  
  
  
  
CS 319 - Object-Oriented Software Engineering  
Design Report  
  
  
Punch For Glory  
Group 1-C

Taner Baygün

Özgür Can Erdoğan  
Saba Betigül Şahinkaya  
Ömer Berk Uçar

**1. Introduction**

**1.1 Purpose of the System**

Punch for Glory is a system aiming to entertain user with well-designed gameplay which includes difficulty levels and some power-ups. Punch for Glory looks very poor in terms of graphics. However, gameplay of Punch for Glory is designed to maximize the sense of achievement for player. Punch for Glory has a user-friendly interface which reduces the complexity of the gameplay. Opponent boxer’s levels are low at the beginning of the game. When, the player increase his position in the league, his opponents will be more difficult. The reason behind it is, increasing the user’s game experience.

**1.2 Design Goals**

* **Efficiency:** The system is going to be responsive and able to run with high performance. The game will run at least 30 fps in order to provide smoothness. This is one of the most important design goal because performance is highly related with the user’s gameplay experience. In order to get optimum performance, memory usage of the game will be minimized, and also, each objects gets allocated memory in order do their own tasks. This will boost the performance of the game significantly because it will decrease the workload.
* **Reliability:** The game will be bug-free and stable in the boundary conditions. The game should not crash at any time. To get this goal, the testing procedures will involve. The testing procedure will continue each stage of the development of the game. Therefore, boundary conditions will be tested carefully in order to avoid unexpected game crashes.
* **Adaptability:** Java is one of the most well-known programing language which provides cross-platform portability. Our game can work all JRE installed platforms, therefore, user should not worry about the operating system requirements. Using Java will cause to sacrifice performance but its adaptability obscures the performance problems.
* **Usability:** One of the most important thing about our game is, it should be easy to play. Therefore, it affects our design goals too. This makes the game more user-friendly. Therefore, the game will be designed such that user can easily understand the concept of the game and user can easily interact with it. However, user-friendly interface doesn’t mean that the gameplay easier. If user-friendly interface and gameplay is easier, the player can be bored immediately.
* **Extensibility:** Object oriented architecture of the game enables system changes without causing any bugs or harming other classes. Modifications are easier by using object oriented architecture. For example, we can add new opponents or change the opponents without having to modify anything in other classes. Therefore, object oriented architecture minimizes the possibility to cause some malfunctions in other classes.

**Tradeoffs**

* **Efficiency – Reusability:** Reusability is not the main concern for our game. Because, our system is not planned to integrate to other systems. Therefore, the classes are designed specifically for our game. Also, it prevents us to being too complex in class implementation. The main design approach is being efficient in our game.
* **Functionality – Usability:** The most important thing for a game, its usability. Because, costumers are our main targets and they should understand the game easily. Therefore, functionality of the game should be basic.We focused on usability rather than functionality. Because our purpose is entertaining the people. The game has simple user interface and easy gameplay. Thus, they can enjoy while playing game rather than suffering.
* **Space – Speed:** The implementation details of our game is planned to be fast. We don’t merge any object to earn memory space. Reason behind it is, making game fast as possible. If we consider our game, there will be plenty of objects that use memory. However, we try to keep our game simple as possible, therefore, memory problem is not the most important one especially in this era. However, if the speed of our game is not enough to provide 30 fps, users don’t like the gameplay. Therefore it should be fast.

**1.3 Definitions, acronyms, and abbreviations**

**Cross-platform:** Cross-platform refers to ability of software to run in same way on different platforms such as Microsoft Windows, Linux and Mac OS X.

**JRE:** Java execution environment is termed as the Java Runtime Environment (JRE). All systems need Java Runtime Environment (JRE) in order to execute the projects which are developed in Java.

