CS319 - Object-Oriented Software Engineering
System Design Report

Run To Live

Group 2

Mehmet Nuri Yumuşak

Ege Yosunkaya

Merve Çapar

Selin Özdaş

1.Introduction

1.1 Purpose of the system

Main intention of Run for Live is constructing enjoyable strategy game with ordinary GUI. The aim of player is escaping from virus in accordance to given statistics about country, the killing percentage of virus etc. What makes game enjoyable and different is its own specific rules in escaping stage of player from virus. System has lots of data and game is constructed on these data about virus. These data vary for every level of game. So, adjustment of level is determined by statistics of virus and countries. The game also increases enjoy and excitement by giving incentives in other words power ups in certain situations. It is 2D game and graphical user interface will not be professional but it will satisfy user with easy and user friendly interface. Run for Live aims gaining to user thinking multidimensional and logical decision making ability besides having enjoy the game.

1.2 Design Goals

After defining application domain, it is important to construct solution domain. Defining solution domain starts with identifying design goals. Nonfunctional requirements and trade-offs are the basis of design goals.

End user Criteria

- **User-Friendliness:** Run for Live is enjoyable game and there should be some criteria to make the game adaptable more easily. In that sense, user-friendliness and design the UI of the game is big concern for who plays it. It may seem to be difficult to understand rules of the game but simple and compact design of user interface diminish all complexity of the rules because although there will be lots of data and statistics, the way that presentation of these information has simple, clear and organized features.

- **Ease of Learning:** If game will give player fun, player should know and adapt the game and understand the aim of the game in a clear way. Our system has special rules. Therefore, it is prominent to learn how to play Run for Live and to determine his strategy in accordance with rules for user. Because of these reasons, system provide user with help documentation and instructive information in a help window.
- **Robustness:** System provides robust software that handles errors and erroneous input of player and developer. Run for Live is a kind of strategy game and it has some amount of statistical data. These numerical data require computational operations to adjust level of the game. Therefore, system should be robust in terms of handle arithmetic exceptions and errors of input. To fulfill such requirement, system is designed robust and responsive for such detrimental cases.
- **Efficiency:** Run for Live is a survival game and when game in play, program should be responsive in terms of speed and visualization. To handle these issue, system use WPF to make actions and movements in time and smooth. These criteria determine the performance and usability of game. WPF uses special libraries: System.Windows.Media, System.Windows.Media.Animation, System.Windows.Media.Effects etc. With usage of less memory, game catches high performance.

Developer Criteria

- **Extendibility:** As an object oriented program, Run for Live should provide easy and clear facility of improvements. After some time, in accordance with bugs, feedback from players in View Credits section program need to be maintained. Because of these improvement issues, classes that will be maintained or modified and related classes should be well organized so that modifications and problems that may occur from changes can be handled with less effort on these.
- **Reliability:** In run-time, system should not crash because of the exceptions that may be occur when unexpected input or unexpected component in calculations like divided by zero. To achieve not to get such errors and to make system bug free, test stage is important for us as developer.
- **Modifiability:** System has extensible and organization or content of classes may be changed in the future. So, modifiability is important concern for developer of the game. Therefore, subsystems of the game are minimized and created clearly.

Trade-Offs

- **Functionality vs Usability**

Run for Live should appeal people from different range of ages. Functionality is important concern for us but usability has priority compared to functionality. Of course, functionality is necessary for software program however we produce this software for people who do not actually interested in how complex the game but how easy and understandable the game for play purpose. Therefore, sacrificing more complex functionality brings us to more usable and addressable for various range of people game to produce.

- **Cost vs Robustness**

Our system must be robust in terms of calculations. As stated analysis design report, Run for Live is a kind of strategy game and in every turn system need to make calculations in accordance with our specific rules of game. These are calculating percentage of dead people, infected people and how virus can influence people in terms of its calculated killing percentage and there are lots of calculations for statistics of countries. To achieve this process, system should be powerful in terms of arithmetic operations but we this will be achieved, cost of calculations namely complexity of the program is the first thing to be sacrificed. We are studying on achieving robust software with minimal complexity.

