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 and 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 Elements and WPF.

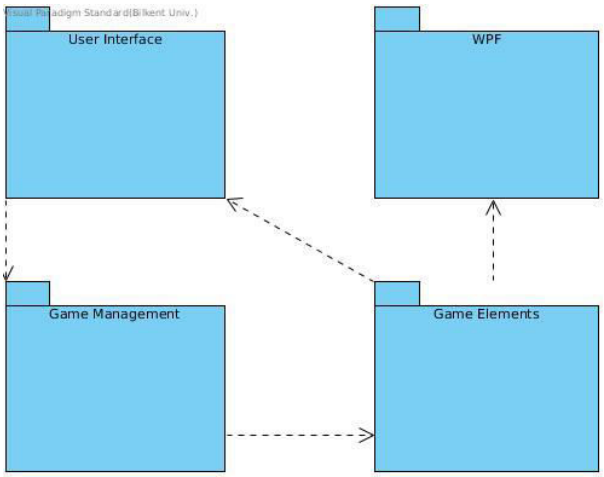


Figure 2.2.1: High level decomposition of the system

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.

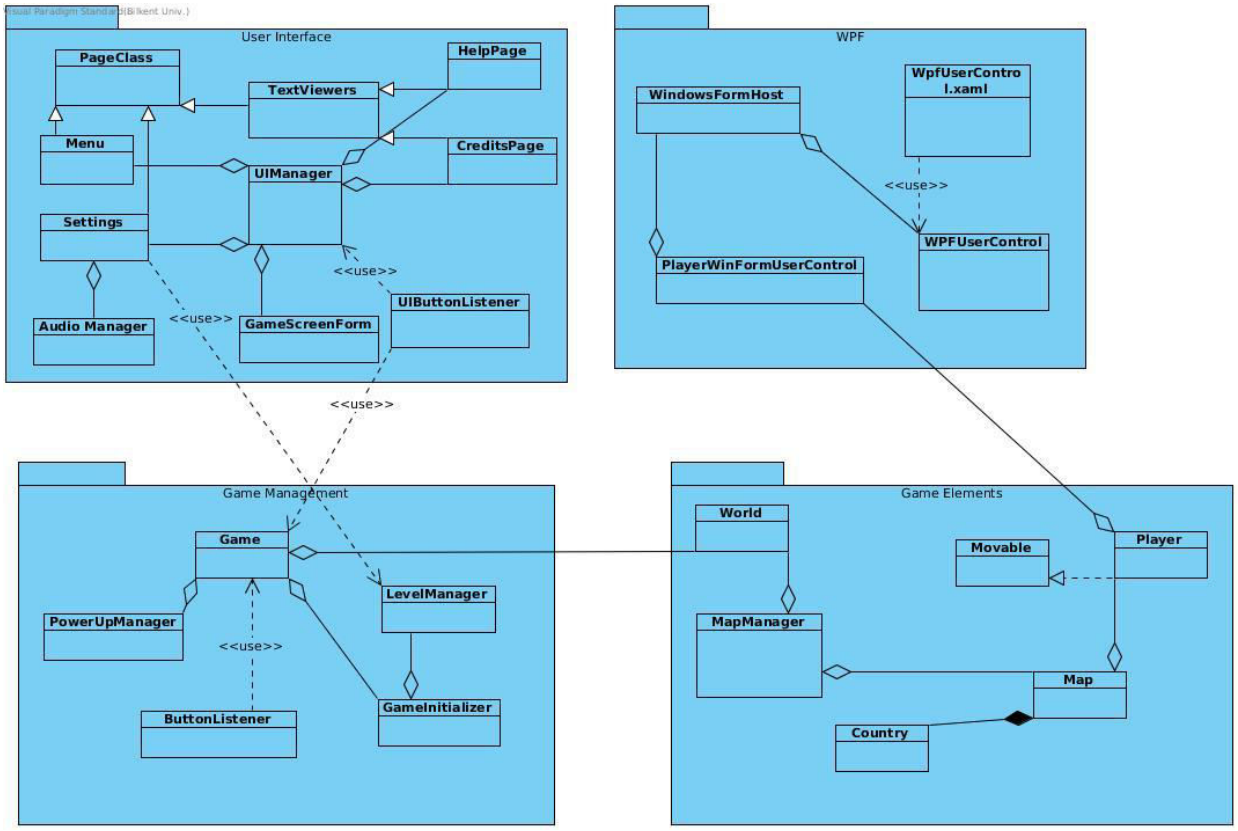
On the other hand Game Management is the one subsystem to control and construct the game and it has all the functionalities to manage the objects.

