1. **Introduction**
   1. **Purpose of the system**

The system of Nightmare Dungeon’s purpose is to make users have fun from the game that presents a different experience in every run because of the random elements of the rogue-like genre such as different item and minion spawns. The gameplay and the controls are inspired by BoE so they are easy to learn so the player can enjoy the game with little effort invested beforehand. Although the game is easy to learn, the game can get challenging so the player can find excitement about the game.

* 1. **Design Goals**
* **Efficiency:** The controls are going to be responsive to keep up with the pace of the game. The game will run smoothly in order to provide a stable fps and enjoyable gameplay.
* **Reliability:** The system should always give the user the promised service so we will make the game such that it will not crash or have game breaking bugs. To achieve this kind of reliability we will frequently test the system and debug the code.
* **Good Documentation:** We will have good documentation of our system so that it is easy to work on, traceable and maintainable. We will do this by saving our drafts and work progress as well as the final product so that we can follow through the development process in its every step as we need it.
* **Extensibility:** The system will be easy to extend so that new features and functionalities will be easy to add. We will accomplish this by following object oriented design and its methods such as abstraction from start to end. By doing so, we will be able to add or modify a class without corrupting the functionalities of other classes or disturbing other elements of the design.
* **Usability:** The game will be user-friendly in its design. A new player will be able to start the game and jump into the game directly with a simple UI that isn’t clustered with confusing buttons and that guides the user around.

**Design Trade-Offs**

* **Efficiency vs. Portability:** Our game needs to be implemented efficiently in order to give the enjoyable experience we intend to give to the user while playing the game since the game is fast paced. We can afford to sacrifice portability since the game is intended to be a desktop/windows game, so we don’t need to think about portability and how to port the game to other systems such as game consoles or other operating systems.
* **Rapid Development vs. Functionality:** Since we have to keep up with the deadlines, we are sacrificing possible functionalities for rapid development. Our game could have been more complex in the sense of functionality such as different mechanisms like jumping over obstacles or shopkeepers where the user could buy upgrades or secret rooms that could be reached by completing different tasks.
* **Cost vs. Robustness:** Our team is consisted of 4 people so the cost is very low. By having few people working on the project, robustness of it is sacrificed so even if we try hard, there will most probably be some bugs and unintended features in the game.
  1. **Definitions, acronyms and abbreviations**

**BoE [1]:** BoE is an acronym used in the Binding of Isaac community to refer to Binding of Isaac.

**Fps [2]:** Fps is an acronym for frame per second which is the count of frames that the screen displays in a second.

**UI [3]:** UI is an acronym for user interface which is a design that allows the user and the system to interact.

**Rogue-like [4]:** Roguelike is a term used to describe a subgenre of role-playing video games that are characterized by a dungeon crawl through procedurally generated game levels, turn-based gameplay, tile-based graphics, and permanent death of the player-character.

* 1. **References**

**[1]** <https://en.wikipedia.org/wiki/The_Binding_of_Isaac_(video_game)>

**[2]** <https://en.wikipedia.org/wiki/FPS>

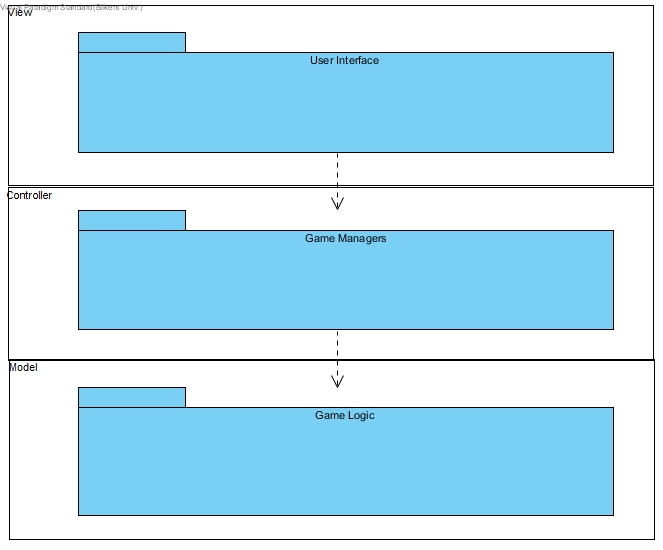
**[3]** <https://en.wikipedia.org/wiki/User_interface>

