CS 319 - Object-Oriented Software Engineering

Design Report

Colony Wars

Supervisor: Bora Güngören

İbrahim Taha Aksu

Ahmet Emre Nas

İzel Gürbüz

Kaan Çakmak

Table of Contents

1. Introduction

1.1 Purpose of the system

Colony Wars is a time based strategy game that requires usage of a considerable amount of strategical thinking. The game aims to provide the users with a leisure time activity that is both challenging and fun to spend time on. The user interface of the game is quite simple which enables even the user who is not familiar with the common flaw of strategy games to get use to it easily. Different playing modes such as campaign and skirmish are added to gameplay to keep the player interested and busy. In the light of all the points provided above the purpose of the system is to serve user an activity that is fun and challenging to spend time on.

1.2 Design goals

1.2.1 Ease of Learn

One of our main aims in constructing this system is to make it's learn easy by providing simple mechanics with the game tutorials. Users do not have to have any idea about the game or overall about strategy game playing. Purpose of tutorials is to teach the game mechanics and rules to the user easily. Therefore, user can understand game mechanics in a minimized amount of time and jump in to the game directly.

1.2.2 Usability

Targeting a user scope that is so broadly open from children to adults our program should have a user interface that is clearly understandable regardless of the user's experience with any software applications or games. Besides having a simple interaction for user the tutorial/help part should be explaining basics and usage of game's features neat enough so that nearly no one would have any problem using it.

1.2.3 Reliability

Provided with any input regardless of being valid or not the game should keep working without any crash and should be well prepared for such cases so that it gives the best service possible to the user. The game should also be bug-free in terms of game play and no matter how long the program is used it should keep it ability to function properly.

1.2.4 Performance

The program is going to refresh frames of the game and doing calculations constantly since it's a real time dynamic strategy game. Thus it should have a fast enough response time and should be stable in spite of the length of the game which comes with more workload by time passing.

1.2.5 Extendibility

Regarding the limited time we as developers have we are not including some of the functions although we are eager to have them. There are also many possible game futures such as multiplayer playing or new arcade game modes. Thus the game should be extendible further in the future.

1.2.6 Memory Usage

According to game mechanics, there are a few maps, basic game animations and collection of integer game data. User also can save his/her process in the campaign mode. System saves process by using map name and integer values of his/her last game. When user wants to save the process again, old saved data will be erased and new game data will be saved. According to these properties, memory usage of the game is not high.

1.2.7 Functionality

The real purpose behind constructing this program is to provide user with an activity that is highly enjoyable and challenging. Thus the game should have several features and functions that would keep the user’s attention on the game and serve him a way in which he/she could enjoy the time spent on the game.

1.2.8 Tradeoffs

We have some specific priorities in which we put several design goals on front of others. There are two cases in which two of our design goals intersect in a way that it is hard to have both of them fully functional. One of these being performance and reliability and the other ease of learn and functionality.

1.2.8.1 Performance-Reliability

In order our system to be reliable that is crash free and working correctly incase of any missing or false inputs we have to trade off from our performance goal since it takes more calculations are more process in order to have a system reliable.

