Software Design Document

for

Crazy Five

Team: Team-Alfonso

Project: Crazy Five

Team Members:   
*Alfonso Ruelas*

*Andrei Kondyrev*

*Pallavi Arivukkarasu*

*Syed Inshal Raza Naqvi*

*Tayyab Munir*

**Last Updated: [4/5/2021 20:36:12 PM]**

**Table of Contents**

[Document Revision History 3](#_Toc68547762)

[List of Figures 4](#_Toc68547763)

[List of Tables 5](#_Toc68547764)

[1. Introduction 6](#_Toc68547765)

[1.1 Purpose of the System 6](#_Toc68547766)

[1.2 Architectural Design Goals 6](#_Toc68547767)

[1.2.1 Architecture 6](#_Toc68547768)

[2. Software Architecture 8](#_Toc68547769)

[2.1 Overview 8](#_Toc68547770)

[2.2 Subsystem Decomposition 8](#_Toc68547771)

[2.2.1 Game Engine 8](#_Toc68547772)

[2.2.2 Controller 9](#_Toc68547773)

[2.2.3 Player 9](#_Toc68547774)

[2.2.4 Menu 10](#_Toc68547775)

[2.3 Use Cases 11](#_Toc68547776)

[2.3.1 Move Player 11](#_Toc68547777)

[2.3.2 Spawn Enemies 12](#_Toc68547778)

[2.3.3 Enemies Firing Lasers 13](#_Toc68547779)

[2.4 Access Control and Security 14](#_Toc68547780)

[2.5 Global Software Control 15](#_Toc68547781)

[3 Subsystem Services 17](#_Toc68547782)

[3.1 Services 17](#_Toc68547783)

# Document Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev #** | **Rev Date** | **Description** | **Rationale** |
| **4** | **20210405** | **Finalizing SDD** | **Finalizing** |
| **3** | **20210402** | **More updates to sections 1, 2, & 3** | **Adding SDD information** |
| **2** | **20210326** | **Updates to section 1, 2, & 3** | **Adding information** |
| **1** | **20210324** | **Initial Creation** | **Initial Creation** |
|  |  |  |  |

# List of Figures

[Figure 1: Multi-Layered Architecture 9](#_Toc68547484)

[Figure 2: Game Engine 10](#_Toc68547485)

[Figure 3: Controller Diagram 10](#_Toc68547486)

[Figure 4: Move Player Use Case 13](#_Toc68547487)

[Figure 5: Spawn Enemy Use Case 14](#_Toc68547488)

[Figure 6: Enemy Firing Use Case 15](#_Toc68547489)

[Figure 7: Access Control 17](#_Toc68547490)

# List of Tables

[Table 1: Player Keys 10](#_Toc68547784)

[Table 2: Hot Keys 10](#_Toc68547785)

[Table 3: Access Control 15](#_Toc68547786)

[Table 4: Services 17](#_Toc68547787)

# Introduction

This Software Design Document (SDD) is intended to explain the design process of the new Crazy Five computer game. This technical document will follow the standards and infrastructure set but multiple standards organizations such as IEEE, and ISO.

## Purpose of the System

The purpose of this SDD is to provide an overview for how the new Crazy Five will be designed and built. The SDD was created to ensure that we meet the requirements specified in the project requirements documentation set by all the stakeholders and follow existing maintenance management practices and tools. This SDD provides an explanation of the system architecture, software, hardware, database design, and security.

## Architectural Design Goals

The Crazy Five video game will make use of the multi-layer architecture with special emphasis on Availability and Performance as design qualities.

### Architecture

By using the multi-layer architecture, this project will be able to split up work between developers to complete tasks more quickly and not risk developers overstepping on each other’s work.

This project will consist of the following layers:

* Presentation Layer
* Functionality Layer
* Application Core Layer

#### Design Qualities

The following design qualities have been identified by all stakeholders as being required.

