**Black Light Project’s Software Design Document**

By SERGIO MADERA

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Author** | **Revision date** | **Corresponding Commit** |
| 0.1 | Sergio Madera | 17/10/2016 | v0.25 |
| 0.2 | Sergio Madera | 07/11/2016 | v0.26d |
|  |  |  |  |

1. **Introduction**
   1. Purpose

This document’s purpose is to provide a detailed description and justification of an architectural solution proposed to satisfy the requirements the videogame demo development initiative “Black Light Project” has. Diagrams and similar graphics will be used when required.

* 1. Scope

This document interprets Black Light Project’s goals as use cases and proposes and describes comprehensively an arquitectural solution to satisfy these. The solution proposal is detailed through a use case view, a component diagram and a sequence diagram for each process.

* 1. References

http://www.ecs.csun.edu/~rlingard/COMP684/Example2SoftArch.htm

1. **Use Cases**

* Player moving
* Player jumping
* Player interacting with NPC
* Player initiating battle
* Player taking damage
* Player attacking
* Player picking up buff
* Player dashing
* Player losing buff
* Player defeating enemy
* Player healing
* Player gaining points
* Player spending points
* Player saving the game
* System updating GameState

1. **Arquitectural Pattern**

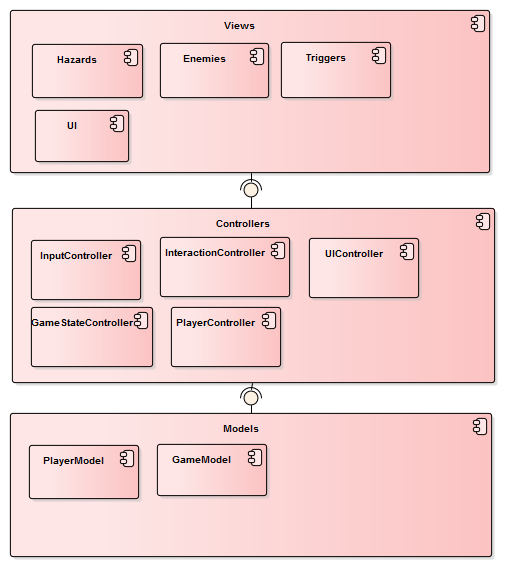
This section specifies the arquitectural pattern used is MVC, where Models contain data not shown to the player, Views manage everything shown to the player and are told by Controllers the current state of Models so they can update themselves and Controllers are entities that contain the logic to detect what the player wants and coordinate other components so the player action is satisfied.

1. **Components**

This section describes the components the arquitecture described in the Arquitectural Representation comprises of.

* 1. **Component diagram**

The following is the component diagram of logic components that will be scripted by the developer (author of this document) in the Unity project. It is not strictly correct, and some components just describe aggrupation of other components.

****

* 1. **Views**

Everything that can be seen by the player, whether rendered in the world or as an interface. May also manage collision detection.

* + 1. UI (View)

What is rendered on screen as the User Interface. Updated exclusively by the UI Controller, based on what is stored in Model components (PlayerModel, for example).

* 1. **Controllers**

Scripts that handle Logic/Workflow.

* + 1. UIController

Handles all logic regarding what is showed, hidden or updated in the User Interface.

* + 1. InputController

Receives all input from the player and acts as a controller for the logic of the game, meaning it coordinates other components necessary to show the result of the input to the user. It also handles input related logic such as raycasting to see if the player is trying to interact with something.

* + 1. PlayerController

Manages changes in the player status such as movement, health, buffs, debuffs, stacks, and similar.

* + 1. InteractionController

Manages interaction with interactable objects in the game. Has a list of all accepted (programmed) interactions. The interactable objects tell they want to interact with the player and the parameters with which they want to do so, and the InteractionController coordinates the corresponding components so it is done. For example, if a GameObject wants to talk to the player, they would get to the InteractionController and it would redirect them towards the DialogController. In detail. Example:

* InputController detects the player presses F when raycasting an interactable GameObject.
* InputController tells InteractionController what just happened and sends him the object that wants to interact with the player
* InteractionController asks the object the way it wants to interact with the player. The interactable GameObject should send a proper interaction identifier (TALK, SCENE\_CHANGE, etc.). Depending on this, the InteractionController forwards it to the proper controller. For a TALK interaction, this would be the DialogController, so the InteractionController tells the DialogController an NPC wants to TALK and sends him the NPC.
* The DialogController (or any other controller from this point on as far as the InteractionController is concerned) should know what must be done to complete the interaction successfully. A little more in detail, however, the DialogController, knowing the NPC wants to TALK, will ask the NPC for the parameters with which it wants to talk to the player (text of the dialog, “sound” of the dialog, the delay between each character been written on the screen…) and then ask the UIController to update itself in the way the NPC described. It is important to note that the DialogController does not just forward the responsibility of the whole process to the UIController, just the “update the UI” part in each step of the logic the DialogController contains.

**GameStateController**

Manages logic to save the game and is the one who updates the GameModel and encrypts it each time it is saved.

* 1. **Models**

Contain data that is not displayed directly. The data stored in them might be retrieved by a controller and passed to a view so it is shown to the player in some way.

* + 1. PlayerModel

Contains data regarding the player's current position, rotation, health, stacks, buffs, etc. Is updated exclusively through the PlayerController.

* + 1. GameModel

Contains information regarding current game world status, this is, what NPC should say currently, what quests are available, what enemies spawn, what event should happen next in the story. Is the object that is encrypted when saving the game.

**General Player Input Flow**

* Player inputs
* InputController gets input
* InputController decides from what’s information is available to him what the player wants to do (following the game’s control scheme)
* InputController calls other components so the input can be answered

**Player Movement Workflow**

Player inputs a movement key (W, A, S or D for example)

InputController identifies the player wants to move to a certain direction (forward for W, for example) and tells the PlayerController to do so.

PlayerController does what it is told to by the InputController.

1. **Processes**

This section describes the processes through which the system satisfies each use case and justifies the solution provided. There is at least one process per use case described in the section 2 of this document.

* 1. Interacting with an NPC

