System Design Document
Purpose

The system design is documented in the System Design Document (SDD). It describes additional design goals set by the software architect, the subsystem decomposition (with UML class diagrams), hardware/software mapping (with UML deployment diagrams), data management, access control, control flow mechanisms, and boundary conditions. The SDD serves as the binding reference document when architecture-level decisions need to be revisited.

Table of Contents

1. Introduction

1.1 Overview

The present document provides insight on the technical details of the system design for the Space Invaders game.
It will be presented the design goals and solutions for incoming problems, it will be justified the chosen architecture style for the proposed system and the design decisions made.
Furthermore, there will be presented the decomposition of the system in subsystems and services provided. The hardware/software mapping is described and defined through diagrams. Finally, persistent data manager, the access control and security, global software control and boundary controls are addressed.

1.2 Definitions, acronyms, and abbreviations

TSI – TUM Space Invaders

1.3 References

More information on features and motivation of this project can be found in the Requirement Analysis Document of the respective system.

2. Design Goals

The TSI game have the design goals described as follows.

- Usability : Since the game has the purpose to be simple, it is important that the user interface is easy to understand and the app is overall easy to use.

- Extensibility : The game is design to have few base functionalities in the beginning and can be developed into a more complex user experience, so it is important that that the application is easy to extend.

- Performance: It is important to provide a system with real time deliverable actions when talking about a game. The ideal is that a player does not experience any latencies during the game due to performance inefficiency.

- Scalability: although the game does not require a big amount of data transference, it has to be possible for more than one user to send and receive data at the same time.

In addiction the game must be developed in Java for desktop operating systems, the version control system must be git and it must be playable offline.

In order to develop a game with strong base structure and high usability, there was the need to implement less functionalities, so the game is not complex in the beginning stage but has a high perspective of extensibility. Also it is important to notice that there is a few information as possible in a non-local data base so that we can decrement the possibility of delay times.

3. Subsystem decomposition

For the specific system designed, the most suitable architecture style is the Model-View-Controller pattern. By choosing this model, the system is clearly separated in three parts: the model, the game logic and the user interface. This helps structure the system in a way that allows to change things without having to rewrite too much code.
In TSI, the Model contains the elements that are structural part of the code, such as the Menu and Level models, that have to be available for updating all the other components of the game. The View will contain everything that is related to what the user can see, the user interface. The controller contains the whole management of the game, where the inputs from the user will be interpreted and turned into game actions.
Underneath there is a simplified structure of the architecture applied to the game.

