Bilkent University

CS 319

Object-Oriented Software Engineering

Fall 2015

Design Report

Project Name: Gold Miner

Group #2

Murat Selman Gündoğmuş	**21000213**
Şükrü Efe Keleşoğlu	**21001407**
Ezgi Nostar	**21101633**
Ahmet Said Sayarlıoğlu	**21101612**

Contents

1. Introduction

We decided to design and implement a game called Gold Miner. It is a basic flash game that users try to collect gold and hidden treasures in limited time. The main purpose of the game is to score maximum point by collecting maximum amount of gold, jewelry and power ups. While doing this, users have to escape the randomly distributed bombs and dynamites to gain more points. Bombs and dynamites are a danger for gold and other collectibles items by destroying the collectible items in the area where they are belong.

In addition to that, the game consists of a number of different levels which are following another. User can play next level by reaching the target of a certain level.

The game that we inspire by is following.

http://www.kraloyun.com/Oyun/Hazine-avcisi

We plan to develop our game with different features. In Gold Miner, we are going to use new features including new power ups, new maps and challenges.

The game will be a desktop application and will be controlled by a keyboard's up and down buttons.

This report contains two parts indeed. Requirement analysis part contains an overview of the game, describes functional and non-functional requirements, constraints and scenarios, presents use case models and user interface. Analysis part contains object model including class diagrams and dynamic model including state chart and sequence diagram.

2. Requirement Analysis

2.1 Overview

Gold Miner is very easy to play and funny. Users just need to click down arrow on keyboard to activate the hook which provides to collect gold and hidden treasures. The main object in this game is the hook which has two states: rotate and rewind. Rotating provides the hook is rotating from left to right at the top of the screen. Rewinding provides the hook returns to its initial position in a straight path and then changes the state to rotate by allowing the player to shoot again. Every time hook is thrown, hook can collect the gold or other items depending on location of the gold and path that hook is thrown. Every time player collects a new item, user can increase the cash according to point of a collectible item to reach the target. If the player reaches the goals by hitting shot without missing any time, player can see the new map by passing the new levels.

2.1.1 Gameplay

The player will need keyboard to play the game. Players have to use the hook and reel to mine gold and other treasures out of the land. When the hook swing back and forth, user press the down button to activate and collect items. Once the hook has grabbed something, it will begin to reel it up. Heavy objects like rocks and large pieces of gold will be more time to pull. PowerUp Bag contains random amounts of cash a strength power-up, or a stick of dynamite. Users have to collect the target amount of money by the end of the level if users do not meet their goal by the end of the level, it's game over. Users' cashes carries over from one level to the next.

2.1.2 Levelling

In Gold Miner, there are 10 levels which are in difficulty order. When the player passes a next level, the new map contains more bombs which destroy the collectible items. In addition to that in next levels, the target point is higher than the previous levels. In this way, user has to reach higher target point in a more dangerous map. Also, even the target point is increased, the given time is not changed and it will be fixed to 60 seconds. The

advantageous point about the leveling is that the game can add new power ups and diamond which are more valuable or cause to decrease the cash point.

Jewels are designed to their values. The different kinds of jewels will come up in each new level and it will see as in ascending value order. Jewels are stone, gold, diamond, ruby, sapphire. In addition to that, to make our game is harder, bombs will appear in higher levels. Also, PowerUp Bag and animals which blocks to collect to gold and jewelry will be appeared.

1st Level: Just stones and golds will be appearing.

2nd Level: Stones, golds and diamonds will be appearing.

3rd: All jewels and bombs will be appearing.

4rd and Higher Levels: PowerUp Bags and animals will be appearing.

2.1.3 Bonus Items

There are several bonus items in the game. Some of them are power ups and some of them are penalties.

Users can either obtain a bonus or a time penalty from the PowerUp Bag in the game. Users will also get a time penalty when they pull back the hook empty. Also the player does not know exactly what type of power-up or power-down coming when they pull the PowerUp Bag. In addition to that, some of power up in the PowerUp Bag such as lots amount of cash is rare compare to other power ups such as dynamites or power downs such us time penalty.

PowerUp Bag: It will be appeared in 4rd and higher levels and provides to reach different bonus items.

Bomb: It causes to destroy collectible items nearby. Also it blocks to the bigger golds.

Dynamite: It provides to reset user's hook when they collect the stones and unwanted items. In this way, user can collect more items without losing any time.

Time Penalty: It comes from the PowerUp Bag. Users will take 3 seconds time penalty.

Extra Time: It comes from the PowerUp Bag. Users will take extra 10 seconds.

2.1.4 Hook

The hook is the only item that user can control. User can collect the item with the hook by using keyboard down button. The hook can keep and pull the items according to their size and weight. If the hook keeps the bigger item or stones, the time for pulling increase and user can lose more time. On the contrary, the item which is keep is small such as diamond; the hook can pull this item easily without wasting a time. Also, hook can be strengthening by power ups and in this way hook speed is increased.

2.1.5 Collectible Items

In Gold Miner, there are different types of collectible items such as bomb, stones, jewelry, gold, animals and power ups and downs which contain dynamite, time, and strength. This different kind of items will come up in each new level. There are different kind of gold according to the size and points, one type of diamond, one type of animal, one type of bombs and different kind of stones according to the their size and weight.

2.2. Functional Requirements

2.2.1. Loading Stage

The game loads all elements which are required for different stages such as images, audio files and panels.

2.2.2. User Interface

User interface of the game consist of three page which are main page, side-page and game page. In main page, there is instruction panel on the bottom of the page to tell how to play the game. Another thing in main page is start button. Start button is interactive with Mouse actions. Also main page has audio file and image to make more attractive the game. Another page type is side-page. In side-page, there is a panel to tell users' target point and remaining time also it shows player the current point and level and the last thing on the side page is

the exit level button. Last page type is game page. This page has hook, and collectable items such as various size of gold and rock, diamond, random powerUP bag and bomb.

2.2.3. Start Button

Start button is interactive with Mouse actions. It directs to game page.

2.2.4. Exit Level Button

Exit level button is interactive with mouse actions. It ensures to user exit the current level and it directs to main page.

2.2.5. Time

It is visible part of the game page. It counts 60 seconds reversely at each level

2.2.6. Point

It is visible part of the game page. User points will increase when user catches the item such as gold, rock, diamond or random pouch. Also if user point is less than the target point, user fails the level and game is over.

2.2.7. Target Point

Target point is differs in each level. It increases when user passes the level.

2.2.8. Level

Each level has difficulty. When user passes the level, level difficulty will increase. Using heavy items and increasing the level target point ensure to make more difficult level regarding to previous level.

2.2.9. Hook

Hook is moving from the left to the right at the top of the game page. It is interactive with keyboard actions. When specific keyboard button is pressed, hook moves in a fix direction until it hits an item and pulls up the item

2.2.10. Items

There are five types pf item such as gold, rock, diamond, bomb and random pouch. If user catches the bomb, there will be explosion around the bomb and it destroys the item which around the bomb. Diamond is another item in the game. There is only one type of diamond, it is light and it has higher point regarding to other items. Because of its' lightness, hook pulls up easier and it takes less time. In Random pouch there is third type of item. It has various items in itself such as dynamite, strength or point. System randomly picks one of them when user catches the random pouch. Also there are rocks and golds in the game. Both of them have different weight and point. Rocks have significantly lower point than golds. Heavier rocks and golds have higher points. However, user spends to more time to pull up the item. Regarding to golds and diamonds, rocks give less points.

