CS 319 - Object-Oriented Software Engineering

Design Report

Syracuse

Supervisor: Uğur Doğrusöz

Group 27

Muhammad Usman

Onur Kulaksızoğlu

Ayşe Kübra Taş

Table of Contents

1. Introduction

1.1 Purpose of the System

Syracuse is a turn based city/state building simulation/strategy game inspired by the game Tropico. In this game the player takes control of the city Syracuse during the antiquity. The main goal of the player is to defend, expand and develop his/her city. The game is designed to be user friendly and easy to learn. It get difficult as the player progresses in order to keep the user interested. The main focus of the game is its gameplay which means that some other things like graphics are not that good as some other games of the same genre. But this game contains loads of interesting missions and quests to keep the player engaged for hours.

1.2 Design Goals

Usability/ User friendliness: We have put a great emphasis on making the game really easy to use.The user can easily switch between different layers to like the city layer or Sicily layer to get a better understanding of what's going on and make decisions easily. Apart from the we have a turn based system which means that the player can play the game at any pace and whenever they want and once he completes all the missions, he can ask to go onto the next level/age. All these features make the game really easy and friendly to use.

Ease of learning: Often strategy games contain a lot of stuff going on at the same moment and the player has to look at all of them and makes decisions accordingly. It's the same with our game. That's why we have the Sicilopedia. It is an encyclopedia which contains information about each and every aspect of the game and the player can refer to it whenever he is facing any difficulty. So this feature helps the user in learning the ins and outs of this game.

Maintainability/ Modifiability: We have chosen to use Java as the programming language for our game. As it is pretty easy to use and maintain than some other languages like C++. So that's why our game will be really easy to maintain and update whenever we feel the need to do it.

TradeOffs:

Rapid Development vs Eye Candy: We have chosen to focus on core gameplay elements of the game and not putting very much emphasis on the graphics so that we can easily and fastly code the game and put it out to be used by the users.

Efficiency vs Portability: We are releasing this game for Windows only so that we can make it as fast and efficient as possible and not worry about making it cross platform as it might require us to implement some features that might overall decrease the game performance.

Cost vs Robustness: We are choosing to reduce the cost by implementing everything really fastly and using just the java native libraries. So the game might not be as fast and responsive as some other strategy games. Plus we have chosen not to focus much on graphics, we are going to reduce the cost by using simple/free graphical tools instead of buying a professional graphical design tool.

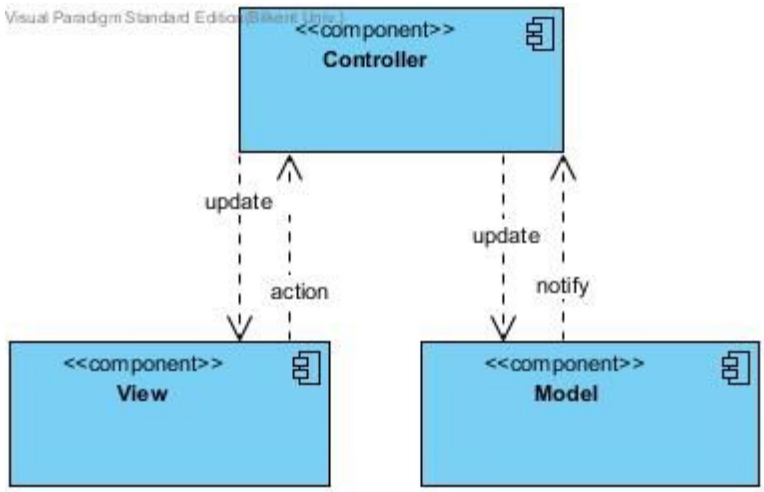
Speed vs Space: As our game doesn't use heavy graphical elements, the loading and saving of game will be pretty fast as there will be not much of the data to be read or written and this in turn will also reduce the space requirements for our game.