#### Availability

Crazy Five will focus on having the game be available 100% once it starts. This means that once the game starts, there should be no crashes or failures that cannot be captured or handled.

The developers for this project will make it a priority to write code that will prevent the application from crashing or displaying failures to the users in a way that will need a restart of the game. This means that while the game is in play, any errors/faults/failures will be handed in a way that will keep the game play from stopping.

Special focus from the developers will be to handle exceptions and process them in a way that will not only handle the error but also to save the state of the game for future error troubleshooting.

#### Performance

Performance is, “all about time”. The Crazy Five development team will focus on any action that the user performs, whether it be a button click or a mouse movement, should not contain any lag, as lag severely impacts game play in real-time.

Considering that they are only a few possible actions that the user can perform, these actions will be benchmarked and tested throughout the game while testing.

# 2. Software Architecture

## Overview

As mentioned earlier, the Crazy Five project will utilize the multi-layered software architecture to design the game. The multi-layered architecture is the most widely used architecture and easiest to implement in an environment such as a video game.

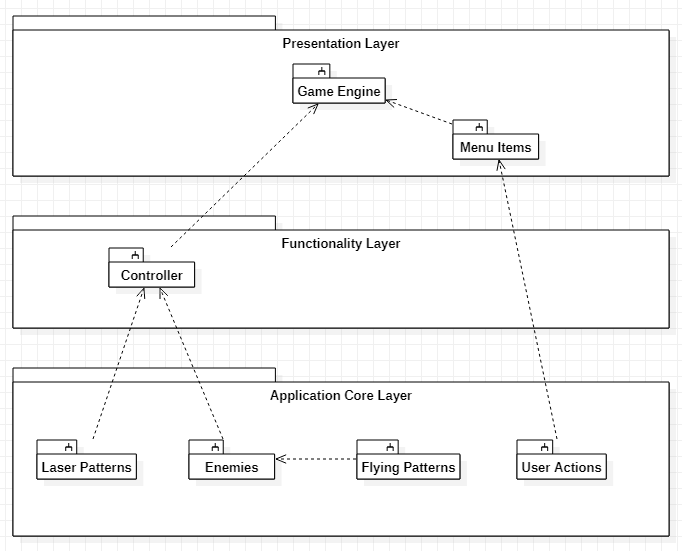


Figure 1: Multi-Layered Architecture

## Subsystem Decomposition

### Game Engine

The game engine is used as an entry point for the application/game. From this point, one can initialize the game settings, load content, draw, and update.

The GraphicsDeviceManager and SpriteBatch are used for the MonoGame engine, whereas the GameController and ActorFactory are developed objects to handle new logic.

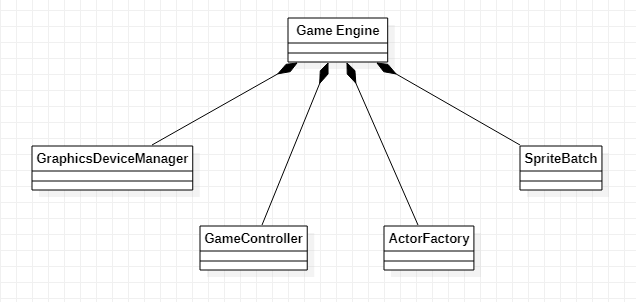


Figure 2: Game Engine

### Controller

The controller will manage all aspects of the enemy’s actions, including creating the enemies, lasers, and the patterns that the enemies will use to move.

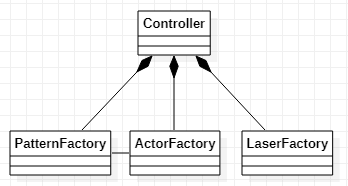


Figure 3: Controller Diagram

### Player

The user actions subsystem will capture all user actions such as movements. At the time of this writing, the following button clicks are captured:

|  |  |
| --- | --- |
| User Action | Description |
| Up Arrow | Move the player sprite up |
| Down Arrow | Move the player sprite down |
| Right Arrow | Move the player sprite to the right |
| Left Arrow | Move the player sprite to the left |

Table 1: Player Keys

In addition to these basic moves, the player can also be moved diagonally (i.e, holding up arrow and right arrow simultaneously, for example) across the screen, giving the player a total of 8 directions to move.

