3. Subsystem Services

3.1. Design Patterns

Facade Design Pattern (Encapsulating Subsystems)

Facade Design Pattern is a structural design pattern that aims to reduce coupling between a set of related classes and the rest of the system. To do that, a single Facade class implements a high level class for a subsystem by invoking the methods of lower level classes. Facade provides a shield for caller so that it does not access the lower level classes directly. (BRUEGGE & DUTOIT, 2010)

In this project, Facade class is gameEngine which can be seen under Game Algorithm / Game Visual Subsystem. gameEngine communicates and associate other lower level classes.

3.2. User Interface Subsystem

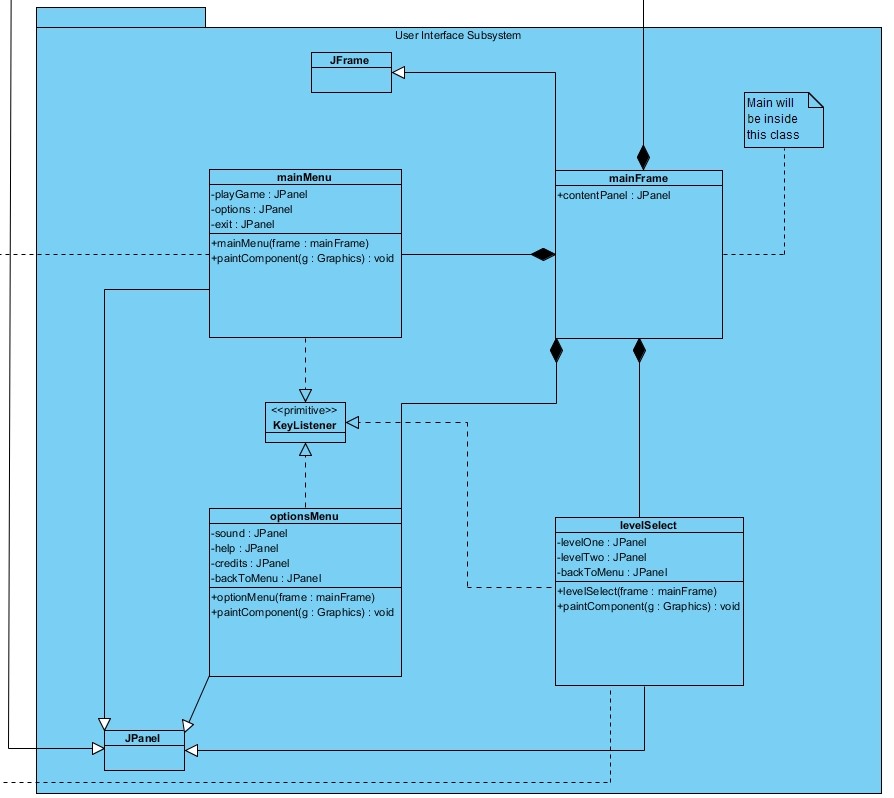


Figure 3.2: User Interface Subsystem

User Interface Subsystem is consisted of 4 different classes which provides graphical components for menu screens. In the current iteration of the design gameEngine handles the game screen. It also implements JPanel; however, it also interact with other classes. Therefore it will be analyzed under Game Algorithm / Game Visual Subsystem.

3.2.1. mainFrame Class

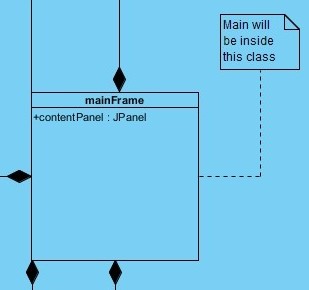


Figure 3.2.1: mainFrame Class

Attributes

Public JPanel contentPanel: This attribute will initilize the border of the content which mainFrame will take from optionMenu, levelSelect, mainMenu and gameEngine.

Constructors

Public mainFrame: Initialize the bound, layout and tittle of the window.

Design Choices

The main method of the project will be called by mainFrame class.

This class will extends JFrame.

3.2.2. mainMenu Class

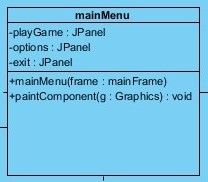


Figure 3.2.2: mainMenu Class

Attributes

Private JPanel playGame: This attribute will be work like a button. When the user presses the appropriate keylistener, it will call the levelSelect class.

