Bilkent University

CS319 Project / Group: 2B

Endless Dungeon: Progressive Dungeon Crawler RPG

Design Report

Can Özgürel, Alperen Kaya, Alemdar Salmoor, Batuhan Erarslan

Supervisor: Uğur Doğrusöz
TA: Hasan Balcı

Iteration 2 Design Report

Apr 17, 2018

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object-Oriented Software Engineering course CS319.

Table of Contents

Table of Contents 2

1. Introduction 3
1.1 Purpose of the system 3
1.2 Design goals 3

2. Software architecture
2.1 Subsystem Decomposition 7
2.2 Hardware/software mapping 11
2.3 Persistent data management 11
2.4 Access control and security 12
2.5 Boundary conditions 12

3. Subsystem services 12

4. Low-level design 15
4.1 Object design trade-offs 15
Development Time vs Performance 16
4.2 Final object design 17
4.3 Packages 18
4.4 Class Interfaces 18
4.4.1 User Interface Subsystem 18
GameFrame Class 19
PauseMenu Class 20
Menu Class 20
MainMenu Class 21
GameMenu Class 22
SoundManager 22
SettingsMenu Class 23
4.4.2 Game Management Subsystem 23
FileManager Class 24
WaveManager Class 25
GameManager Class 25
InputManager Class 26
4.4.3 Game Objects Subsystem 27
Inventory Class 27
PlayerType Class 28
MageType Class 28
AssassinType Class 28
WarriorType Class 29
Player Class 29
GameObject Class 30
Stats Class 30

Enemy Class 31
EnemyType Class 31
BossType Class 32
RegularType Class 32

5. Glossary & references 32

1. Introduction

1.1 Purpose of the system

Endless Dungeon is the RPG genre game that is falls into entertainment category.At the beginning of a game, the user presented with 3 different types of character types to endorse nonlinear gameplay. The user playing from a first person perspective will have to defeat enemy non player characters (NPC) through as many “enemy waves” as possible. User has wide range of in game items that affects the flow and progress of the game and adds to the interactiveness and variability of the game. As with many contemporary games the user has at least a minimal number of game settings to alter in addition to quite standard save progress and load progress options.

1.2 Design goals

Modifiability

The entertainment and gaming applications are in particular very interactive with the user. If there is some flow of events or circumstances that user considers as “illogical” or unacceptable in the context of the game, it should be modified to suffice the user preference

the game have to be modified. Additionally, if the user starts to get bored by the features the game provides the game should be modified to suffice the user. Also, if there other applications of the similar type that started to provide more options the game should once again be modified to suffice the market competition. All in all, modifiability is always the concern for the game applications.

High-performance

The Endless Dungeon, the game developed by our group will be highly interactive with the user. For those reasons the game should have a relatively good performance. That is to say, the user should not miss his to click his items and abilities during the game. As pointed, the game will have a rich input output sequences, and this input data should be processed quickly and output to the user before the next input arrives. Therefore the user experience will highly depend on the performance of the system.

User-friendliness

For the reasons, that our system is the game application it should posses user friendly gameplay, so that users continue playing it and not give up on it. By user-friendliness we imply that buttons be placed in habitually similar places, in game items have descriptive names and visuals, the game should have properly designed goal and reward cycles.

Reuse of components

Our game will provide different range of enemy characters as well as multiple user played characters. The game will have wide range of in game items divided into categories by their purpose and functions. The game will have an alternating stage for each enemy wave. If designed properly many components may be reused multiple times.

Ease of Learning

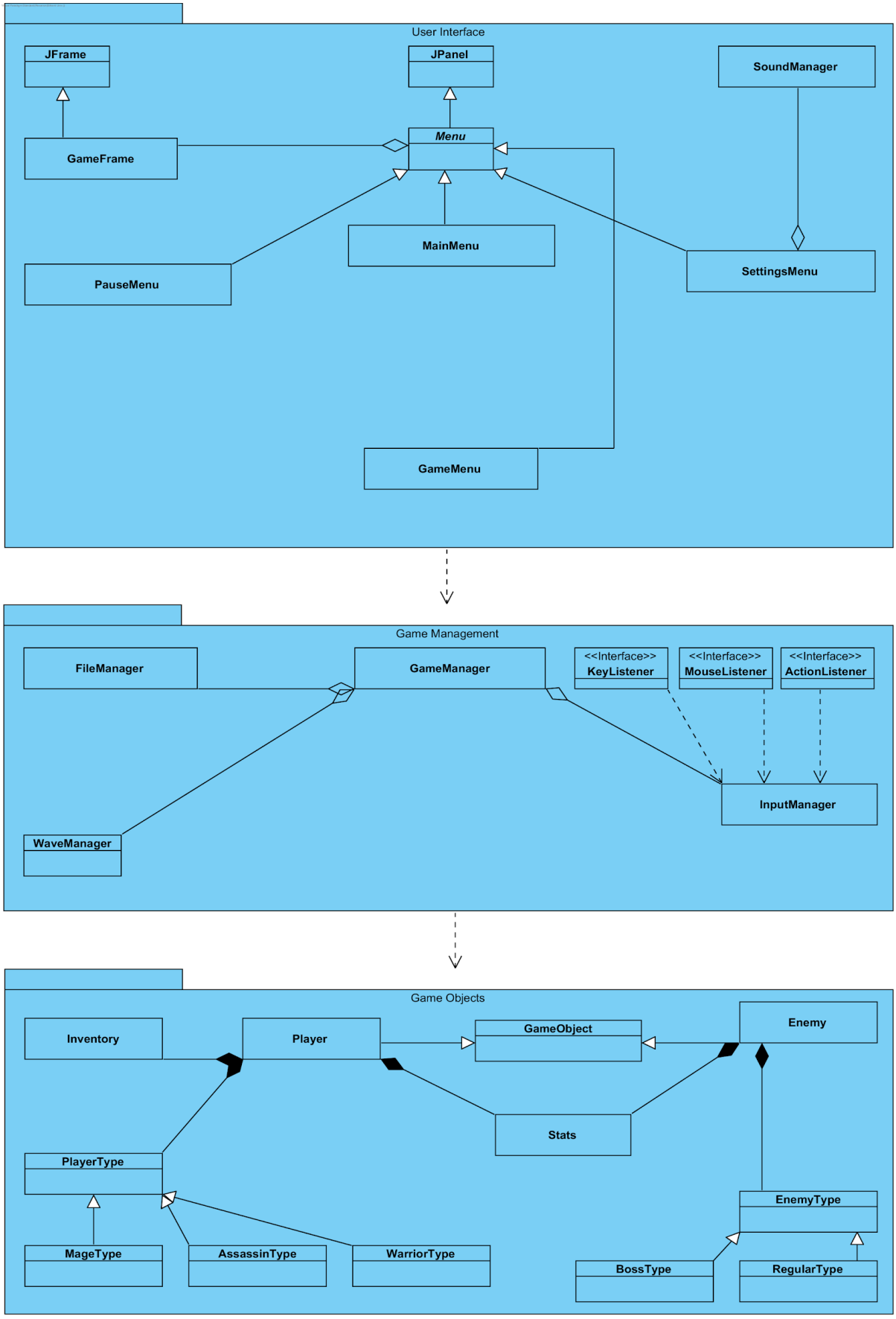
In addition to user friendliness, ease of learning will determine if the application is going to be successful or abandoned once and for all. By ease of learning, we want to make sure that the game is not very complicated in terms of what is requires from the user. We want to design a system in such a way that it includes many simple patterns that can be followed by the user and remembered quickly. This way, since the gameplay will include many simple patterns we will achieve the goal of teaching the game to the user in short amount of time. As a result, will be able to gain more loyal users.

Rapid Development

Endless Dungeon is the game that is to be developed in strict time constraints. The game is to be developed during one academic semester. The design of the system should make sure that these time constraints are feasible to be met.

Cost-effectiveness

Considering this game is developed by 4 undergraduate students the budget allocated for the development of the system is very limited. However,



assuming that we have definite goals that we want to achieve by completing our system we

should design the game in such a way that it will be cost-effective. That is, use free resources than their paid alternatives, etc.

2. Software architecture

2.1 Subsystem Decomposition

During the decomposition of the system, we decided to select the Model View Controller architecture to design our game for the reason that it is the most applicable design architecture that fits our system. Our system has three subsystems according to MVC model which are responsible for different cases:

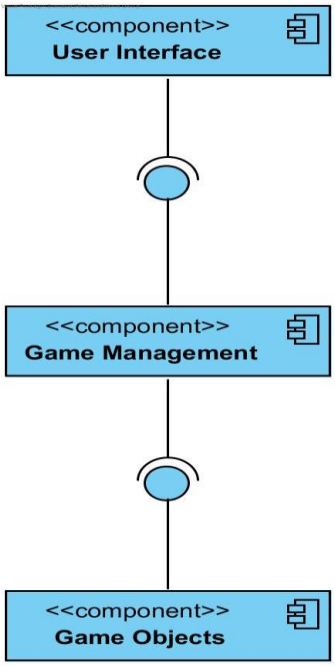


Figure 1 - Basic Subsystem Decomposition

-User Interface Subsystem: This subsystem corresponds to the view part of MVC architecture. It is responsible for providing User Interface and interface components for our system. The interaction between the User Interface Subsystem and Game Management Subsystem is provided only between GameManager component of the Game Management Subsystem.

