BILKENT UNIVERSITY

ENGINEERING FACULTY

DEPARTMENT OF COMPUTER ENGINEERING

CS 319

Object Oriented Software Engineering Project

e-Bola

Design Report

Group 2B

Can AVCI

Deniz SİPAHİOĞLU

Ergün Batuhan KAYNAK

Esra Nur AYAZ

Course Instructor: Bora Güngören

October 21, 2017

1 Introduction

1.1 Overview

In this section, we represent the purpose of the system, which is mainly entertaining the player. We listed the design goals in this part, in order to achieve our purpose. Our design goals are providing portability, maintaining an easy use of interface, learning the game easily, smooth gameplay and extendable design. We had to decide between some trade-offs during the project which will be explained in part 1.4.

1.2 Purpose of the System

e-Bola is a basic 2-D game which encourages the player to develop a game strategy in order to fight the enemies and the boss. The player will have multiple choices to choose from while choosing the items and attributes. There is no "correct item" or attribute to win the game, so everything can work. This is what makes the game enjoyable and skill-dependent. The game is very basic compared to today's 3-D games, which are made with thousands of dollars budget and an experienced team of developers. With e-Bola, our aim is to ensure that the player enjoys the game while improving his reflexes and decision making.

1.3 Design Goals

1.3.1 End User Criteria

Easy to learn:
Since the game takes very basic inputs from the user (4 to 5 keystrokes), the player will be able to get familiar with the game very quickly. The concepts other than the controls, will be explained in the Tutorial level, which can be accessed from the initial menu. The tutorial will consist of several images which explains the basics of the game. We will try to keep the tutorial as short and as informative as possible. We aim to keep it under a minute so that the player can start the game sooner. The tutorial won't cover everything, so the player will have the excitement of exploring other features of the game by himself.

Easy to use:
When the player initializes the game, the menu navigation and the buttons that he is going to use will be crystal clear. The control buttons will be preassigned and will be the same with the other games. For example, ESC for pausing and viewing the menu, arrow keys or 'WASD' for movement, just like the other games that the player might've played.

1.3.2 Maintenance Criteria

Extendibility:
We are aware of what keeps the game going is its extendibility. No matter how good the game is, the player will get bored after some time if the developers don't create new content for the game. Since our game is level based, it is very important to make it expendable. We designed the classes in a way that it is very easy to add new levels, monsters, attributes and even new territories to the game. It is only limited to the creativity of the developer. Also, we will add the developer's contact information to the credits page, so that the user can give suggestions of what he wants to see in future versions of the game.

Portability:
There are lots of operating systems to choose from, but in terms of portability and ease of use, we choose to develop our game in Java. This way we will be able to reach more players, since Java Virtual Machine provides platform independency and our game will be accessible from every computer that runs java.

Reliability:
Every possible scenario will be tested out level by level, so that the user won't witness any crashes or face unexpected events in the game, since these will be handled accordingly.

Modifiability:
Modifiability is very high at this game. Even without the source code, the user can change the text file related to the levels, and create custom levels. However the game may not start if it's done incorrectly.

1.3.3 Performance Criteria

Response Time:
This game requires immediate action from the system when there's any kind of input from the user. That's why the response time will be one of our main focuses. Animations will also respond to the players' input to create smooth gameplay environment.

1.4 Trade-offs

1.4.1 Efficiency and Reusability

Instead of adding more features such as more weapons, attributes and monsters, we kept things as simple as possible to make the game easier to play

and learn. However, the game has so much potential to add functionality, so we can increase the functionality of the game later on if it's necessary.

1.4.2 Memory Use and Efficiency

We store most of the data in text files. Using text files increases our extendibility and modifiability. However, if the user knows how to make the system and hidden files visible, it can easily be used to cheat. Such user can reach any data he has on the game and modify it, such as level, attributes, attack and defense points. Also, the user can damage the file by mistake, and this may cause problems in the initialization of the game.

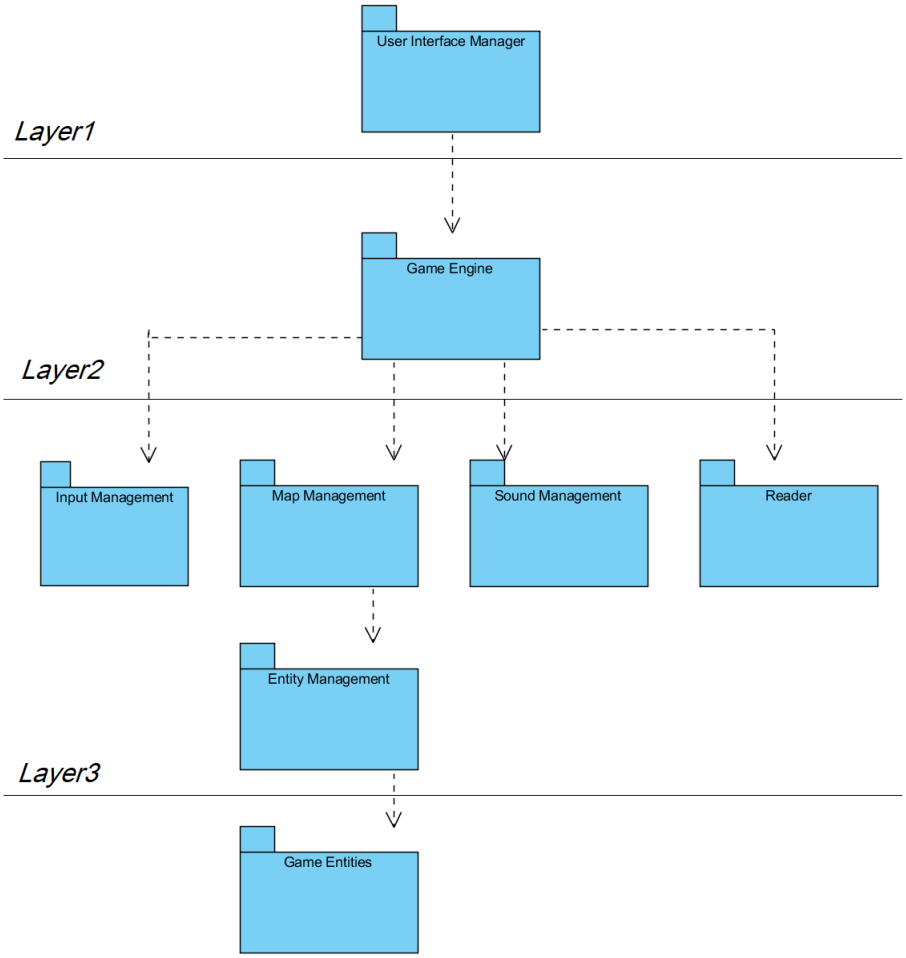
2 Software Architecture

2.1 Overview

e-Bola is designed to be programmer-friendly. For this purpose, the software system is decomposed into subsystems. Decomposition of the project will slightly increase the workload but will also improve the readability, reusability, and ease of maintenance of the project. In addition, decomposition is helpful to implement Model View Controller (MVC) architectural pattern.