2.2.11. End Game

The game should end if point is less than target point or user clicks the "exit level" button during the game. In this situation, it directs to main page.

2.3. Non-functional Requirements

2.3.1. Extendibility

Game should available to update the functional requirements in the future. For instance, new levels or new items can be added. It can be change the value of items or add new obstacles.

2.3.2. Easy to Play

Game is easy to play. Children, teens and adults can play this game easily. There is no complex user interface. Also there is a description of gameplay at the main page.

2.3.3. Availability

Game will be played in most type of computers even old fashioned systems in the market.

2.3.4. Performance

Because of the limitation of time during the game, system response time has utmost importance. So the game should be high performance.

2.4. Constraints

- User should have flash player on their computer to play this game

- If developer wants to upgrade or change the game, java programming language should be used.

2.5. Scenarios

- User enters the game and sees the game instruction

- User clicks “Start Game” button and sees the target score.

- During the game, user shoots the hook with keyboard actions and collects the items

- If user catches gold, rock, diamond or random pouch, user gains points. If user catches TNT, user loses time.

- If user gain target score, user will play next level.

- If user click the “Exit Level” button or does not collect target points, the game will be end.

2.6 Use Case Models

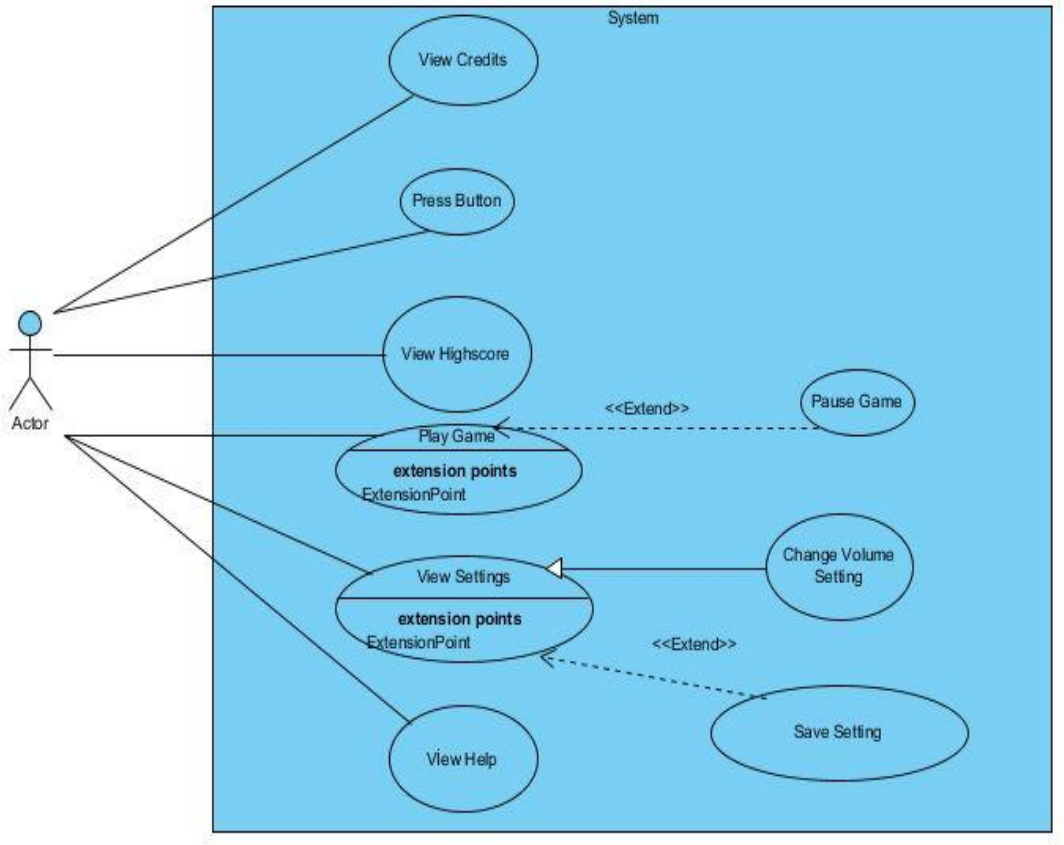


Figure 1 Use Case Diagram

Use Case Model #1

Use case name: Play Game

Participating actors: Player

Entry Condition: Player presses the play game button and starts the process.

Exit Condition:

- Player has completed all the levels successfully OR
- Player has chosen to exit the game in pause menu OR
- Player has not reached the target at the given time.

Main Flow of Events:

1. Player starts the game.
2. The system constructs the first map.
3. Player reaches the target point
4. The system generate new map.
5. Player finishes all the level.
6. The system shows the score of the player.
7. Player enters the name.
8. The system saves the score and adds the score to the high score table.
9. Players return the main menu.

Alternative Flows of Events:

- Player has not reached the target point when the time is over.(display game over message and go steps 6)
- Player has chosen to exit (go to step 9).

Use Case Model #2

Use case name: Change Settings

Participating actor: Player

Entry Condition: Player presses the settings button and starts the process

Exit Condition: Players make the changes and return the main menu

Main Flow of Event:

1. Player chosen to view settings menu.
2. Player can adjust the settings of the game.
3. The system saves the changes
4. The system returns the main menu.

Alternative flow of the event:

- Player does not change any settings(go to step 4)

Use Case Model #3

Use Case Name: View High Score

Participating Actor: Player

Entry Condition: Player presses the high score button and starts the process

Exit Condition: Players return the main menu

Main Flow of the Event:

1. Player chosen to view high score.
2. The system display high score.
3. Players return main menu

Use Case Model #4

Use Case Name: View Help

Participating Actor: Player

Entry Condition: Player presses the help button and starts the process

Exit Condition: Players return the main menu

Main Flow of the Event:

1. Player chosen to view helps and hints.
2. The system display help and hints.
3. Players return main menu.

Use Case Model #5

Use Case Name: View Credits

Participating Actor: Player

Entry Condition: Player presses the credits button and starts the process

Exit Condition: Players return the main menu

Main Flow of the Event:

1. Player chosen to view credits.
2. The system display credits.
3. Players return main menu

Use Case Model #6

Use Case Name: Pause Game

Participating Actor: Player

Entry Condition: Player presses the pause button and starts the process

Exit Condition:

- Player chose to continue to play game OR
- Player can chose to return Pmain menu OR
- Player can exit the game.

Main Flow of the Event:

1. Player presses the pause button during the game.
2. The system displays the pause menu.
3. Players continue the game

Alternative Flow of Events:

- Player chose to return main menu.
- Player directly exit the game

Use Case Model #7

Use Case Name: Press Button

Participating Actor: Player

Entry Condition: Player presses down button after the play game button is pressed and starts the process.

Exit Condition: After the hook is thrown, process is completed until the it comes initial state.

Main Flow of the Event:

1. Player presses down button during the game.
2. Hook is thrown.
3. Hook comes the initial state.

2.7. User Interface

2.7.1 Main Menu

Figure 2 Main Menu

When player open the game, s/he sees screen like this with 4 choice Play Game, high Score, Credits and Exit Game

2.7.2. Game Screen

Figure 3 Game Screen

This screen is a example of our game screen. After pushing the button of the “Play Game” at the main menu player have a screen like that. There will be several differences in our game for example there is only points instead of money and coins but it will be generally like this.