-Game Management Subsystem: This subsystem corresponds to the controller part of MVC architecture. It has four components which are:

-FileManager component deals with saving and loading files.

-InputManager component deals with user ınput.

-WaveManager component deals with wave progress.

-GameManager component deals with actual game. It has functionalities such as starting game, stopping game and exiting from the game. Additionally, it controls the game objects via GameObject component of Game Objects Subsystem.

-Game Objects Subsystem: This subsystem corresponds to the model part of MVC architecture. According to data coming from GameManager which belongs to the part of controller part of the system, they update their instances and also UserInterface Subsystem will deal with the changes that occured in that layer . It has four components which are:

-GameObject component provides the interaction between GameManager Subsystem and GameObjects Subsystem.

-Player component is an interface to the player object and its types .

-Enemy component is an interface to the EnemyType which allows us to create different types of enemy objects.

-Stats component updates the stats of both player and enemy objects.

Consequently, we believe that our system decomposition will offer high cohesion and low coupling to us.

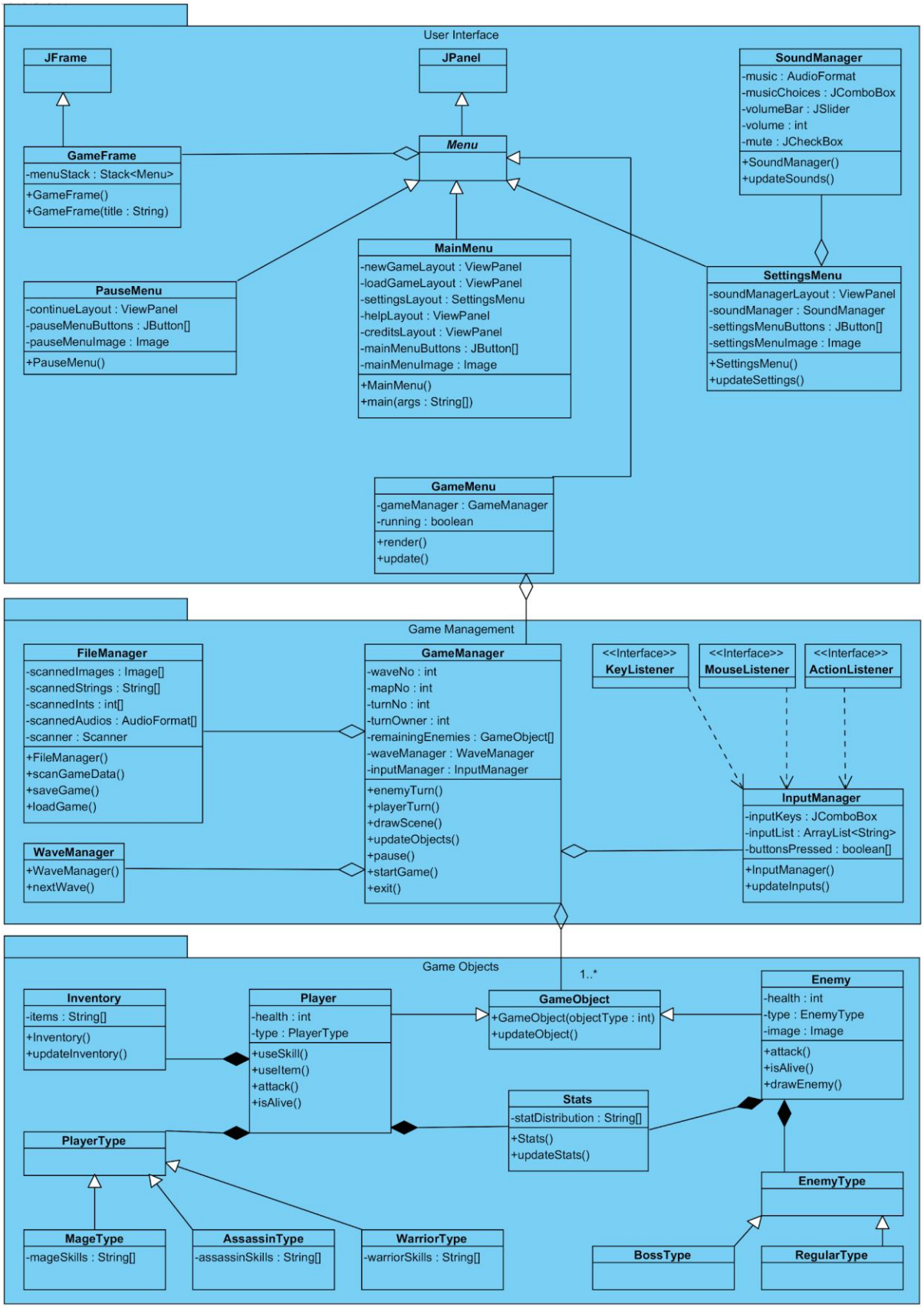


Figure 2 - Detailed Subsystem Decomposition

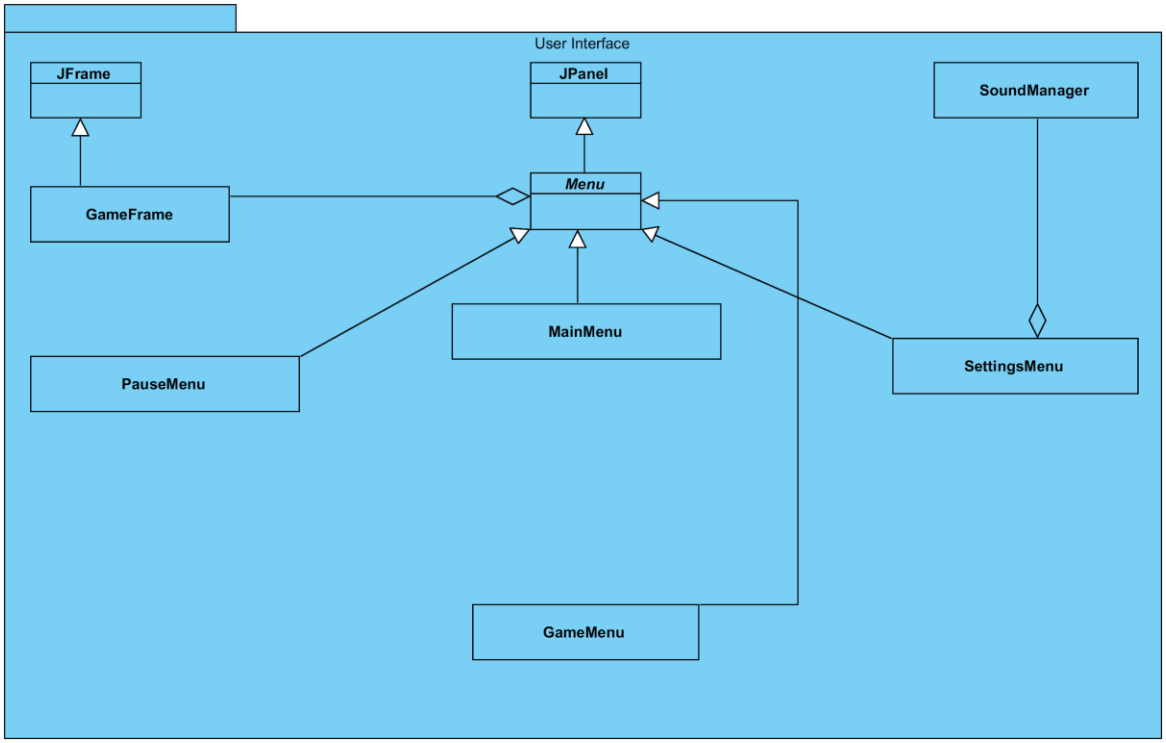


Figure 3 - User Interface Subsystem

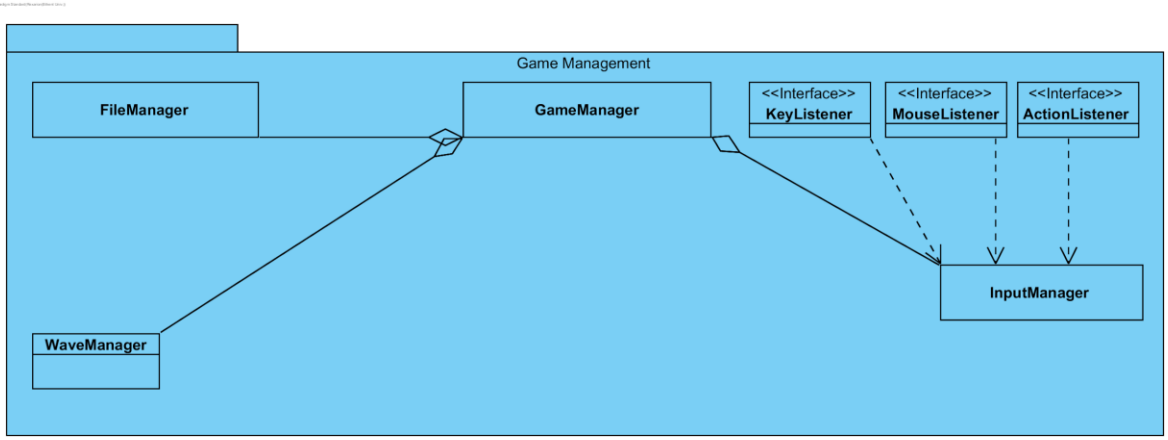


Figure 4 - Game Management Subsystem

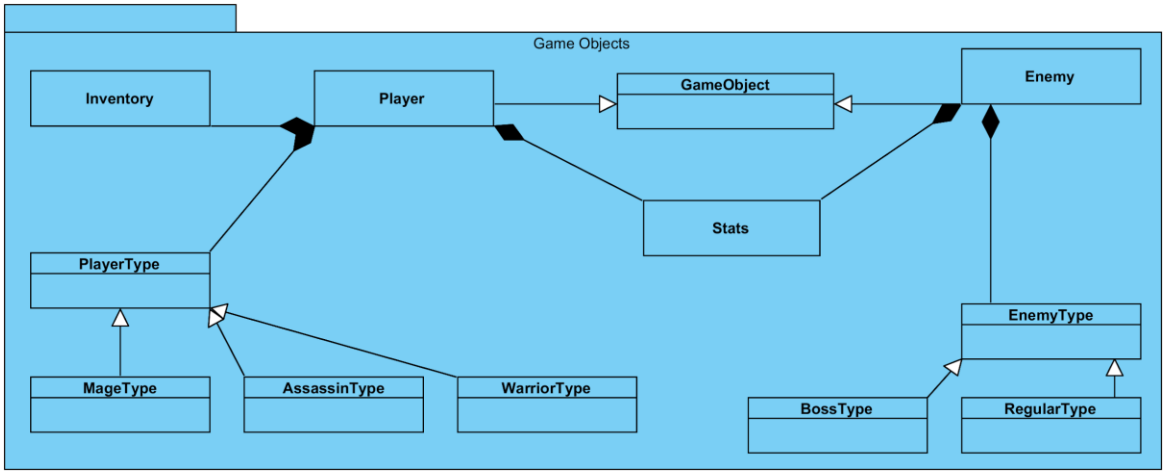


Figure 5 - Game Objects Subsystem

2.2 Hardware/software mapping

Input/Output Performance

The user will interact with the game through mouse and keyboard that will be inputs of the system. The game will output data through the screen and audio system of the computer machine.

Processor allocation

Endless Dungeon is the game developed in Java programming language and therefore requires a computer machine that can run Java Runtime Environment. According to Oracle Corporation, developer of the Java Development Kit and Java Runtime Environment, require at a minimum Pentium 2 226 Mhz processor. However, it is preferable to run our application on a multi core processor because it involves many sequences of CPU burst followed by IO interactions.

Memory allocation

Java Runtime Environment require a minimum 128 MB of memory. However, it is advised to have at least 512 MB of memory to run our application since the game involves a relatively comprehensive amount of image data.

