CS 319 - Object-Oriented Software Engineering

Design Report

Left For F

Group 2A
Şekip Kaan Ekin
Ali Can Zeybek
Ömer Faruk Geredeli

1.Introduction

1.1 Purpose of The Game

Left 4 F is a system aiming to entertain users with a well-designed gameplay. The system has a user-friendly interface which enables the user to learn how to play easily. Game rules challenges user to learn them and play the game efficiently. Therefore, our main purpose is to design the game easy to learn hard to master.

1.2 Design Goals

- **Adaptability**[1]:

Java® is one of the few programming languages which provide cross-platform portability. This attribute of Java® enables our system to work in all JRE installed platforms; therefore, user will not have to worry about the operating system requirements. In order to fulfill this adaptability feature, we preferred to program the game in java by sacrificing some of the performance advantages of other programming languages such as C# or C++.

- **Reliability**:

System will be bug-free and consistent in the boundary conditions. The system should not crash with unexpected inputs. To achieve this goal, the testing procedures will continue simultaneously with each stage of the development. Besides, boundary conditions will be evaluated very carefully not to miss any unconsidered situation which may crash the system.

- **Usability**:

Easiness in the usage is an important design goal in such games in terms of user's comfort. It makes the game friendlier and attractive. Therefore, the system will be designed such that user can easily interact with the system without any prior knowledge. The game will have a commonplace outline in the main menu and in-game menu. However, user friendliness of the system does not imply making the gameplay easier, which might make the player bored sooner than expected.

- **Extensibility**:

Object oriented architecture of the game enables system customizations without causing any bugs during modifications. For instance, number of maps and episodes in the game could be extend without making any changes to other classes.

- **Performance**

We will design our game in a way that reactions of the user inputs will take no more time than 2 seconds on average.

1.3 Trade-Offs

- Memory vs Performance

Since object oriented programming used to design our game, memory usage is insignificant to us. This allows us to gain run time efficiency that helps us gain performance.

1.4 Definitions, Acronyms and Abbreviations

- **Cross-platform**[2]:

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**[3]**:**

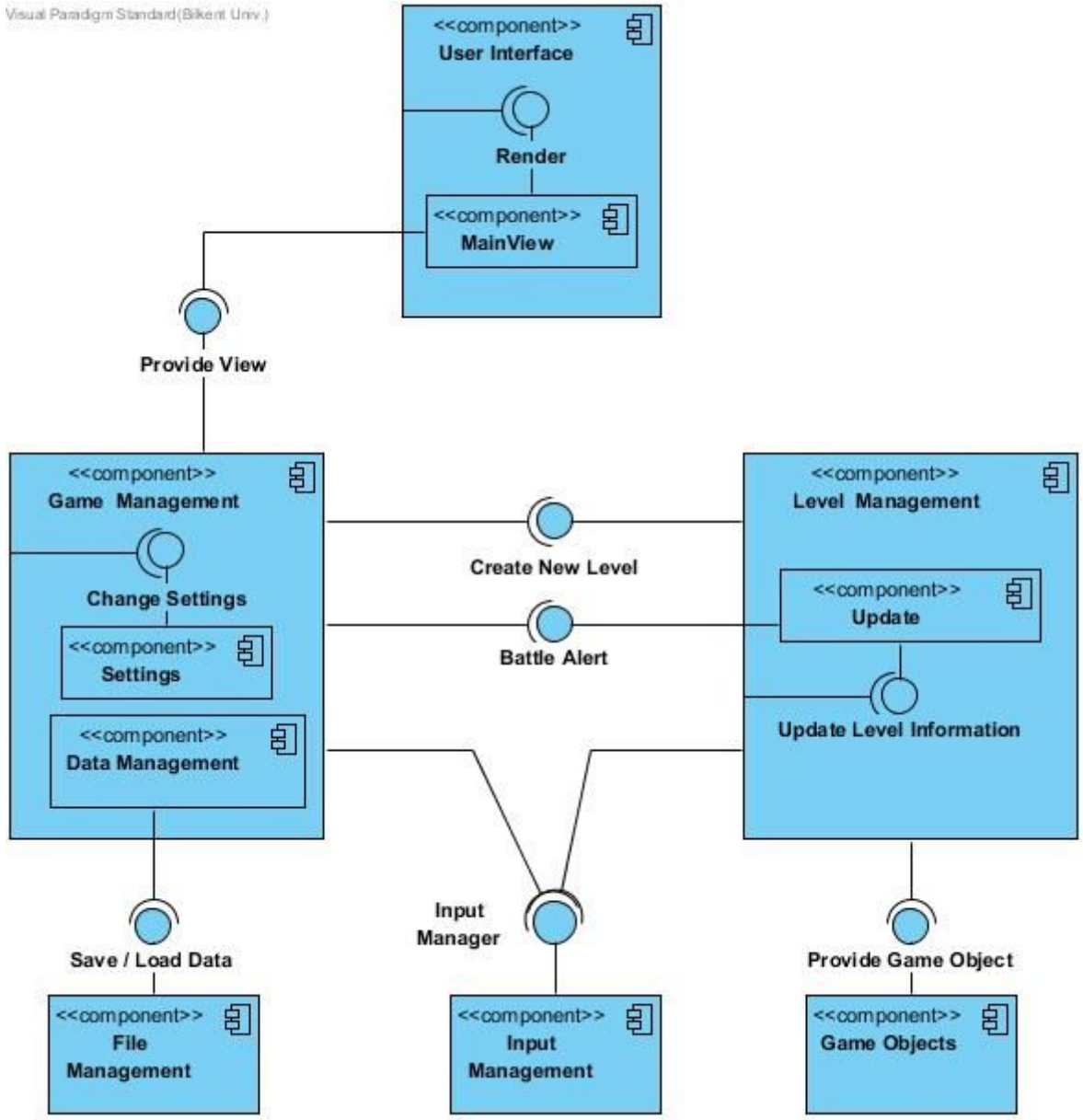
Java execution environment is termed as the Java Runtime Environment (JRE). All systems need Java Runtime Environment (JRE) in order to execute the projects which are developed in Java.

- **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



2.2. Hardware Software Mapping

Our game will require the Java Runtime Environment to be executed since it is developed by Java. We will require a keyboard while playing our game. There will be standard 2D graphics so every standard computer will be able to work our game.

2.3. Persistent Data Management

We will use text files as database part in our game. We will load all the necessary files on the memory. Users will be able to upload their personal images instead of our enemy characters' pictures. Another images will be .jpg or .png in system files.

2.4. Access Control and Security

Left 4 F will be a setup game. Therefore, it will not be required internet connection. We will have a single user, the player, using the game. We do not need an Access Matrix to represent the actors of the system because we have a single actor. We do not have any security option like that having user account and password. The user will be able to access his load game directly.

2.5. Boundary Conditions

Initialization: Left 4 F will not be included in some file which will be opened as .exe but it will be an application for the users to be downloaded and clicked on so that they can start the game.

Termination: Left 4 F can be closed by clicking "x" button in main menu. If player wants to quit during game play, system provides a pause menu by which player can return to main menu and perform it.

Error: Left 4 F will have inner game errors that will be seed by the player. There will be any error which cause data loss. It means that if any problem occurs while playing. Left 4 F, resources and levels that belong to the player will not get lost in database. First inner game error is when you want to invest but you are being under attack, you cannot invest and you will get warning message that is "You cannot invest during the battle!".

3. Subsystem Services

3.1 Interface Layer

User Interface component is responsible for displaying the present view to the user. Present view is provided by Game Management component.