WPF subsystem is the one responsible for the UI management of the player

The last subsystem, called Game Objects, 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 User Interface and Game Management subsystem are only Game and Level Manager classes which will be affected by the changes in the User Interface subsystem. At the same time, the Game Management has a connection which is the World class managed by MapManager class. Lastly, Player class of Game Objects has a connection with WindowsFormHost class of WPF. 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 four 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 User Interface subsystem partitions will be visible for the user/player. Game Management subsystem is a connector between the user and application. Game Objects subsystem is holding the game related objects and WPF subsystem manages the player actions.



2.3 Layers

Since we have used MVC(Model View Controller) in our software, We have divided our system into three parts which are the Model, Controller and View.

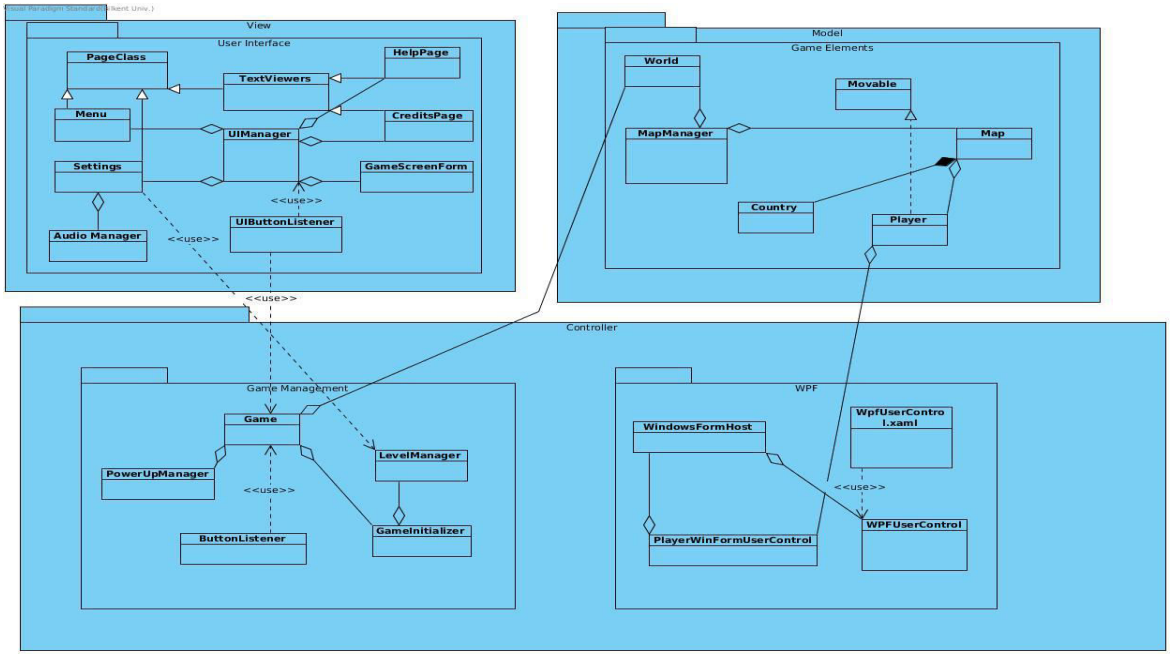
Model part includes the Game Entities subsystem since it manages the data, logic and flow of the events.

View is the one including UI Management subsystem since it is the one generating the output according to the updates coming from model.

Controller as the last one includes Game Management and WPF subsystems since this part is the one to manipulate model part and (only in some cases) to update the view according to the user's changes on the related software partition.

Benefits:

- The separation of these three components, allows us to re-use the logics across applications.
- User interface design can be improved or changed without becoming stuck with the other components of the system.
- Since design pattern makes the UI independent from the model, model can be changed without concerning about the UI.
- This decomposition and bundling of the system makes it easier to work independently and concurrently for us.
- Revisions and edits lead only the related part to change.



2.4 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)

2.5 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.

2.6 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.