2.3 Persistent data management

Because our game does not require complex queries to retrieve in game entity objects, or concurrent accesses. Additionally, our game does not require multiple applications for the same data set. For the reasons above and because most of the data in our application is voluminous data(e.g. images) and temporary data (e.g. core files), we favor flat file storage. Moreover, because file abstraction is relatively low level it enables to perform a variety of size and speed optimizations.

2.4 Access control and security

The user playing the game for the sake of entertainment is the only actor interacting with the system. The game is designed in way that the user cannot alter game specific variables like hit points of the character for specific wave, or number of starting items. We decided not to impose user authentication once the use wants to load a saved game because it hinders user friendliness. Though this approach make possible a scenario when a user loads and overwrite a game progress of another user. However, this is the convention in game industry nowadays because usually it expected that a user playing the game is the single owner of the machine that is hosting the game or if it is shared with someone then they have mutual respect to not interfere with each others data.

2.5 Boundary conditions

Endless Dungeon does not need any installation progress to start, because it does not have the .exe file. There will be an executable .jar file to give a start to the game. Game will go back to the main menu in two conditions:
-if health of the player becomes zero or,
-player chooses to pause game and presses to the main menu.
If one of these conditions matched, game will stop and return to the main menu.

3. Subsystem services

3.1 User Interface Subsystem services

User Interface Subsystem is responsible for menu operations such as opening option, help and credits pages, and it can also start the game loop by user choice of loading previous game or starting a new game. It also has the SoundManager for sound related operations.

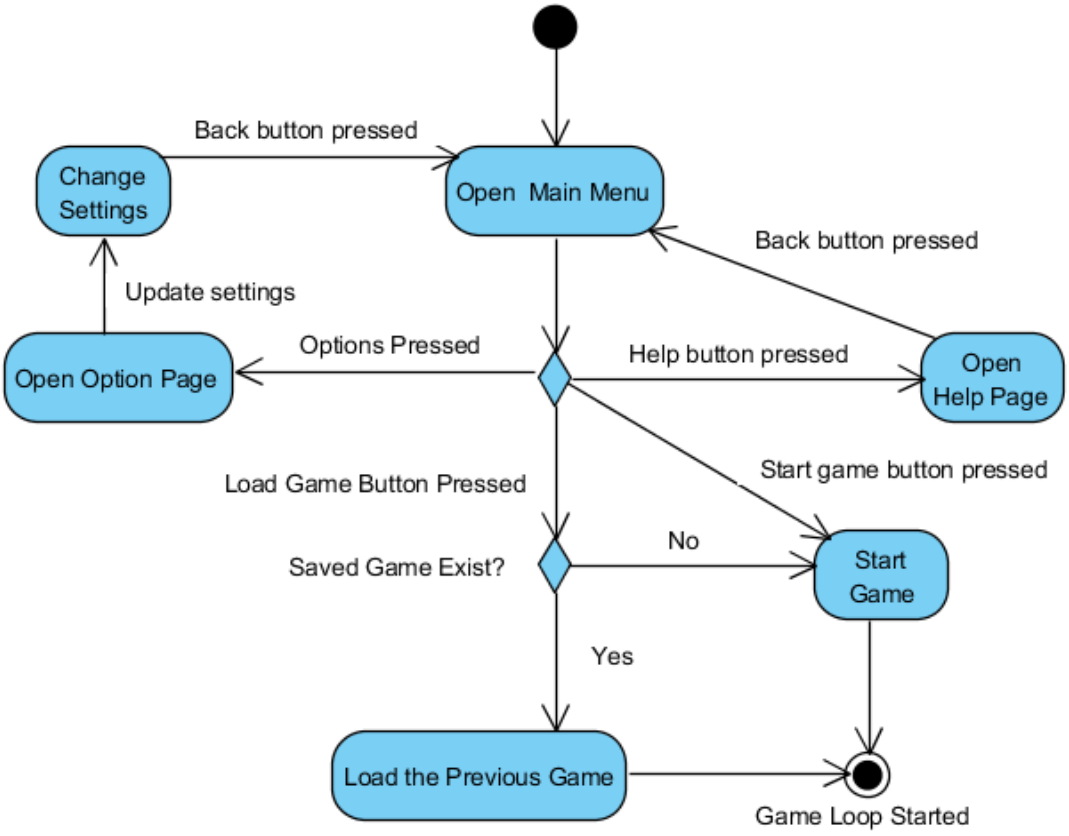


Figure 6 - Activity Diagram that shows fundamental functions of User Interface Subsystem

3.2 Game Management Subsystem services

Game Manager subsystem is responsible for game dynamics. The main functions of this subsystem is that it creates game objects and it handles the necessary operations according to given inputs. Game Loop is controlled under this subsystem (Pausing game loop is implemented here.).