2.2 Subsystem Decomposition

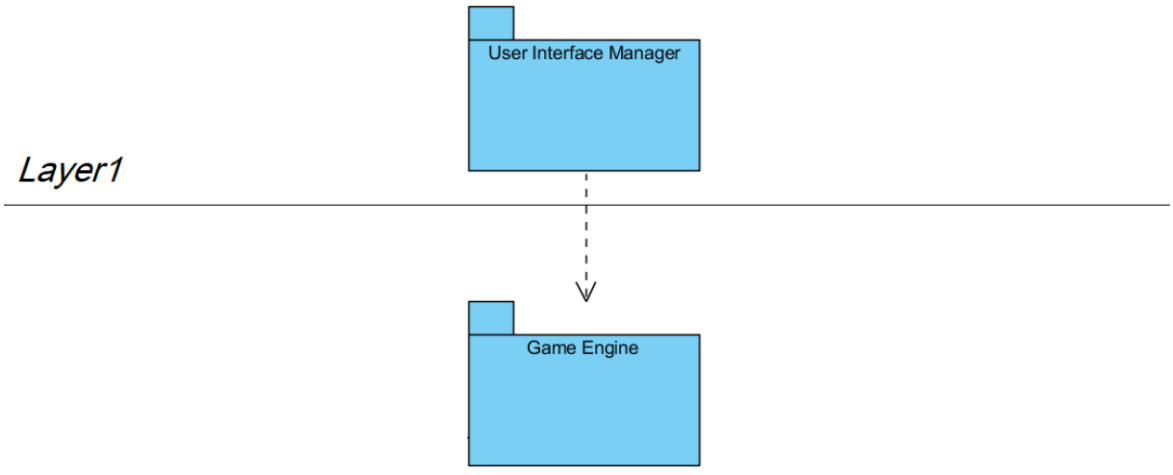
The three-tier architecture is an important design. It suggests to have three tiers, one for presentation, logic, and data. The three-tier principle is also suitable for the design of e-Bola project. Presentation tier's only content is "User Interface Manager". Logic tier can be investigated under two parts: one part for "Game Engine" and one part for management units whose controller is "Game Engine". "Game Entities" is in charge of all the data as the only member of data dies. Design of e-bola is divided into three layers as shown:



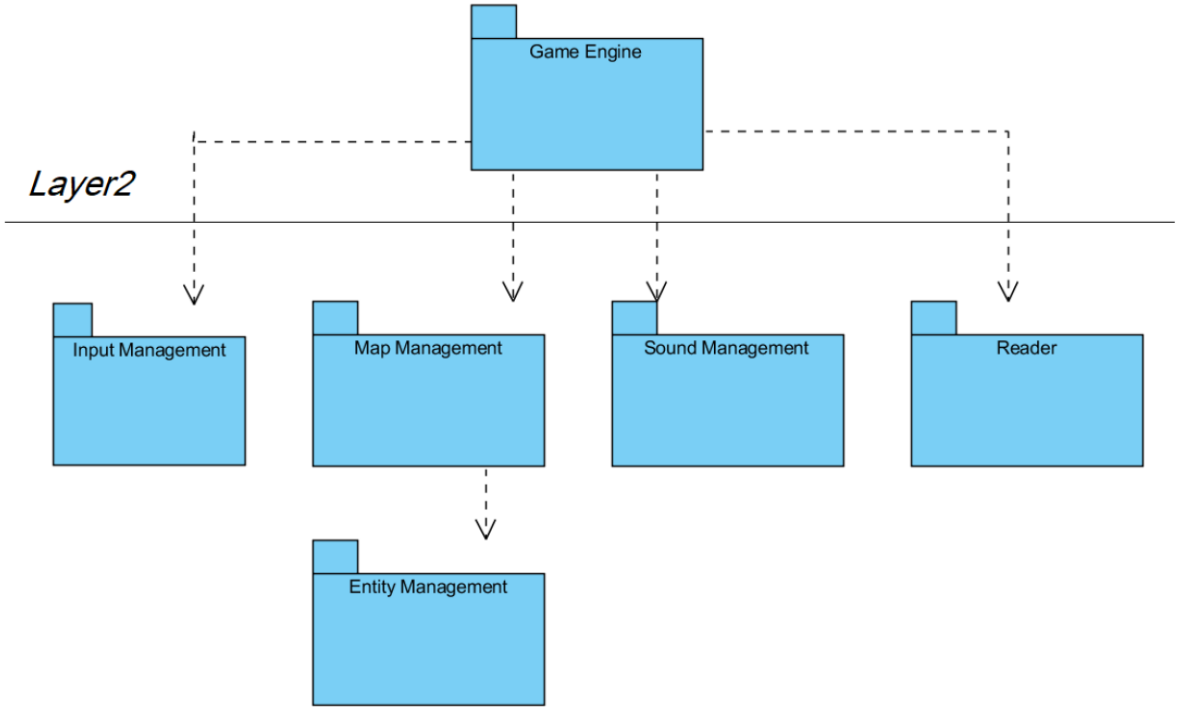
2.3 Architectural Styles

2.3.1 Layers

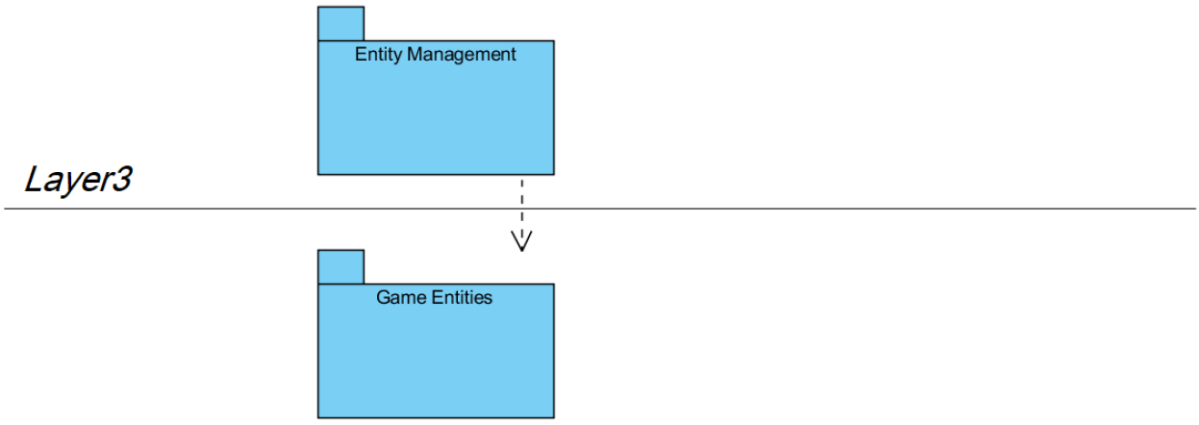
Layer1: The first level of e-Bola is the User Interface Manager and Game Engine. The User Interface Manager is responsible of how the menu is shown and how the user interacts with it. The UI Manager works synchronously with the Game Engine, and updates itself accordingly.