1.2.8.2 Ease of Learn-Functionality

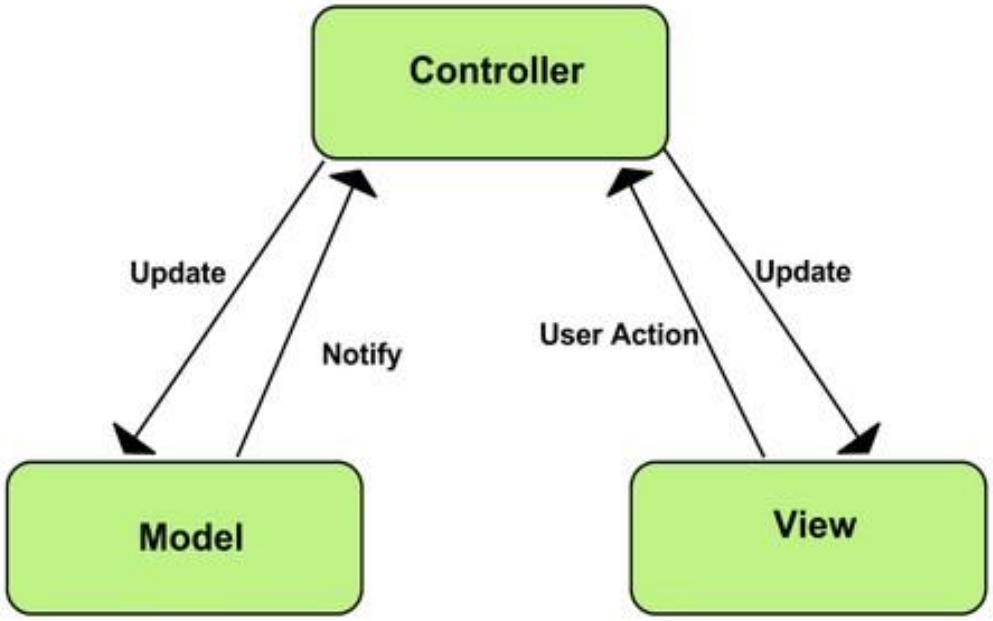
As we as developers seek to target a wide scope of users including adults too, we have to provide a game that is challenging which might require some complexity in the game mechanics to make it hard to achieve the objectives of the game plays. However, more functionality comes with a more complex system making it hard to learn to play the game easily. Thus we will have a trade off in the functionality since we would lose a great user scope by making the game complex considering the game would have a lot more young users than adults.