1.3 Definitions, Acronyms and Abbreviations

GUI: [1] Graphical User Interface

WPF: [2] Windows Presentation Foundation

1.4 References

[1] http://www.computerhope.com/jargon/g/gui.htm

[2] https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx

2. Software Architecture

2.1. Overview

This section is about the fundamental structures of the system, creation of those structures and their documentation. Object Oriented software design allows us to divide our project into subsystems having their own unique functionality. On the other hand, we are going to use Model-View-Controller system design pattern to establish the connection between User Interface and the coded subsystems. This way is chosen to make our system more comprehensible and make our subsystems compatible with each other. Those design patterns hopefully will lead the system to satisfy both functional and nonfunctional facilities we have defined in the analysis report.

2.2. Subsystem Decomposition

Our application system is divided into three subsystems which having their unique functionality (Figure 2.2.1). According to those functionality, decomposed parts are named as User Interface, Game Management and Game.

User Interface part is mainly coming after the UIManager class, which is the more important one since all other classes in this subsystem must use it.
On the other hand Game Management is the one subsystem to control and construct the game and it has all the functionalities to manage the objects.
The last subsystem, called Game, is the one including the objects which the game logic has to have. If we see those objects as identities, it would not be wrong to call Game Management subsystem as identity controller.

The only connection between the first and second subsystem is the GameManager class which will be affected by the changes in the first subsystem. At the same time, the second one and third one has a connection which is the Game class managed by GameManager class. Those decisions has been made to minimize the workload and performance slippage of the overall system, if there has to be a change in one of the subsystems.

Those three subsystems are designed to be independent as more as it could be to minimize the error effection has been made in one of those subsystems and to satisfy our design goals. Only the first subsystem partitions will be visible for the user/player. The second one is a controller system and the third one is the game structures.