**[4]** <https://en.wikipedia.org/wiki/Roguelike>

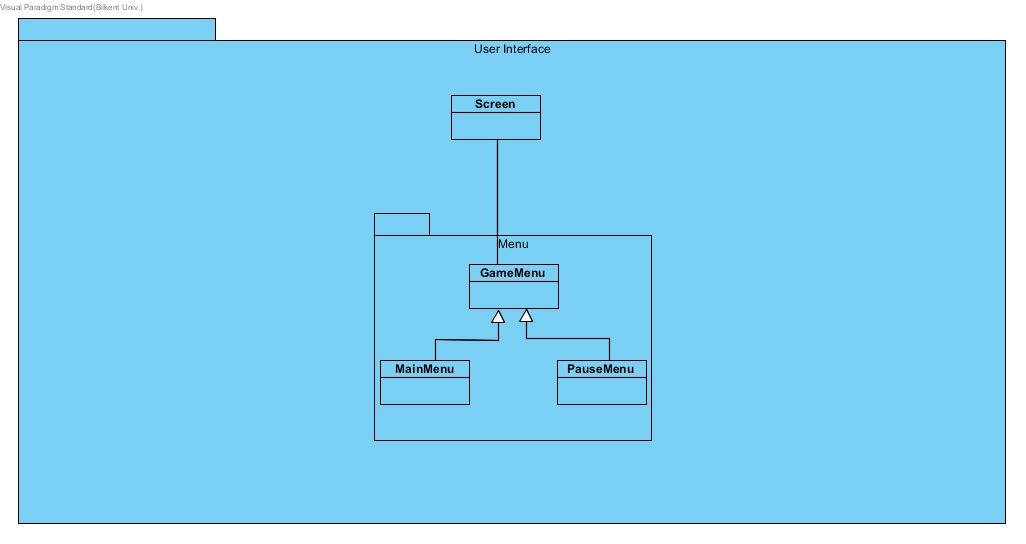
1. **Software Architecture**
   1. **Subsystem Decomposition**

As for the architecture, we chose to go with the three-tier architecture for the design of the system since most games are heavily dependent on game logic, user interface and the interaction between them so we thought that these three main elements that make up most of the games can be categorized into three layers. These three layers will provide MVC design by acting as view (User Interface), controller (Game Managers) and model (Game Logic). View is the top layer and it will act as an interaction between the user and the system and it will depend on controller. Controller is the middle layer and it will act as a connection between view and the controller. Model is the bottom layer and it will handle the data flow of the game. By following MVC we are aiming to create a system with high coherence and low coupling. We used opaque layering as a layering method.

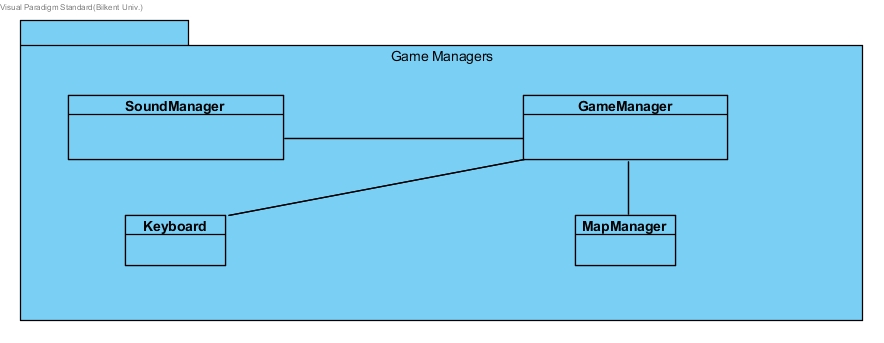
Sound Manager main menuye baglı olunca packageerde sorun cıkıyor ve benzeri. MapManager ve map, Screen(silinecek)



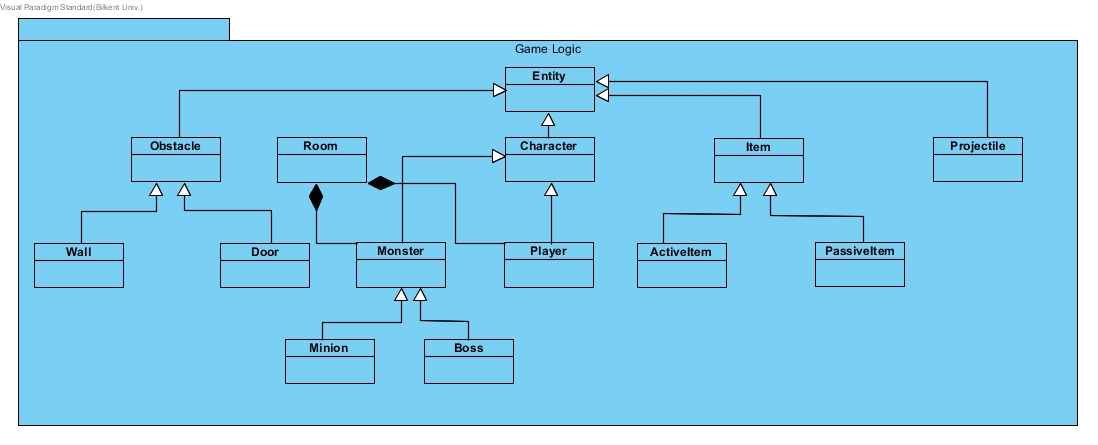
High Level View of Subsystem Decomposition



Contents of View



Contents of Controller



Contents of Model

* 1. **Hardware / Software Mapping**

As software configuration, we will implement the game in Java.