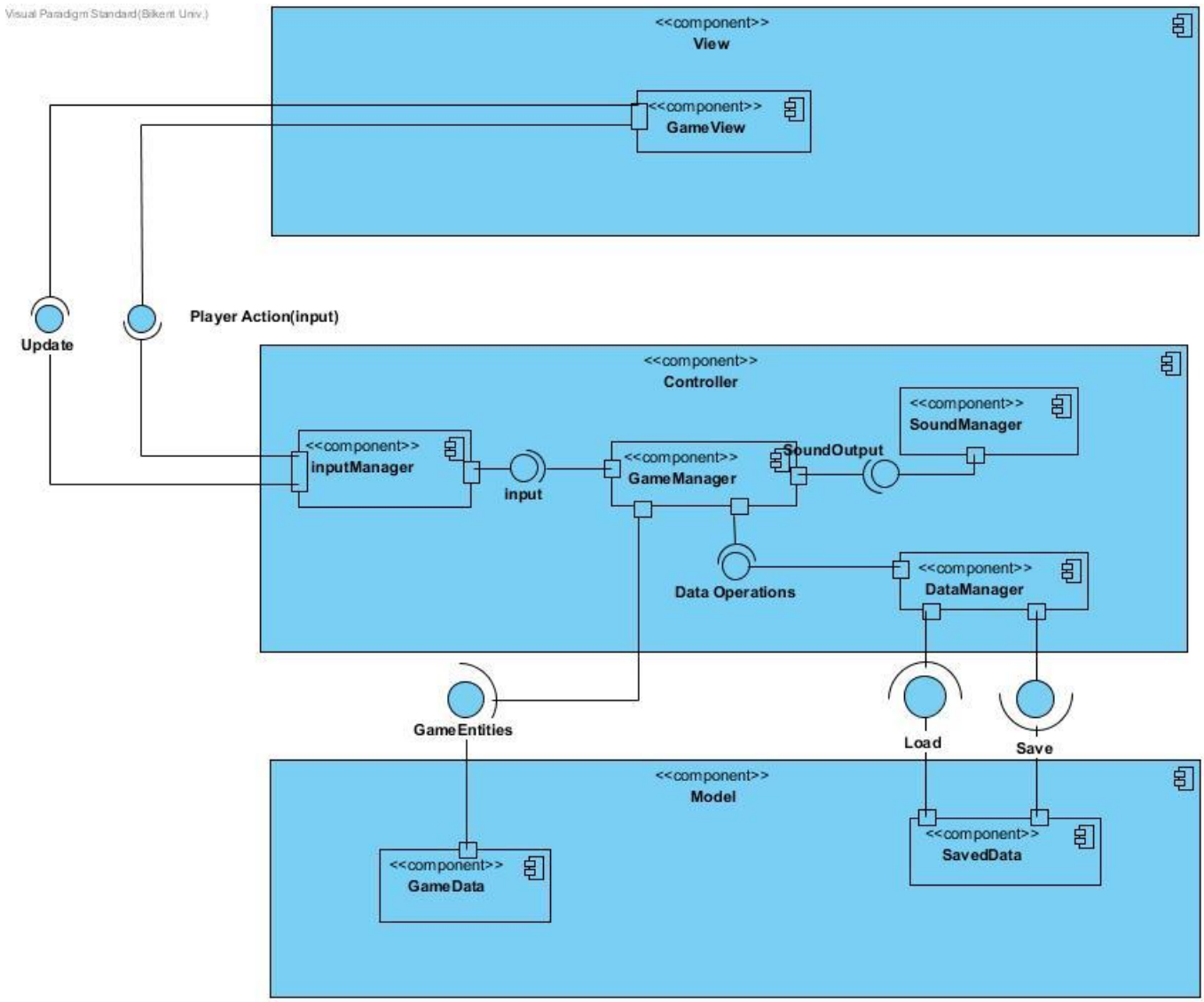
2.Software Architecture

2.1 Subsystem decomposition

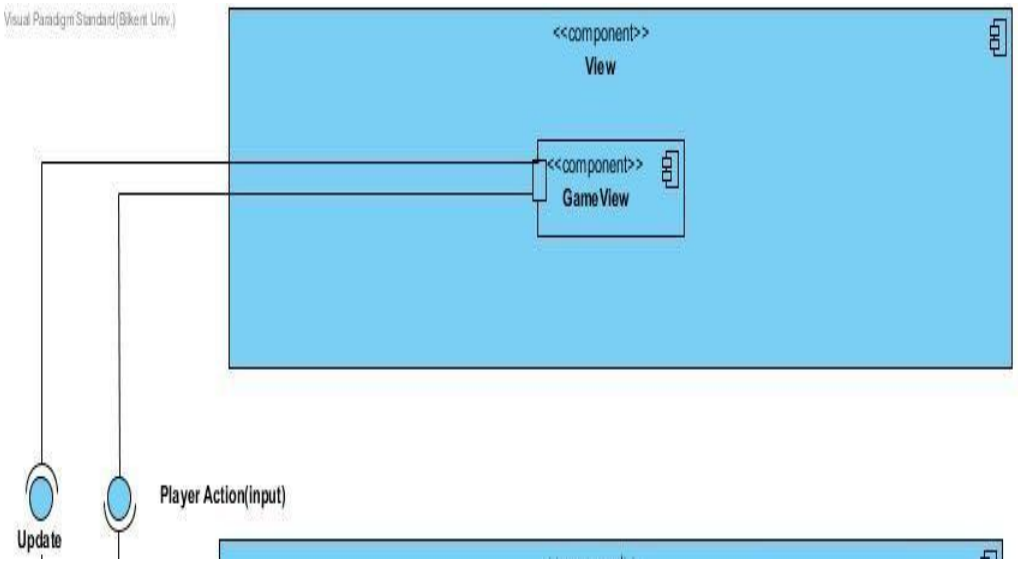
With the purpose of building a strategy game in which visualizations take a considerable part in the design as much as computations do we decided that the best architectural style to use for modeling the subsystems is MVC. Where we can handle all user interface related issues in

the view subsystem while doing the computations that needs to be done behind the scenes in the controller subsystem. Controller subsystem is going to handle the user input make the necessary computations and transmit the new data to model subsystem. Controller subsystem would also notify view subsystem which will update the view accordingly.