### Menu

The user actions subsystem will capture all user actions such as button clicks. At the time of this writing, the following button clicks are captured:

|  |  |
| --- | --- |
| User Action | Description |
| S | Slow the game down |
| K | Toggle auto-fire; On/Off |
| Spacebar | Fire Projectile |
| ESC | Exit Game |

Table 2: Hot Keys

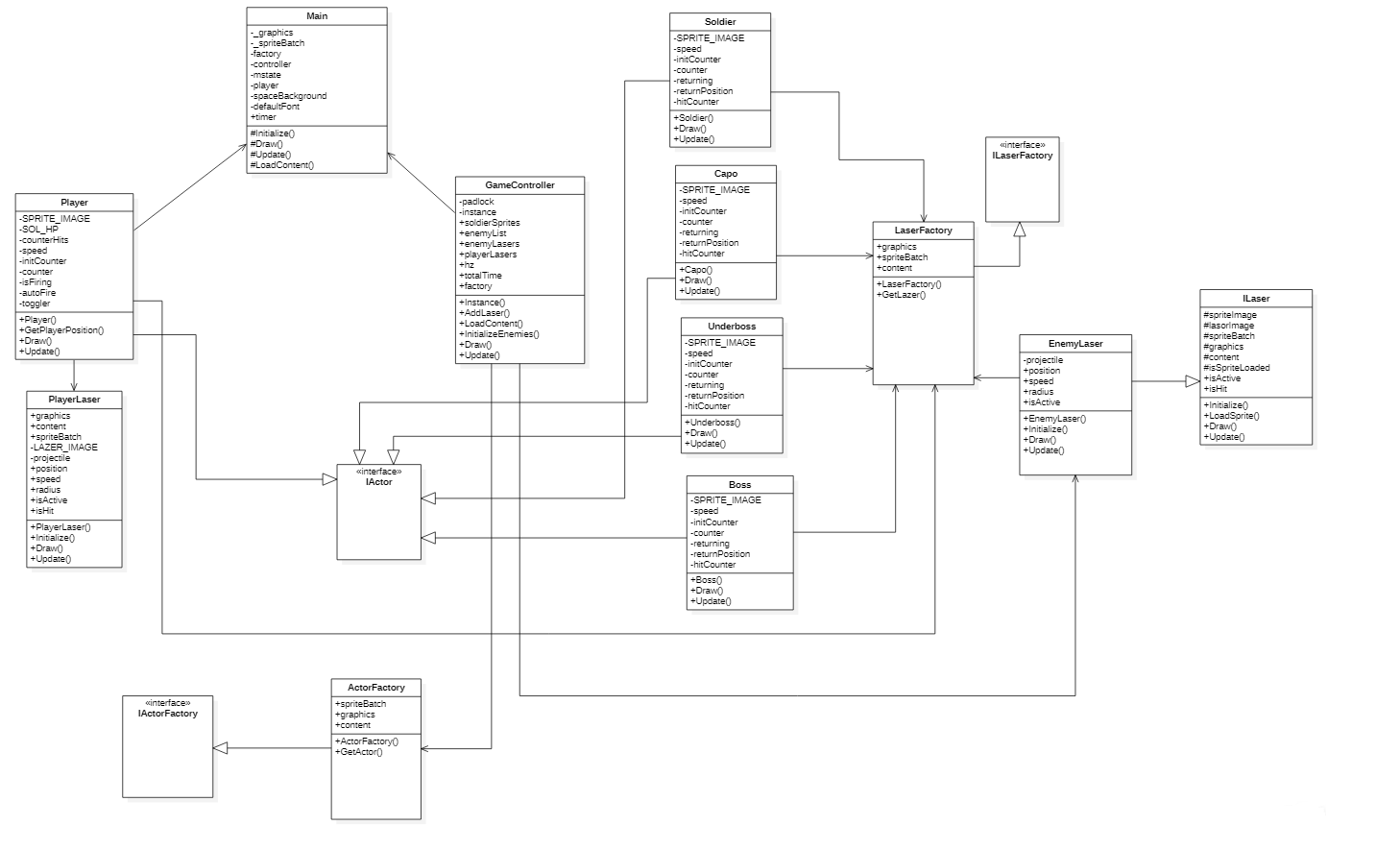


Figure 4: Player Classes

## Use Cases

### Move Player

The movement of the player comes from the clicking of pre-configured keys (See Section 2.2.4). When the player, for example, clicks on the up key, the Game Engine handles that action and decides if it’s a valid key press. If it is so, then the key press is passed to the player object for further processing.

In this example, the player object will calculate how much to move and whether or not it’s in slow motion. After this step is done, the player is adjusted from the movement object.

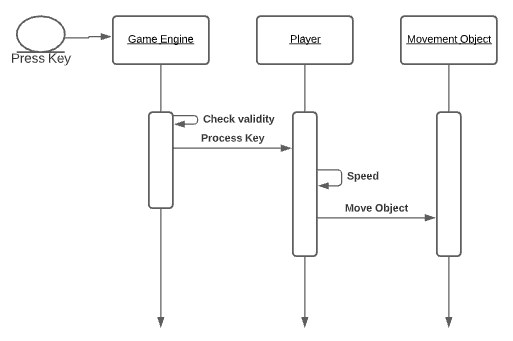


Figure 5: Move Player Use Case

### Spawn Enemies

The spawn enemies are processed a bit differently than the Move Player case. In this case, the Game Engine passes control of the enemy objects to the controller. The controller doesn’t only handle all aspects of the enemies, but it also controls the firing of the lasers as well.

