highest 3 scores that the player has acquired up until that point will be shown. Once the player hits the “Start” button, the game will switch to “gamePage”. In the game screen, the user will see Enemies, the Paddle, the number of balls left, the current ball, the total score, and the current level. After the game ends “endPage” will appear showing the total score and a return button that returns to the “mainPage”.

### **Component design description**

### **Workflows and algorithms**



### **Software requirements mapping**

SRS-SB-007.1

SRS-SB-008.2

SRS-SB-010.1

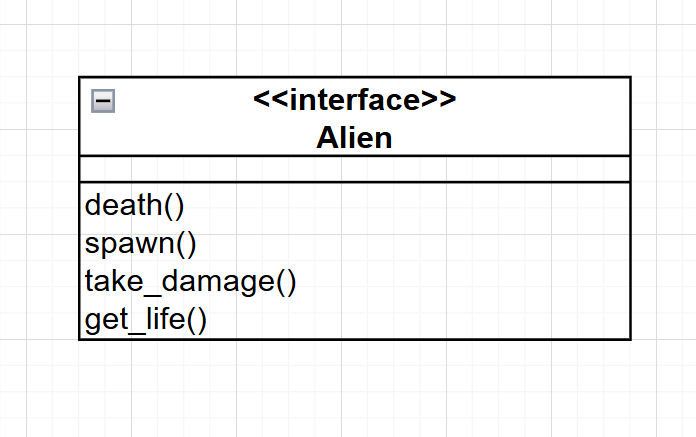
## Model (Domain Layer)

### **Component interfaces**

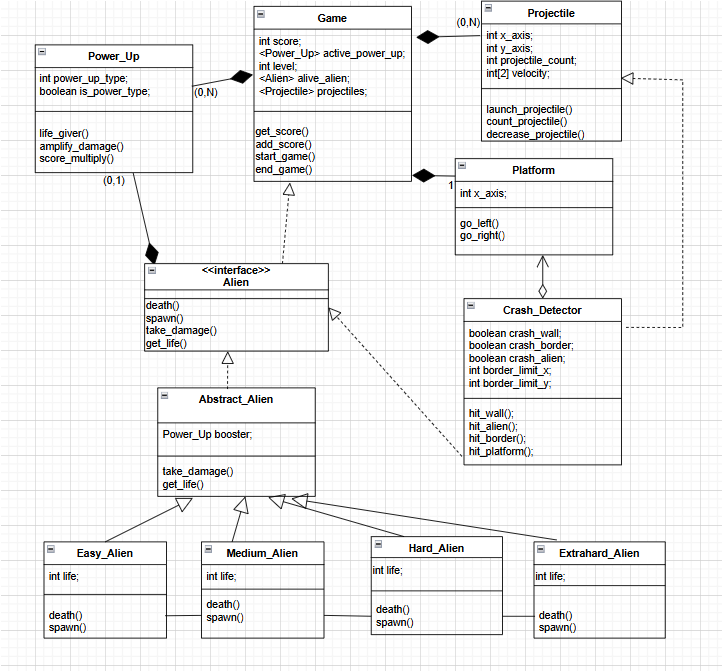
The game contains 3 types of moving objects which are Projectile, Platform, and Alien.

<<Alien>>

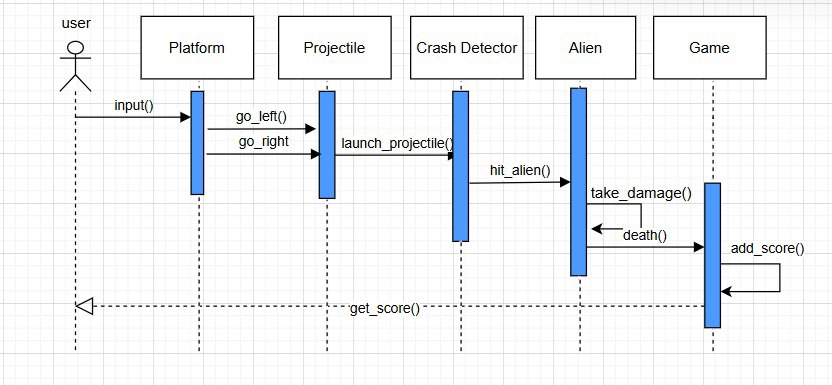
Each alien has a health value that decreases upon being hit by a projectile. An alien is eliminated when its health reaches zero. When an alien is eliminated, the player earns points based on the type of alien. The type and difficulty of aliens vary based on the progression of the game level. Additionally, some aliens may carry power-ups, which are dropped upon their elimination.