3.2 Management Layer

Game Management component is responsible for selecting which menu to show, changing the settings, creating a new level and saving or loading a game. This component will take the signal from *Input Management* class or Battle Alert from *Level Management* and if the new chosen menu is not the one shown at the time, *Game Management* will provide a new view menu to *User Interface*.

Game Management has 2 sub managers. *Settings* component is responsible for changing the settings which user decides. *Data Management* component saves a game according to its implementation or loads a game if the user asks to.

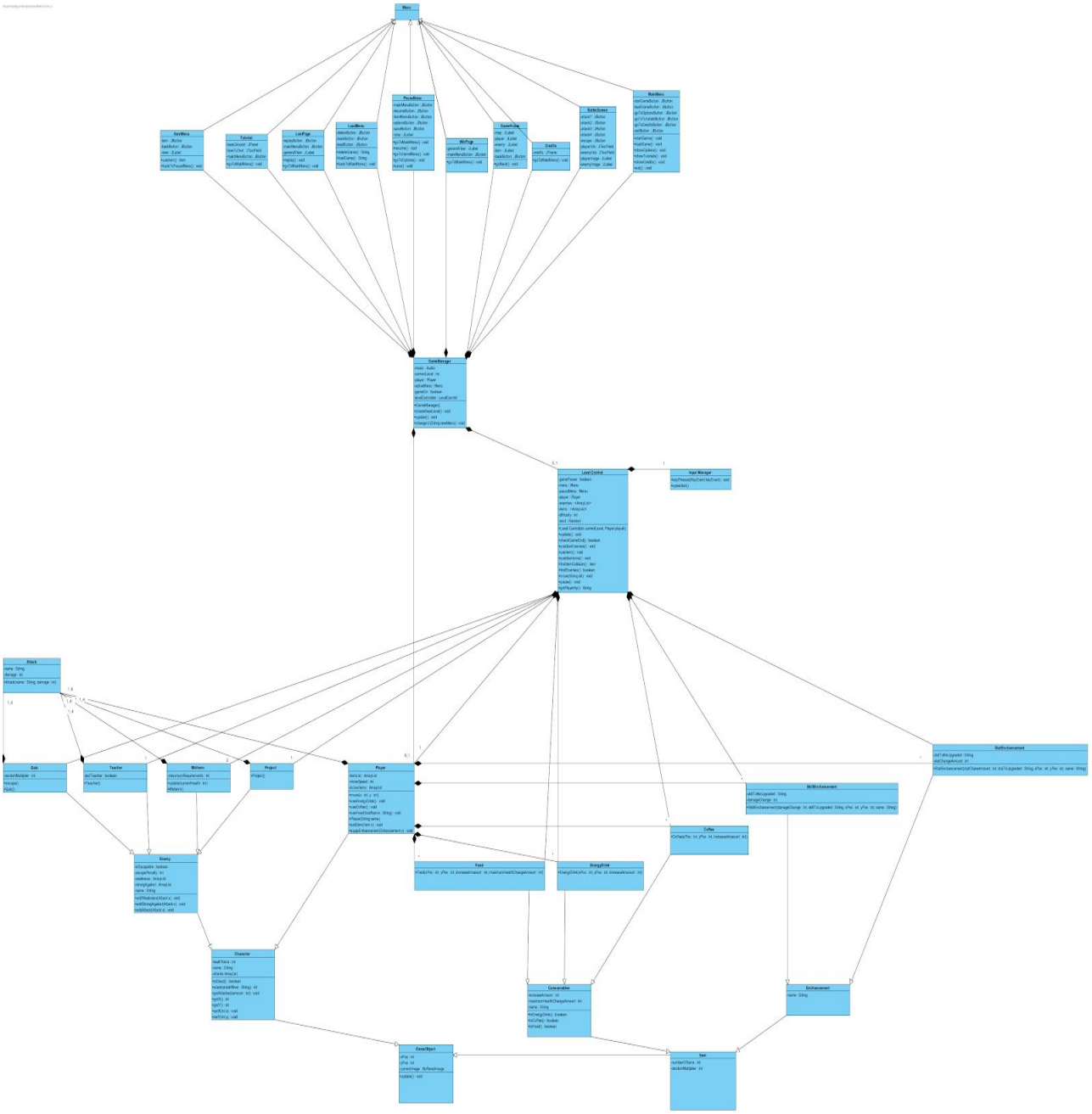
Level Management component is responsible for handling the current played level which is given by *Game Management*. *Update* component updates the information of the characters and game objects according to user's movements or actions inside a game loop. It checks for any collisions. If it is an object, *Level Management* handles it. If it is an enemy, it lets the *Game Management* component to know if there is a battle or not via Battle Alert, so that *Game Management* component can change the view and *Level Management* can control the battle scene.