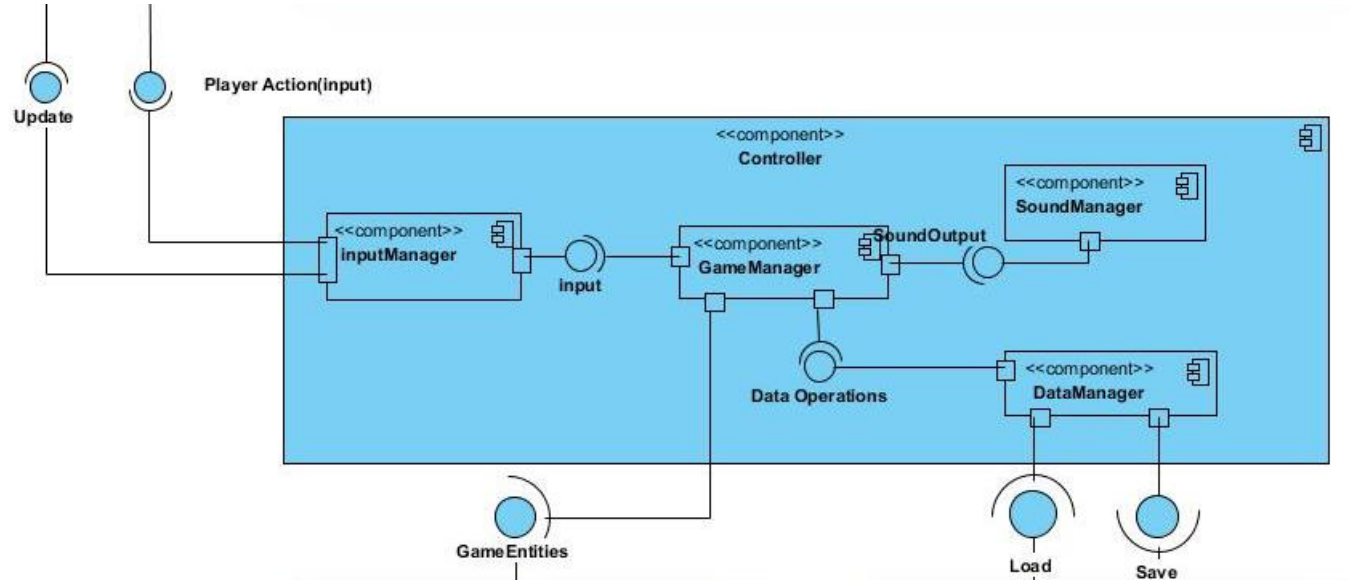




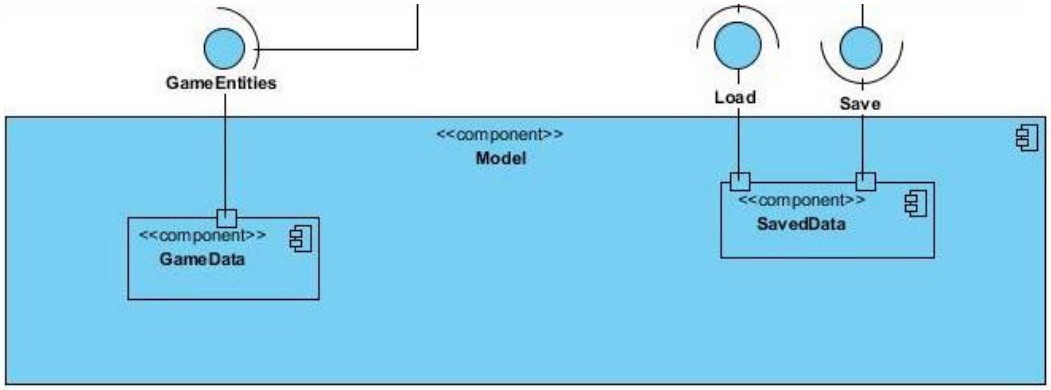
The MVC modeling architecture has 3 subsystems included in it: Model, view and controller. The controller is in relation with both model and view as it is the central subsystem for the program taking care of the user input, making the necessary computations and transmitting the new data to model subsystem.



The relation between controller and view subsystems is about the handling of user interface. The view interface includes game view component in it which handles the visualization of the frames of game including all the menus, sub menu pages and gameplay. The view provides the controller subsystem with user actions to which it has access through user interface(The mouse and keyboard actions for our particular case) . Controller uses these action update to calculate some specific data and process the flow of the game. Controller subsystem in return provides the view subsystem with the updates that are necessary to be visualized in the upcoming frame to inform the user. View subsystem draws every new frame doing these necessary changes coming from the controller subsystem.