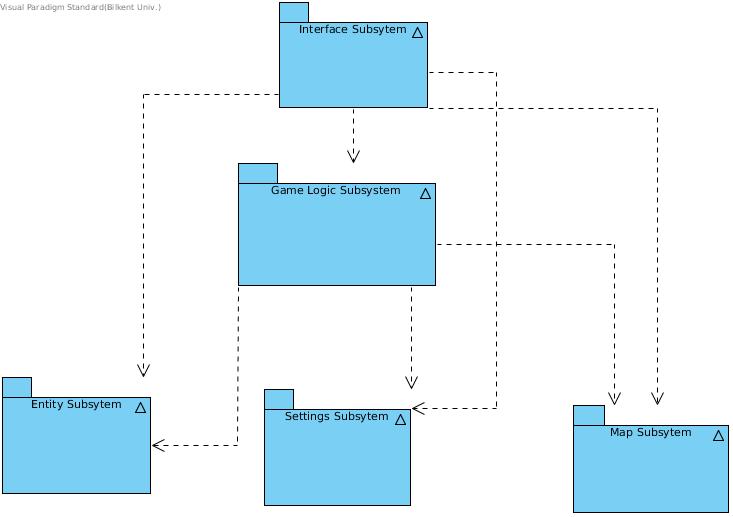
**Fps:** Abbreviation of “frames per second”. Fps represents the number of graphical layouts can be prepared by the system each second.

**Boundary conditions:** Conditions of the system which may generate run-time errors. They are exceptional cases according to the normal flow of the program. These conditions must be handled carefully for robustness of the system.

## 2. Software Architecture

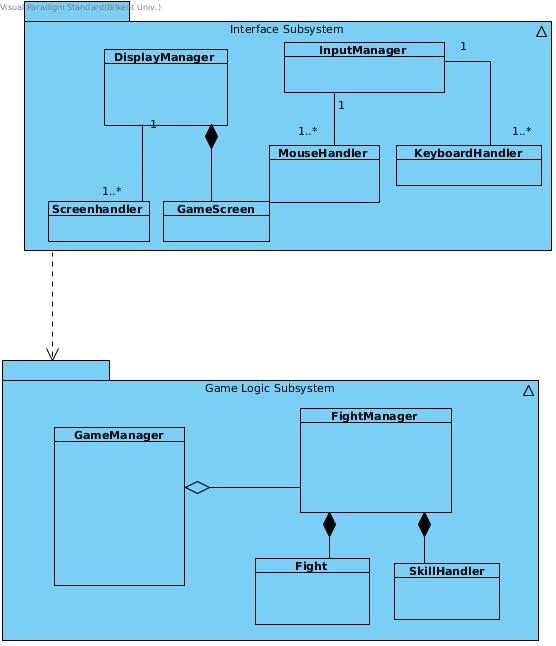
**2.1 Subsystem Decomposition**

We choose three-tier architectural decomposition for our game system because it is the best option for our design. Subsystem decomposition includes Game Logic Subsystem, Interface Subsystem, Entity Subsystem, Settings Subsystem, Map Subsystem as shown in figure 1.