2. Software Architecture

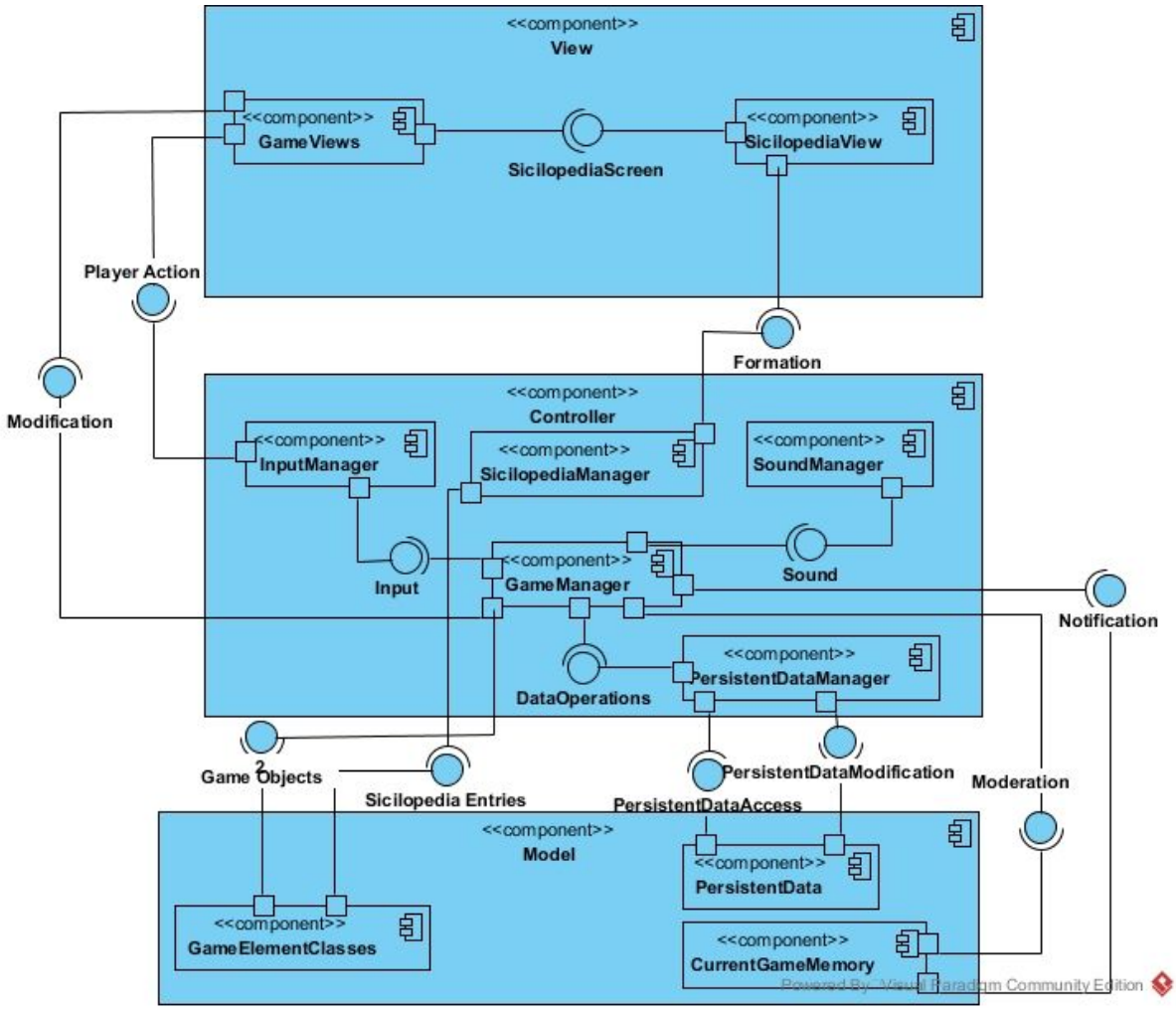
In this section, we will decompose our system into subsystems to make it more understandable and to facilitate our job during implementation . We will also explain which hardware and software tools are needed and how to manage database system. Additionally, access control and security issues will be mentioned and boundary conditions will be examined under three subsections.

2.1 Subsystem Decomposition

Our game, as a strategy game, includes many screens and pretty much logic behind these screens which need to be considered separately. For this screen-logic separation we decide to use MVC pattern which provides minimizing coupling between main subsystems and maximizing cohesion within a subsystem that is while view subsystem is handling all user interface stuffs, model subsystem is dealing with background logic issues and controller subsystem interacts with user while updating model and view subsystems and receiving actions from view.



There are subsystems of model, view and controller components in the diagram below. Relationships between these subsystems are also defined in the diagram.



2.2 Hardware/Software Mapping

We choose Java SE platform that uses object-oriented Java language to implement our game with its Java development Kit (JDK) and Java Runtime Environment (JRE). The packages we mainly use are AWT package which handle basic GUI operations, swing package which handles widgets and util package that contains many framework, event model etc. The only hardware need for our project is a keyboard and and a mouse.