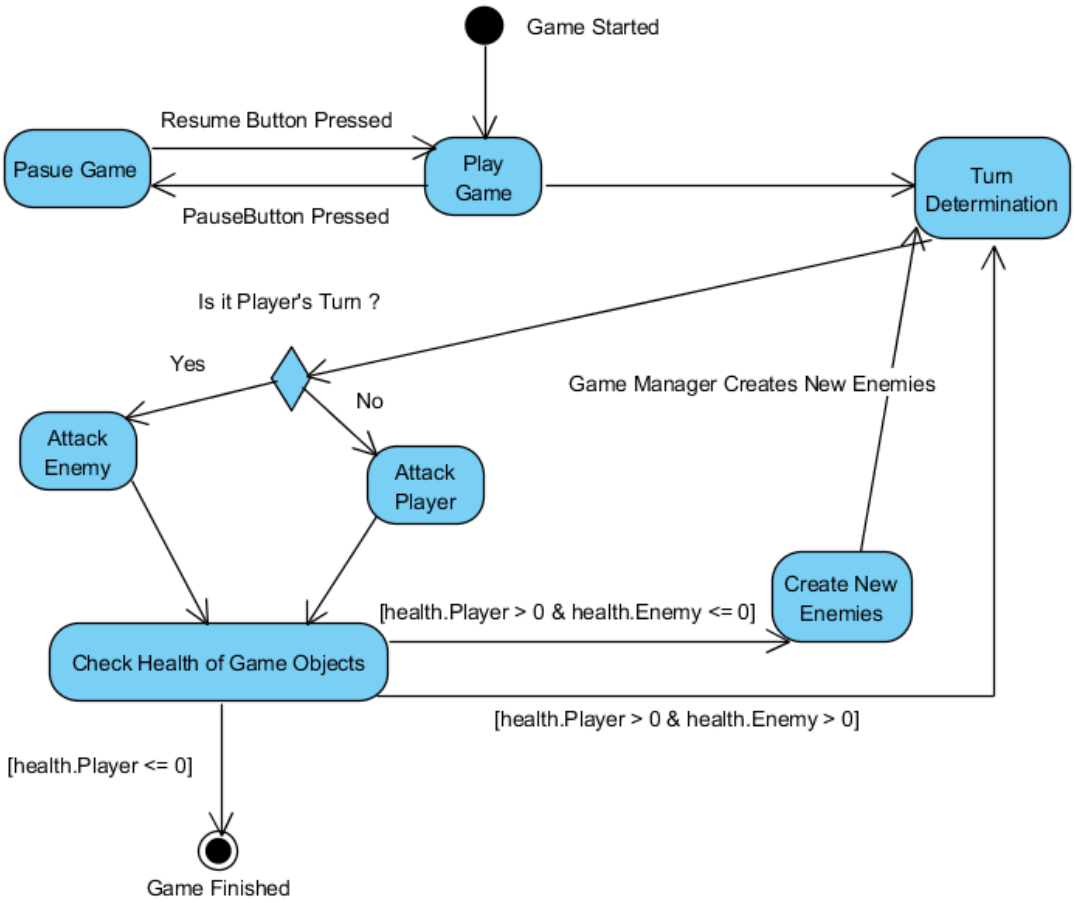


Figure 7 - Activity Diagram that shows fundamental functions of Game Management Subsystem

3.3 Game Objects Subsystem services

Game Objects subsystem is responsible for game object status. Main function of this subsystem is that it gives the necessary information about certain objects that need to be known by the Game Management subsystem to process the game dynamic. These informations are the player's and enemies' health conditions, damage contributions of skills and items.The properties of these objects are updated by the help of Game Management subsystem.

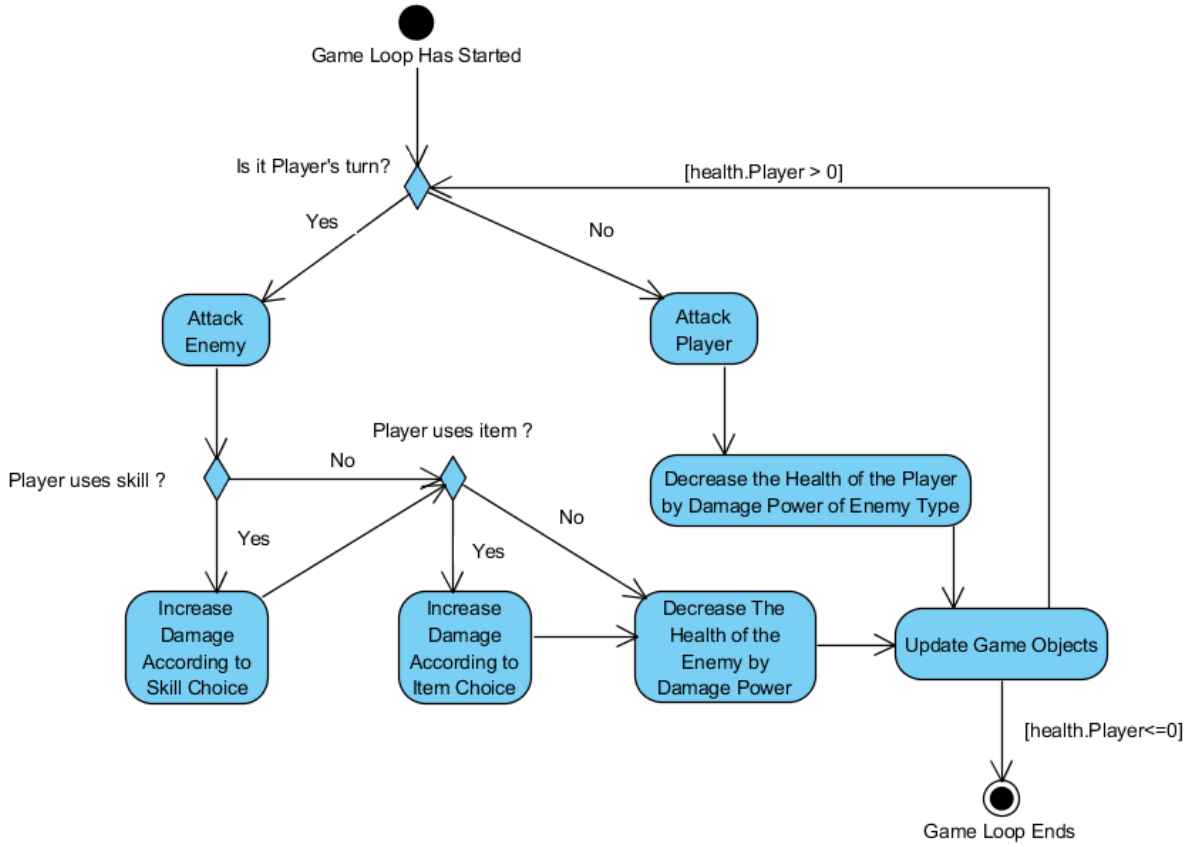


Figure 8 - Activity Diagram that shows fundamental functions of Game Objects Subsystem

4.Low-level design

4.1 Object design trade-offs

Performance vs Space

The common part of RPG genre games is that they are highly interactive with the user. As pointed at design goals, our purpose is to make this game as fast as we can because we do not want to disturb the user by delays that occur due to memory management. Therefore, we decided to sacrifice the memory usage and give more attention to the performance of the game.

Functionality vs User-friendliness

Nowadays, the most important features of applications are ease of use and ease of learning not to bother the user with complex functionalities.For the reason above, it is critical to take user-friendliness into consideration rather than making the game more complex, because we believe that the games with user-friendly style will draw more attention than games with complex functionalities.

Development Time vs Performance

Most programmers agree that C++ provides better performance than Java does. However, it is also known that C++ is not the best choice when it comes to implement a game. Therefore, we decided to use Java for decreasing the time used for dealing with memory leaks but this decision reduces the performance of the game.