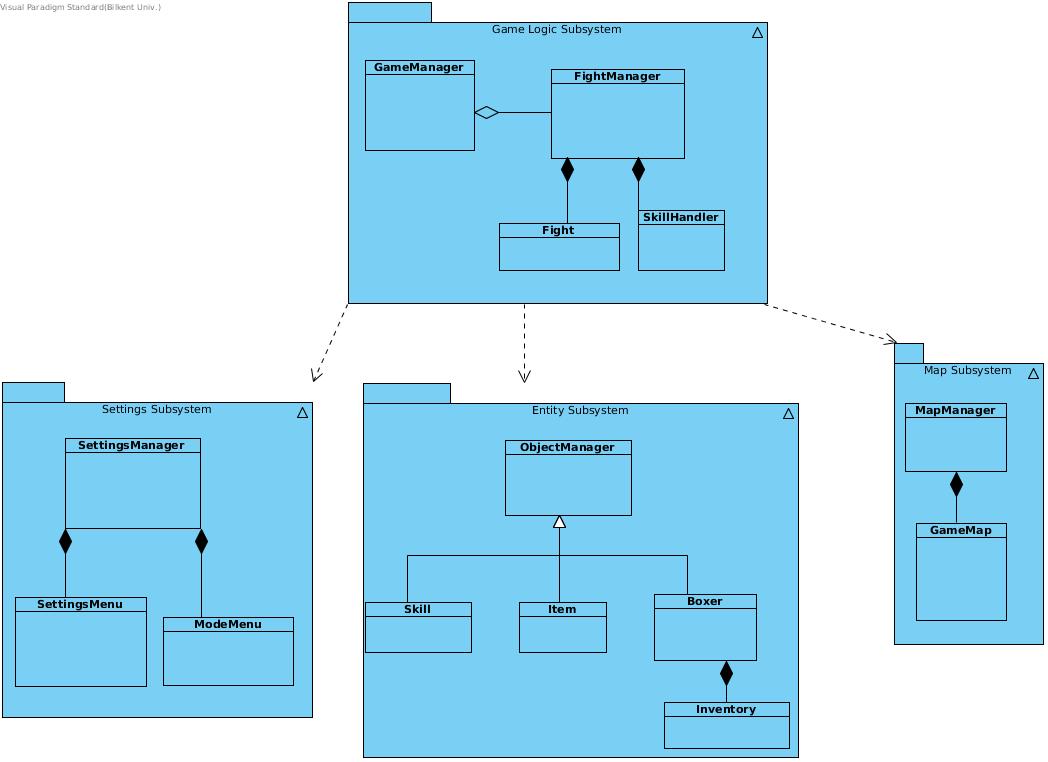


Interface Subsystem is in the first layer. User see user interface of the system first and interface subsystem is responsible for user interface related things. It has DisplayManager and InputManager classes for user interface events. It has open architecture so interface subsystem connected with both classes in tier-two and tier-three. Then with the help of user interface, user will start game, change settings etc.

User control other system components over GameManager class. Game Logic Subsystem responsible for game and fight related issues.



GameManager Class create, update and delete tier-three subsystem classes. It connected with other subsytems' Façada classes. With the relation between them GameManager Class control over everything in the game.



**2.2 Hardware / Software Mapping**

Fight for Glory game will require the Java Runtime Environment to be executed because it is developed in Java platform. As hardware requirements, the game mostly requires a mouse and a partially keyboard to interact with the game.

In terms of I/O requirements and graphical requirements of the game, it requires an average computer for today’s standards as a requirement. Java’s 2D graphic library uses GPU when it is available, therefore, having a GPU results better graphical performance. On the I/O side, the game requires very small requirements which is a mouse and a keyboard. Thus, it cannot be considered as a big problem in today’s computer environment.

**2.3 Persistent Data Management**

Game data will be stored in the client’s local hard disk drive. Our game will not use any database system since the data that is used in the game needs to be accessed in real-time. Thus, all the necessary files and data will load on to memory. When the game requires these files, these files are accessed directly from the memory. The background images, boxer’s images, item’s images, skill’s images, and other important user-interface images will be stored on to client’s hard disk drive.

**2.4 Boundary Condition**

The game will give an error if the file or the saved data on the memory are corrupted. After getting the error message, these content will be deleted and the game will return to main menu if the user wants to start a new game.

## 3. Subsystem Services



## 3.1 Settings Subsystem

### 3.1.1 SettingsMenu Class

### designClasses

#### Attributes:

**Private int red:** This int holds a value for character’s color.

**Private int green:** This int holds a value for character’s color.

**Private int blue:** This int holds a value for character’s color.

**Private int[] appearance:** Appearance is an array includes color variables. It is using to pass parameters to stickman on screen until player find the color he wants to use then it passes to player instance of boxer class.