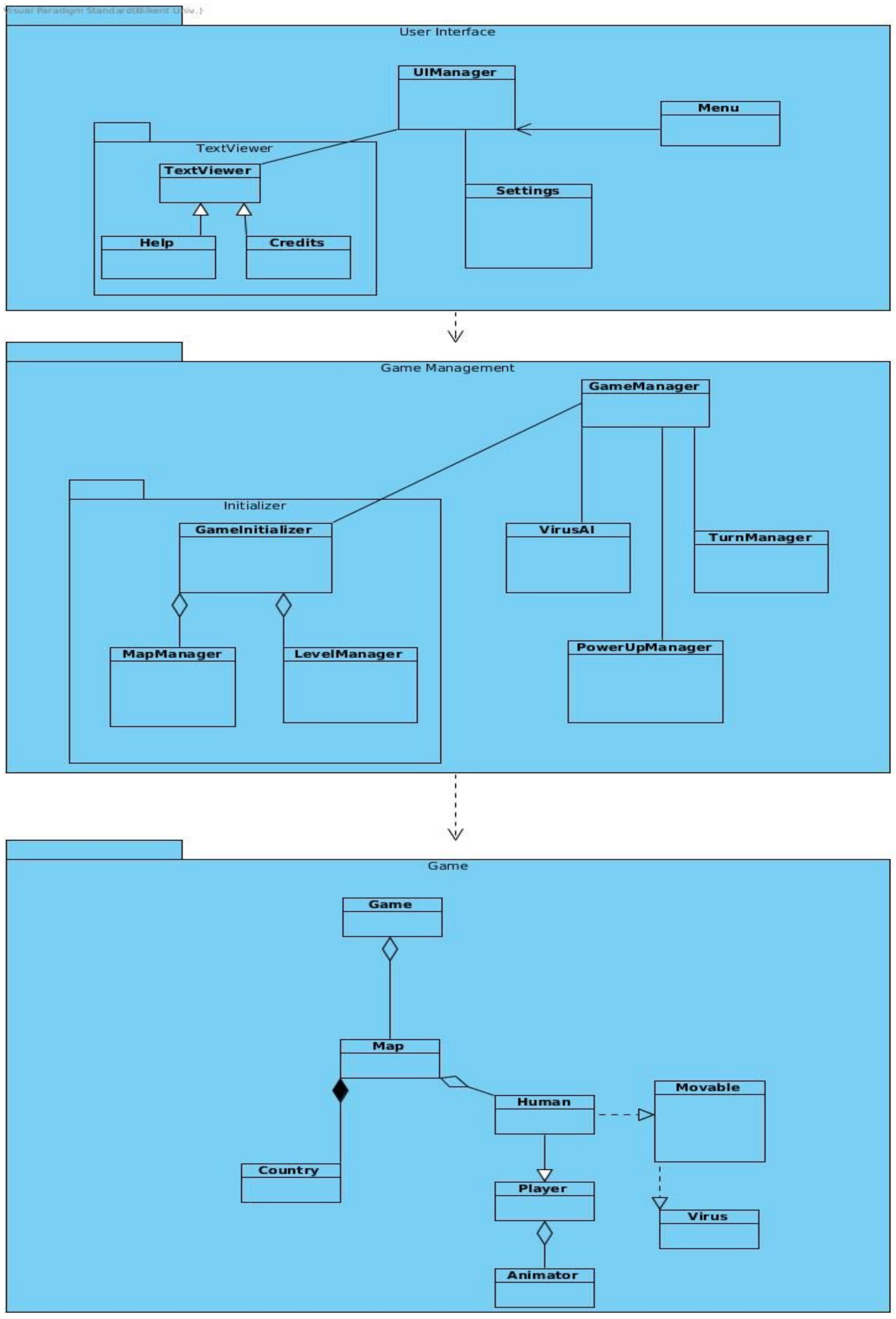


Fig 2.2.1: Basic System Decomposition (Detailed figure for system decomposition: http://tinypic.com/r/30sd73n/9 **)**

2.3 Hardware / Software Mapping

Our application, Run to Live, will be implemented in C# via using Visual Studio SDK on .NET platform.

Software requirements:

- .txt support (to store information)
- Any C# compiler to compile and run the application

Hardware requirements:

- Any computer which can work with basic applications
- OEM input device(Mouse)

2.4 Persistent Data Management

Run to Live will be an lightweight application so that data storage will be minimum and restricted with .waw .gif .jpg .txt files for the audio, image and information.

2.5 Access Control and Security

Runtime of our application does not have any dependency on Internet and the user since it does not have any save/load operations. Any access control system and restrictions due to security issues will not be needed.