2.7.3. Help Screen

Figure 4 Help Screen

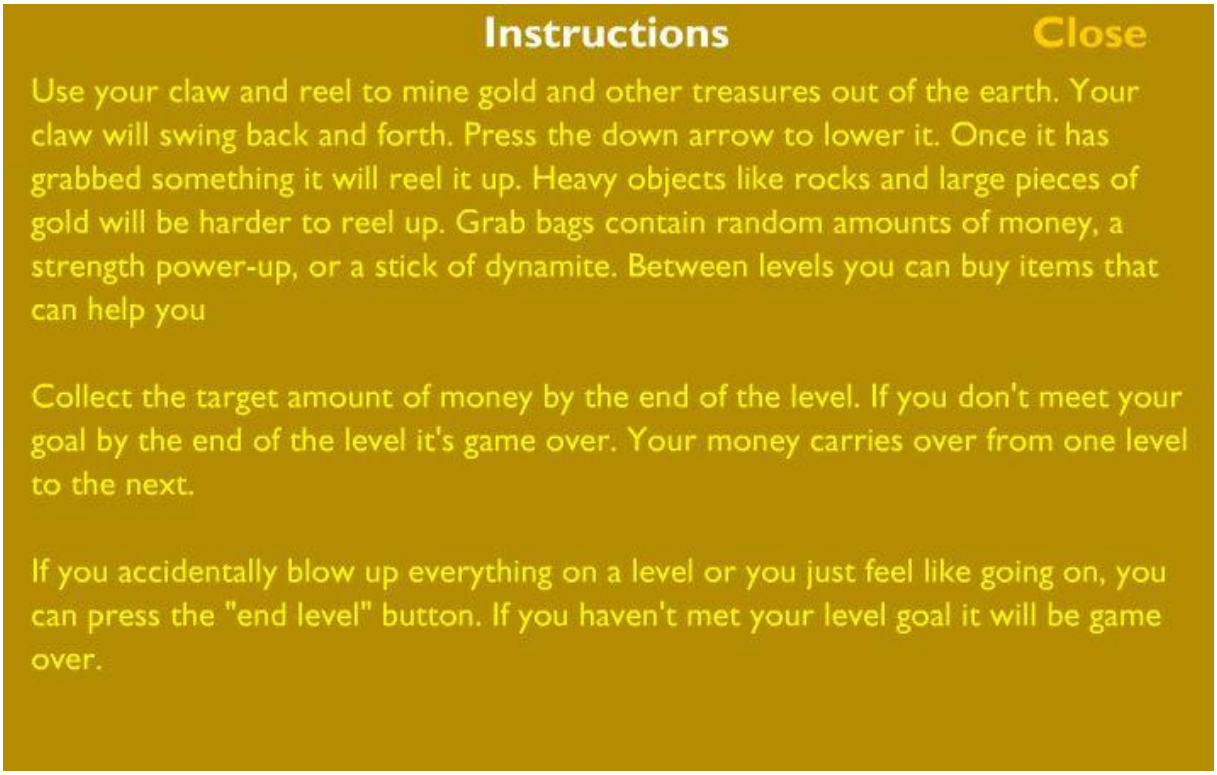


Figure 5 Instructions

The player can enter this screen by pushing the button of “Help” in the main menu. This is our help screen of the game. Player can learn instruction of the game and general information of the game.

2.7.4 Sound Setting

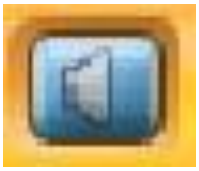


Figure 6 Sound Settings

This button is in the settings page and we can reach setting page from the main menu.

2.7.5. Collectibles

Figure 7 Gold

This is system main source of the point. In the map there are several f gold with different sizes. Size indicates that amount of point that the gold has.

Figure 8 Rock

In the game like gold there are several of rocks. But these rocks do not have any points. And like gold there is different size of rocks. Bigger ones are harder to pull and consume much more time than smaller ones

Figure 9 PowerUp Bag

In the game there are 3 powers up and they are hidden in the powerUp bags.

Figure 10 Diamond

In the game after first level there are diamonds in the game map which are smallest element in the game but the amount of point to of the diamond is highest in the all game

Figure 11 Animal

This is the last element of the collectibles. It has animal shape and has the lowest point of the game.

3. Analysis

3.1 Object Model

3.1.1 Domain Lexicon

The object model of the gold miner consists of 20 classes.

In our design Game Manager is center of the all class interaction It's manage the all user interaction by the system also it is the manager class of our system that carries out the task of the organization of all game. Our other classes explained below.

- **CollectionManager:** This class is responsible for managing the collection issues of game objects that we collect.
- **Menu:** This class provides menus of the game to the game manager
- **Input Manager:** This class handles player's inputs.
- **SoundSetting:** This class handles the sounds volume of the game
- **MapManager:** This class manages the map itself and background image of the map.
- **GameMap:** This class contains GameObject class. It also contains dynamic objects like hook
- **Sprite:** This class consist of objects images and positons so all game objects at least have one sprite
- **GameObject:** This class basically contains all entity objects of the system which include hook and collectible class. Also collectibles class has 6 child classes.
- **Hook:** This class is the main object of the game. It collects the collectibles.
- **Collectibles:** This class contains objects which can be collectible by the hook in the game. There are 6 child classes of the collectibles.
- **Bomb:** This class is the child class of the Collectibles. This objects main purpose is the make the game harder with the explosion of the bomb.

- **Rocks:** This class is the child class of the Collectibles. This objects main purpose is the make the game harder with occupation of the place with lower point.
- **Jewelry:** This class is the child class of the Collectibles. This is a collectible object which has most point in the game.
- **Gold:** This class is the child class of the Collectibles. It is the main source of the point in the game.
- **PowerUps:** This class is the child class of the Collectibles. Its purpose is the make the game easier to play with the several power ups for the hook. This class has 3 child classes.
- **Dynamite:** This class is the child class of the PowerUps. Its purpose is the reset the hook if the player took the heavy and pointless object.
- **Time:** This class is the child class of the PowerUps. Its purpose is the increase the remaining time of the game.
- **Strength:** This class is the child class of the PowerUps. Its purpose is the decrease the pullHook time.
- **Animals:** This class is the child class of the Collectibles. . This objects main purpose is the make the game harder with occupation of the place with lower point.