The Controller subsystem has several duties such as realasing the input coming from the View via the input manager. Use the realized input to calculate new state of the frame and update the view with the information of the new state. The calculations are done in the game manager subsystem processing the input and already existing data. Sound manager is to manage the sound output of the game and last but not least the data manager is to handle the issues of loading and storing from/to data files in the model subsystem.



Finally the model subsystem is where all the data is stored regarding the system. The saved data which includes the data files recording the information of the level the user has access to and preferred settings and also game data holding the game entities which are needed to initialize the game.

2.2 Hardware/software mapping

The game will be developed using Java so Java Runtime Environment is needed to execute the game. Regarding the hardware requirements, our game is played with a mouse. Moreover the game will use swing awt library for graphics.

2.3 Persistent data management

The data will be stored in the user hard disk and the game doesn't require a database . The data will be accessible at real time so, documents of the game will be stored in the memory to be accessible when the game engine works. Other components of the game such as images are stored as to allow user to make changes to create the game according to personal choices.

2.4 Access control and security

All users who play at the same computer will have a unique id and non-unique password. These id and password will be registered at the beginning of the game. Therefore, every player will use only their account. When user entered the his/her account, user can only load his/her saved data which he did previous game. By this future, users can only access to their game data and according to these, game security and user data protection will be provided.

2.5 Boundary conditions

2.5.1 Initialization

System will be an executable jar file. It will not have .exe extension. That's why, there is no need to install the game.

2.5.2 Termination

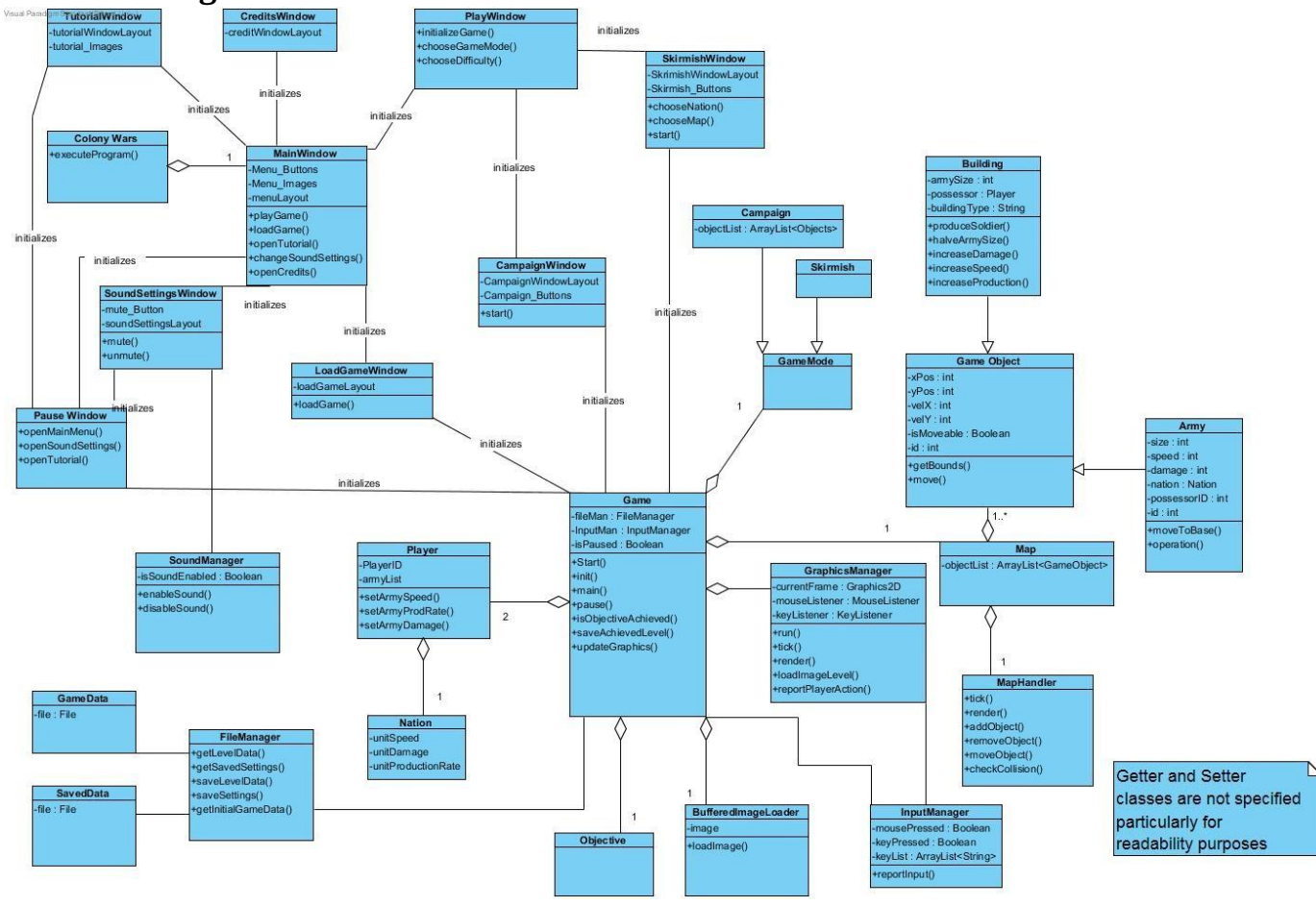
User can use the "x" button which is located at the top-right of the frame or s/he can go to main menu by using the pause menu and then click to quit button for termination.