4.2 Final object design

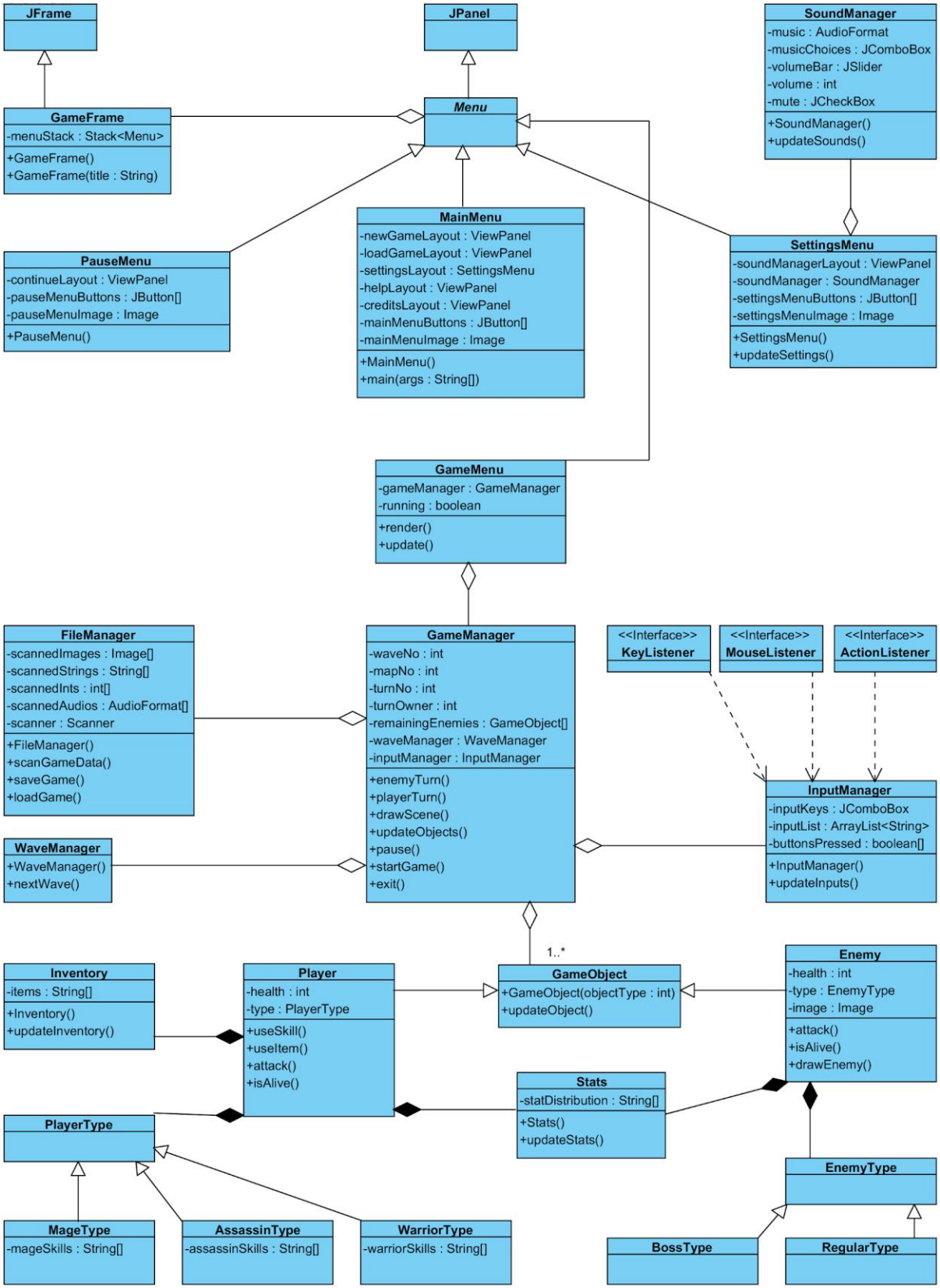


Figure 9 - Final Object Design

4.3 Packages

java.lang

java.lang contains the classes that are fundamental to the design of the Java programming language. It contains the Object class, String class and many more that will be used throughout the development of the Endless Dungeon game application.

java.util

java.util contains many useful classes that will aid the development such as Arrays, ArrayList, Date, Random, Scanner, Timer

java.io

java.io is a an important class that handle input output operations in Java. It contains classes for reading data from input stream, for writing data to output stream. Classes that manipulate data on the local file system. Also, classes for object serialization.

javax.swing

java.swing is “model-view-controller” graphical user interface (GUI) library for desktop Java applications. The framework provides a layer of abstraction between the code structure and GUI

4.4 Class Interfaces

4.4.1 User Interface Subsystem

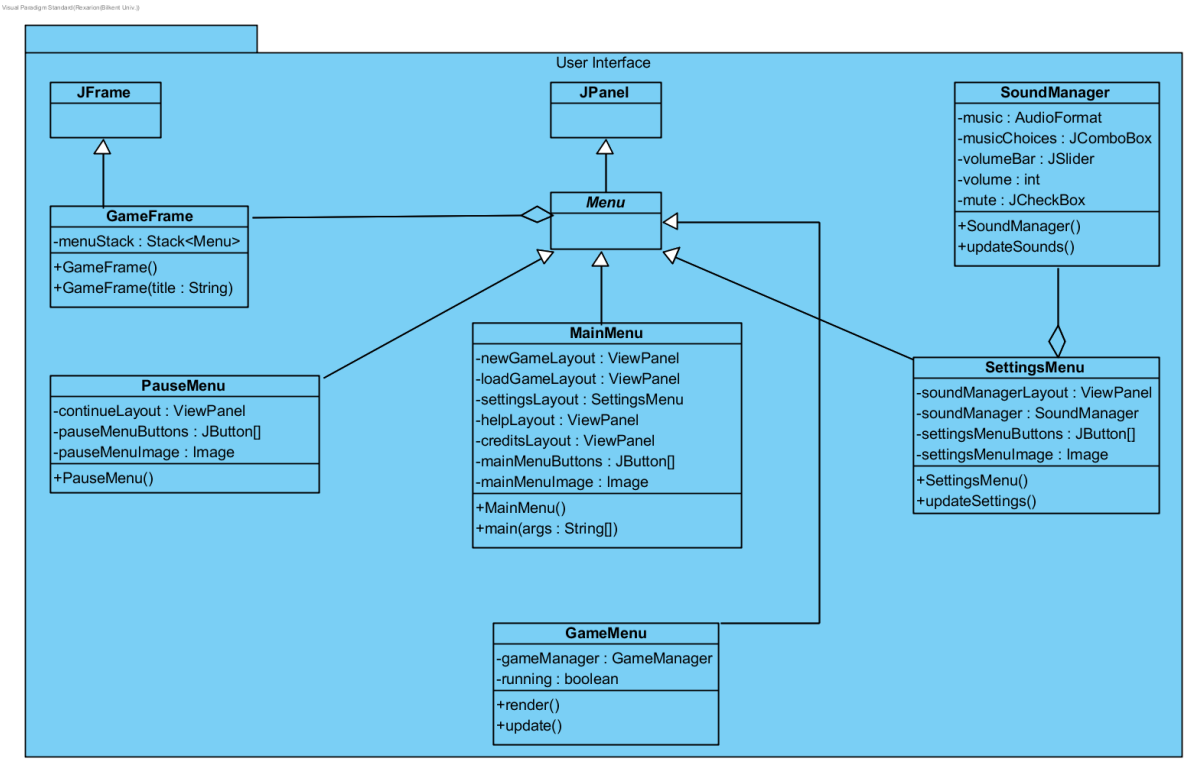
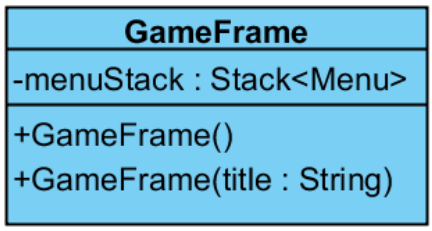


Figure 10 - Detailed User Interface Subsystem

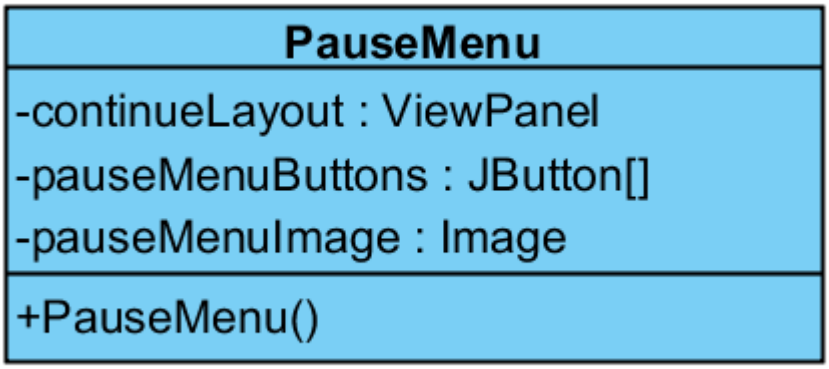
GameFrame Class



“GameFrame” class extends a JFrame and it has a stack of Menus as JPanels to be shown in accordance to the player.

PauseMenu Class

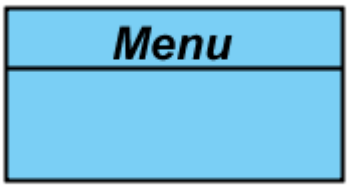
Visual Paradigm Standard(Rexarion(Bilkent Univ.))



“PauseMenu” extends a “Menu class”. It includes a ViewPanel layout array of buttons for the pause menu and image that accompanies the pause menu properties. Instantiated when a user pauses the game.

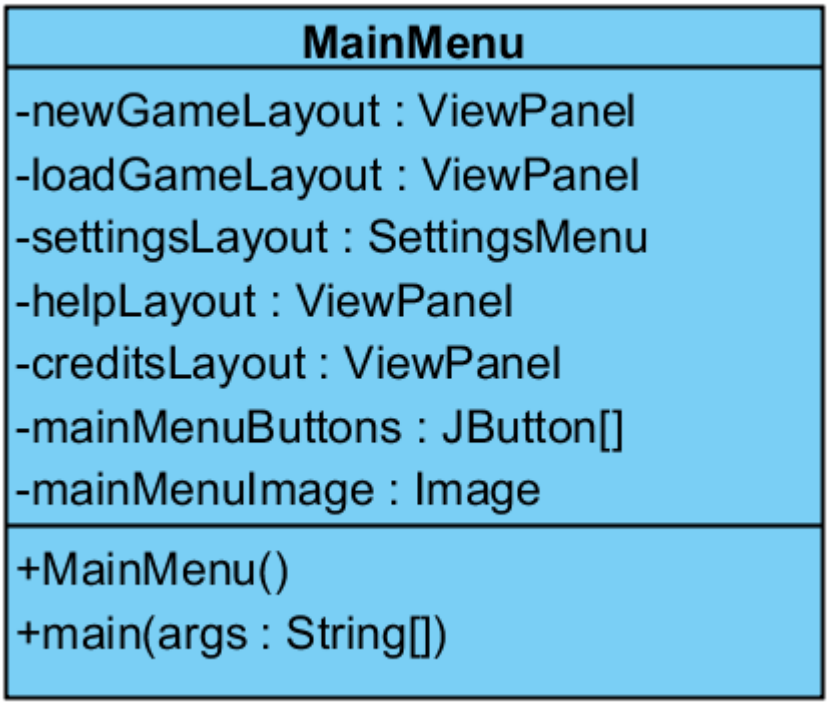
Menu Class

Visual Paradigm Standard(Rexarion(Bilkent Univ.))