2.7 Boundary Conditions

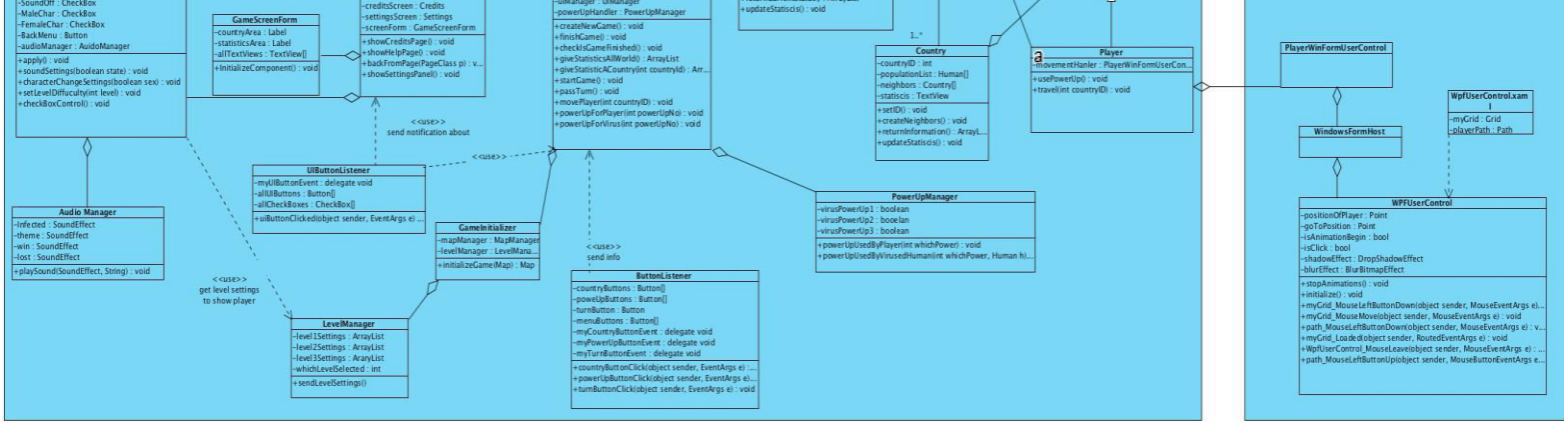
Initialization: Game does not need any installation process, users only need to to run the program on the C# compiler. Game will be stored as an executable .jar file instead of an .exe file.

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.