2.5.2 Starting

User can load his/her last game by using the load button from the menu or can start a new game. User can only load his/her own data.

3.Subsystem services

3.1 Class Diagram



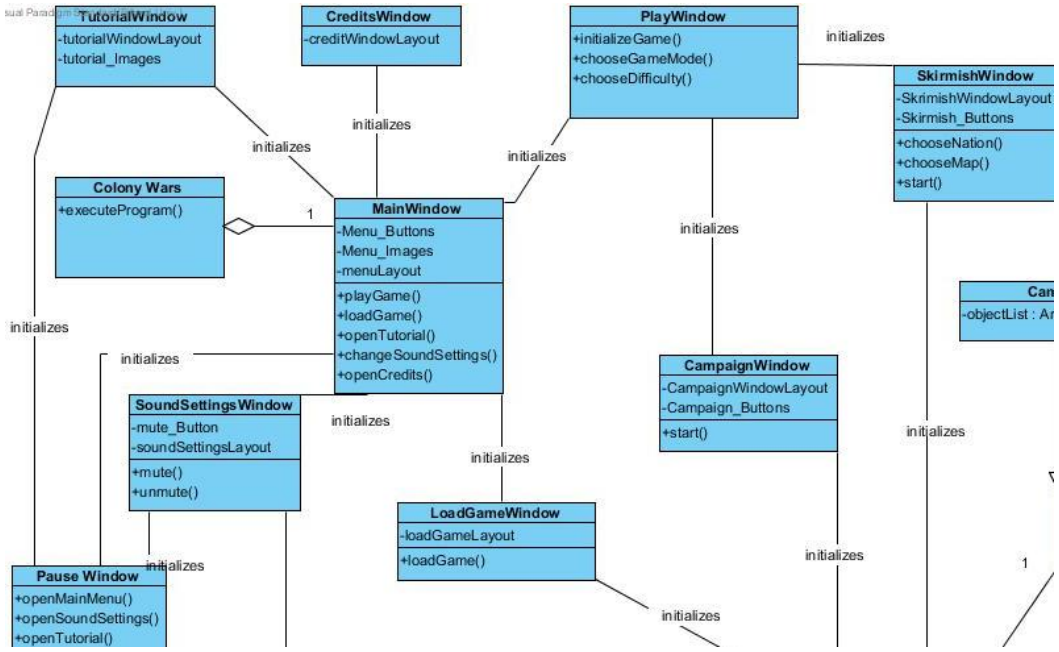
3.2 View Subsystem

The classes included in this subsystem are basically to handle the graphical user

interface of the program.