3.1.2 Class Diagram

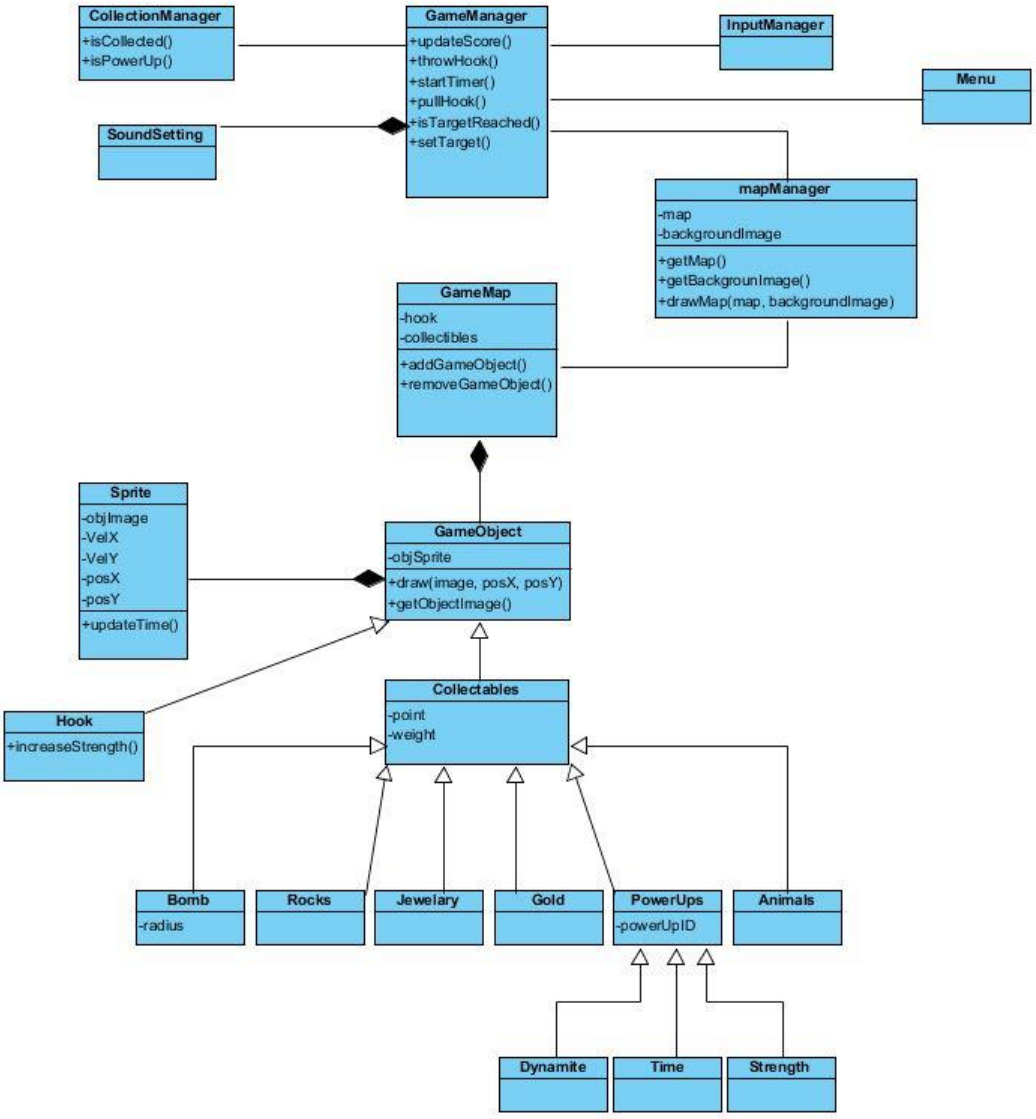


Figure 12 Class Diagram

3.2. Dynamic Models

3.2.1 State Chart Diagram

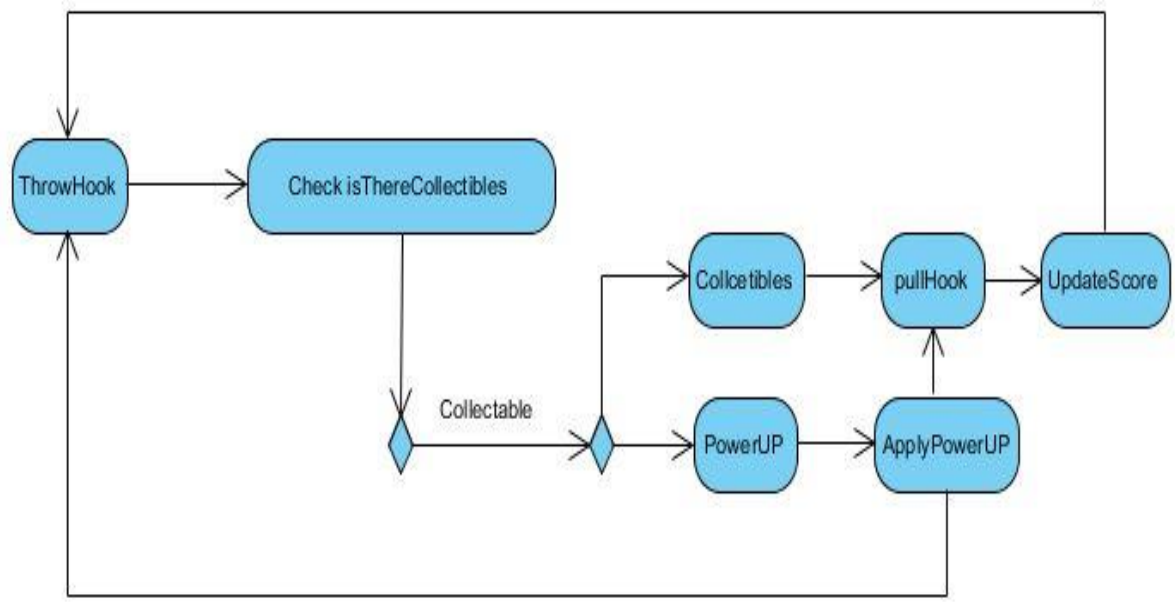


Figure 13 State Chart Diagram

Activity diagram show the main flow of the game event. In every cycle, Game Manager controls the hook and collected items simultaneously. When user select the Play Game the system start to initialize game which contains generate map, set timer and set target functions.

After the creation of the map finished, system wait for the player input from keyboard to start the process of the throwing hook. After the throwing hook process finished the system control if there is a collectible item where the hook is thrown. If there is not any collected item hook return back the initial state without any collected item. After this state, system control the remaining time for the rest of the game. If there is enough time player can throw hook again. If there is collectable item where the hook is thrown, collection manager check if the collected item is a power up or cash point. If it is a power up, system apply the power up to the hook and hook can pull the item rapidly. If the collected

item is cash point, system adds the value of the cash point item to the players point and updates the score. Then system checks the remaining time and the system see that the time is over, the system check if the total point exceeds the target point of the level. After checking the target point of the level the system decides whether the new map generated or not. If the target is reached, system generate new map of the next level and also set the new target point and the new time. If the target is not reached, current game is over and the system displays the high score screen and wants from player to enter his name, after entering the name of the player the system displays the high scores. Then the system returns the main menu.