Private JPanel option: This attribute will be work like a button. When the user presses the appropriate keylistener, it will call the optionMenu class.

Private JPanel exit: This attribute will be work like a button. When the user presses the appropriate keylistener, it close the program.

Constructors

Public mainMenu: Load the panels and images.

Methods

Public paintComponent: It will draw the screen by using Java Graphics2D library

Design Choices

For Donkey Kong type of arcade games, client usually uses keyboard listeners to move around selections in the menu. Thus, JPanels will be used instead of JButtons. The panel will glow to indicate that user is on that segment of selection.

This class will extends JPanel and implements KeyListener.

3.2.3. optionMenu Class

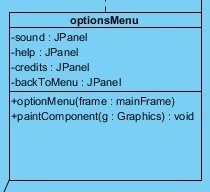


Figure 3.2.3: optionMenu Class

Attributes

Private JPanel sound: This attribute will be work like a button. When the user presses the appropriate keylistener, it will mute the sound effects. Therefore it will glow according to current selection.

Private JPanel help: This attribute will be work like a button. When the user presses the appropriate keylistener, it will pop up another panel to show how to control the Jumpman.

Private JPanel credits: This attribute will be work like a button. When the user presses the appropriate keylistener, it will pop up another window to show the credits with close option.

Private JPanel backToMenu: This attribute will be work like a button. When the user presses the appropriate keylistener, it take you back to mainMenu class.

Constructors

Public optionMenu: Load the panels and images.

Methods

Public paintComponent: It will draw the screen by using Java Graphics2D library

Design Choices

This class will extends JPanel and implements KeyListener.

3.2.4. levelSelect Class

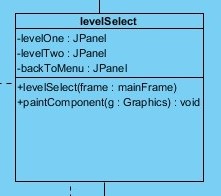


Figure 3.2.4: levelSelect Class

Attributes

Private JPanel levelOne: This attribute will be work like a button. When the user presses the appropriate keylistener, it will call the gameEngine class and load proper level.

Private JPanel levelTwo: This attribute will be work like a button. When the user presses the appropriate keylistener, it will call the gameEngine class and load proper level.

Private JPanel backToMenu: This attribute will be work like a button. When the user presses the appropriate keylistener, it take you back to mainMenu class.

Constructors

Public levelSelect: Load the panels and images.

Methods

Public paintComponent: It will draw the screen by using Java Graphics2D library

Design Choices

Number of the level panels depends on how many level the design team is going to do

This class will extends JPanel and implements KeyListener.

3.3. Input Management Subsystem

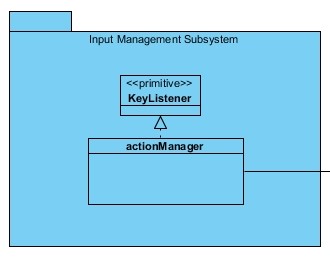


Figure 3.3: Input Management Subsystem

This part is still in progress. Require more research and more thinking. actionManager class aims to make a separate class for keylisteners during gameplay.

3.4. File Management Subsystem

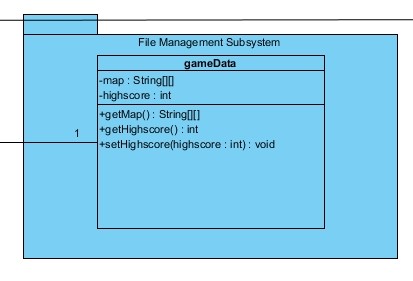


Figure 3.4: File Management Subsystem

3.4.1. gameData Class

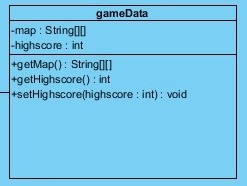


Figure 3.4.1: gameData Class

Attributes

Private 2D String Array map: This attribute will contain the map data which gameData will read from txt files.

Private int highscore: This attribute will contain the highest score as integer which also be readed from a txt file.

Constructors

Public gameData: Take the level as input and load the appropriate level and high score.

Methods

Public getMap: Get map information, this method will be called under gameEngine class via gameData object.

Public getHighscore: Get the highest score for that level, this method will be called under gameEngine class via gameData object.

Public setHighscore: Set the highest score via overwriting the txt file, this method will be called under gameEngine class via gameData object.

Design Choices

There is no setMap method because of the fact that levels will be design before game execution and game algorithm will be programmed according to that data.