3.2.1 Game View

This subsystem is to handle the user interface of the software system. It basically includes all the classes related to the topic. GUI classes thats aim is to visualize main menu and other sub menus the main game screen are builded in this subsystem. This subsystem is also responsible for realizing the user input via interactivities between the user and the interface such as a button click or a mouse drag. Sending those realizations to the controller component it eventually gets updates from it and updates the game view accordingly.



3.2.1.1 MainWindow Class

Main window is the first window that is going to be instantiated when the game is launched. It includes buttons for directing the user in another window which are Play Game, Load Game, Open Tutorial, Change Sound Settings and Open Credits buttons.

3.2.1.2 Play Window Class

This window is the one user encounters when they click the Play Game button in main window. This window has several buttons such as Choose Game Mode where the user chooses one of two game modes, Choose Difficulty where the user chooses from one of the 3 levels of difficulty and Initialize Game which directs the user either to Campaign Window or Skirmish Window according to their choice of game mode.

3.2.1.3 Skirmish Window Class

This window has 3 function 2 of which are choosing a map for the game to be initialized on and choosing the nation to be played with. The last function called start starts the game by creating the instance of the game class.

3.2.1.4 Campaign Window Class

This window is pretty simple as it only has one function which starts the game by calling the game by creating the instance of the game class. The reason of it differentiation from Skirmish Window is that one does not have to choose nation and map in campaign mode since those are automatically assigned according to the level of campaign being played.

3.2.1.5 Credits Window Class

This window is basically aims to show plain text giving information about the developers.

3.2.1.5 Tutorial Window Class

This window is basically aims to show plain text giving information about the game mechanisms and basic gameplay.

3.2.1.6 Sound Settings Window Class

This Window is to provide user with the ability to un/mute sound of the game. The window has a button for each function.

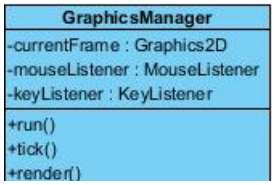
3.2.1.7 Pause Window Class

This window is accessible through the gameplay when the users chooses to pause the game. It has 3 functions which are Open Main Menu, Open Sound Settings, Open Tutorial and has a button for each function.

3.2.1.8 Load Game Window Class

This Window is to load a specific level that played in the campaign mode. It has buttons for every level that the user already played and won and the next level he could play which he can load by clicking the related button.

3.2.1.9 Graphics Manager Class



This Class is the class that draws the frames of the gameplay. It interacts with the Game class from controller subsystem to inform it of user actions (inputs) which it does through reportPlayerAction() operation and takes the updates from the game class to redraw the frame accordingly. It draws the frames using the methods run(), tick() and render().It has 3 attributes one being the current frame which specifies the condition of the current frame and the others are a Mouse Listener and Key Listener which are useful for realizing the user interaction.

3.2.1.10 Buffered Image Loader

This class loads the images that represent the gameobjects in the game such as buildings, armies and also background image using the operation loadImage().

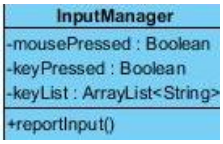
3.3 Controller Subsystem

This subsystem is for realising the interactaion between the user and the interface. It realizes the input provided by view manager and computes/changes game entities accordingly. Computing new entities the controller subsystem stores new data to model subsystem . The system also handles the sound that is played during the game and also handles the issues related to saving and loading the game.

3.3.1 Input Manager

Input manager is responsible for realising the interactaion of the user with user interface. Provided with the action of user from the view model the input manager transmits this information to the game manager.

3.3.1.1 Input Manager Class



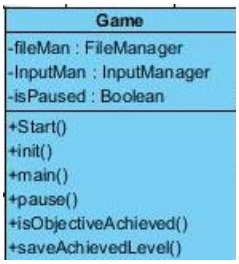
This class has 3 attributes whose 2 are Boolean values for checking if any inputs are realized from the graphic manager : mousePressed and keyPressed. The last one is a Arraylist of strings which specify the list of keys from the keyboard that are valid inputs to the game.

3.3.2 Game Manager

There are various duties that are handled by the game manager. Its basic purpose is to manage the main game logic.