Layer2: The second level of e-Bola is the interaction between the Game Engine and Input Management, Map Management, Sound Management and Reader. This level is responsible of the game logic, and how all entities interact with eachother.



Layer3: The third and final level of e-Bola is responsible of the Entities themselves and how they report to the Entity Manager.



2.3.2 Model View Controller

Data tier constitutes the model for our system. *User Interface Management* package represents the view for the system. Logic tier constitutes the controller of the system.

2.4 Hardware/Software Mapping

On hardware side, e-Bola requires a keyboard and a mouse for its input needs. For output, a speaker (optional) and a screen is necessary.

On software side, Java Runtime Environment (JRE) is required to run since the game will be developed using Java programming language. Java uses Java Virtual Machine (JVM). JVM can run Java code on any machine that it is installed. This language choice lets the game be independent from platforms. Therefore, software and hardware needs will be minimal. Also, no network connection is required.

2.5 Persistent Data Management

e-Bola game will store maps, audio files and entity images in the user's disk. It does not require a complex system to store the necessary data. The game instance will be saved in a text file.

2.6 Access Control and Security

e-Bola does not use any network-related input or output (doesn't have login issues) and does not let multiple users play the game at the same time. Such restrictions makes e-Bola not accessible for outsiders.

2.7 Boundary Conditions

2.7.1 Initialization

The necessary data will be provided by the game, so there is no need for the user to do anything at the initialization step. e-Bola does not need to be installed and it does not need to be registered.

2.7.2 Termination

If Celly dies, the game ends automatically and starts from the last checkpoint. If the user wants to end the game temporarily, he can save the current state of the game and continue playing whenever he wants. If the user wants to start a new game, the user can close the game and restart it, and select the new game option.

3 Subsystem Services

3.1 Overview

The *User interface* package is responsible for the system's interaction with the user. It consists of "Menu Panel", which implements "Menu Button Listener" interface and has "Menu Button"s, and "Game Panel".

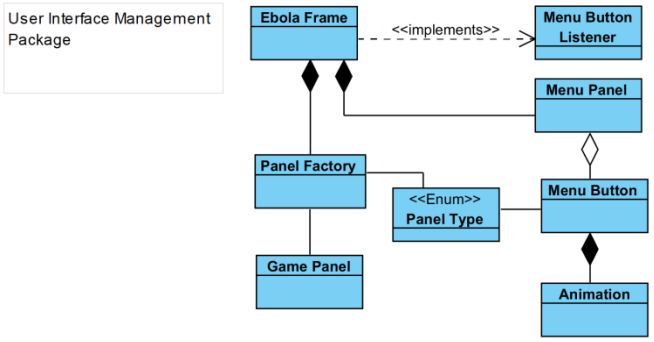


Figure 1

The *Game Engine* controls all the other managers: "Input Manager", "Map Manager", "Sound Manager", and "Reader" packages. "Map Manager" is in charge of "Entity Manager".

The *Input Manager* controls the mouse and keyboard inputs such as commands like moving left or clicking a button.

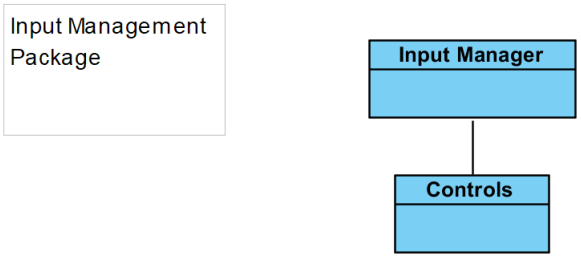


Figure 2

The *Map Manager* manages "Map"s which manages "Room"s.

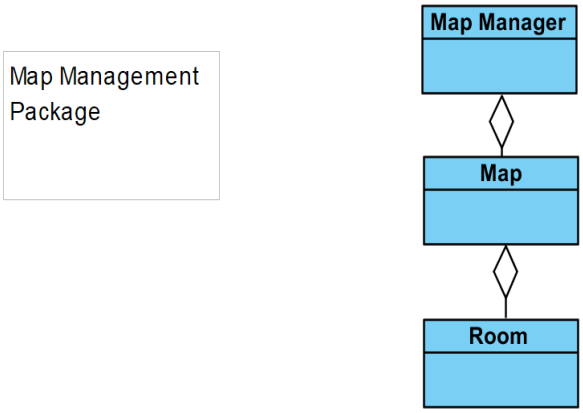


Figure 3

Sound Manager package involves “Sound Manager”. “Sound Manager” is responsible of controlling background sounds via “Background Sound” and interactable sounds such as the sound of an “Alive” entity dying or firing.

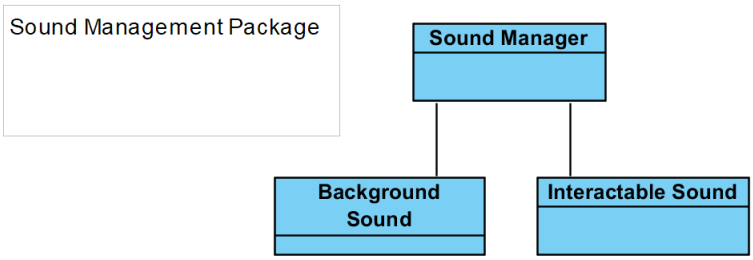


Figure 4

Reader package is responsible of reading from pre-existing files and directing needed data to “Game Engine” to create entities in the game.

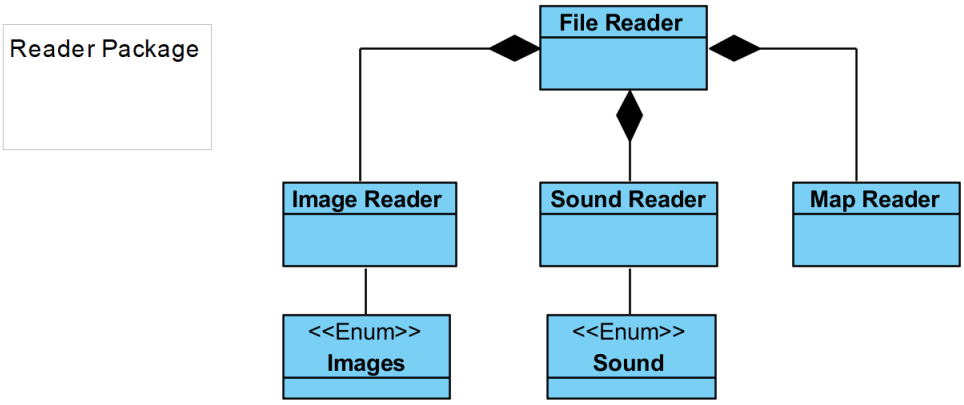


Figure 5

Entity Management has “Celly Manager” to manage Celly and “Enemy AI Manager” to manage the movement and action of Celly’s enemies. “Entity Manager” also manages “Tile Generator”. “Tile Generator” generates tiles using “Tile Factory”.