3.3 Storage Layer

Storage layer consists of three components. File Management component is responsible for saving the current data to the save file and loading it back up when requested. Input Management is responsible for alerting Game Management components for any input coming from the user. Lastly, Game Objects component is responsible for providing game object classes to the Game Management components.

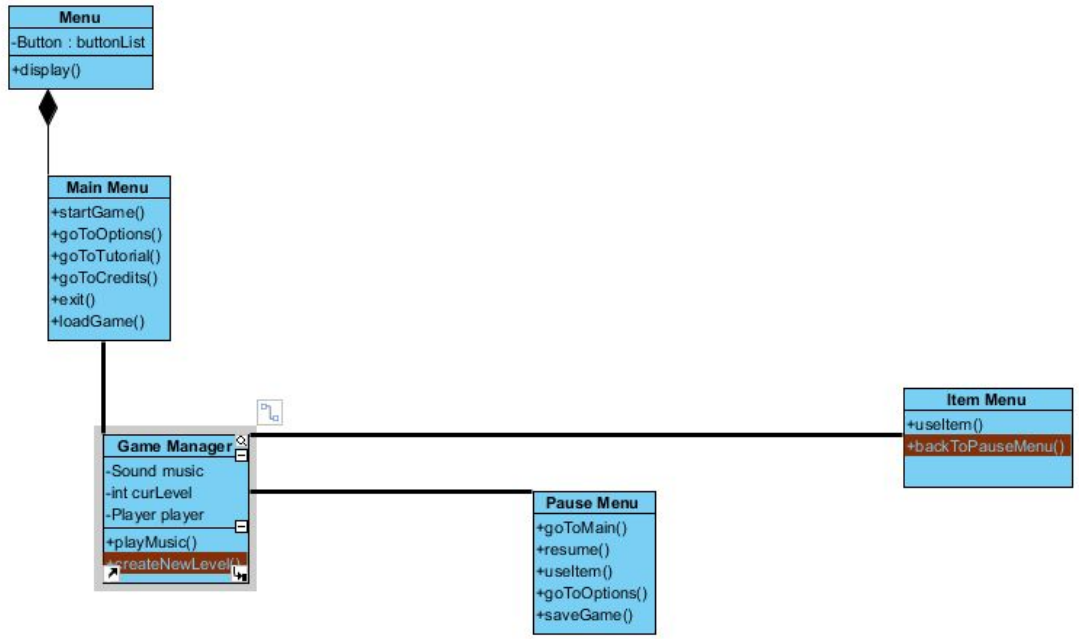
4. Low-Level Design

4.1 Final Object Design



4.2 Layers

4.2.1 Interface Layer

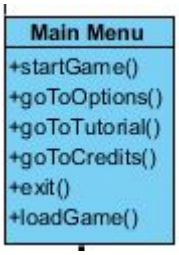


This diagram represents relationships between menus in our design. Game Manager class is associated other menu classes.

Menu Class

Menu class includes button attribute which provides us buttonList. This attribute include all buttons in our game that will provide to control.