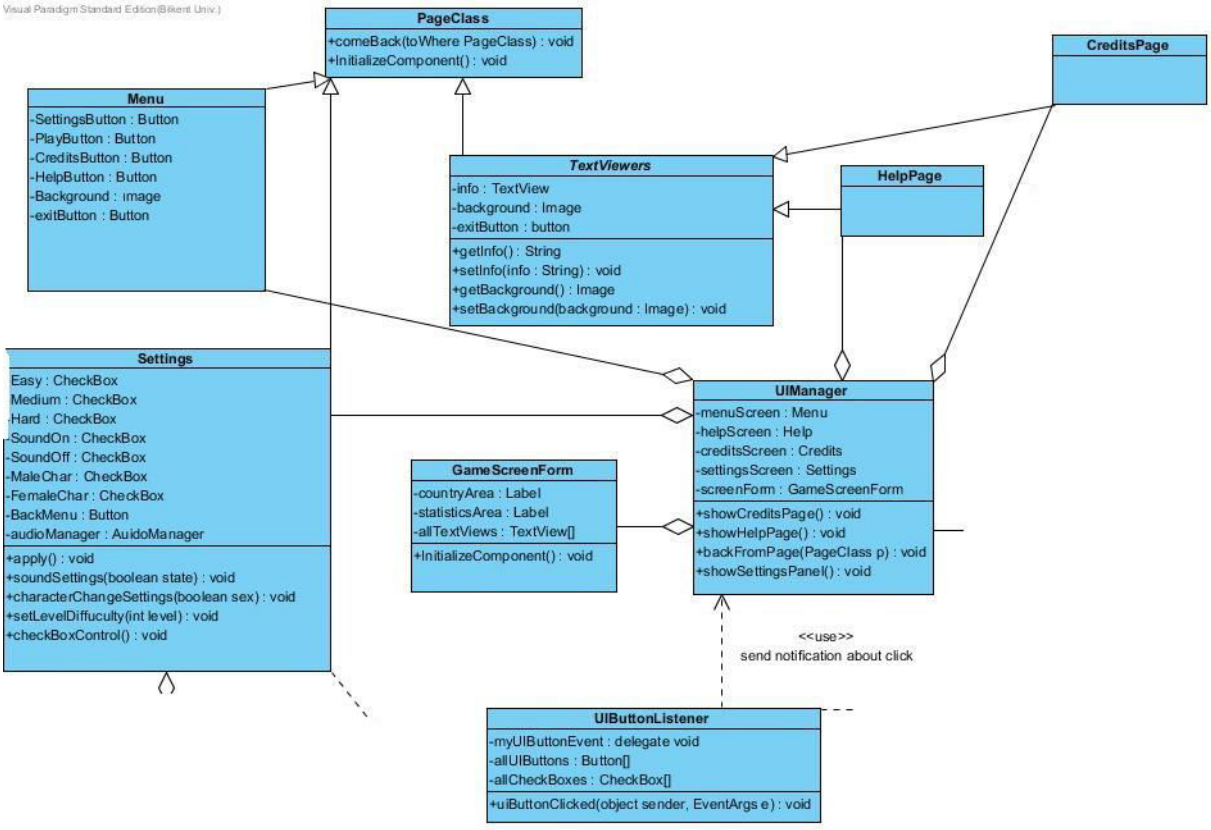
3. Subsystem Services

3.1 Detailed Object Design

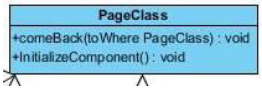

-SoundOff : CheckBox
-MaleChar : CheckBox
-FemaleChar : CheckBox
-BackMenu : Button
-audioManager : AuidoManager
+apply() : void
+soundSettings(boolean state) : void
+characterChangeSettings(boolean sex) : void
+setLevelDiffuculty(int level) : void
+checkBoxControl() : void
GameScreenForm
-countryArea : Label
-statisticsArea : Label
-allTextViews : TextView[]
+InitializeComponent() : void
-creditsScreen : Credits
-settingsScreen : Settings
-screenForm : GameScreenForm
+showCreditsPage() : void
+showHelpPage() : void
+backFromPage(PageClass p) : v...
+showSettingsPanel() : void
<<use>>
send notification about
<<use>>
UIButtonListener
-myUIButtonEvent : delegate void
-allUIButtons : Button[]
-allCheckBoxes : CheckBox[]
+uiButtonClicked(object sender, EventArgs e) ...
Audio Manager
-Infected : SoundEffect
-theme : SoundEffect
-win : SoundEffect
-lost : SoundEffect
+playSound(SoundEffect, String) : void
<<use>>
get level settings
to show player
GameInitializer
-mapManager : MapManager
-levelManager : LevelMana...
+initializeGame(Map) : Map
LevelManager
-level1Settings : ArrayList
-level2Settings : ArrayList
-level3Settings : AraryList
-whichLevelSelected : int
+sendLevelSettings()
<<use>>
send info
ButtonListener
-countryButtons : Button[]
-poweUpButtons : Button[]
-turnButton : Button
-menuButtons : Button[]
-myCountryButtonEvent : delegate void
-myPowerUpButtonEvent : delegate void
-myTurnButtonEvent : delegate void
+countryButtonClick(object sender, EventArgs e) :...
+powerUpButtonClick(object sender, EventArgs e)...
+turnButtonClick(object sender, EventArgs e) : void
-powerUpHandler : PowerUpManager
+createNewGame() : void
+finishGame() : void
+checkIsGameFinished() : void
+giveStatisticsAllWorld() : ArrayList
+giveStatisticAtCountry(int countryId) : Arr...
+startGame() : void
+passTurn() : void
+movePlayer(int countryID) : void
+powerUpForPlayer(int powerUpNo) : void
+powerUpForVirus(int powerUpNo) : void
+ updateStatistics() : void
1..*
Country
-countryID : int
-populationList : Human[]
-neighbors : Country[]
-statistcis : TextView
+setID() : void
+createNeighbors() : void
+returnInformation() : ArrayL...
+updateStatistcis() : void
Player
-movementHanler : PlayerWinFormUserCon...
+usePowerUp() : void
+travel(int countryID) : void
PowerUpManager
-virusPowerUp1 : boolean
-virusPowerUp2 : boolean
-virusPowerUp3 : boolean
+powerUpUsedByPlayer(int whichPower) : void
+powerUpUsedByVirusedHuman(int whichPower, Human hi...
PlayerWinFormUserControl
WindowsFormHost
WpfUserControl.xaml
-myGrid : Grid
-playerPath : Path
WPFUserControl
-positionOfPlayer : Point
-goToPosition : Point
-isAnimationBegin : bool
-isClick : bool
-shadowEffect : DropShadowEffect
-blurEffect : BlurBitmapEffect
+stopAnimations() : void
+initialize() : void
+myGrid_MouseLeftButtonDown(object sender, MouseEventArgs e)...
+myGrid_MouseMove(object sender, MouseEventArgs e) : void
+path_MouseLeftButtonDown(object sender, MouseEventArgs e) : v...
+myGrid_Loaded(object sender, RoutedEventArgs e) : void
+WpfUserControl_MouseLeave(object sender, MouseEventArgs e) : ...
+path_MouseLeftButtonUp(object sender, MouseButtonEventArgs e...