2.6 Boundary Conditions

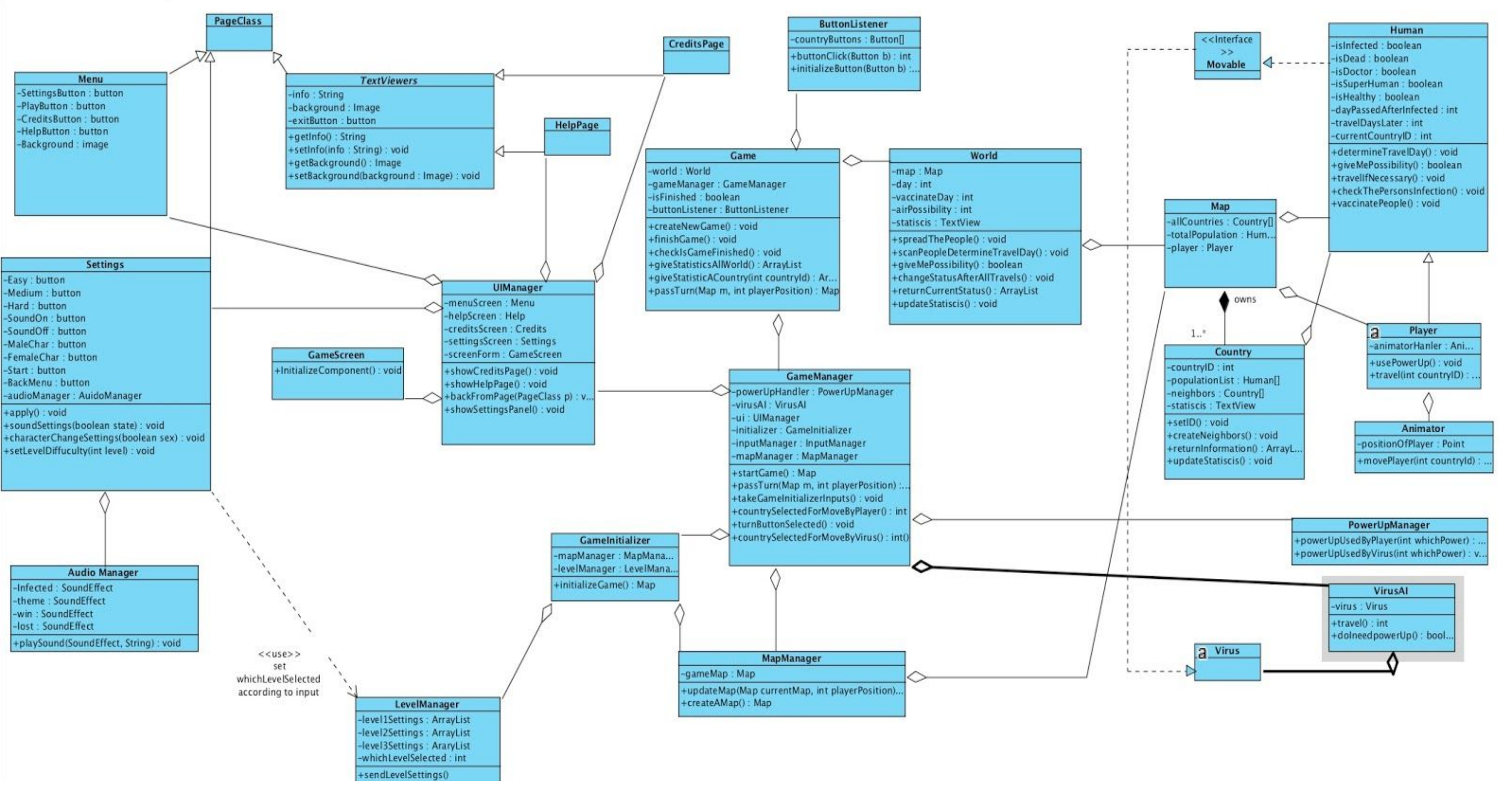
Initialization: Game does not need any installation process, users only need to to run the program on the C# compiler. Game will be stored as an executable .jar file instead of an .exe file.

Termination: User may leave the game whenever he/she wants to to by clicking on the related quit operation button.

Error: Since any user information will not be stored after the termination, any error occurrence may lead user to lose all of data related to game, user has to restart the game. If external game files can not be loaded by the system, game will terminate itself.