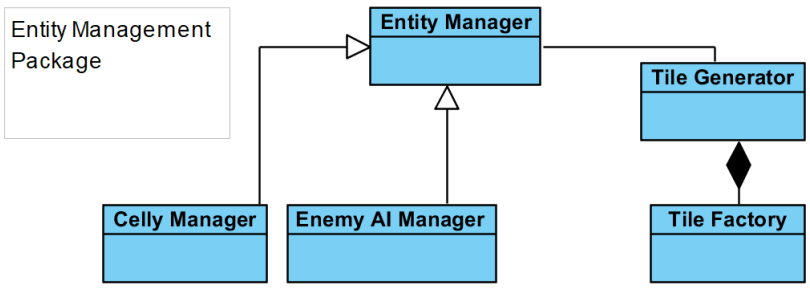


Figure 6

Game Entities package contains all of the other entities. Entities are divided into three categories: “Alive”, “Interactable” and “Tile”. “Alive” entities are “Celly”, “Virus”, and the ultimate enemy “EBOLA”. “Interactable” entities are “Chest”, “Key”, and “Portal”. “Celly” will be able to interact with them. “Tile”s are neither alive nor interactable. “Grass”, “Rock”, and “Water” is a “Tile”.

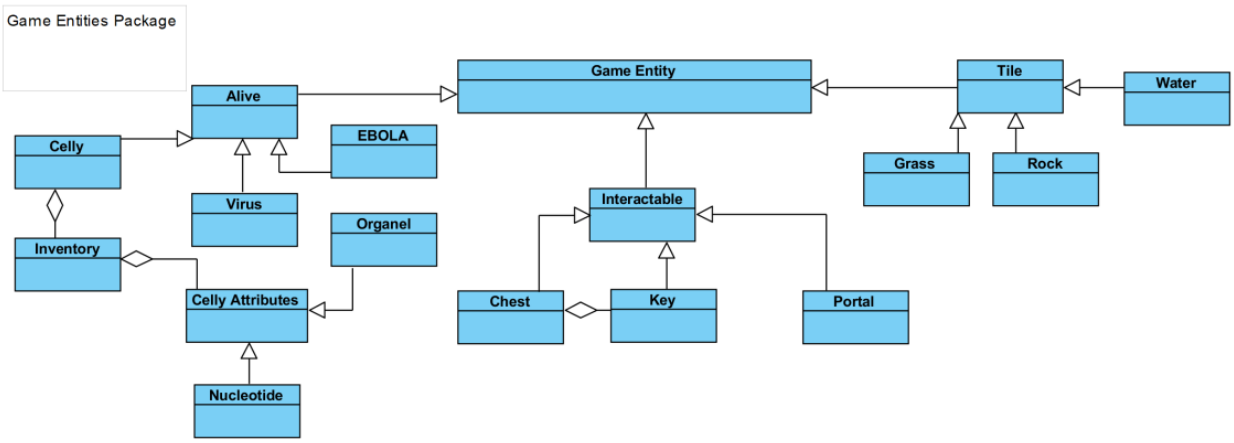


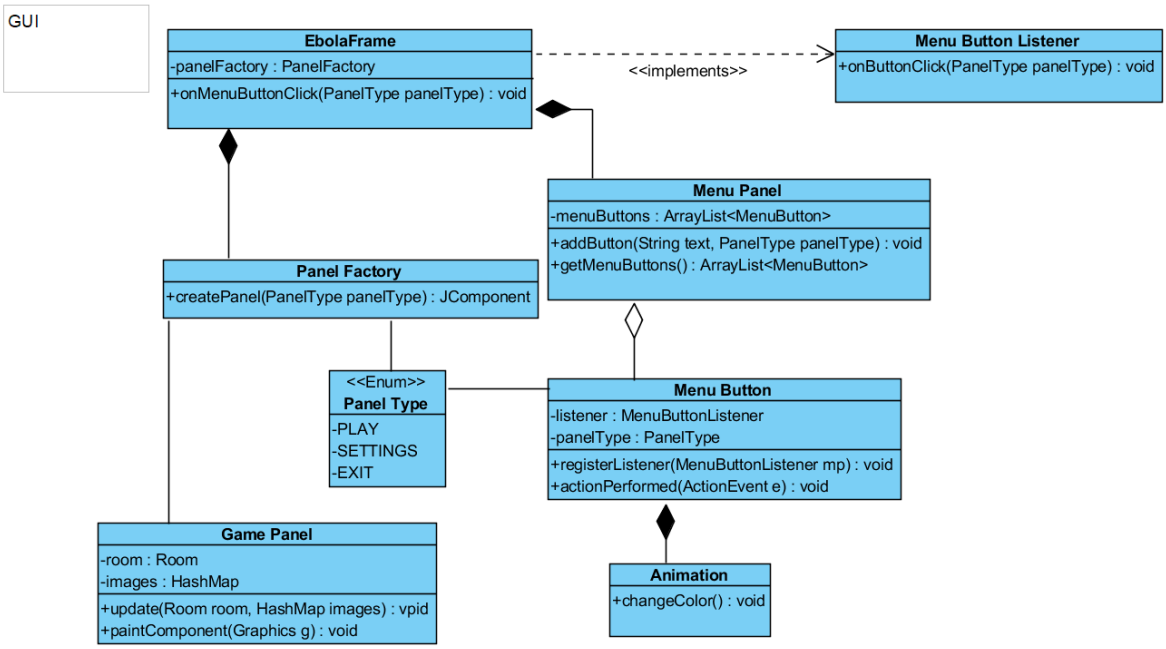
Figure 7

3.2 Detailed Object Design

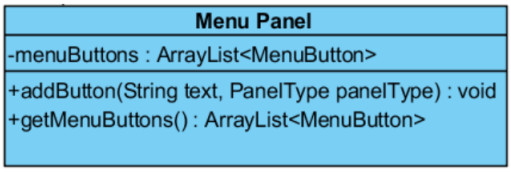
All of the classes of e-Bola, their attributes and their operations are given below. The relationships between classes are also shown in this detailed class diagram. In the rest of section 3, the classes will be explained in-depth, individually.