3.2 User Interface Subsystem

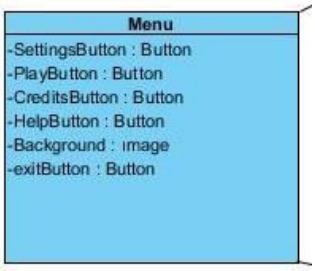
Classes: PageClass, Menu, Settings, TextViewers, CreditsPage, HelpPage, GameScreenForm, UIButtonListener, UIManager



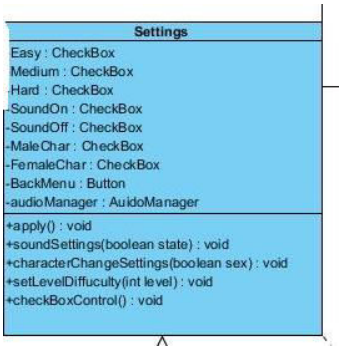
This subsystem is visualization for the player. It has all the page classes to show. UIManager is kind of façade of the system.



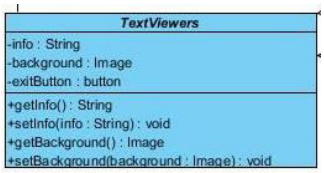
PageClass is an abstract class that defines what is a page actually. All pages extends this class. Because all pages should have back button PageClass has “comeBack” method and this method takes a page parameter that we want to come back.



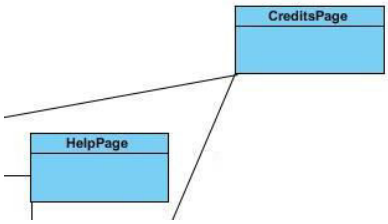
Menu class is the first ui page that player encounter when the game begins. It has standart “Settings” button to arrange settings of the game, beside ui settings game settings is also in this page. It has a background image. “Credits”,”Help” and “Exit” button is also there. All the button events is handled by the UIButtonListener.



This page is to arrange settings about both game dynamics and ui. After you choose the settings by clicking the checkboxes you click apply button and your settings is saved. This class has some methods for instance apply method is used for save the settings and soundSettings method is obviously change the sound according to your choice. CharacterSettings method determines the visualization of the character. SetLevelDifficulty determines the difficulty of the game, difficulties are pre-determined in levelManager class. Lastly, checkBoxContol is for checking if user press both checkboxes that against each other or not.

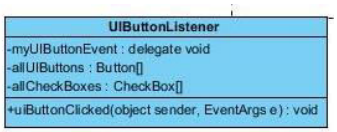


This class is for pages that has only texts and back button in it like “Help” and “Credits” classes.

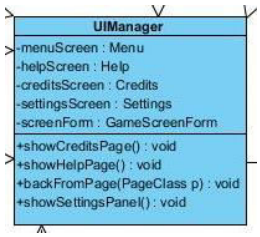


This classes is for show some information to the user. Help class contains informations about how this game is played and credits page contains informations about developers.

This class is where the game is happening. All the countries and statistics and power up buttons and characters and animations will be in this class.