It is responsible for calling the operations that correspond to saving and loading the game, managing the sound functionality of the game via the sound manager. It also updates and realizes the current data by its relationship with the model subsystem through data manager subclass.

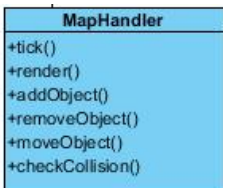
3.3.2.1 Game Class



The game class is the main class in controller subsystem which handles all the calculations and manages the game logic. It has 3 attributes which are file manager, input manager and a boolean value: isPaused which indicates if the game is currently paused or not. The class has init() operation which initializes game objects and all, start() operation which calls the main(). Main() is the main game loop in all calculations and updates are made. Pause() is to pause the game and bring the pause menu. isObjectiveAchieved() checks if the objective is done by either AI or human player. saveAchievedLevel() after winning a

level in campaign mode user saves their process to continue from next level later on. updateGraphics() is the operation that updates the user interface after necessary calculations are made.

3.3.2.2 Map Handler Class

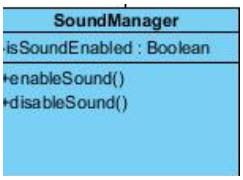


This class handles the conditions of all the game objects that are currently included in the map. It has several operations such as tick() and render() which are used for drawing the game objects on map. addObject() and removeObject() which are used for adding an object to the map or removing one from it. The last operation is called checkCollision () which checks if any two objects in the map collide incase they do there would various operations to take.

3.3.3 Sound Manager

This subclass is responsible for handling the sound output from the system. It provides the gama manager with the corresponding output which eventually play the sound.

3.3.3.1 Sound Manager Class



This class has an attribute of Boolean type that is to check if the sound is on for the game or off. There two operations included in the game which are enableSound() and disableSound(). These operations are to un/mute the game sound.

3.3.4 Data Manager

Data manager is the subclass which game manager calls upon a request of saving or

loading a game that is given by the user. It updates the saved data component if the user asks for saving the game and realizes the saved game if the user requests to load a game.

3.3.4.1 File Manager Class

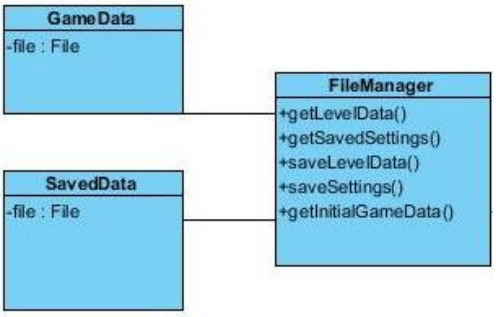
This class takes care of the data storing and loading. The data stored includes both the data needed to initialize the game and the data of saved games. It has 5 operations in total. First one is getLevelData() which is for loading the campaign level that player is able to play next whereas setLevelData() is to store this data in the file. getSaveSettings() is to load the preferred settings that the user chose before and saveSettings() operation is used to store this data in the file. getInitialGameData() is to load the data that is useful for initializing the game such as maps, locations of bases in the map etc.

3.4 Model Subsystem

This Subsystem is the collection of subclasses which are responsible for managing the data related to game entities which are stable information, the data related to the current game which the controller queries to build the current game and provide its continuity and data related to saved games for the user to be able to continue an early saved game by loading it back.

3.4.1 Game Data

This subclass is to hold all the information related to the game that is: Game entities. It holds all the information from maps to constants to be used in the game. When game is built the game manager uses this subclass to access the game entity values.



3.4.1.1 Game Data Class

It is the class where the file for storing the data needed in the initializing of the game exists. The game data class provides information the the controller subclass through File Manager class.

3.4.2 Saved Data

This subclass is to store the values that are related to a specific game session such as map, base and army informations. In case of load request it communicates with the data manager from where data eventually reaches the game manager leading to the rebuilding of the game.

3.4.2.1 Saved Data Class

It is the class that contains the file where information about users accessible level regarding the campaign mode and his/her preffered settings are stored. The saved data class provides information to the controller subclass through File Manager class.