**Private int vol:** It is to determine the volume. It is between 0(Mute) and 100(Max).

#### Constructors:

**public setAll():** Initializes red, green, blue to 0 (black), vol to 50 (med) and appearance to red,green,black.

#### Methods:

**public SettingsMenu():** It calls setAppearance and setVolume methods.

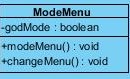
**setVolume( int):** It is to set volume level.

**setAppearance( int, int, int):** It is to set appearance variable as red,green,blue.

**setDefault():**Initializes red, green, blue to 0 (black), vol to 50 (med) and appearance to red,green,black.

**changeSettings(int[], int):** It is to set appearance and volume in one method.

### 3.1.2 ModeMenu Class



#### Attributes:

**Private boolean godMode:** It is to determine difficulty level. Thus Punch For Glory have 2 difficulty levels it uses boolean. If godMode 1 it is hard else it is easy.

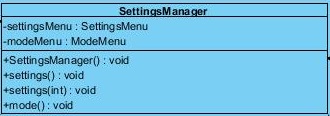
#### Constructors:

**publicModeMenu( ):** It sets godMode 0(easy) as default value.

#### Methods:

**changeMode( ):** It changes godMode value. If current value is 0 it makes 1, if 1 it makes 0.

### 3.1.3 SettingsManager Class



#### Attributes:

**Private SettingsMenu settingsMenu:** It is an object of SettingsMenu class.

**Private ModeMenu modeMenu**: It is an object of ModeMenu class.

#### Constructors:

**settingsManager( ):** It calls settings( )and mode( ) to give default values to them.

#### Methods:

**settings( ):** It calls setDefault( ) of settingMenu class.

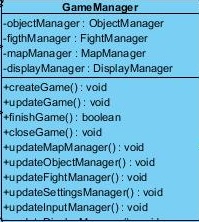
**settings( int, int, int, int):** It calls setAll(int ,int, int, int) to set appearance and volume level according to user input.

**mode( ):** It calls changeMode( ). It is using only when user change mode option in menu and press apply.

## 3.2 Game Logic Subsystem

### 3.2.1 GameManager Class

GameManager Class is the controller of all the manager type classes. It connects with all of them and get and send information for continuity of the gameplay. It is the core of the game system.



***Attributes:***

**objectManager** : It is an instance of ObjectManager Class and use for connect GameManager Class with ObjectManager Class.

**fightManager** : It is an instance of FightManager Class and use for connect GameManager Class with FightManager Class.

**mapManager** : It is an instance of MapManager Class and use for connect GameManager Class with MapManager Class.

**displayManager** : It is an instance of DisplayManager Class and use for connect GameManager Class with DisplayManager Class.

#### Methods:

**createGame( )** : Create all manager classes to start gameplay.

**UpdateGame( )** : Modify changes in gameplay to other manager classes.

**finishGame( )** : Check whether game is finished or not.

**closeGame( )** : Delete all manager classes which are created to end the game.

**updateMapManager( )** : Send modification information about map to MapManager Class.

**updateObjectManager( )** : Send modification information about entities to ObjectManager Class.

**updateFightManager( )** : Send modification information about fight event to FightManager Class.

**updateSettingsManager( )** : Send modification information about settings to SettingManager Class.

**updateInputManager( )** : Send modification information about inputs to InputManager Class.

### 3.2.2 FightingManager Class

FightManager Class is created when boxer goes to the arena or street which means he is going to fight. FightManager Class manage the general control mechanisms of fights. It communicate with SkillHandler class and after skills are used FightManager Class calculate results.

### designClasses (3)

***Attributes:***

**fight** : A Fight Object instance. Use for connect FightManager Class with Fight Class

**skillHandler**  : A SkillHandler Object instance. Use for connect FightManager Class with SkillHandler Class

#### Methods:

**calculateDamage( )**: calculate the given damage to opponent in the round.

**calculateMiss( )** : calculate the miss rate of given damage.

**updateHealth( )** : update the health information of the boxer after takes opponent’s damage.

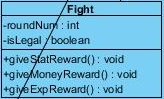
**updateStamina( ):** update the stamina information of the boxer.