3. Subsystem Services

3.1 Detailed Object Design


PageClass
Menu
-SettingsButton : button
-PlayButton : button
-CreditsButton : button
-HelpButton : button
-Background : image
TextViewers
-info : String
-background : Image
-exitButton : button
+getInfo() : String
+setInfo(info : String) : void
+getBackground() : Image
+setBackground(background : Image) : void
CreditsPage
HelpPage
ButtonListener
-countryButtons : Button[]
+buttonClick(Button b) : int
+initializeButton(Button b) : ...
Game
-world : World
-gameManager : GameManager
-isFinished : boolean
-buttonListener : ButtonListener
+createNewGame() : void
+finishGame() : void
+checkIsGameFinished() : void
+giveStatisticsAllWorld() : ArrayList
+giveStatisticACountry(int countryId) : Ar...
+passTurn(Map m, int playerPosition) : Map
World
-map : Map
-day : int
-vaccinateDay : int
-airPossibility : int
-statiscis : TextView
+spreadThePeople() : void
+scanPeopleDetermineTravelDay() : void
+giveMePossibility() : boolean
+changeStatusAfterAllTravels() : void
+returnCurrentStatus() : ArrayList
+updateStatiscis() : void
<<Interface>>
Movable
Human
-isInfected : boolean
-isDead : boolean
-isDoctor : boolean
-isSuperHuman : boolean
-isHealthy : boolean
-dayPassedAfterInfected : int
-travelDaysLater : int
-currentCountryID : int
+determineTravelDay() : void
+giveMePossibility() : boolean
+travelIfNecessary() : void
+checkThePersonsInfection() : void
+vaccinatePeople() : void
Map
-allCountries : Country[]
-totalPopulation : Hum...
-player : Player
owns
1..*
Country
-countryID : int
-populationList : Human[]
-neighbors : Country[]
-statiscis : TextView
+setID() : void
+createNeighbors() : void
+returnInformation() : ArrayL...
+updateStatiscis() : void
Player
-animatorHanler : Ani...
+usePowerUp() : void
+travel(int countryID) : ...
Animator
-positionOfPlayer : Point
+movePlayer(int countryId) : ...
Settings
-Easy : button
-Medium : button
-Hard : button
-SoundOn : button
-SoundOff : button
-MaleChar : button
-FemaleChar : button
-Start : button
-BackMenu : button
-audioManager : AuidoManager
+apply() : void
+soundSettings(boolean state) : void
+characterChangeSettings(boolean sex) : void
+setLevelDiffuculty(int level) : void
UIManager
-menuScreen : Menu
-helpScreen : Help
-creditsScreen : Credits
-settingsScreen : Settings
-screenForm : GameScreen
+showCreditsPage() : void
+showHelpPage() : void
+backFromPage(PageClass p) : v...
+showSettingsPanel() : void
GameScreen
+InitializeComponent() : void
GameManager
-powerUpHandler : PowerUpManager
-virusAI : VirusAI
-ui : UIManager
-initializer : GameInitializer
-inputManager : InputManager
-mapManager : MapManager
+startGame() : Map
+passTurn(Map m, int playerPosition) : ...
+takeGameInitializerInputs() : void
+countrySelectedForMoveByPlayer() : int
+turnButtonSelected() : void
+countrySelectedForMoveByVirus() : int()
PowerUpManager
+powerUpUsedByPlayer(int whichPower) : ...
+powerUpUsedByVirus(int whichPower) : v...
VirusAI
-virus : Virus
+travel() : int
+doIneedpowerUp() : bool...
Virus
Audio Manager
-Infected : SoundEffect
-theme : SoundEffect
-win : SoundEffect
-lost : SoundEffect
+playSound(SoundEffect, String) : void
GameInitializer
-mapManager : MapMana...
-levelManager : LevelMana...
+initializeGame() : Map
<<use>>
set
whichLevelSelected
according to input
MapManager
-gameMap : Map
+updateMap(Map currentMap, int playerPosition)...
+createAMap() : Map
LevelManager
-level1Settings : ArrayList
-level2Settings : ArrayList
-level3Settings : AraryList
-whichLevelSelected : int
+sendLevelSettings()

3.2 User Interface Subsystem

Classes: UIManager , TextViewers , Help , Credits , Settings , Menu ,PageClass , GameScreen

UIManager class provides screen according to decisions have been made by the user. The attributes of the help , credits , settings and menu screens are provided by the corresponding class to the UIManager class.

UIManager class initializes GameManager class in the Game Management package.

3.3 Game Management Subsystem

Classes: Initializer Package (GameInitializer , MapManager , LevelManager) , GameManager , VirusAI , PowerUpManager

Game management subsystem initializes game element objects , process the information and updates the corresponding objects.

GameManager class initializes GameInitializer and GameInitializer creates LevelManager and MapManager. LevelManager holds predefined settings for 3 difficulties and selected difficulty , furthermore it provides the information to GameInitializer. MapManager creates a map when starting a game then updates the map throughout the game.

GameManager class creates Game object.

3.4 Game Elements Subsystem

Classes: Game , Map , Country , Human , Player , Animator , Movable , Virus

Game logic subsystem consist of elements of the game such as Player , Virus other Humans and Countries etc. These classes holds information and provide it to game management system when needed.

Game class holds the information about the game and process it. It checks whether the game is finished or not and gives statistics about the whole countries and about every single country.