Entity Manager
Game Entity
Map Manager
GUI
Reader
<<implements>>
Entity Generator
-aliveEntities : ArrayList<Alive>
-interactableEntities : ArrayList<Interactable>
+generateEntities() : Entity[][][]
Entity Factory
+createObject(char c) : Entity
Entity Manager
Enemy AI Manager
-virus : ArrayList<Virus>
+checkStatsChange() : boolean
+simpleRandomAction() : void
Celly Manager
-celly : Celly
+decideAction(KeyEvents event) : void
+attack() : void
+interact() : void
+move(KeyEvents event) : void
+checkStatsChange() : boolean
Input Management
Game Engine
MapManager
-map : Map
+checkInteraction(Point point) : boolean
+changeRoom(Gate gate) : void
Map
-currentRoom : Room
-rooms : Room[]
+changeCurrentRoom(Room room) : void
Room
-entities : Entities[][]
-gates : ArrayList<Gate>
Menu Button Listener
+onButtonClick(PanelType panelType) : void
EbolaFrame
-panelFactory : PanelFactory
+onMenuButtonClick(PanelType panelType) : void
Menu Panel
-menuButtons : ArrayList<MenuButton>
+addButton(String text, PanelType panelType) : void
+getMenuButtons() : ArrayList<MenuButton>
Menu Button
-listener : MenuButtonListener
-panelType : PanelType
+registerListener(MenuButtonListener mp) : void
+actionPerformed(ActionEvent e) : void
Animation
+changeColor() : void
Panel Factory
+createPanel(PanelType panelType) : JComponent
<<Enum>>
Panel Type
-PLAY
-SETTINGS
-EXIT
Game Panel
-room : Room
-images : HashMap
+update(Room room, HashMap images) : vpid
+paintComponent(Graphics g) : void
GameEntity
Alive
-hp : int
-pointCoordinates : Point
-movementSpeed : int
+updateHp() : void
+updateMovementSpeed() : void
+updateCoordinates(Point point) : void
Virus
Celly
-key : Key
-inventory : Inventory
+updateInventory() : void
Inventory
Point
Tile
-movementSpeedModifier : int
-damageOverTime : int
Rock
Water
Grass
CellyAttributes
Nucleotide
Organel
Portal
-destination : Portal[]
Interactable
Key
-keyType : int
Chest
-key : Key
-cellyAttributes : CellyAttributes[]
+tryKey(Key key) : boolean
Sound Management
Sound Reader
-soundMap : HashMap<EbolaSound, AudioInputStream>
+AudioInputStream audioStream()
+HashMap getSoundMap()
<<Enum>>
EbolaSound
-LEVEL1 BG
File Reader
+openFiles() : File[]
Map Reader
-rooms : String[][]
+readRoom() : String[]
Image Reader
-imageMap : HashMap<EbolaImage, BufferedImage>
+BufferedImage readImage(String className)
+HashMap getImageMap()
<<Enum>>
EbolaImage
-CELLY
-GRASS
-CHEST

3.3 User Interface Management Subsystem

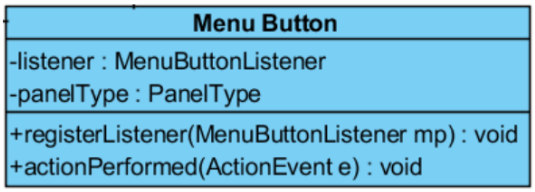


3.3.1 Menu Panel



MenuPanel is the panel that welcomes the user when the game launches. It contains buttons, an instantiation of the MenuButton class, which are used to navigate to other panels (e.g. game settings panel). The structure of User Interface package makes adding new buttons easier with the MenuButton class and MenuButtonListener interface.

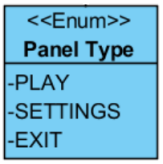
3.3.2 Menu Button



MenuButton class is a child of Java JButton class. It has methods to make the Ebola Frame listen to any instance of MenuButton and also holds values from Panel Type enumeration to help Ebola Frame understand which panel to create when the Menu Button is pressed.

3.3.3 Panel Type

Panel Type enumeration holds the values that correspond to the panels that can be created.

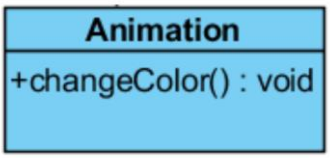


3.3.4 Menu Button Listener

Menu Button Listener interface forms a contract between Ebola Frame class and the Menu Button objects. This way, Ebola Frame knows which buttons is pressed, when it is pressed, and which panel should be created.

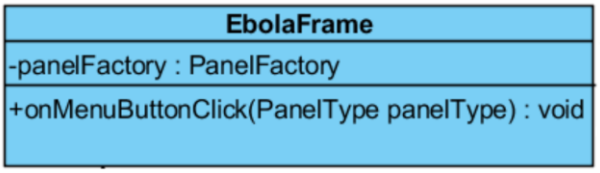
3.3.5 Animation

Animation class is used by MenuButton objects to make the interaction with the user more responsive.



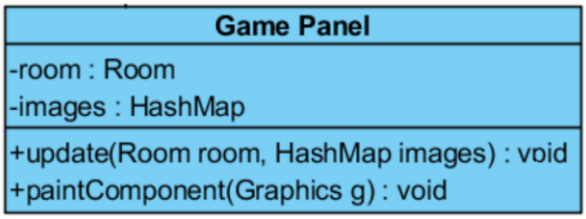
3.3.6 Ebola Frame

EbolaFrame class holds the main application frame. It implements the MenuButtonListener interface and listens to MenuButton clicks. When there is a MenuButton click, EbolaFrame asks the Panel Factory to create the appropriate panel, using the PanelType enumeration.