3.2.2Sequence Diagram

Sequence Diagram for Start game

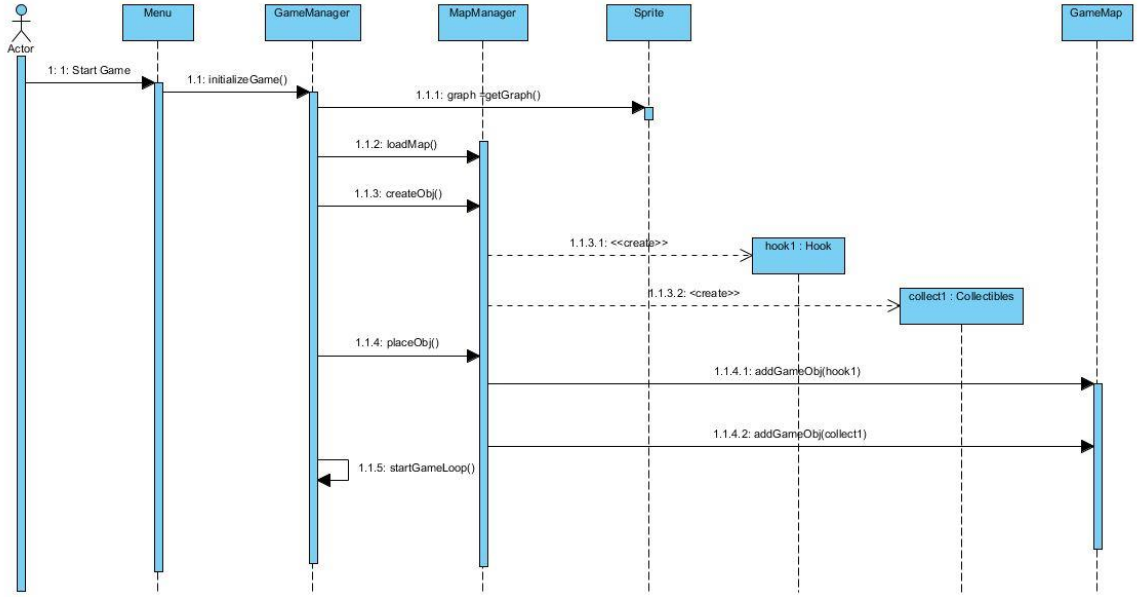


Figure 14 Sequence Diagram for Start Game

Scenario for Start Game: Player starts the game by pressing the start game button from the Main Menu. After the system get graphic images and objects to current window. System loads the current map by generating objects for the game and placing them randomly. Then system starts the game loop.

Sequence of the Game Loop

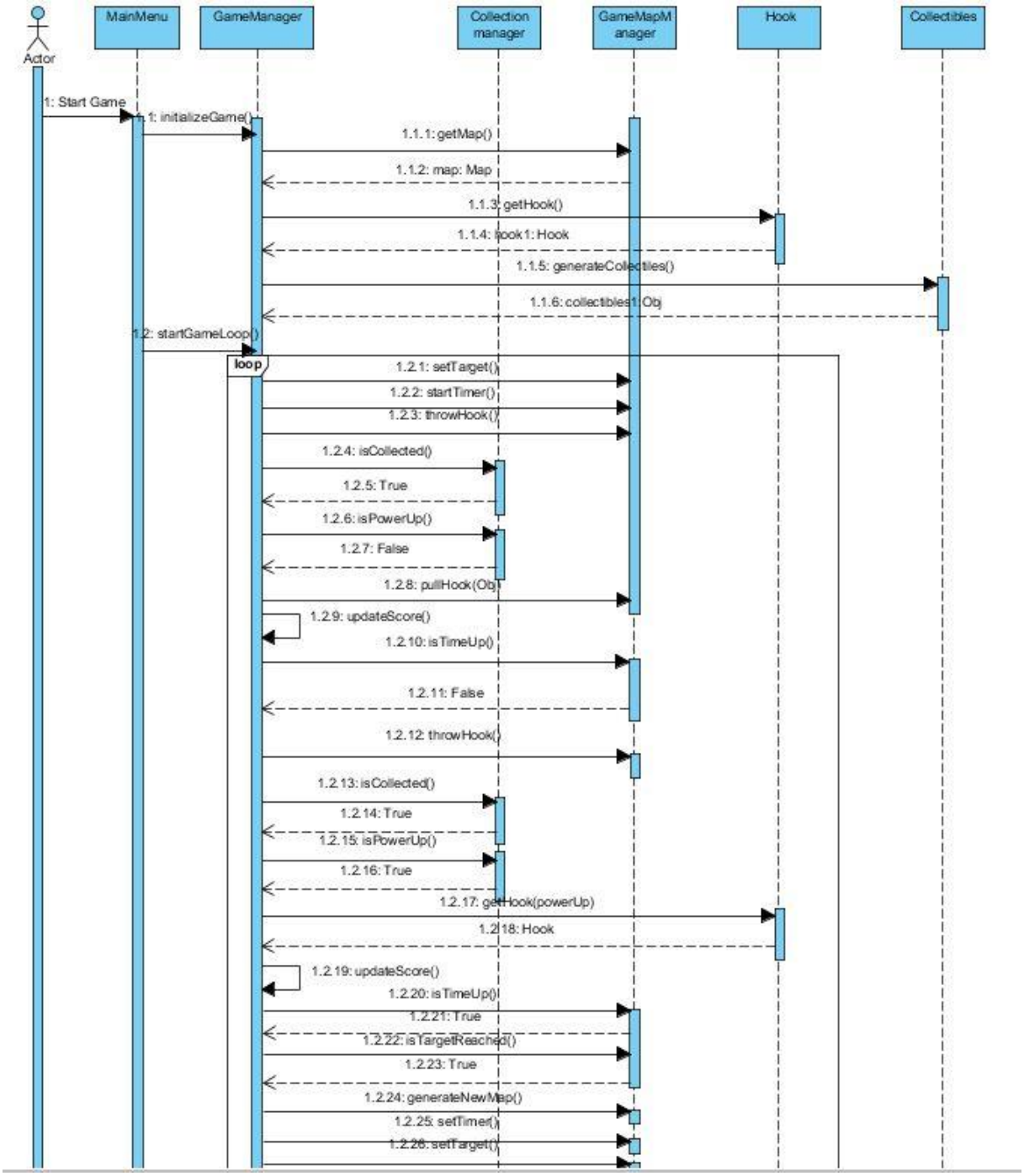


Figure 15 Sequence Diagram for the Game Loop

As seen above collection manager handles the collecting objects and communicates gameManager which manages the actual game dynamics by interacting with other control objects of the system

Scenario for Game Loop: Player starts the game by pressing the start game button from the Main Menu. First the system gets the map, hook and object to the map, and then we entered the game loop. Start of the game loop, the system set the remaining time and target point with the constant unchangeable values. Then player throw the hook where he want to throw, after throwing the hook, the system check if the location of the hook where we throw and location of the collectibles match. If it matches the system activates the collection manager which handles the collection of the collectibles. After that the system updates the score according to point of the collected object. If collected object is a powerUp system activate the special function for the powerUp and gets the new upgraded hook which decreases the time of the pullHook() function.