**createFight( )** : Start a fight by creating a Fight Class instance.

**endFight( )** : End fight by deleting Fight Class instance.

### 3.2.3 Fight Class

Fight Class is created and finished by FightManager Class. It is model part of fighting event. It includes information about the fight and its effects.



***Attributes:***

**roundNum** : Number of total round in the fight instance. Fight will be finished after the round number reached.

**isLegal** : It is the attributes that shows whether fight is legal or not.

#### Methods:

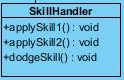
**giveStatRewards( )** : send the reward information regarding stats to boxer.

**giveMoneyRewards( )** : send the reward information regarding money to boxer.

**giveExpRewards( )** : send the reward information regarding experience to boxer.

### 3.2.4 SkillHandler Class

SkillHandler Class is responsible for activating skills boxer uses during fights. Boxer has three skills and use them during fight rounds and SkillHandler Class get the input and apply chosen skills. It communicate with FightManager Class, send results, for continuity of fighting.



#### Methods:

**applySkill1( )** : send the request of applying skill 1 with the information of skill 1.

**applySkill2( )** : send the request of applying skill 2 with the information of skill 2.

**dodgeSkill( )** : send the request of applying dodge skill with the information of dodge skill.

**3.3 Map Subsystem**

Map Elements Subsytem is responsible for creating and loading places in the game. It consist of MapManager Class and GameMap Class.

### 3.3.1 MapManager Class

MapManager Class is controller of maps in the game. Its main duty is creating and open the map with all the places we determine.



***Attributes:***

**map** : It is an instance of GameMap Class and connect MapManager Class with GameMap Class.

#### Methods:

**createMap( )** : Creates the map with all the places in the map determined beforehand.

**openMap( )** : Send display request of opening the general map of the game.

**goArena( )** : Get the arena map information from GameMap Class and request displayin it.

**goHome( )** : Get the home map information from GameMap Class and request displayin it.

**goStreetArena( )** : Get the street arena map information from GameMap Class and request displayin it.

**goGym( )** : Get the gym map information from GameMap Class and request displayin it.

**goWorkplace()**  : Get the workplace map information from GameMap Class and request displayin it.

### 3.3.2 GameMap Class

GameMap Class is used for sending information of map areas or general map on the screen. Its purpose is loading places on map.



#### Methods:

**loadMap( )** : retrieve general map information from database and send it.

**loadArena( )** : retrieve arena map information from database and send it.

**loadHome( )** : retrieve home map information from database and send it.

**loadGym( )** : retrieve gym map information from database and send it.

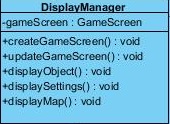
**loadStreetArena( )** : retrieve street arena map information from database and send it.

**loadWorkplace( )** : retrieve workplace map information from database and send it.

**3.4 Interface Subsystem**

### 3.4.1 DisplayManager Class

Control mechanism of display of game. Get requests and display them on the screen.



***Attributes:***

**gameScreen** : An instace of GameScreen Class. It connect DisplayManager Class with GameScreen Class.

#### Methods:

**createGameScreen( )** : Create an instance of GameScreen Class.

**updateGameScreen( )** : Modify GameScreen Class instance.

**displayObject( )** : Display requested ObjectManager instance on the screen.

**displaySettings( )** : Display requested SettingsManager instance on the screen.

**displayMap( )** : Display requested MapManager instance on the screen.

### 3.4.2 GameScreen Class

GameScreen Class is the class directly responsible of screen view.

designClasses (3)

#### Methods:

**updateScreen( )** : Update the screen view.

### 3.4.3 ScreenHandler Class

Handler for screen view. Connecting DisplayManager instance with screen.