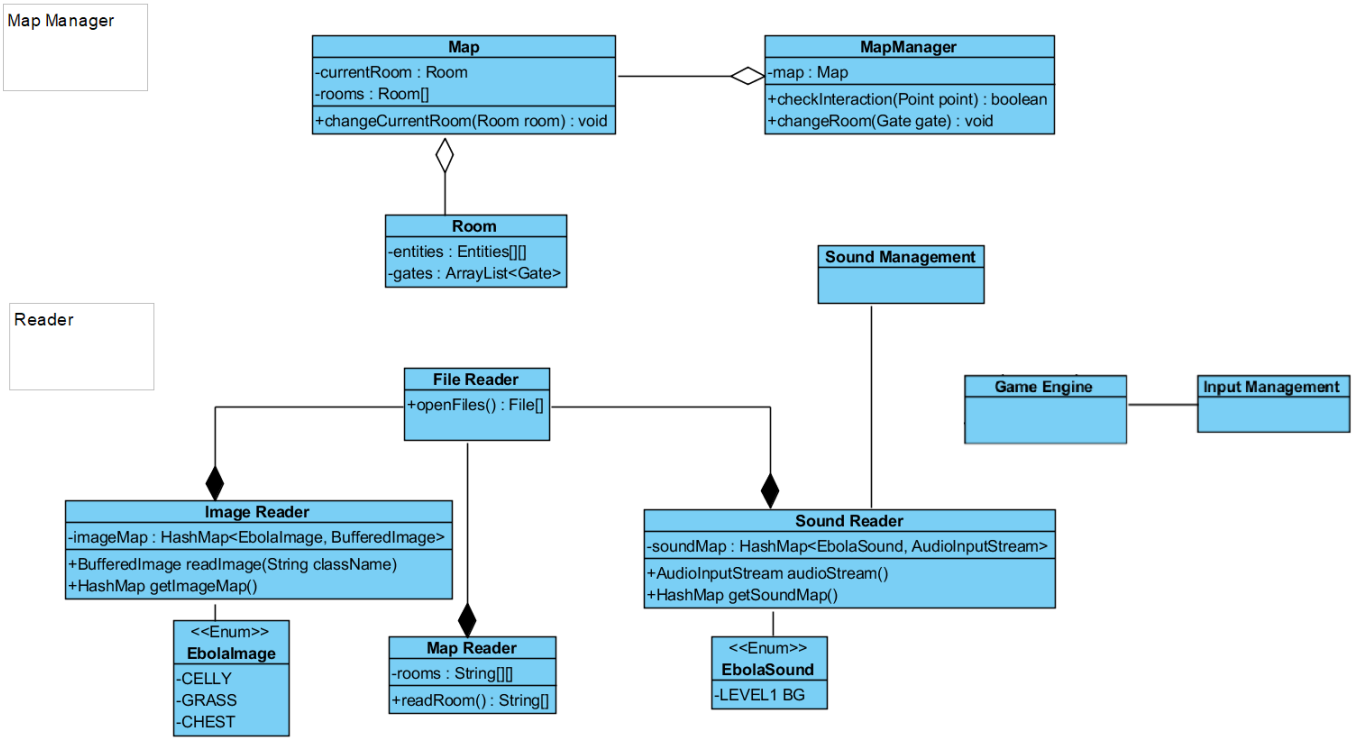


3.3.7 Game Panel

GamePanel is the panel where the actual gameplay is shown. When the user clicks the “Play” button, GamePanel is created along with Game Engine. Game Panel uses ImageReader class from the Reader package to obtain a hashmap. Game Panel retrives information about the game status from the GameEngine via GameEntity objects. Game Entity objects are then painted to the screen using this map.



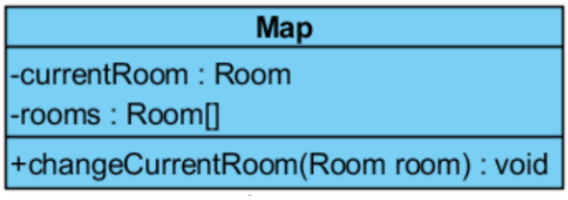
3.4 Game Management Subsystem Interface



Map Management Package

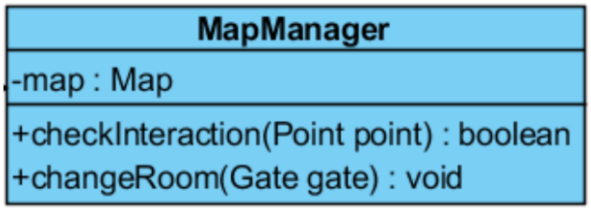
Map Management Package consists of classes that are related to the objects in a game level and their internal statuses.

3.4.1 Map



Map class contains the Room objects of the current game level. One of the rooms in the map will be the current room in the game while all others will stay dormant until the player proceeds to another room. The current room will be the room that the player can interact with, so other room objects will not be manipulated.

3.4.2 Map Manager

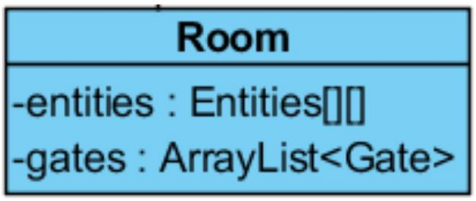


Map Manager is used to hold all the information about the current Map in the game. Depending on the current state

of the game, the Map Manager will check if there are any interactions with any Portal objects. In the presence of an interaction, it will change the currentRoom variable to a different Room object.

3.4.3 Room

Room class contains the Game Entity objects as a two dimensional array. Dimensions of this array corresponds to the rows and columns of the Game Panel. Room also contains the Portal objects that link rooms to other rooms.

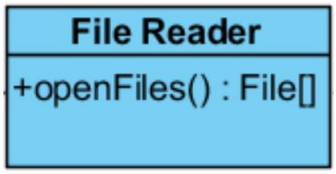


Reader Package

Reader package is responsible of reading all game related data into memory. Advantages of this is further explained in the trade-offs section (1.4).

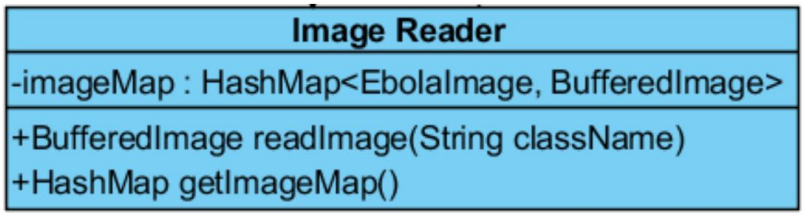
3.4.4 File Reader

All readers use the File Reader class for their reading processes. The main aim of the File Reader class is to locate the game files and verify their integrity. If there are no problems, the files are opened and sent to the respective readers for the actual reading process.

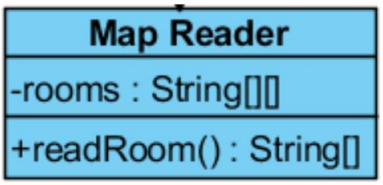


3.4.5 Image Reader

The Image Reader reads all image files into the memory. These images are than mapped to Game Entity objects. At runtime, this mapping will be used by the Game Panel to paint images corresponding to the objects that are present at that current screen. The mapping is done between EbolaImage and BufferedImage. EbolaImage is an enumeration whose values correspond to the image file names and names of Game Entity classes. BufferedImage is a part of Java awt package.

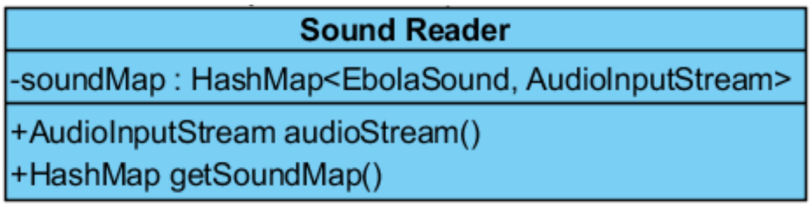


3.4.6 Map Reader



The Map Reader reads the text files that correspond to the Room objects at runtime. Since these text files contain multiple attributes about the room, it is important to differentiate between information and send them to object creation in an orderly manner. The way the files are read will determine the connection between the Portal objects. The Entity Generator will use this information to create Room objects and their connection with the other Portal Objects.