Sequence Diagram for the Settings

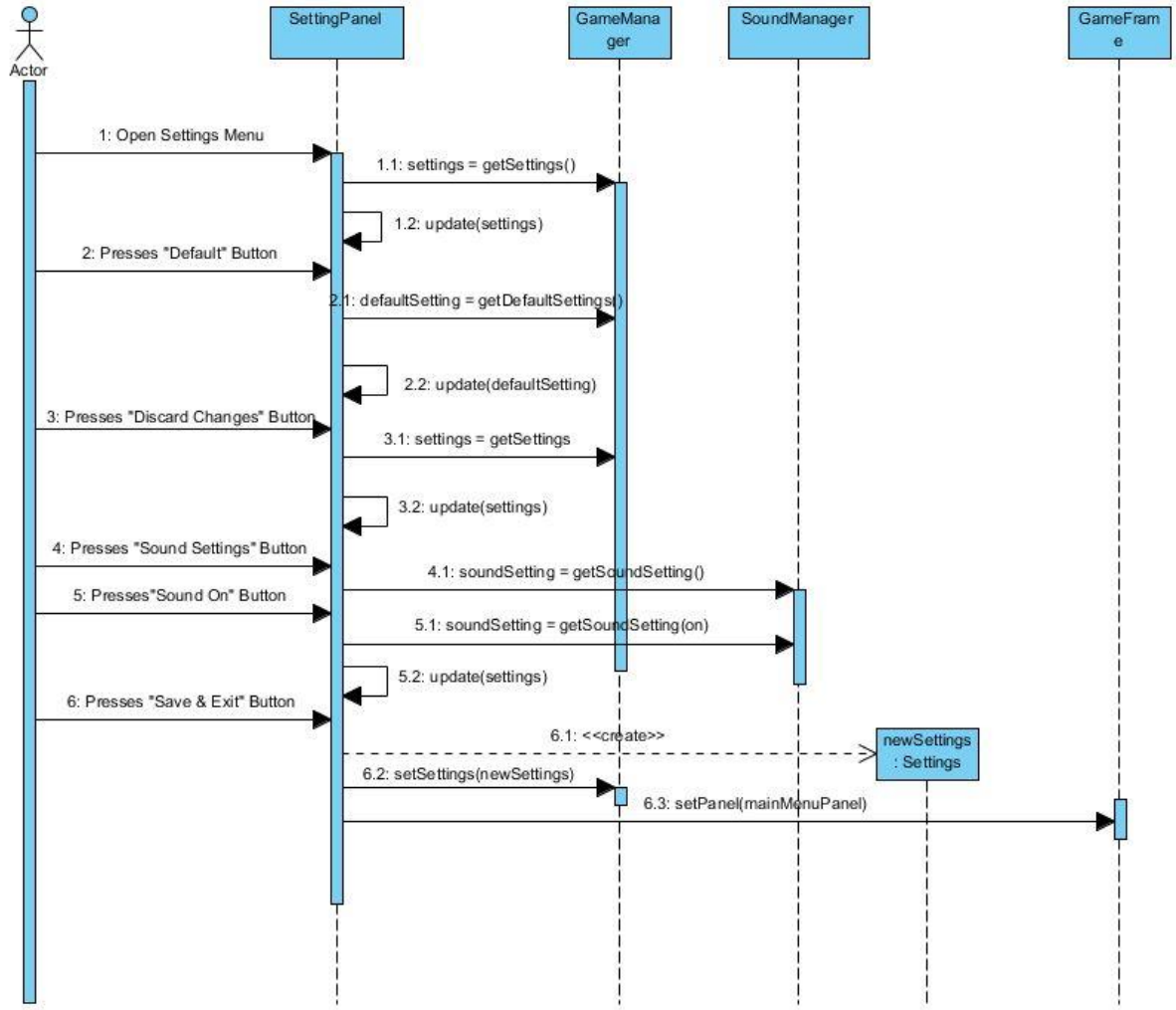


Figure 16 Sequence Diagram for the Settings

Scenario for Settings: This diagram shows how player can make changes setting of the game

Player enters the setting menu by using Settings button in the main menu. Player can press default button for default settings. Player can discard the changes by using the Discard Changes button and lastly player can change the volume level by volume setting. Then use save & exit to return main menu.

4.0 Design

4.0.1 Purpose of the System

Our system's purpose is to provide enjoyable time with Gold Miner. The game that we inspired from http://www.kraloyun.com/Oyun/Hazine-avcisi. In our system, we use new features including new power ups, new maps and challenges. Adding new futures provides the pleasure of achievement for users. The system starts with easy level that provides to guide the user how to play simply. As user progress in levels, the games become more difficult to supply a challenge to the player. In order to get the attention of the player, we have supplied bonuses throughout the game. The purpose of all this is to make sure that the player is entertained and challenged at the same time.

4.1 Design Goals

Designing the game goal has utmost importance because it ensures us to focus on some specific qualities which makes program better in case of end-user criteria, maintenance criteria and customer criteria. These qualities base on non-functional requirements. Our design goals are described below.

4.1.1 End-User Criteria

Ease of Use

In contrast to complex user interface, Gold Miner will consist of simple and smooth user interface menus which end-user will find desired operations then performs. Also Gold Miner will played with two keyboard actions up and down button which ensures easy control to end-user.

Ease of Learning

There is no need to know any knowledge about Gold Miner gameplay before. The instruction panel in the game will provide information to end-user about how to play Gold Miner, loss-win conditions and also it will be clarified game items such as powerUp bag, bomb or dynamite about how it works.

4.1.2 Maintenance Criteria

Extendibility

Adding new levels, items and functions is very important in developing computer world to catch up new age. Because of this situation, Gold Miner will be suitable to add new levels, items and functions to existing system.

Modifiability

Modifying the existing items, functions and level is very easy in our program. For instance, if customer wants to change target point or game time, Gold Miner would be modified easily by not only us, but also most of programmer who knows the language because we decompose and minimize all the parts of the game .

Reliability

The game will be consistent and it will not crash with unexpected input. The game should be ready to all possible scenarios and inputs to achieve this situation.

4.1.3 Customer Criteria

Efficiency

Systems' response time is very important because of the limitation of time during the game. So, system will respond end-users' actions immediately.

4.1.4 Tradeoffs

Ease of Use-Ease of Learning vs Functionality

We will try to design a simple user interface menus and functions in Gold Miner because our aim is make end-user comfortable. We want to make all people with different age able to play Gold Miner without troubling. So, we gave higher priority to ease of use and ease of learning than functionality while we determining our design goals.

Efficiency vs Portability

In Gold Miner, response time has utmost importance in gameplay because game will be buit on end-user timing. When user press the keyboard, hook should move immediately. However in portable graphics code, sometimes we might not get the response times we are looking for. Because of this situation, we prefer efficiency rather than portability when we determine our design goals.

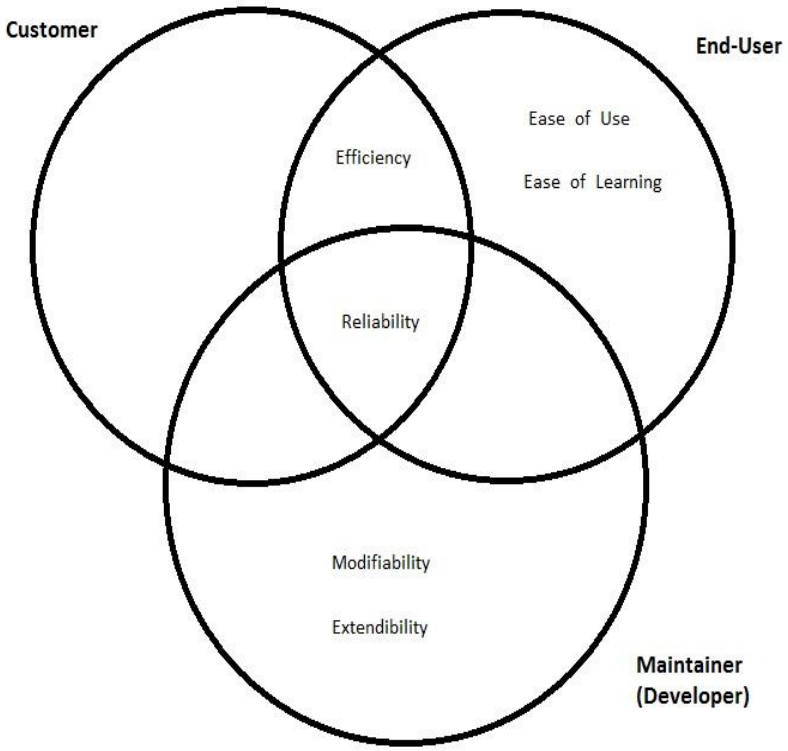


Figure 17: Distribution of design goals among stakeholders

4.1.5 Definitions, Acronyms, and Abbreviations

List of acronyms and abbreviations that are used is following:

MVC: Model, View, Controller [1]

GUI: Graphical User Interface [2]

UI: User Interface [2]

JRE: Java Runtime Environment [2]

J2SE: Java 2 Platform, Standard Edition [2]

JDK: Java Development Kit 4 [2]

4.1.6 Overview

At this stage, we explained purpose of the system, which is entertained and challenged a player at the same time, to achieve this purpose we defined our design goals in this part. Our design goals are specified in three criteria, end-user, maintainer and customer. We determined ease of use and ease of learning for end-user; extendibility, modifiability, reliability for maintenance; and efficiency for customer. Realizing our goals, we made switch between efficiency and portability, ease of use&learning and functionality.