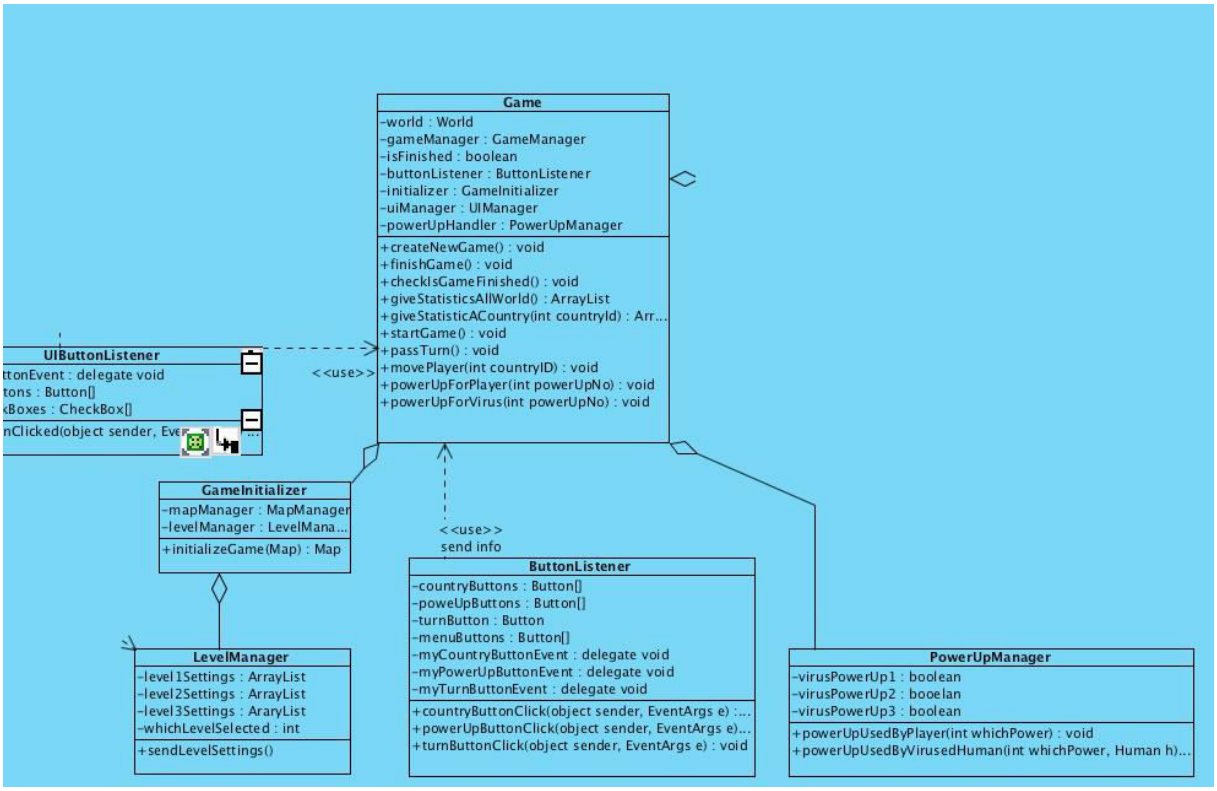


This class makes connection between ui page classes and the managers. It listens if any button is clicked in the all page classes and share this information with the UIManager class. To do this job obviously it should have all the buttons and checkboxes of the whole project. it has one method to determine which button is clicked and according to this it gives necessary informations to UIManager class or GameManager class directly. If click is related to page show or something like that it send to UIManager but if it is like setting level it send GameManager.



This class is kind of façade of this subsystem. It manages this subsystem. It shows the pages. It controls the transitions between classes.

3.3 Game Management Subsystem



Classes: Game,GameInitializer, LevelManager, PowerUpManager, ButtonListener, UIButtonListener

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

Game Class: Game is a core controller class of the project. It controls and arranges the things.