3.5. File Management Subsystem

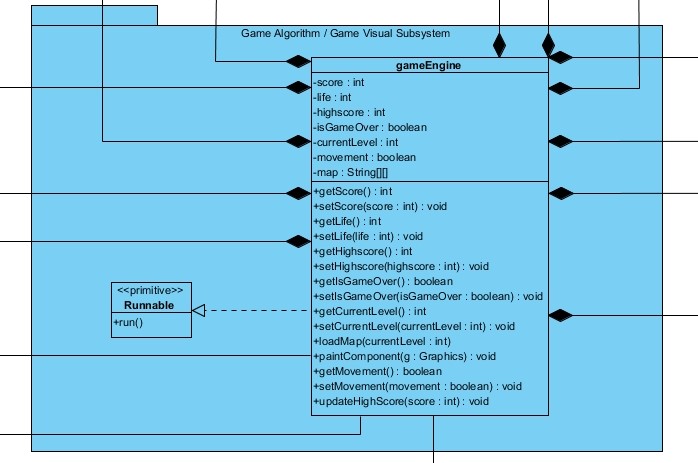


Figure 3.5: File Management Subsystem

3.5.1. gameEngine Class

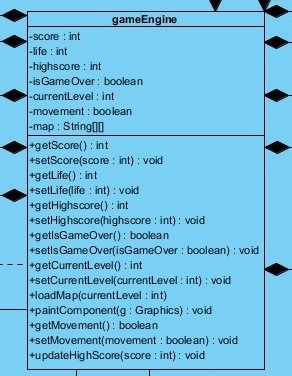


Figure 3.5.1: gameEngine Class

Attributes

Private int score: This attribute will contain the current score while game is running.

Private int life: This will indicate how many life points player has. Every level start with 3 and it will decrease every time player dies. When it reaches 0, game will end.

Private int highscore: This attribute will contain the highest score of the current level. It will be loaded by gameData object which will be created during runtime. When isGameOver true, it will be updated depend on the score.

Private Boolean isGameOver: This attribute is created for future implementation purposes. It help us to implement methods and there could be a pause option during gameplay. Thus, it currently has no purpose other than making the implementation easy.

Private int currentLevel: This attribute will be selected by user under levelSelect class.

Private Boolean movement: This attribute is similar to isGameOver. To make the game different, there will be some power-ups in the game. One of them is going to decrease the speed of objects other than Jumpman so that player will be able to jump and kite easier. Basically it indicates the game speed.

Private 2D String Array map: This attribute will contain the map components and depends on the value inside gameEngine will draw the map. It will be loaded by gameData object which will be created during runtime.

Constructors

Public gameEngine: Take the level as input, draw the appropriate level and render it during runtime.

Methods

Public int getScore: Get the score during run time.

Public void setScore(int score): Update the score during run time.

Public int getLife(): To check the life during run time to finish the game.

Public void setLife(int life): There is another power-up and it increase your life count by 1.

Public int getHighScore(): Load the highest score by using gameData object before game start executing.

Public Boolean getIsGameOver(): Determine the game is finished or not.

Public void setIsGameOver(Boolean isGameOver): Every time player dies, game algorithm will check if it is 0 or not then call this function with true.

Public int getCurrentLevel(): Get the currentLevel to update highscore after game is finished.

Public void setCurrentLevel(int currentLevel): Set the currentLevel before execution of the game to reach it later.

Public void loadMap(int currentLevel): Will do the interaction with gameData class.

Public void paintComponent(Graphics g): It will draw the screen by using Java Graphics2D library and render it inside run() method.

Public Boolean getMovement(): Check whether or not player reached movement power-up.

Public void setMovement(Boolean movement): If player object and movement power-up collides, slow down the game.

Public void updateHighScore(int score): After game isGameOver is become true, game algorithm will check if the score is higher than highscore or not. Then if score is higher, this method will be called and next time the game executed, the new score will be shown.

Design Choices

This class will extends JPanel to draw the game screen and implements runnable to run the game by using run operation by overriding it.

Gameplay algorithms will be initiate under run() method.

We are planning to separate this subsystem into 2 pieces named as follows: Game Algorithm Subsystem and Game Visual Subsystem; however, we need to think over how to interact them.

3.6. Game Entities Subsystem

3.7. Overall Design

# References

BRUEGGE, B., & DUTOIT, A. H. (2010). *Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition.* Prentice-Hall.