“PauseMenu”, “SettingsMenu”, “MainMenu”, “SettingsMenu”, “GameMenu” classes all extend “Menu” class. The “MainMenu” class itself extends a JPanel and has an instance of “GameFrame” class.

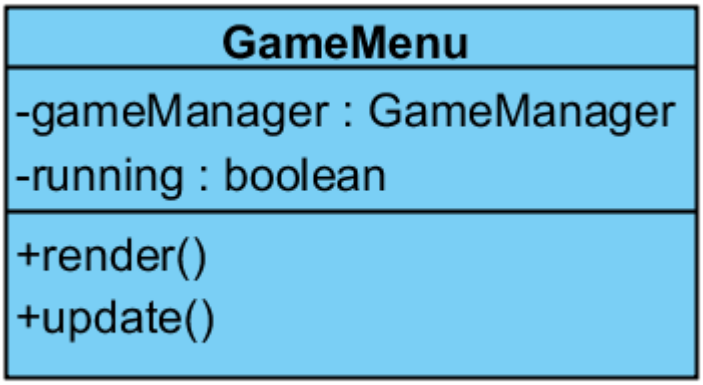
MainMenu Class



“MainMenu” extends a “Menu” class and as the name implies is the central menu in the game.
-newGameLayout: ViewPanel for a new game.
-loadGameLayout: ViewPanel for a game that is to be loaded.
-helpLayout: ViewPanel for the help user may require about the game.
-creditsLayout: the information about the contributors to the game development.
-settingsLayout: settings of the game.
-mainMenuButtons: array of buttons to control the main menu.
-mainMenuImage: the background image of the main menu.
+MainMenu(): Constructor.
+main: main method.

GameMenu Class

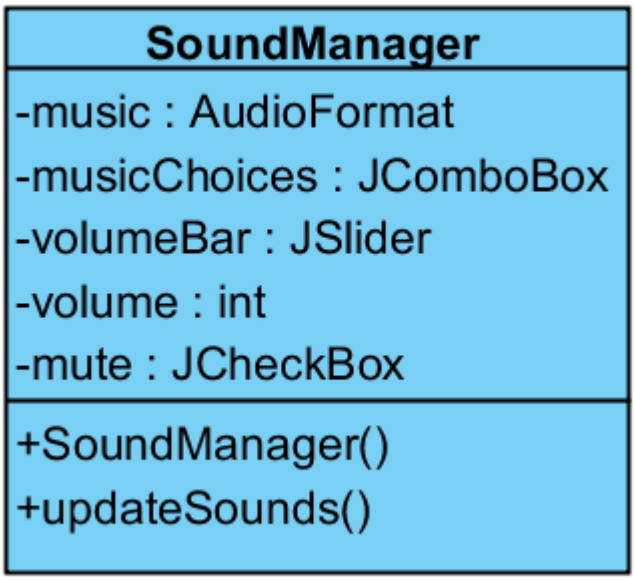
Visual Paradigm Standard(Rexarion(Bilkent Univ.))



“GameMenu” extends a “Menu” class”. It has an instance of GameManager class and boolean property that signifies whether Game Manu is running or not.
+update(): update the GameMenu.
+render(): render the changes.

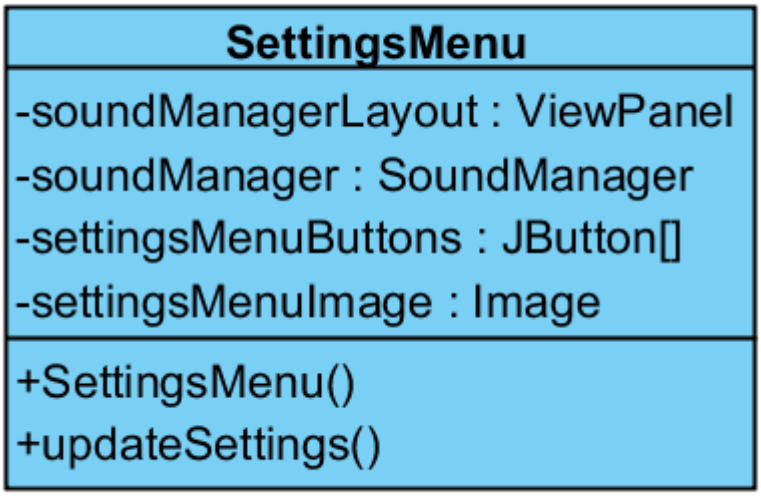
SoundManager

Visual Paradigm Standard(Rexarion(Bilkent Univ.))



SoundManager is responsible for in game sounds.
-music: attribute for the in game music
-musicChoices: attribute for the user choices of the music
-volumebar: attribute for the Jslider that controls the volume of the in game music
-volume: integer value that marks the sound intensity of the in game music.
-mute: a checkbox to mute or unmute the in game music.
+SoundManager: Constructor of the “SoundManager” class
+updateSounds(): update the sound changes applied.

SettingsMenu Class



“SettingMenu” class is responsible for the Settings of the game:

-soundManagerLayout: View that is presented in Settings Manager.
-soundManager: an instance of sound manager.
-settingsMenuButtons: array of buttons to control the Settings Menu.
+SettingsMenu(): Constructor.
+updateSettings(): update user applied changes

4.4.2 Game Management Subsystem

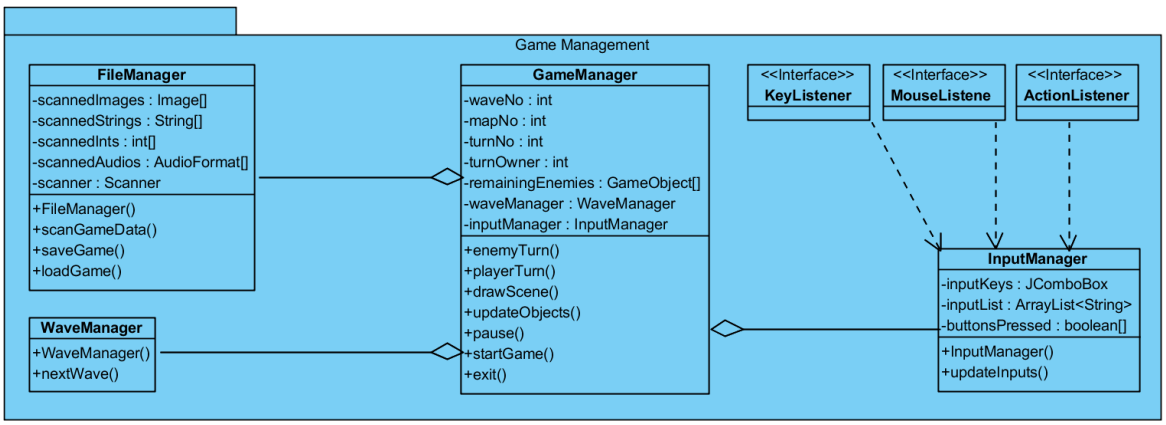
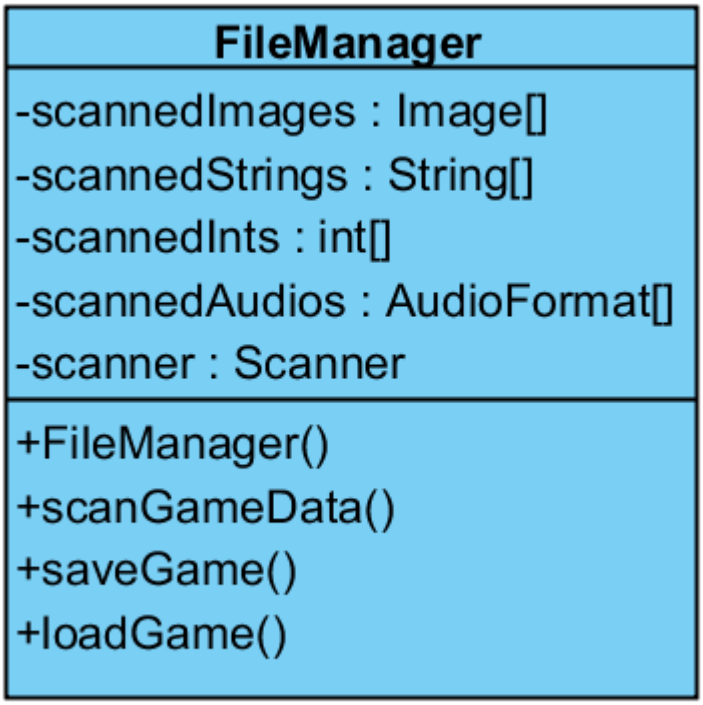


Figure 11 - Detailed Game Management Subsystem

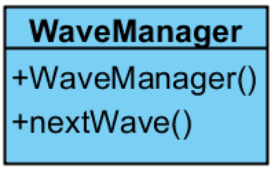
FileManager Class



FileManager class will be used to manage the file system for the game. Files will be used to save the games to be continued after the game has been closed by the player.