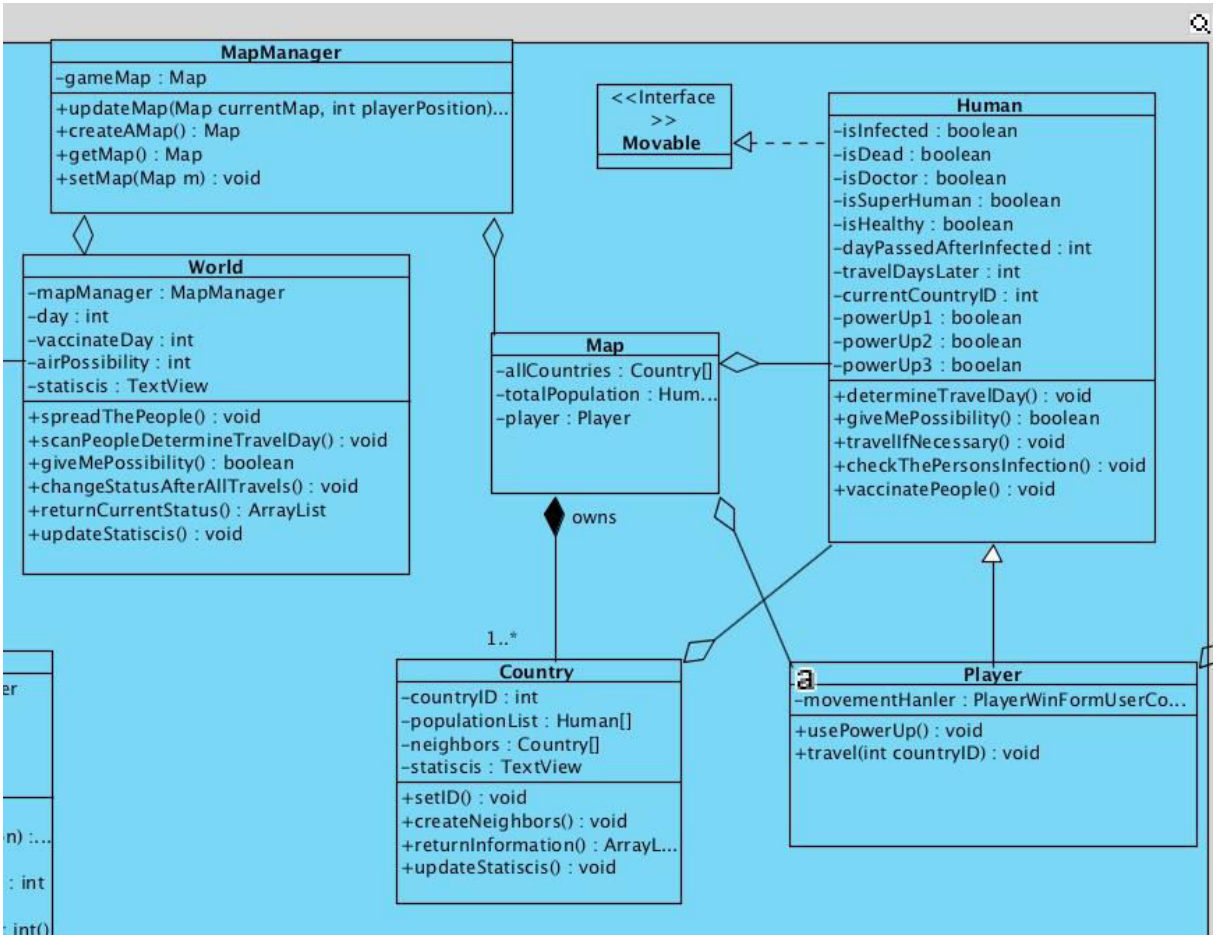
GameInitializer Class: Initializes the game according to the information coming from Level Manager.

LevelManager Class: Responsible to send the chosen level information

PowerUpManager Class: Keep track and send information about power ups to Game.

ButtonListener Class: Initilizes and utilizes the buttons.

UIButtonListener Class: Send information if a button is pressed.



3.4 Game Objects Subsystem

Classes: Map, MapManager, World, Country, Human , Player,<<Interface>>Movable

Map Class: Holds the Country array(represents all the countries in the map), Human array(represents all the humans in the map) and a Player object.

MapManager Class: Holds a map object. Updates the map with the player position parameter.

World Class: Holds a MapManager object, current day, vaccinate day and airPossiblity as integer and statistics of the game as a TextView object. Moreover, functionalities of the class are following:

spreadThePeople(): distiribute the people to the map.

scanPeopleDetermineTravelDay(): TODO

giveMePossibility(): returns a boolean value of the ??

changeStatusAfterTravels(): Adjusts the variables of the class after the travel happens.

returnCurrentStatus(): retunrs a ArrayList of the ???

updateStatistics(): updates the statistics variable of the class.

Country Class: Every country object has a unique ID(countryID) , holds human list specific to the country(populationList: Human[]), list of its neighbors(Country[]), and its statistics as TextView object. Functionality of country class is following:

setID(): sets ID value of the country.

createNeighbors(): set neighbors of the country.

returnInformation(): returns an ArrayList containing information about the Country object.

updateStatistics(): updates the TextView object of the country.