Main Menu Class

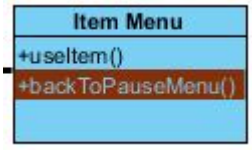


Main Menu class includes the attributes which are startGame(), goToOptions(), goToCredits(), goToTutorials(), exit(), loadGame().

startGame() method provides to start game.
goToOptions() method provides to go to options menu.
goToTutorial() method provides to go to tutorials.
exit() method provides to exit game.
loadGame() provides to load game.

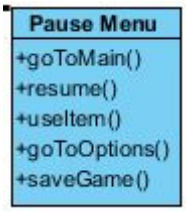
We have explained Game manager class.

Item Menu Class



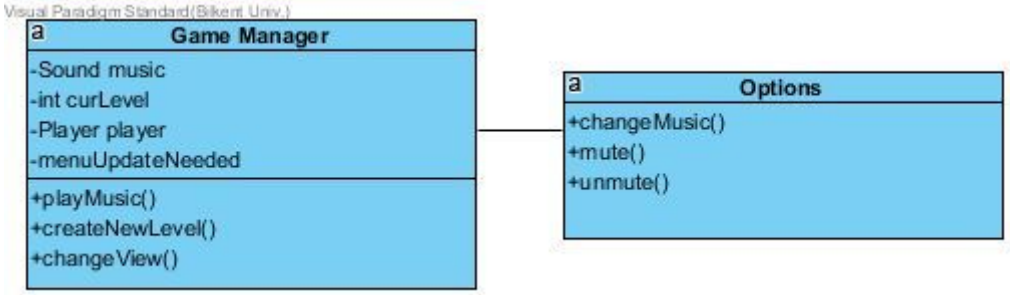
Item menu class includes useItem() and backToPauseMenu() methods. useItem() method provides to use items that belongs to the player. backToPauseMenu() method provides to back to pause menu.

Pause Menu Class



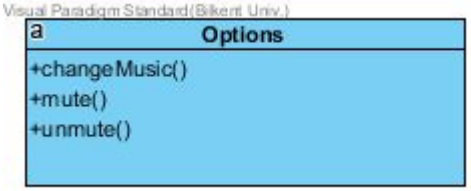
Pause Menu class includes goToMain(), resume(), useItem(), goToOptions(), and saveGame() methods.
goToMain() method provides to go to main menu while playing.
resume() method provides to resume while player in pause menu.
goToOptions() method provides to go to options menu.
saveGame() method provides to save game if player wants to save his/her game.

4.2.2 Game Management Layer



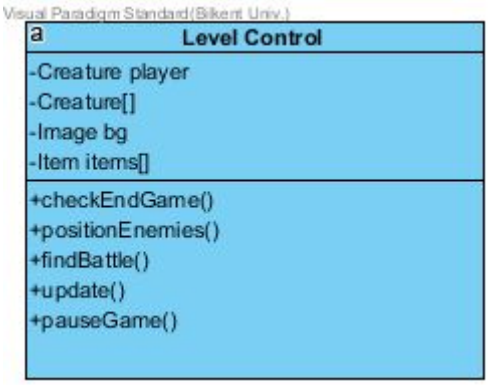
Game Manager Class

"GameManager" will be instantiated when the game starts. It is responsible for changing the current view, holding the current settings and creating a new level.



Options Class

Options class is responsible for changing the current settings which are hold inside the Game Manager instance.