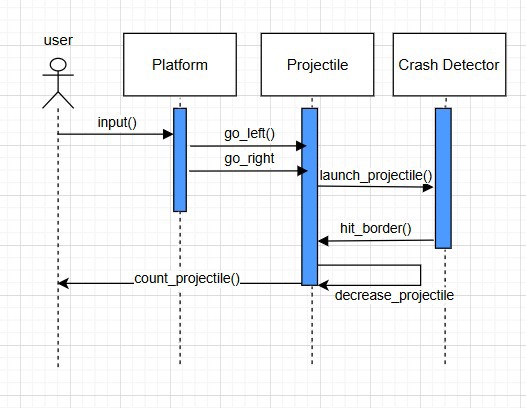


### **Component design description**



### **Workflows and algorithms**





### **Software requirements mapping**

SRS-SB-004.1

SRS-SB-005.1

SRS-SB-005.2

SRS-SB-008.1

SRS-SB-011.1

SRS-SB-013.1

SRS-SB-014.1

## Player Controls (Application Layer)

### **Component interfaces**

It has 3 classes:

1. **PlayerControlsManager**:

* Acts as the controller for player input.
* Interprets keyboard inputs and invokes methods in the Platform and Game classes.

1. **Platform**:

* Represents the player-controlled platform.
* Handles movement logic (left/right).

1. **Game**:

* Manages the overall game state (e.g., score, ball count, power-ups, level).
* Synchronizes game rules with player actions.

Input:

* Player input commands (e.g., keyboard arrow keys for left and right movement).
* Game state updates from the Game class (e.g., ball\_count and level).
* Game events from the Event Manager to enable or disable controls based on the game state (e.g., "pause", "game over").

Output:

* Updates to the Platform object's position (go\_left(), go\_right()).
* Communication with the Game class for any game state adjustments (e.g., triggering events on level transitions).

### **Component design description**

The Player Controls component facilitates interaction between the player and the game environment, ensuring smooth and responsive platform movement. It connects with the Game class to monitor and respond to the current game state, such as active power-ups, score updates, and ball count.

PlayerControlsManager

has 4 attributes:

1. Platform platform: Reference to the controlled platform.
2. Game game: Reference to the main Game instance.
3. int boundary\_left: Define movement limits.
4. int boundary\_right: Define movement limits.

has 3 methods:

1. processInput(input): Interprets player commands and invokes appropriate platform methods.
2. updatePosition(): Adjusts the platform's position while ensuring it remains within boundaries.
3. handleGameState(): Adjusts platform behavior based on Game state (e.g., pausing on game over).

Platform

has 1 attribute:

1. int x: Current x-coordinate of the platform.

has 2 methods:

1. go\_left(): Moves the platform left.
2. go\_right(): Moves the platform right.

Game:

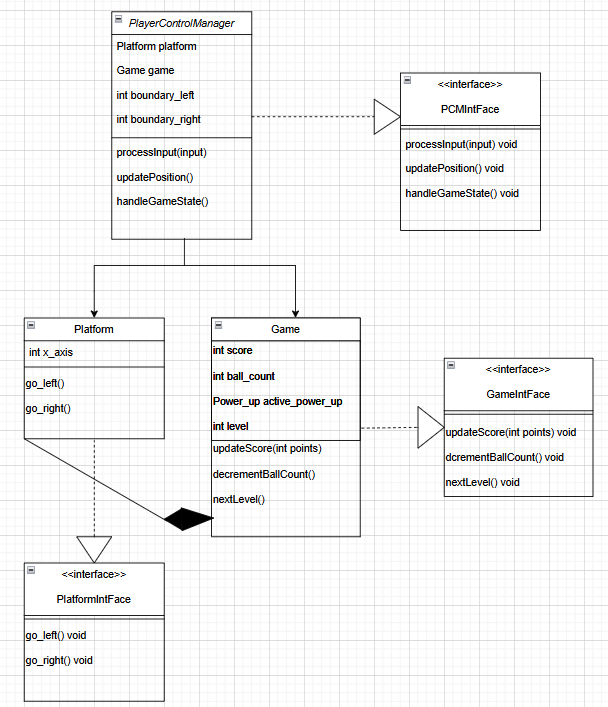
has 4 attributes:

1. int score: Tracks the plater’s current score. (initially 0)
2. int ball\_count: Number of balls left (starts at 3).
3. power\_up active\_power\_up: Tracks any active power-up.
4. int level: Current game level.