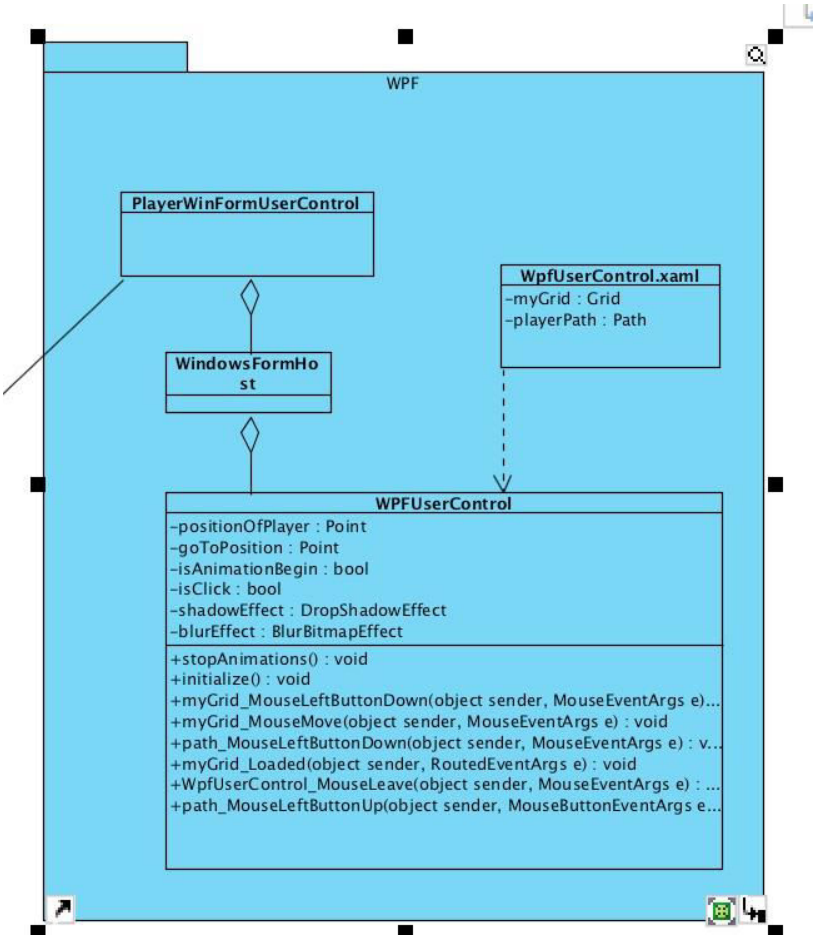
Human Class: Holds important status informations about humans and gives information about itself with related functions.

Player Class: Player “is a” Human. Holds a PlayerWinFormUserControl object which handles the movement UI interactions of the player in a WPF subsystem. It has two functions, usePowerUp, enables user to use power ups and travel(int CountryID), moves player to the given country.

Movable Interface: Human class implements this interface to make sure Human objects apply those rules.

3.5 WPF Subsystem

Classes: PlayerWinFormUserControl, WindowsFormHost, WPFUserControl, WpfUserControl.xaml



Our design uses WPF technology for the player user interface.

PlayerWinFormUserControl: Our project is Windows Form Application and we need WinForm user control to add this control into our project. Because we create and implement user control in WPF type, this WPF control should be transformed into WinForm type. Therefore, PlayerWinFormUserControl class makes possible this transformation. Namely, via this class WPF user control is embedded into WinForm user control.

WindowsFormHost: This class allows you to host WPF user control into WinForm application via .dll file of WPF user control. with adding Solutions file of WinForm project .dll file, hosting is done.

WPFUserControl: It implements fundamental actions of the designed components that is player picture inside the panel. This class includes System.Windows.Media, System.Windows.Media.Animation, System.Windows.Effects libraries to use animation and effect object. The big convenience of these libraries is they provide ready functions like for ease, shadow, gradient etc. You don't have to redesign animation functions, you only use these functions as specified parameters taht what you want.All attributes of animation, effects, implementation of actions, event handlers are coded in this class. WPFUserControl class is WPF type of class. However, our project is Windows Form type application.Therefore, this WPF user control should be embedded into WinForm via WindowsFormHost class.

WpfUserControl.xaml: This class is different type of class in terms of language. This class is coded with XAML(Extensible Application Markup Language) instead of C#. XAML provide developers to control the use of user interface components. XAML has like HTML code style that is all components like button, panel, grid is defined in the tag with special names. In these tags, attributes of created object are defined. This class also provides visual form of your user control. Therefore, we as developers has opportunity to adjust attributes of components via properties window easily.