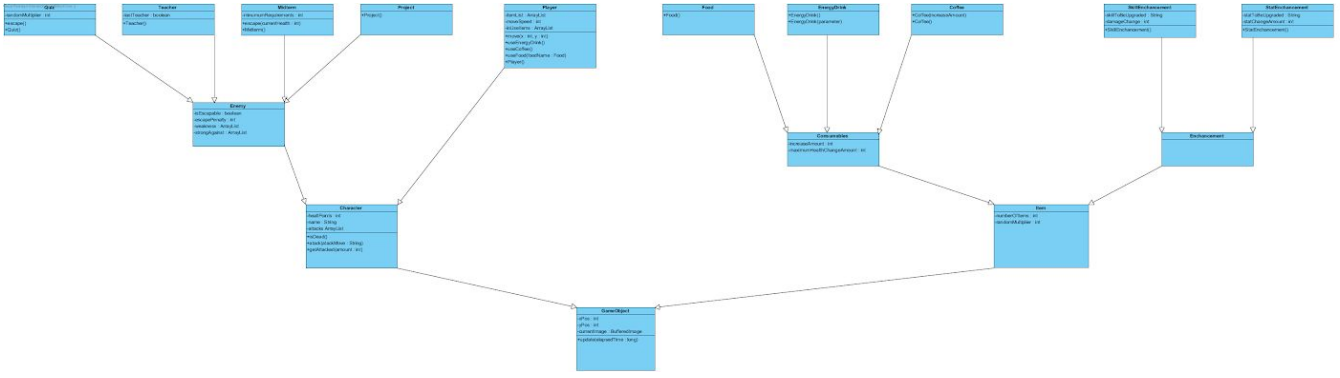


Level Control Class

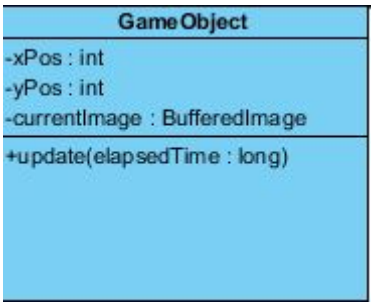
Level Control Class is initialized by the Game Manager when a level is created. It hold the player and the enemies and collectibles as its attributes.checkEndGame(), positionEnemies(), findBattle() and update() functions are required for the in-game loop implementation. pauseGame() function pauses the game and signals the Game Manager to open up the Pause Menu.

4.2.3 Data Management Layer

“Game Objects Subsystem” declares the objects to shown in screen while the game is actually running. It has many crucial objects such as “Player”, “Enemy” and “Item” with their subclasses.

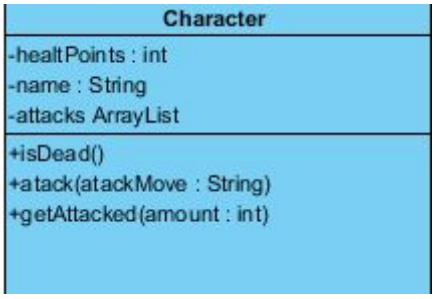


GameObject Class



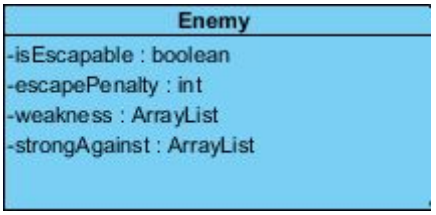
"GameObject" will be instantiated after user decides to play the game. All fundamental objects use the "GameObject" abstract class as a parent class since they all need to have position information such as "xPos" and "yPos". "update()" is defined as an abstract 18 operation since all objects need to act differently to this operation. We require to draw all of the objects on to the screen therefore "GameObject" class has an image that will be drawn by the "MainView" class.

Character Class



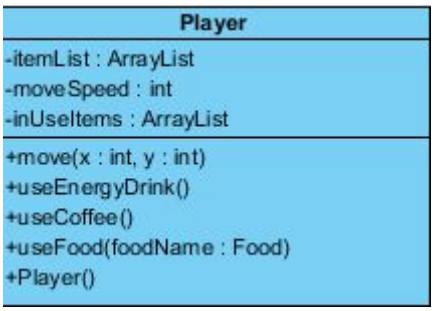
"Character" class has additional attributes to GameObject class's attributes, such as "healthPoints" since all characters in the has to have a health point stat in order to battle with each other. Additional operations such as "isDead" are necessary for battles to be start and end.