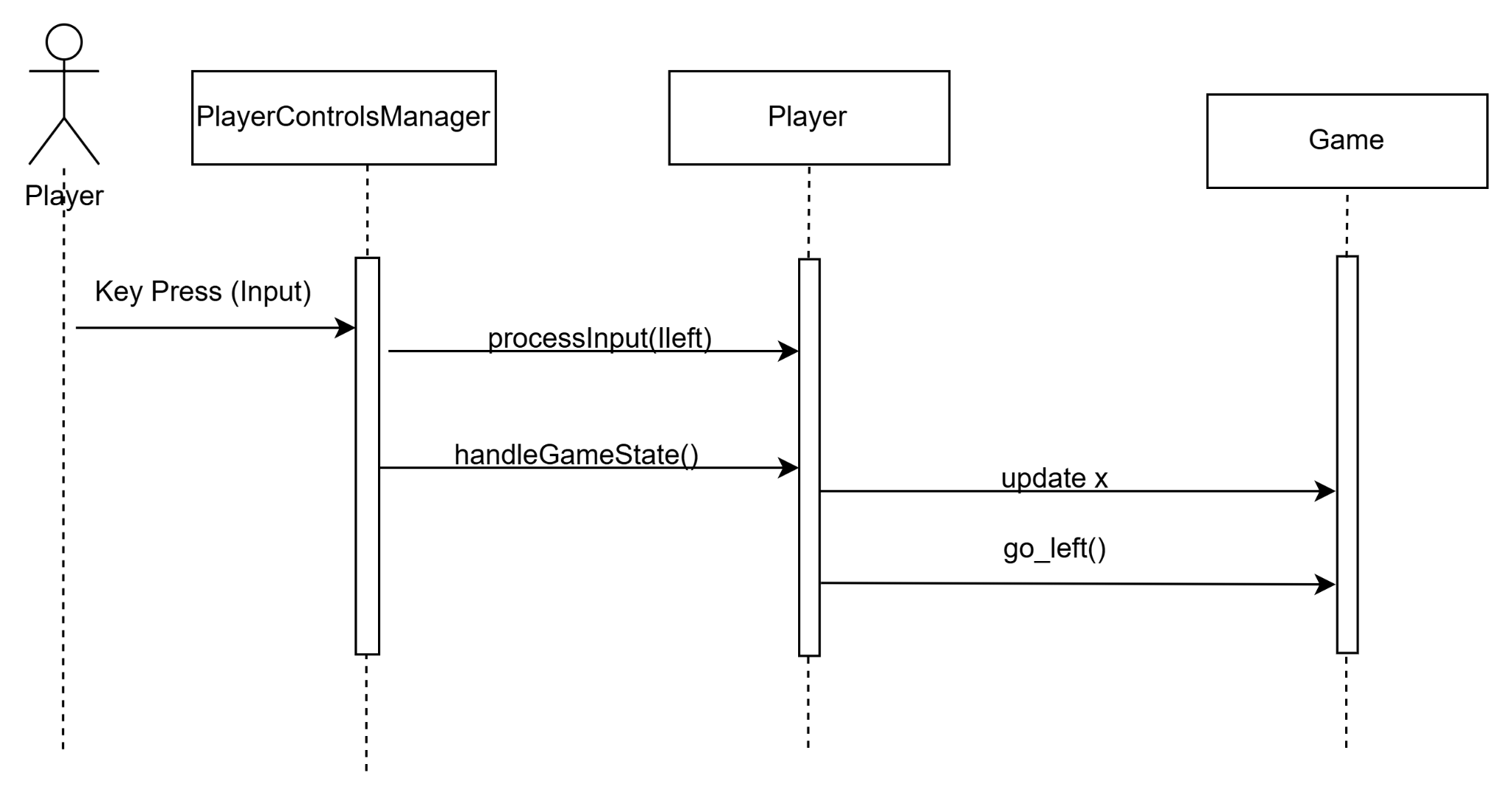
has 3 methods:

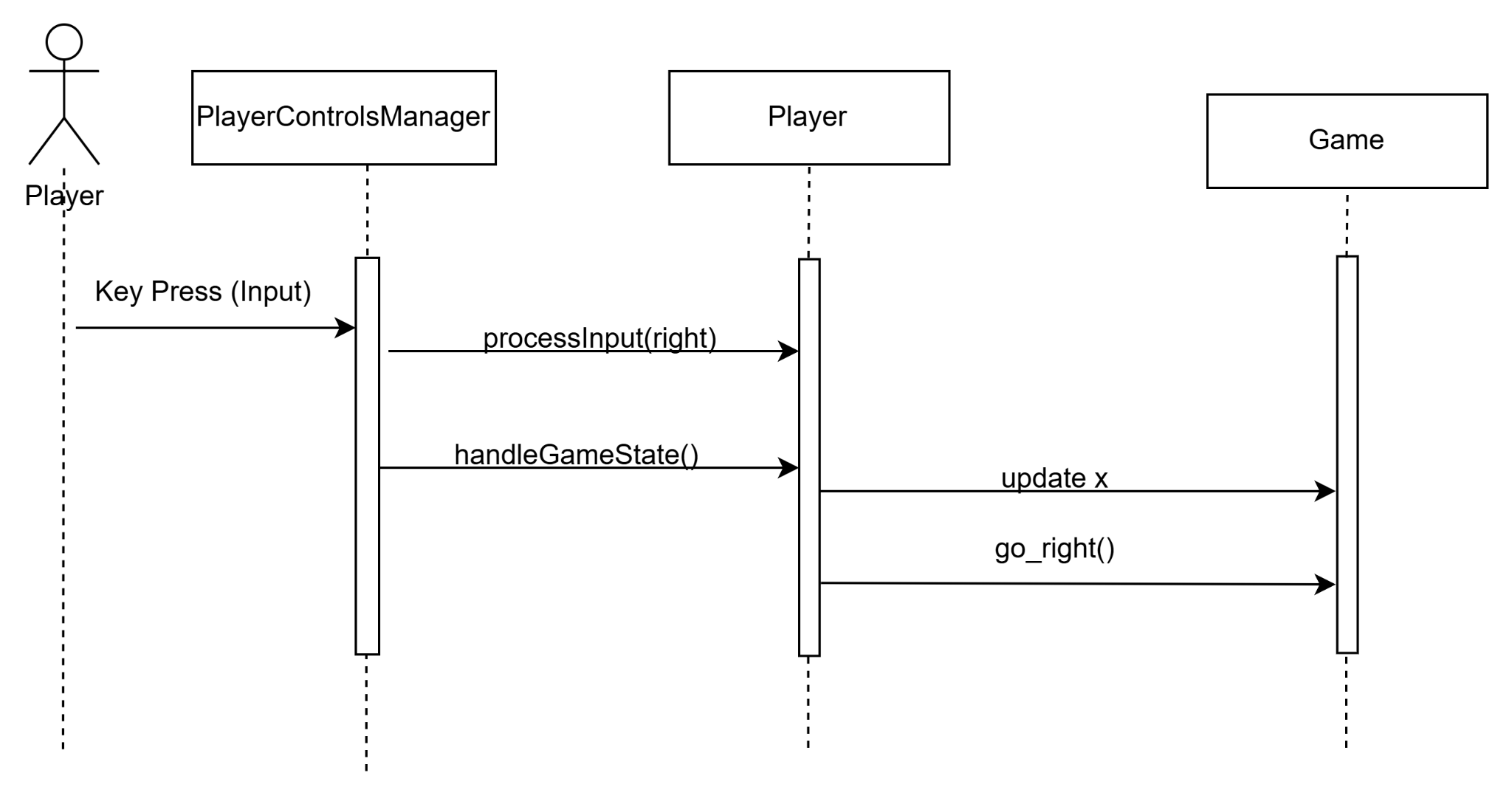
1. updateScore(int points): updates the current score with amount of points given.
2. decrementBallCount(): decrements ball count by one every time balls fall below the paddle.
3. nextLevel(): increments the level by one when all aliens spawned get killed.

Class diagram is shown below.



### **Workflows and algorithms**





### **Software requirements mapping**

SRS-SB-003.1

SRS-SB-008.1

SRS-SB-010.1

SRS-SB-011.1

SRS-SB-014.1

# **COTS Identification**

COTS (commercial of the shelf) libraries used in SB are the following:

* libGDX, 1.13.0, <https://libgdx.com/>
* Java, 8 Update 271, <https://www.java.com/tr/download/>
* JDK,15.0.1, <https://www.oracle.com/tr/java/technologies/javase-jdk15-downloads.html>