Following the diagram below (Figure: 5) displays the process for creating a new enemy.

As stated before, once the game starts, the Game Engine passes all control of the enemies to the controller where it will handle the processing. Once the time is achieved, meaning a specific time elapses (configurable through a configuration file) the controller will create an enemy based on the time. When it’s ready to create it, it will call upon the Enemy Factory object to return a new enemy of our choosing.

The factory returns it to the controller and the controller in return, stores it for later modification of it.

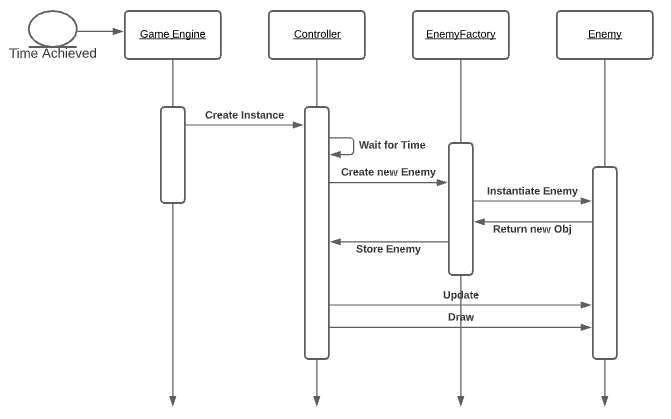


Figure 6: Spawn Enemy Use Case

### Enemies Firing Lasers

In this use case, we describe the enemy’s firing mechanism.

Since stated before, the controller handles all aspects of the enemy, we will start there. Upon the creation of a new enemy, the controller will grab a new firing pattern using a factory and assign it to the enemy.

Depending on the speed of the game (which also is controlled by the controller), the controller will set a timer for firing and will keep track of it.