designClasses (3)

### 3.4.4 InputManager Class

Take care of all mouse and keyboard related inputs. It is connected with GameManager and send inputs to other system classes.

designClasses (3)

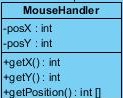
#### Methods:

**getMouseInfo( )** : Receive the mouse input info from MouseHandler Class.

**GetKeyboardInfo( )** : Receive the keyboard input info from KeyboardHandler Class.

### 3.4.5 MouseHandler Class

### Get mouse input informations and transmit them to InputManager Class.



***Attributes:***

**posX** : Position of mouse on X axis.

**PosY** : Position of mouse on Y axis.

#### Methods:

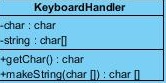
**getX( )** : Get position information of mouse on X axis.

**getY( )** : Get position information of mouse on Y axis.

**getPosition( )** : Combine position information of mouse on both X and Y axis and find the total position of mouse.

### 3.4.6 KeyboardHandler Class

Get keyboard input information and transmit them to InputManager Class.



***Attributes:***

**char** : Character input information.

**String** : Joined version of character inputs which makes string variables.

#### Methods:

**getChar( )** : Get character input information of keyboard.

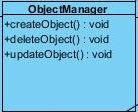
**makeString( )** : Create strings from united character inputs.

**3.5 Entity Subsystem**

Entity Subsystem declares the objects to show in the screen while the game is running. It has many important objects such as “Boxer”, “Item”, “Skill”, “Inventory” and ”ObjectManager”.

**3.5.1 ObjectManager Class**

“ObjectManager” will be instantiated after user decides to play the game. All fundamental objects use the “ObjectManager” abstract class as a parent class in order to update their information. “Boxer”, “Skill” and “Item” classes are subclasses of the “ObjectManager” class.



***Methods:***

**Public void createObject():** is an abstract operation which creates all the game objects since all objects need to act differently.

**Public void deleteObject():** this method deletes the game objects from the game.

**Public void updateObject():** this is an abstract operation which updates the game objects.

**3.5.2 Boxer Class**

****

**“**Boxer**”** class is one of the main objects of the game. These objects are instantiated after the instantiation of the “ObjectManager”class

***Attributes:***

**private int vitality:** This attribute is used for representing the boxer’s stats

**private int strength:** This attribute is used for representing the boxer’s stats

**private int agility:** This attribute is used for representing the boxer’s stats

**private int money:** This attribute is used for representing the boxer’s money

**private int experience:** This attribute is used for representing the boxer’s experience

**3.5.3 Item Class**

**“**Item**”** class is one of the main objects of the game. These objects are related with the boxers. Each object will have different stat effects on the boxer’s stats. These objects are instantiated after the instantiation of the “ObjectManager”class



***Attributes:***

**private int vitality:** This attribute is used for representing the item’s stats

**private int strength:** This attribute is used for representing the item’s stats

**private int agility:** This attribute is used for representing the item’s stats

**private int value:** This attribute is used for representing the item’s price

**private string type:** This attribute is used for representing the item’s type

**private boolean isLegal:** This attribute is used for representing the item’s legality.

**3.5.4 Skill Class**

The “Skill” objects initiated after the initiation of “ObjectManager” class. These Skill objects are used by boxer during fights.

designClasses (3)

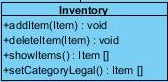
***Attributes:***

**private int power:** This attribute is used for representing the power of the skill

**private int missRate:** This attribute is used for representing the miss rate of the skill

**3.5.5 Inventory Class**

The “Inventory” class is for getting and storing items to boxer’s inventory.



***Methods:***

**Public void addItem():** This method adds item objects into boxer’s inventory

**Public void deleteItem():** This method deletes item objects from boxer’s inventory

**Public item[] showItems():** This method shows all item objects in the inventory.

**Public item[] setCategoryLegal():** This method shows only legal item objects in the inventory