Enemy Class



"Enemy" class has additional attributes to Enemy class's attributes, such as "escapePenalty" since the interactions between player and enemies are special for each enemy.

Player Class



"Player" class has special attributes such as "inUseItems" which holds the enhancements in usage. Special operations such as "useCoffee" are also special to "Player" class since no other object on the game could get use of coffees.

4.3 Description of the Interactions between Classes According to the Use Cases

For all of the use cases, "MainMenu" class that has the main method will be instantiated. It constructs all the JButtons and JPanels for the main menu shown in the beginning of the program. The "GameManager" object will be created even before the game starts and wait for the instructions from the "MainMenu".

LoadGame: The "Load Game" button in the "MainMenu" will go to *Game Manager* instance and call LoadGame() function. This function will look for a "savedgame.txt" by *File Manager*. If no such file exists, there will be an error showed. If it will return true, *Game Manager* will take the file and read it. It will create a new "Level Manager" object with a new Player object with the data from the file. Also it will take the last "difficulty" attribute so that it can create a new level accordingly. After the level is created, *Game Manager* will set the MainView to "Level" and destroy "MainMenu". Play Game use case will take over from this point.

Tutorials: The "Tutorials" button in the "MainMenu" will go to *Game Manager* instance and call tutorials() function. This will initialize a "Tutorials" object that will display the tutorials. Then loadTutorialsDoc() of the "GameManager" will be called, which calls loadTuorialsDoc() of the "FileManager" that gets the tutorials document from the file. Consequently, retuned value will be shown in the JPanel of the "Tutorials" class and "MainMenu" will be destroyed.

Control Settings: The “Options” button in the "MainMenu" will go to *Game Manager* instance and call options() function. This will display the currently saved settings, destroy the “MainMenu” and other choices that user can select. Then it calls the loadGameSettings() method of the "GameManager". Lastly, "GameManager" calls getSettings() to grab the saved settings from the file. “Options” object will use to these returned values to construct its user interface. After the change button is pressed, "*Game Manager*" updates music. At the end of the program, when the user clicks the exit game button, all settings will be written back to the file system using saveGameSettings() of "GameManager".

Play Game: This is the case that user wants to start a new game with the game data of level 1. Then "MainMenu" calls the startGame() method of the "GameManager" with the isNew parameter set to true. Afterwards “GameManager” calls “CharacterSelectionMenu” class and let user pick a character. After user picks a character, loadLevel() method is called in the "GameManager" with the level parameter set to 1. This method calls the getLevelData() function of the "FileManager" with the same level parameter. After the game data is returned to the "GameManager", it initializes the “LevelManager” object with this data. Consequently, the main game loop starts in the "LevelManager". In this loop, first upDateObjects() will be called to update “GameObjects”’s positions. Then the checkBattle() method is called to handle any battles starting with random enemies and checkAction() method is called to handle any interaction with items and predefined enemies. If there is a battle “BattleScreen” object is initialized with given parameters or if there is an item found addItem() function of the “Player” class will be called to add the item. After all the checks and modifications are done, drawObjects() method in the "GameManager" that draws the player, items and enemies in the map that are returned by the “LevelManager” class. Lastly, isGameOver() is called to decide whether to continue or break the loop.

Pause Game: In this use case isPaused attribute of the "GameManager" will be set to true, which will pause the inner game loop and wait for the user to continue.

5. References

- [1] http://www.klabs.org/history/history_docs/sp-8070/ch4/4p1_design_tradeoffs.htm
- [2] http://www.cse.chalmers.se/research/group/idc/ituniv/kurser/08/hcd/literatures/group%202%20ElviraKim.pdf
- [3] http://www.webopedia.com/TERM/C/cross_platform.html
- *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition*, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.(POWERPOINT SLIDES)
- *Principles of Object-Oriented Software Development*, by Anton Eliens, Addison-Wesley, 1995, ISBN: 0-201-62444-3.
- Visual Paradigm Enterprise.