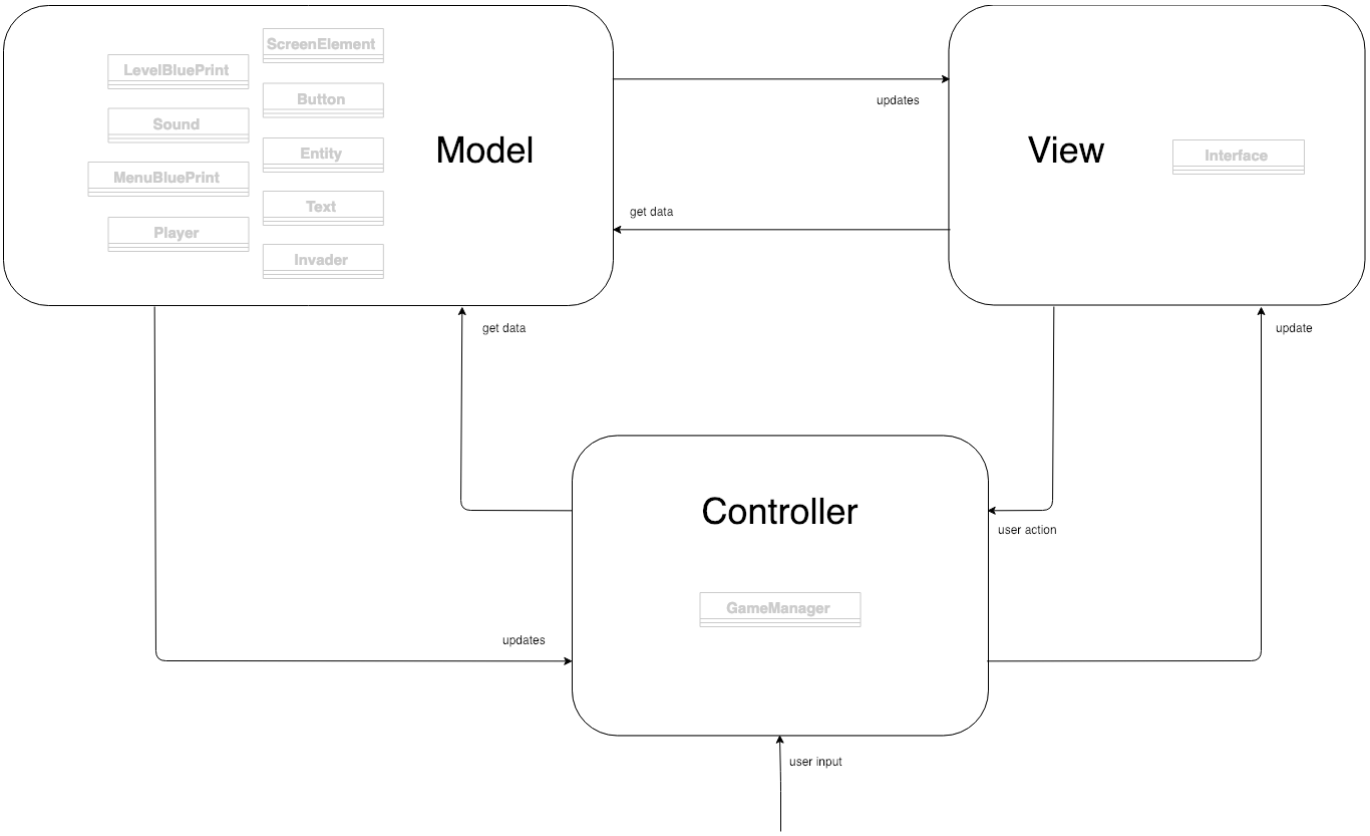

Figure 1 - Model-View-Controller pattern for TSI

4. Hardware/software mapping

The Hardware/ Software mapping for the TSI shows what the ideia of the game would be in its final stage. It uses 2 nodes and each of them contains various different components.

The Desktop : Client node includes all the components that make the game functional, that includes the logic manager, Game Manager, and Sound archive. Then the game manager is connected to the 2 different interfaces: the Game Interface, that depends on the Level Blue Print, and gives the user the game view; the Menu Interface, that depends on the Menu Blue Print, and gives the user the menus' view.
The Input node, that is handled from an outside perspective, contains an Input Platform and the necessary Input Platform Connector to interact with the game.

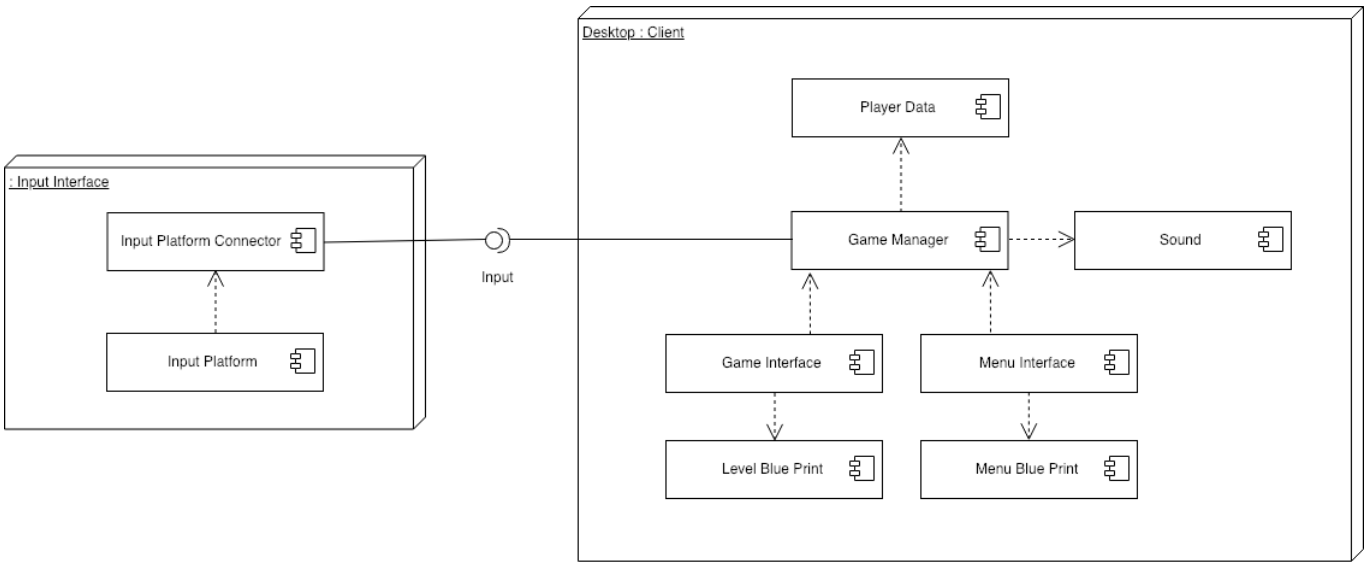

Figure 2 – Deployment Diagram

Additionally, a more complex Hardware/ Software map is represented in Figure 3, representing the general idea of a future perspective of what the TSI game could be in terms of extensibility.
The added Server is a node that contains all player's information, including their name and high scores. That is saved on a remote database called Player Database, so that the players can access real results from each other.
Also, the Input node represents any type of input that could be implemented in the future to make the game more interactive, for example, instead of using a keyboard, it could be used a camera with AI coding so that a person could be able to move the space ship by moving themselves.