- Image[] scannedImages: Images that were scanned in the last game saved to be loaded later on
- String[] scannedStrings: Strings that were scanned in the last gave saved to be loaded later on
- Int[] scannedInts: Integers that were scanned in the last gave saved to be loaded later on
- Scanner scanner: Scanner object to be used to scan the current game object in order to write them to a file
+ FileManager(): Constructor of the FileManager class
+ scanGameData(): Scans the data of the current game to be saved in order to make it ready to be saved
+ saveGame(): Saves the game with the data that was scanned last
+ loadGame(): Loads the chosen game file to be continued from where the player left off

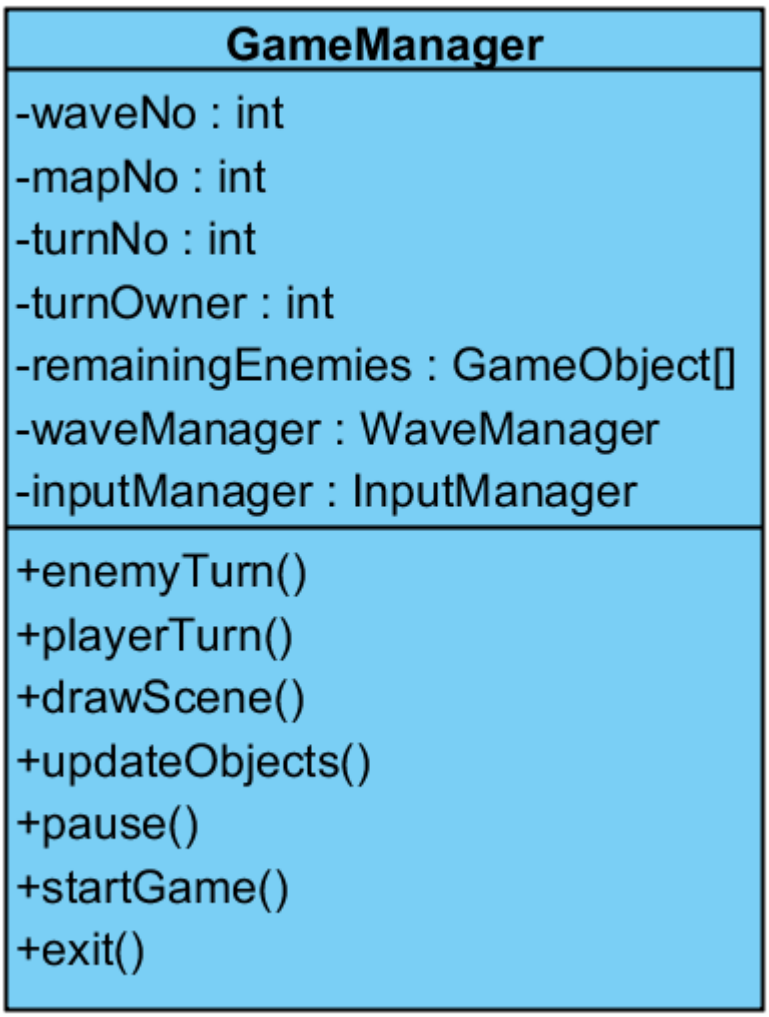
WaveManager Class



This class is design to manage Wave operation to help the GameManager class.

+ WaveManager() method gives the information to the “GamaManager” class about the current wave.
+ nextWave() method gives the information to the “GamaManager” class about the next wave if the current wave passed.

GameManager Class

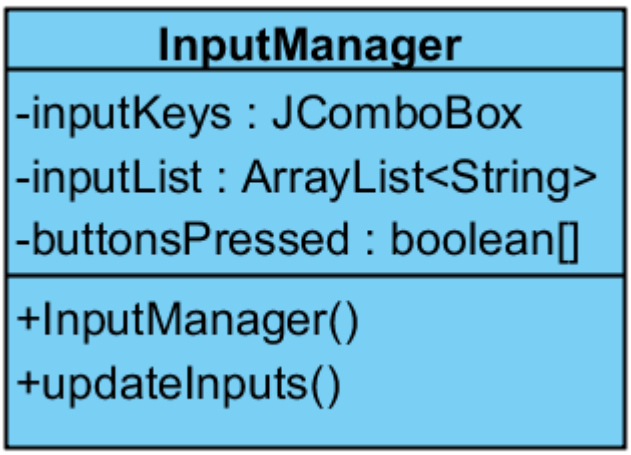


This class is the Facade class of the Game Management subsystem. It operates the operations given by the User Interface Subsystem.

- int waveNo: This attribute is used for determining the Wave number.

- int mapNo: This attribute is used for determining the background map.
- int turnNo: This attribute is used for determining the count of the turns that played.
- int TurnOwner: This attribute is used for determining the whose turn to attack (enemy's or player's turn).
- gameObject[ ] remainingEnemies: This attribute is used for determining the alive enemies by the help of GameObject type of array.
- WaveManager waveManager
- InputManager inputManager: This attribute is a inputManager object, by which GameManager class associates with proper methods of InputManager class.
+ enemyTurn(): This method gives the information about it is enemy's turn to attack, by this method GamaManager can perform the necessary task.
+ PlayerTurn(): This method gives the information about it is player's turn to attack, by this method GamaManager can perform the necessary task.
+ drawScene(): This method gives necessary information to the User Interface Subsystem to update the latest game screen.
+ updateObjects(): This method gives necessary information to the Game Objects Subsystem to update the latest game object entities (such as health of the player, health of the enemy, etc.).
+ pause(): If this method called from GameManager will break the game loop.
+ startGame(): By the help of this method, GameManager can start the game loop.
+ exit(): By the help of this method, GameManager can stop the game loop and exit from it.

InputManager Class



InputManager class is designed for to understand the inputs that are coming from the User Interface Subsystem and do the necessary operations according to these inputs.

- JcomboBox inputKeys: Drop down menu to show the available input keys to be used by the player
- ArrayList<String> inputList: This array holds the input sequences to perform them in the given order.

- boolean[ ] buttonPressed: This array holds the all buttons that are created. If they pressed, this array will be updated so GameManager can perform the necessary operations.

+ InputManager(): This method gives the information to the “GamaManager” class about the what to the with the current Input.
+ updateInputs(): This method gives the information to the “GamaManager” class about the current Input that is pressed.

4.4.3 Game Objects Subsystem

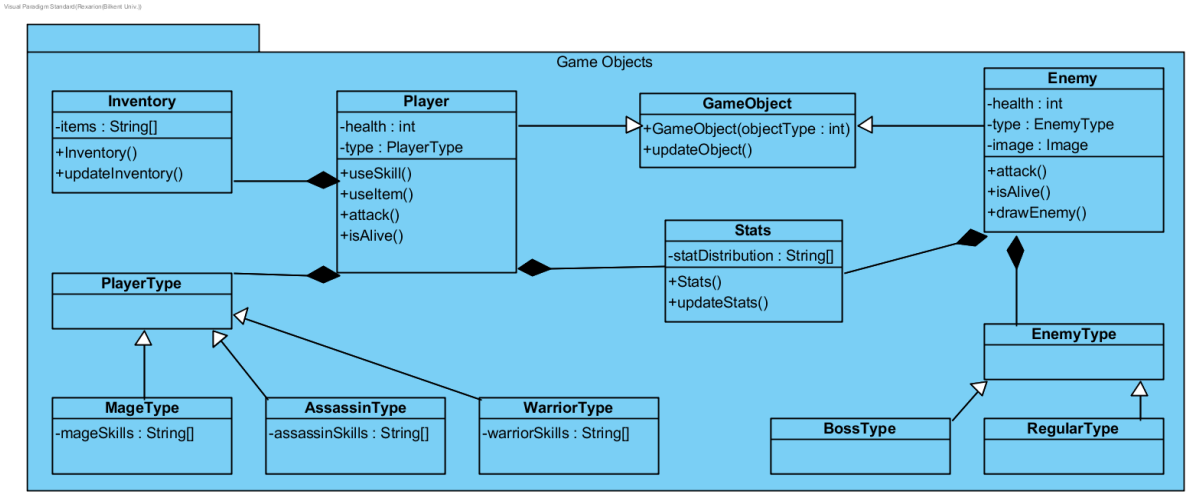
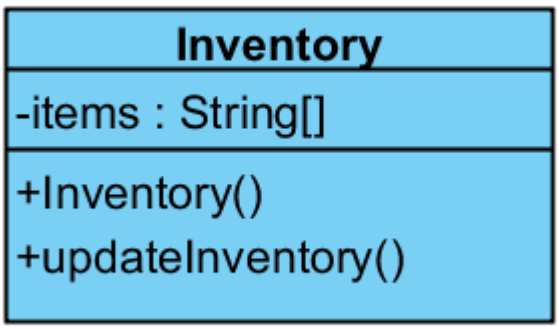


Figure 12 - Detailed Game Objects Subsystem

Inventory Class



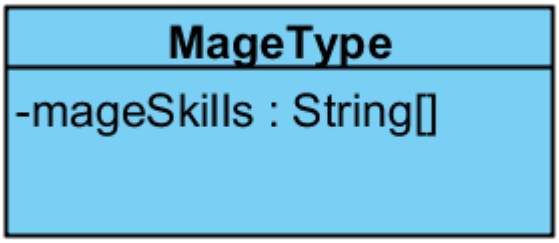
Inventory class is for managing the Player’s inventory of items. Items that are dropped by enemies will gather in the Player’s inventory and they will be ready to used by the Player.