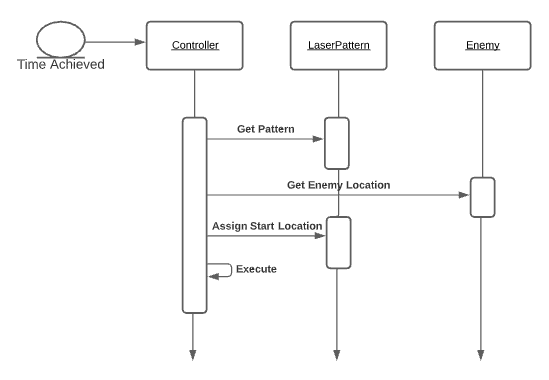


Figure 7: Enemy Firing Use Case

## Access Control and Security

Access matrix:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | IActor | GameController | ILaser | Utilities |
| Enemy | <<*create*>>  GetRadius()  LoadSprite()  GetSprite() | AddLaser() | GetLazer() | GetReturnPosition() |
| Hero | <<*create*>>  LoadSprite()  GetSprite() | AddLaser() | GetLazer() |  |

Table 3: Access Control

Directory permissions: 755

Such mode, in terms of Confidentiality, Integrity and Accessibility, provides us with Accessibility (+x), Integrity (-w) and there is no need in Confidentiality (+r).

Security Issues:

There is no need in encryption. But if we need one – we can use OpenSSL library to generate keys.

RSA 2048 bit would be more than enough. We can make a key request to Let’s Encrypt. So the encryption would be asymmetric. Ideally, we would execute it on a machine in Kubernetes cluster and all the keys are stored in kubeSecret. But if we don’t have a pre-built k8s cluster with a good ACL, the efforts aren’t worth it.

## Global Software Control

Threaded control. Since we have various mechanisms that can be executed simultaneously. And we also operate with objects.

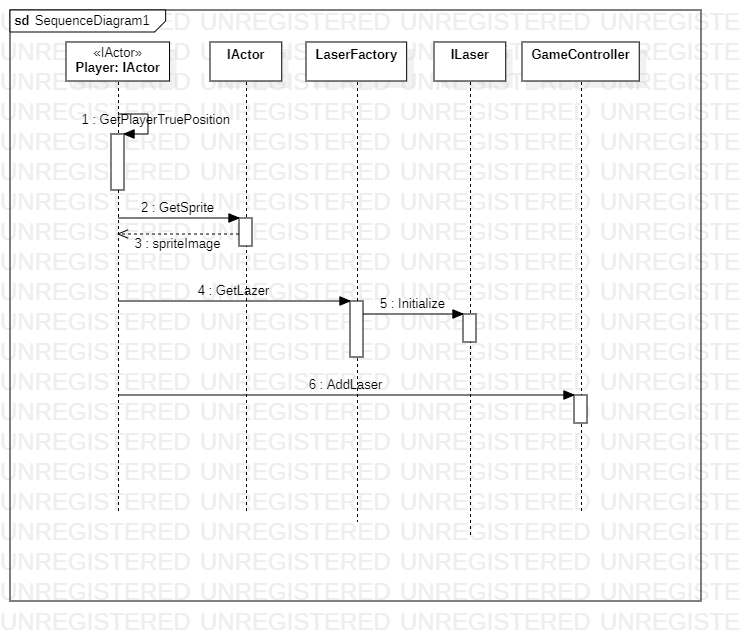


Figure 8: Access Control

# Subsystem Services

## 3.1 Services

The following section will contain the services contained within each subsystem.

|  |  |
| --- | --- |
| Subsystem | Services |
| Menu Item | The menu service displays early in the game - it shows how to start the game and the options the player can use. |
| Controller | The controller service initiates the enemies and controls them based on the configuration file. |
| Player | The player service takes in keyboard inputs and processes them. |

Table 4: Services