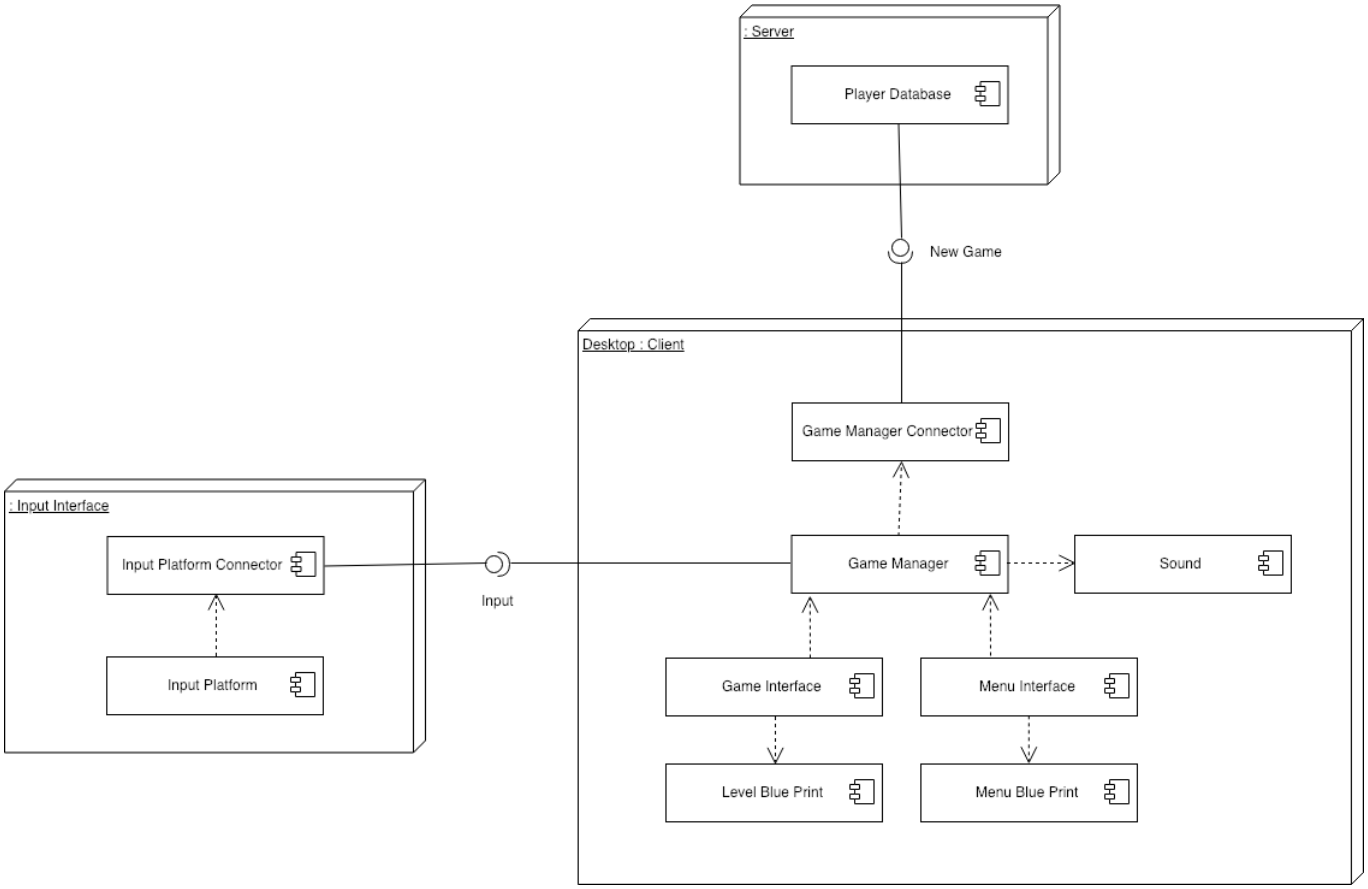

Figure 3 – Extended Deployment Diagram

5. Persistent data management

The TSI is implemented in a file system storage scheme, so data is always available locally and has less failure probability (connection issues, package failure).

6. Global software control

The strategy for this system is the event-driven control flow, since the operations that occur during the game are influenced and require an external event (input of the user) so that the game can flow.
Although there are internal command executions, made and received by components of the system, the whole game does not leave the stationary state without an event outside the control of the component that handles that event, that can be seen as the initial requests for all actions.

Since the game manager and the interface are always updating themselves and communicating with other components, a synchronized flow of requests and answers have to exist. This will be assured because all the modules have to respect an internal clock that influences and determines the time of the actions of all components. This way, no request can interrupt or corrupt others.

7. Boundary conditions

This sections is used to describe how the system starts, shuts down and the failure cases.

The first module to be initiated has to be the the game managers that will make everything accessible to the user interface and consequently to the user. Then the user will input the values that will be recognized by the software and the other components will be called for their specific tasks.
In order to shut down the system, and because all the workflow will end up needing the game manager, this will be the last component to shut down.

If any of the components fail, the system has to be restarted. This will happen because the game cannot work without the game manager, that will execute and manage all the other components; although it can work without the interface, it wont have the initial purpose for its implementation; and if any component of the model fails sending their respective data, the game will crash and cannot be recovered.