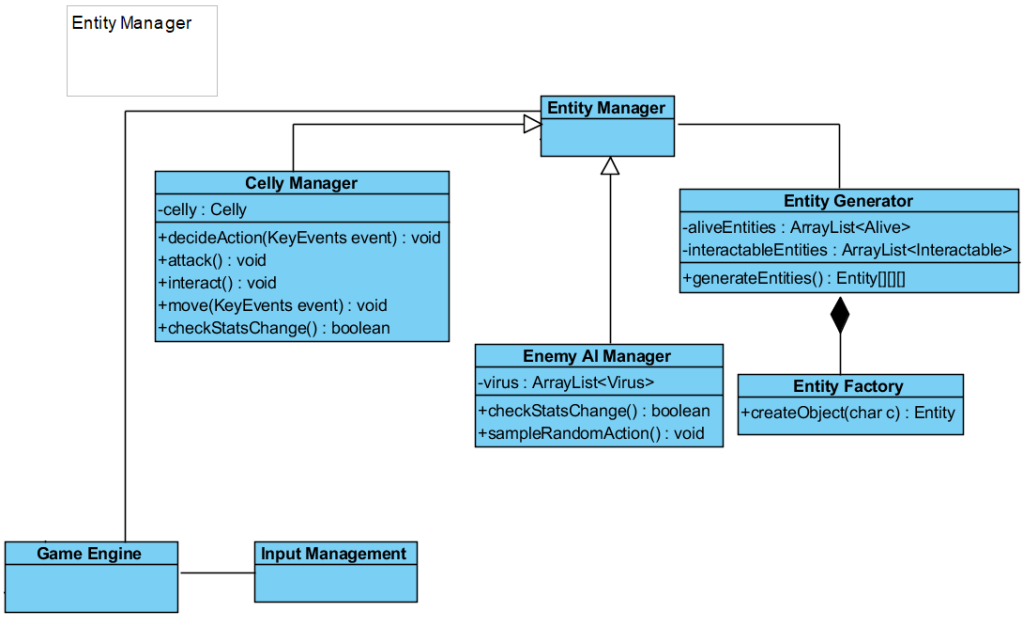
3.4.7 Sound Reader



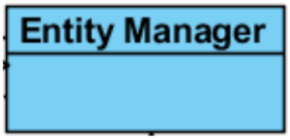
The Sound Reader class is very similar to the Image Reader class, but is related to the in-game sounds. The Sound Reader reads all of the sound files into the memory. These sounds are than mapped to Room objects, or actions of the Alive objects. At runtime, this mapping will be used by the Sound Manager to play sounds depending on the atmosphere, room or action. The mapping is done between EbolaSound and AudioInputStream. EbolaSound is an enumeration whose values correspond to the sound file names. AudioInputStream is a part of Java sound package.

Entity Management Package

Entity management package is used to create and maintain the model objects of the game, namely the classes in the Game Entity Package.

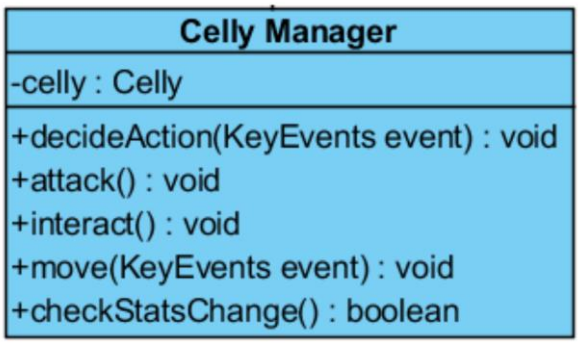


3.4.8 Entity Manager



EntityManager is the core of the GameEntity object lifecycle management. It asks the Entity Generator to create objects whenever GameEngine needs them. It then orders this object data and creates its children, CellyManager and Enemy AI Manager, and passes the relevant data to them. Whenever the GameEngine updates the game status, EntityManager checks the new status and gives appropriate actions to CellyManager and Enemy AI Manager. Since changes in the status of entities can often conflict with or concern other entity statuses, EntityManager, CellyManager and Enemy AI Manager constantly share data to make sure all actions are legal within the constraints of game logic.

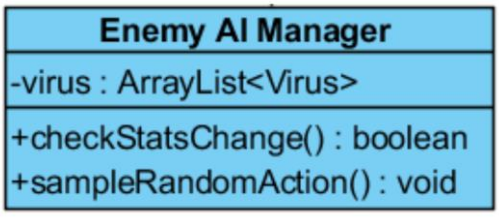
3.4.9 Celly Manager



Celly Manager is a class that manipulates the Celly object depending on the environmental input (tiles, interactables, viruses) and user input (movement keys). Certain game statuses can cause Celly to interact with objects. The User input allows Celly to move. AI of viruses can create situtations where Celly gets updated without any user input (e.g. losing health points after being attacked by viruses).

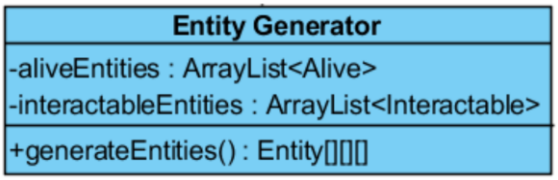
3.4.10 Enemy AI Manager

Enemy AI Manager is a class that determines what the computer controlled viruses (Virus objects) will do in the current status of the game and manipulates the Virus objects accordingly.



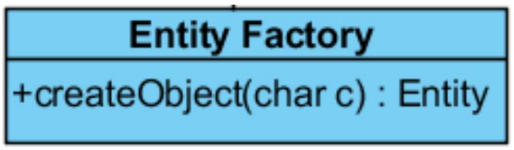
3.4.11 Entity Generator

EntityGenerator creates the objects and sends them to EntityManager for ordering.



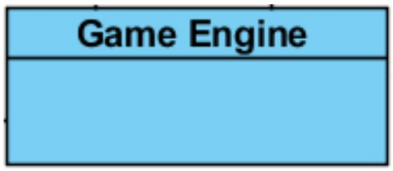
3.4.12 Entity Factory

EntityFactory creates objects using the data provided by the MapReader class (from Reader package).