2.3 Persistent Data Management

In the game, we should keep data to save game and for achievements. For this purpose, a complex database system is not needed but instead we can use simple text files. Text files which keep saved games and achievements would not be used in the current game. These are just for the player so that he can reach when he wants. Also, we are planing to use some gifts and music files which can be used in the current game.

2.4 Access Control and Security

Our game is a single-player game which does not require any network connection. Player needs to download it once and then, only people who can access the game are those who can access the computer in which our game exist. Therefore, there would not be a security problem as long as the computer is in safe.

2.5 Boundary Conditions

Initialization

The game will start after clicking on the icon on the desktop or any place in the computer. The extension of this icon will be .jar.

Termination

Player can leave the game by clicking “Quit Game”. Then we would ask if the player wants to save his played game. If he chooses to save, game will be saved

and the player can continue later. There no need to pause game because game is a term-based game.

Error

For any reason, if game gives any error, game will not be saved. Player can start a new game or he continue with previously saved games. Only current games can be affected by the errors since we are saving played games in text files.

3.Subsystem Services

The parts below in this section describes the brief duties of the each subsystem.

3.1. View Subsystems

Because of our MVC design architecture all the subsystems that handles the interface are collected in this larger subsystem.

3.1.1. Game View

The game view is the subsystem which includes most of the user interface classes in the Syracuse, it handles all the graphic related things. Other than that forming the graphical interface of the Syracuse, this subsystem takes user inputs and sends them to the input manager. This subsystem uses “Sicilopedia View” subsystem to create Sicilopedia in our game, this subsystem is modified by Game Manager subsystem which is one of the controller components.

3.1.2. Sicilopedia View

Sicilopedia View is a subsystem for creating the graphics of the Sicilopedia in our game, it is called when the user chooses so it is lifetime is dependent on the “Game View” interface. Sicilopedia View accesses to Sicilopedia entries by making calls to the Sicilopedia Manager subsystem, which directly accesses to the entries of the game and modifies the Sicilopedia View with the new chosen entries.

3.2. Controller Subsystems

Controller subsystems in our game handles all data accesses and modifications, and also modifying the values in user interfaces. Controller subsystems also handles game logic, sound effects in the game, and input management.

3.2.1. Input Manager

Input Manager is responsible for taking inputs from the player and sending them appropriately to the Game Manager, Game Manager calls the appropriate functions according to these inputs.

3.2.2. Game Manager

Game Manager is responsible for handling all the core game logic, modifying the current game data and the game graphics while also doing them according to the user inputs. Game Manager also calls appropriate functions from the subsystems of Sound Manager and Persistent Data Manager, again according to the current game situation and user inputs. This is the most central and important subsystem of our game.

3.2.3. Sound Manager

Sound Manager subsystem provides an interface to the Game Manager for playing sounds and musics dependent on the game’s current situation. Other than that this subsystem has no function.

3.2.4. Persistent Data Manager

Persistent Data Manager subsystem provides an interface to the Game Manager to be used when player wants load-save games and trophies. Persistent Data Manager subsystem handles all the things related to the data in the hard disk and provides modularity in our system design.

3.2.5. Sicilopedia Manager

Sicilopedia Manager subsystem's duty is to modify Sicilopedia view by the data it accesses from the Game Element Classes.

3.3 Model Subsystems

Model subsystems in our design represent both the data that is persistent on the disk like saved games and the data on the memory about the current game like maps etc.

3.3.1. Current Game Memory

This subsystem's purpose is to hold the memory objects special to this session of the game, like the player's settings, his current diplomatic relationships, buildings in his city, which age he is in etc. This memory is modified-created by Game Manager subsystem and is again only accessed by this subsystem. When a game is saved only this part will be stored as persistent data.

3.3.2. Persistent Data

Persistent Data subsystem represents the data that are saved in the hard disk in our game, the data for the trophies and the saved games are part of this sub-file

system. Persistent data can only be accessed and modified by the Persistent Data Manager subsystem when it's ordered by the Game Manager.

3.3.3. Game Element Classes

"Game Element Classes" model subsystem includes the data that are not unique to only one session of the game and are also not persistent in the hard disk. Objects for Sicilopedia entries, individual building features, unit features, agendas of the other factions and many other data fragments required by the game are organized in this model subsystem. To give an example between the difference of previous parts and this part: this data part holds the information about which raw resource is required for manufacturing a good that is same in every game, while the "Current Game Memory" subsystem holds the information about the current resources owned by the player in the city.