4.1.7 References

1. Object-Oriented Software Engineering, Using UML, Patterns, and Java, 3rd Edition, by Bernd Bruegge and Allen H. Dutoit, Prentice-Hall, 2010, ISBN-10: 0136066836.

2. https://en.wikipedia.org/wiki/Java_(programming_language), website

4.2 Subsystem Decomposition

Our system divided into independent parts makes the developer job easier and make the system well organized. Also decomposition of the system has significant effect on the system like modifiability extendibility and performance. Decomposition is crucial for meeting the requirements of non-functional requirements.

Figure -18 is the subsystem figure we show subsystems classes into the subsystem. Its importance is that when there is a change occurred in the subsystem that change in the subsystem does not affect the other subsystems

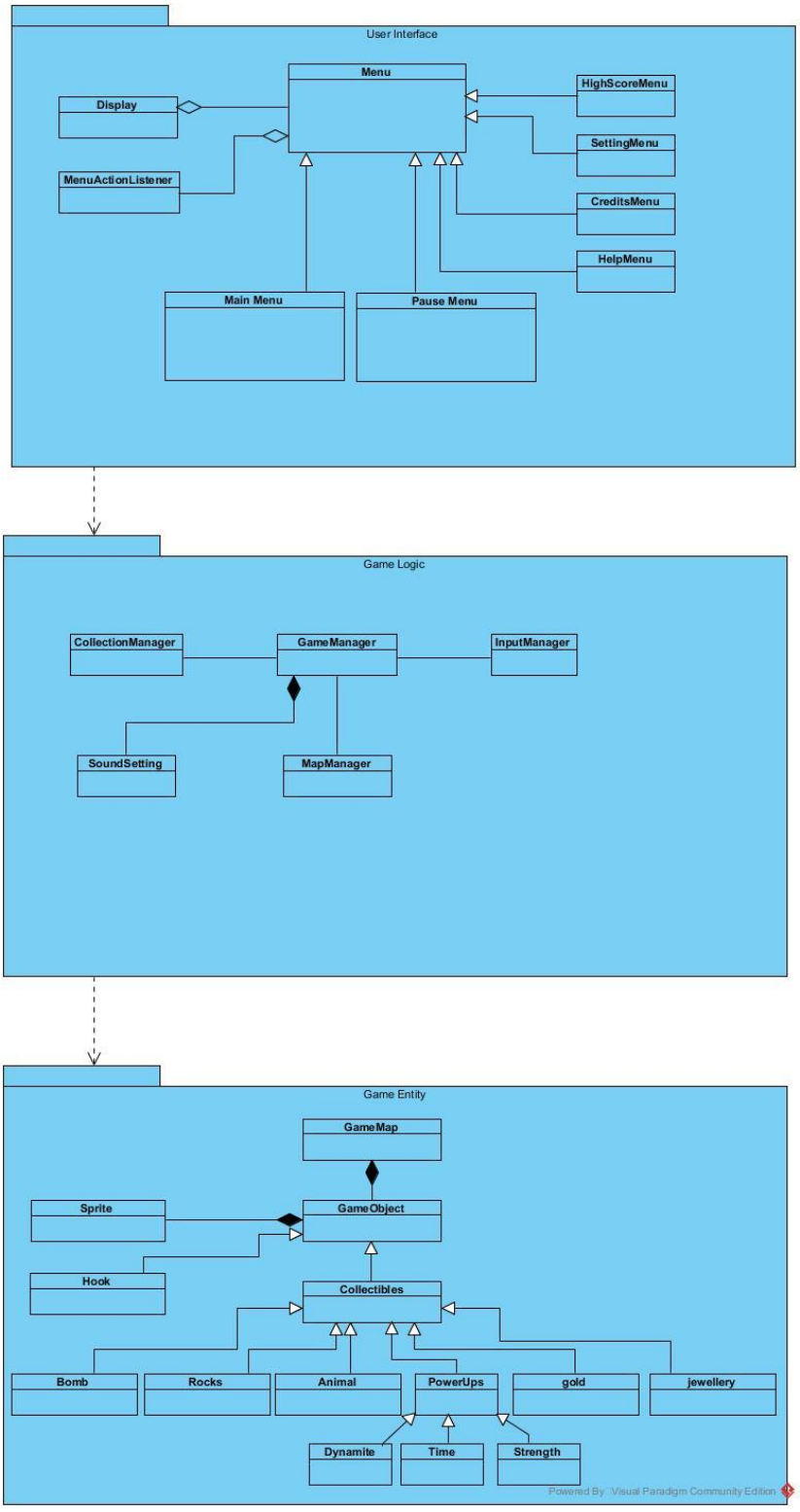


Figure 18 Subsystem Decomposition

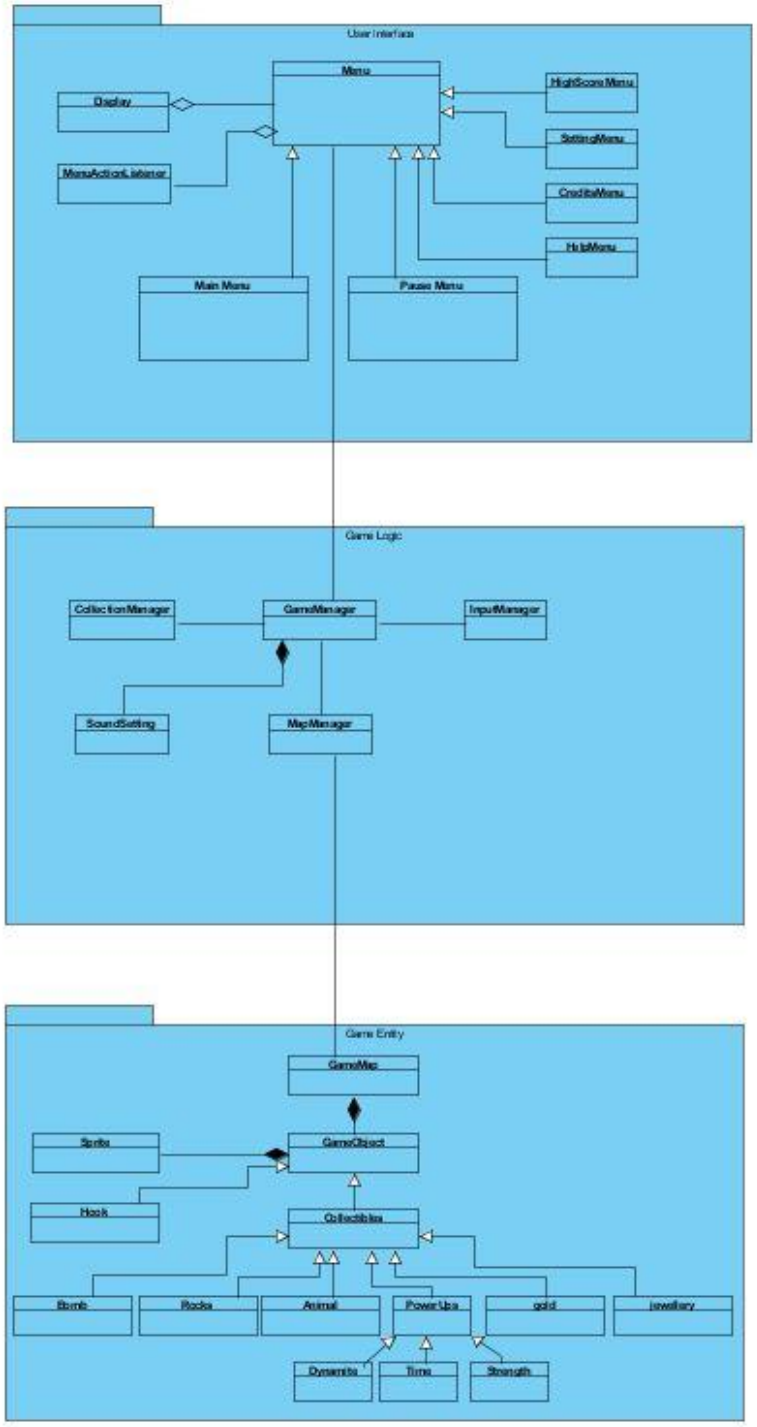


Figure 19: Detailed Subsystem Decomposition

Figuru-19 shows that the detailed decomposition of the subsystems

4.3 Architectural Pattern

Layers

We choose the tree layer architecture design to decompose our system according to model view architecture design pattern. We try to increase cohesion of the subsystem while try to reduce coupling between de subsystems.

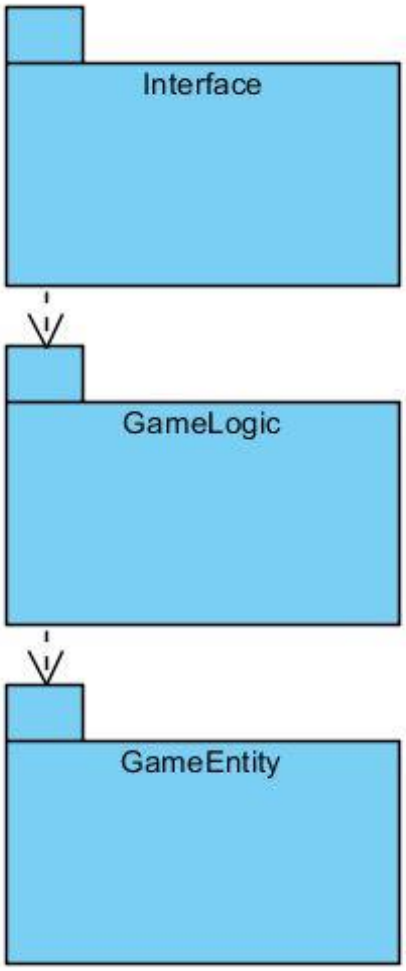


Figure 20: Layers

In our system, we divide the system into three layer which are Interface, gameLogic and GameEntity. Our top layers is Interface layer which provide interaction with the user and it gives interaction related services to user by using other subsytems while it cannot be used

by any other layer below. Our middle layer is the GameLogic layer which provides the game loop actions in it. It uses the below subsytem to create the game loop and it provides service to the above loop. Our bottom subsytem is the game entity subsytem which provide the basic game elements to the game logic and the game interface.

Model View Controller

In this architectural style, we divide our system into tree subsytem to achive this atchitectural patterm. Our Interface shows the view part of this design pattern, GameLogic represent the controller part of this design while game entites represent the model part of the model view controller. We create this system to meet our system goals and achieve high cohesion and low couplng which increase the flexibility of the program. In this pattern our subsytem does not effect from any changes

4.4 Hardware/Software Mapping

Java will be used as an implementation language so JDK need to be installed for play the gold miner. The game has two input devices and one of them is mouse. User can choose the menu options that s/he wants to display by mouse. The other input device is the keyboard. User plays the gold miner with the keyboard. Gold Miner can be played by two buttons which are up and down arrow. The Gold Miner system requirements will be minimal as it is does not many hardware requirements. It needs a platform that can run Java because of the implementation platform is Java and .txt file which required for the storage unit for the Gold Miner. A platform that can satisfy these requirements can run the Gold Miner without any problem.