As hardware configuration, the user will need a keyboard for controlling the character and writing his/her name on the high scores list and a computer screen for the user to interact with the system. A PC with Microsoft Windows and Java compiler will be able to run the game so the system requirements are minimal. We will use local files such as .txt files to store game data such as high scores list so the user won’t need an internet connection to play the game or submit his/her high score to the top 10 list.

* 1. **Persistent Data Management**

We will store our game’s data locally so the user won’t need internet and we won’t

need to build a database system. We will store the high scores in a single file and get the data from there every time player opens the game. We will store the character models in sprite-sheets as sprites so the game will get the pixel data from the sprite-sheet file when needed. We will also have sound files such as .mp3’s for the background music and the sound effects.

* 1. **Access Control and Security**

Nightmare Dungeon doesn’t have any interaction with internet connection and it

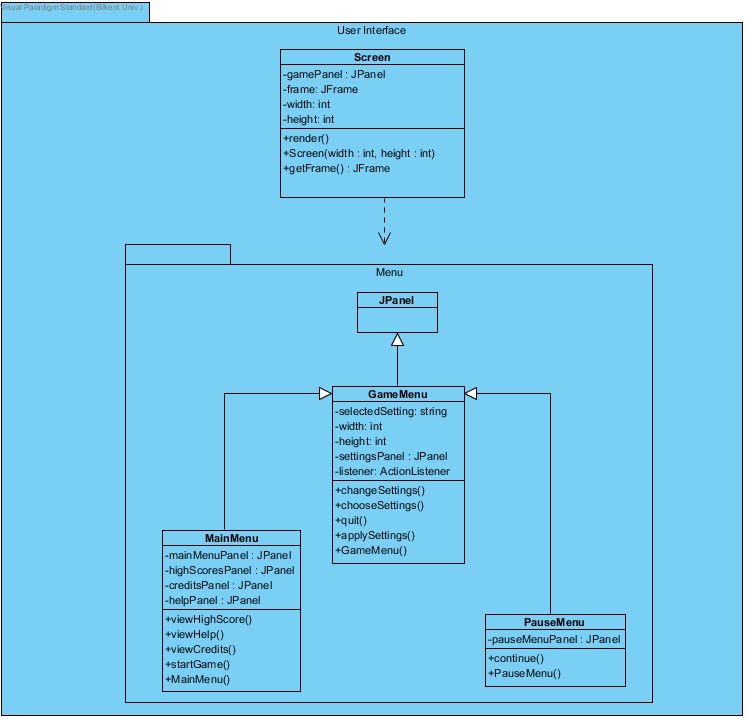
doesn’t have a database that holds user information. The only user information the game is going to need is a name when one of the top 10 high scores is beaten to replace the position whose score was beaten by the current player. At this point the player can use any alias he/she wants. Since this is the case, players won’t need an account so the system doesn’t have any access control or security issues.

* 1. **Boundary Conditions**

The maps will have boundaries on the edges, the implementation will include cases where the player tries to go beyond these boundaries and the game will not allow going beyond the boundaries. We will still catch exceptions about boundaries even if we have checks controlling the boundaries of the map to avoid program crashes. Another boundary case is the situation where all of the player’s lives are gone. When this happens, the game will be over and it will start again. Another boundary case is when the player manages to kill the last boss. When this happens, the game will be over and the players score will be displayed, if the player beats one of the top 10 scores previously saved, then the game will request a name from the user to save it to the top 10 board along with the score of the player.

1. **Subsystem Services**

**User Interface Subsystem**

The interaction between the user and the system is provided by the User Interface subsystem. This interaction is done with the use of graphical components. The User Interface subsystem consists of 4 classes which are GameMenu, MainMenu, PauseMenu and Screen. 

**Screen Class**

Screen class handles graphics objects such as JFrames and JPanels. Every component related to graphics will be put over the JFrame created by the Screen class. The main JPanel of the game called gamePanel will be created and as data changes in the Game Logic section, the screen class will render() to display these changes to the user.

**JPanel Class**

The JPanel class will act as a parent to all the other classes in the Menu package.

**GameMenu Class**

The GameMenu class is a child of the JPanel class and the parent of MainMenu and PauseMenu classes. GameMenu class has the common funclionalities of the MainMenu and PauseMenu classes such as quit(), changeSettings(), selectedSetting, etc.

**MainMenu Class**

MainMenu class is the first class to be created when the game is opened first with the use of the constructor MainMenu(). The MainMenu class presents 4 choices unique to the MainMenu class when the user is in the main menu. These choices are: viewHelp(), viewHighScore(), viewCredits() and startGame(). startGame() method initiates the game. MainMenu class also has changeSettings(), chooseSettings(), quit() and applySetting() methods that it shares with the PauseMenu class. MainMenu class also holds JPanels of the options the user can choose while he/she is in the main menu.

**PauseMenu Class**

PauseMenu class has the common methods mentioned in the MainMenu class explanation above. It also has a unique continue() method that allows the user to exit from the pause menu and continue playing the game.