- String[] items: Manages the items in the inventory in a String array
+ Inventory(): Constructor of the Inventory class
+ updateInventory(): Updates the inventory of the Player according to the changes that occurred during the last turn

PlayerType Class

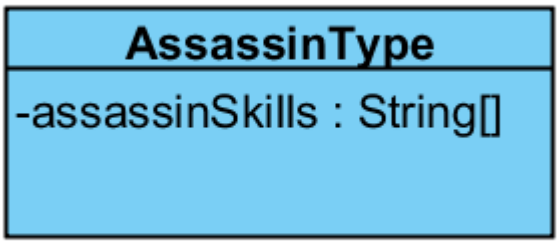
PlayerType class is used to denote the type of the Player. Player's stats and skills will be determined according to the type of that particular Player.

MageType Class



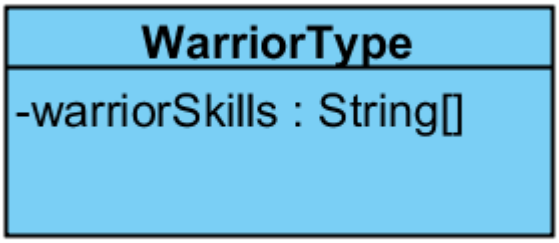
MageType class is the Mage type (commonly known as the Mage "Class" in RPGs) that will be used by Players that are willing to play the game depending more on their skills rather than their basic attacks and items. Players with the Mage type are going to be more fragile compared to the other class but their skills will be equally stronger.

AssassinType Class



AssassinType class is the Assassin type (commonly known as the Assassin "Class" in RPGs) that will be used by Players that are willing to play the game depending more on their items and their luck to actually obtain those items. Their basic attacks and skills will be the 2nd strongest out of the 3 classes as well as their health.

WarriorType Class



WarriorType class is the Warrior type (commonly known as the Warrior “Class” in RPGs) that will be used by Players that are willing to play the game depending more on their basic attacks and high health points. These players will go in battles head on and they will not be able to depend on their skills as much as the other types can.

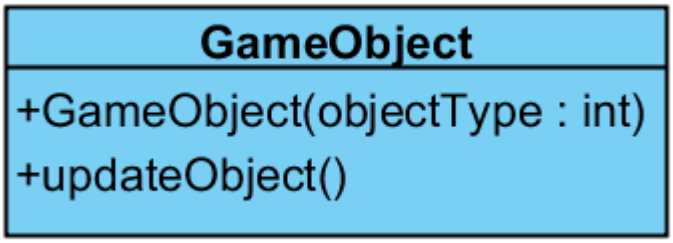
Player Class

Player class is the class used to denote the Player of the game. The Player will not be visible on the game screen as our game uses a retro First-Person perspective.

- Int health: Current health points of the Player
- PlayerType type: The type of the Player
+ useSkill(): Uses the skill of the chosen PlayerType
+ useItem(): Uses an item from the inventory
+ attack(): Attacks the chosen enemy
+ isAlive(): Checks if the Player is still alive

GameObject Class

Visual Paradigm Standard(Rexarion(Bilkent Univ.))

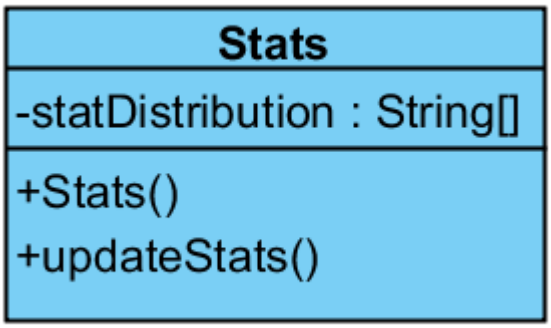


GameObject will represent the objects of the game such as the Player and the Enemies. It will be the Façade class of the Game Objects Subsystem, hence the name.

+ GameObject(int objectType): Constructor of the GameObject class that takes an integer as a parameter
+ updateObject(): Updates the GameObject according to the changes that took place during the last turn

Stats Class

Visual Paradigm Standard(Rexarion(Bilkent Univ.))

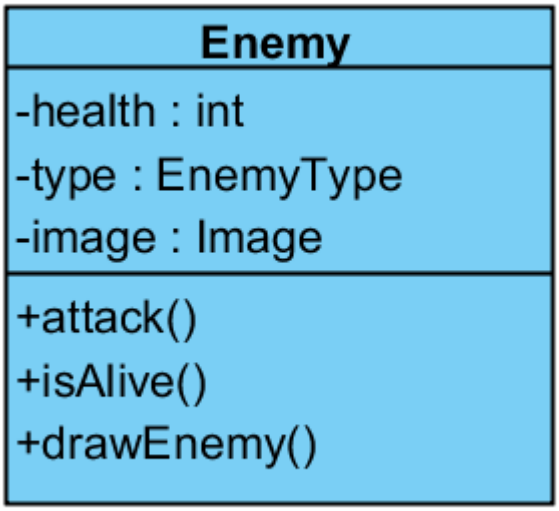


Stats class will be used to denote the statistics of the GameObjects. It will hold several RPG related stats such as Strength, Agility, Intelligence etc..

- String[] statDistribution: The distribution of the statistical components
+ Stats(): Constructor of the Stats class
+ updateStats(): Updates the Stats of the GameObject according to the changes that took place during the last turn (added in case we can add the levelling system and stat-enhancing items)

Enemy Class

Visual Paradigm Standard(Rexarion(Bilkent Univ.))

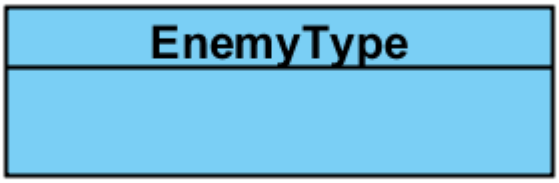


Enemy class will be used to show the Enemies that the Player will have to fight in order to progress.

- int health: Current health points of the Enemy
- EnemyType type: The type of the Enemy
- Image image: The image to be used to represent the Enemy on the screen
+ attack(): Attacks the player to deal damage
+ isAlive(): Checks if the Enemy is still alive
+ drawEnemy(): Draws the Enemy on the screen

EnemyType Class

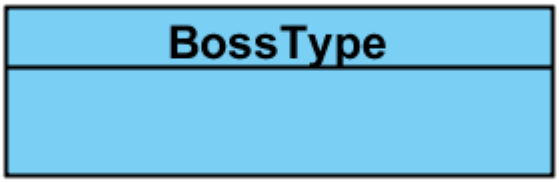
Visual Paradigm Standard(Rexarion(Bilkent Univ.))



EnemyType class is used to denote the type of an Enemy. The stats of the Enemy will be determined according to its type.

BossType Class

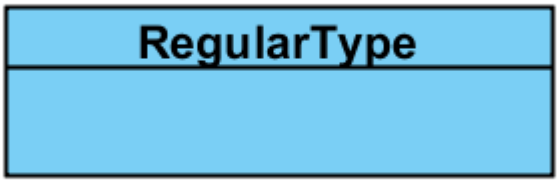
Visual Paradigm Standard(Rexarion(Bilkent Univ.))



BossType class is used to denote the Boss Enemy that will be positioned as the middle enemy and it will be stronger than the Enemies on both sides of the Boss. The Players will have to defeat all enemies, in waves of 3, to progress in the game and the Bosses will prove to be the hardest ones to defeat.

RegularType Class

Visual Paradigm Standard(Rexarion(Bilkent Univ.))



RegularType class is used to denote the Regular type Enemies that will be positioned on both sides of the Boss Enemy and they will be weaker than the Boss. They will act as the lackeys of the Boss to try to defeat the Player.

5. Glossary & references

[1]https://docs.oracle.com/javase/7/docs/webnotes/install/windows/windows-system-requirements.html

[2]https://docs.oracle.com/javase/7/docs/api/