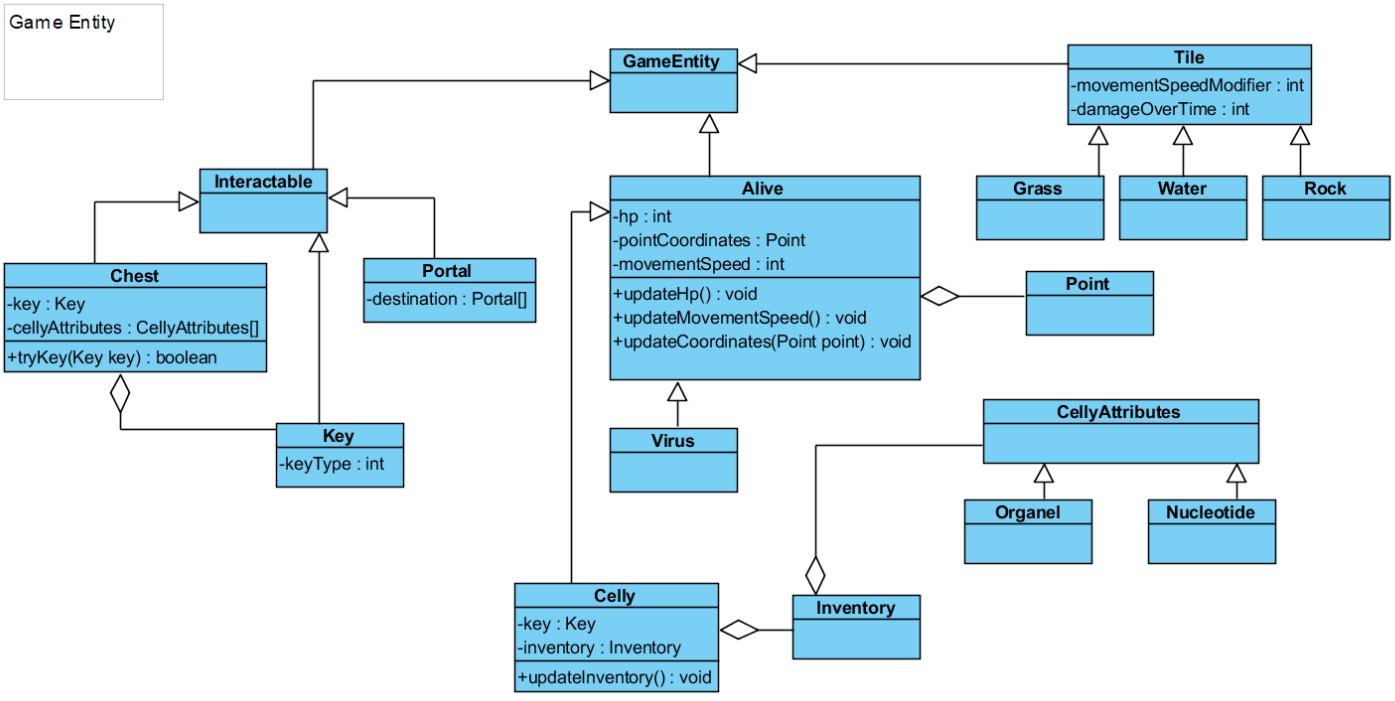
3.4.13 Game Engine

Game Engine is the core of e-Bola game. It connects all the managers across all packages and transmits data among all of them. The main game loop runs within this class and on every iteration, the updated status gets evaluated. The result of this evaluation determines which managers to signal to.



3.5 Game Entities Subsystem Interface

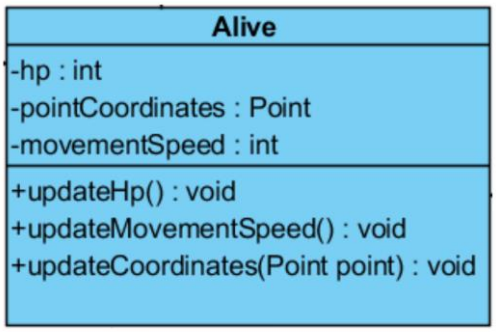
GameEntity Package contains all the entities in the game that are used by other classes. Classes in this package act as models. These models are used to create objects or manipulated depening on the evaluations of manager classes. Most of the model classes that are in the game are located in this package, except some classes that need high coupling with their managers (e.g. Map & Room models with MapManager).



3.5.1 Game Entity

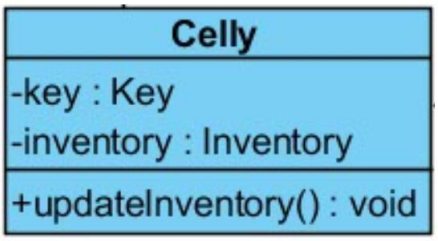
GameEntity class is the main abstraction of all models, it is used to pass all the GameEntity objects in a single polymorphic relationship. It is also conveys the meaning that all GameEntity objects (that are descendents of this class) can be drawn into the GamePanel.

3.5.2 Alive



Alive is the abstraction for “moving and mortal” objects of the game. Descendents of Alive have health points that allow them to maintain their presence in the game. They have a certain place in the coordinate system (determined by the Point class) of the GamePanel and they can change their position and the speed at which they move to a new position.

3.5.3 Celly



Celly is a concrete entity which represents the cell the user plays as. The distinction between Celly and other concrete Alive entities is that Celly has an Inventory object and can hold a Key object. Also, Celly is a special object that has its own manipulator (CellyManager class) that uses the inputs given by the user.

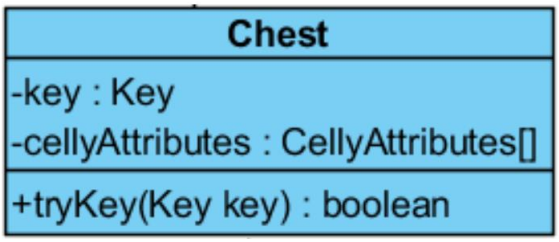
3.5.4 Virus

Virus is a concrete entity which represents the enemies of the game. The distinction between Virus and other concrete Alive entities is that it is controlled by an AI in its actions. Virus class has its own manipulator (Enemy AI Manager) to decide on these actions.

3.5.5 Interactable

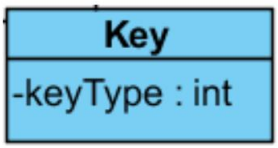
Interactable is an abstraction for obejcts that “do something when triggered”. Descendents of Interactable, trigger the GameEngine when Alive entities come too close (in terms of location in the coordinate system [again determined by Point object that Alive enitites have]). GameEngine then asks the appropriate managers to manipualte game state.

3.5.6 Chest



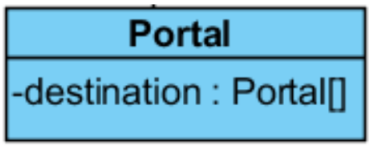
Chest is a concrete entity which represents chests that can be opened by by the user using a fitting Key object. Chest objects contain CellyAttribute objects that the user can take.

3.5.7 Key



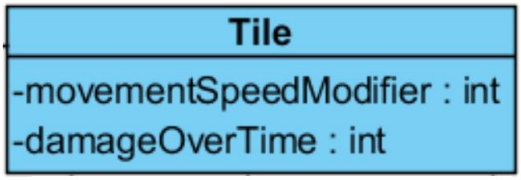
Key is a concrete entity that the user collects to open chests.

3.5.8 Portal



Portal is a concrete entity that is used to switch between rooms of the map. Each Portal object connects to another Portal object that is contained in another Room object within the same level. Connection between Portal objects are determined by the EntityGenerator, which uses room text files as input.

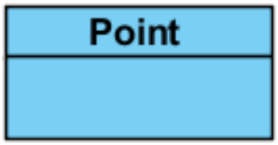
3.5.9 Tile



Tile is an abstraction for objects that represent the floor of the game board, where “Alive enitites walk on”. Descendent of Tile class have different values for modifiers and different images that are assigned to them during painting of the GamePanel. Modifiers can change the movement speed of Alive entities depending on the tile they are currently contained within.

3.5.10 Point

Point class represent the current coordinates of Alive entity objects.

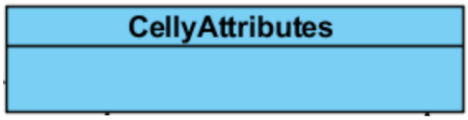


3.5.11 Inventory

Inventory class holds objects of the CellyAttribute class. It represents a limited amount of CellyAttribute objects Celly can hold.

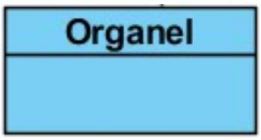
3.5.12 Celly Attribute

CellyAttribute class is a polymorphic abstraction to objects Celly can have to increase its attributes.



3.5.13 Organel

Organel class represents objects with certain attributes that can be held in the Inventory of Celly.



3.5.14 Nucleotide

Nucleotide class represents the main currency in game.