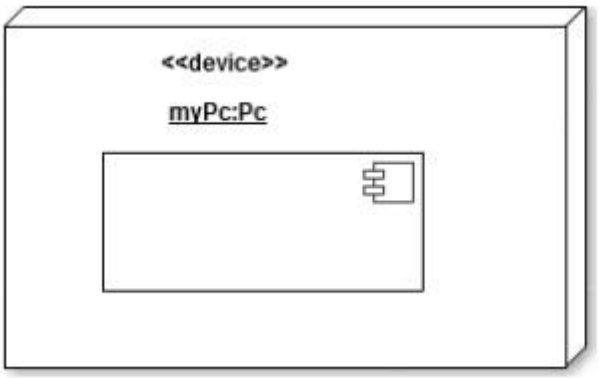


Figure 21: Deployment Diagram

4.5. Addressing Key Concerns

4.5.1 Persistent Data Management

The Gold Miner does not need any complex database system. In the implementation of the game The Gold Miner has static and dynamic data. Map's background and soundtrack of the game are static data that will be saved in texture file. The gold miner's dynamic data are sound settings and the objects locations in the map. Sound setting has a default setting at the execution but the user can change this variable after the execution. Object locations will be determined at the creation of the levels which will be random.

4.5.2 Access Control and Security

The Gold Miner can be played any platform that we explained in the Hardware/Software Mapping part of this report. In addition to that user does not need any authentication to play the game so every user can access to the all objects that the game has. It does not have any access control management because of having only one type of user. Also, because of having no hierarchy among users in the game, Gold Miner does not need to take any personal information from the user. It indicates that our system has not any protection system.

4.5.3 Global Software Control

The software control of Gold Miner is event driven control. As an architectural pattern, model-view-controller is used and therefore event driven control can be proper and logical. Event driven control follows the events, waits for occurring the events and then it sends the proper object. In MVC, when the event is tapping and event occurs. The model reports the view part of the MVC after updating the system. In view part, screens are shown and these screens can read a data from the model part of the MVC and the update the screen.

4.5.4 Boundary Conditions

Initialization

The Gold Miner has the .jar extension since the Gold Miner does not have to be installed.

Termination

The Gold Miner termination process is simple. In main menu, if player wants to quit the game s/he just needs to click the quit game button. If the player wants to quit the game while they playing, s/he opens the pause game menu by clicking pause button. After that, if user wants to return the main menu options, they can click the quit button. In addition to that, if the system crash is occurred, pop up screen which indicates user there is system crash will be shown. Then the user's activities in the game will be recorded. However, if there is a fatal error, subsystem terminates and the system should be rebooted

5. Object Design

5.1. Pattern Application

In the Gold Miner Singleton, Façade, Bridge patterns are used. In below the problems and the solutions are explained.

Singleton Pattern

Problem In the Gold Miner some of the classes have to be single. For example our manage classes such as gameManager and MapManager. They have to be instantiated only once because preventing the conflict among the objects.

Solution For solving this problem we use the singleton pattern In singleton pattern a class need to be instantiated once also the class enable to provide global access. Our management class are centralized with the singleton pattern. The example of Gold Miner's manager class can be seen below.

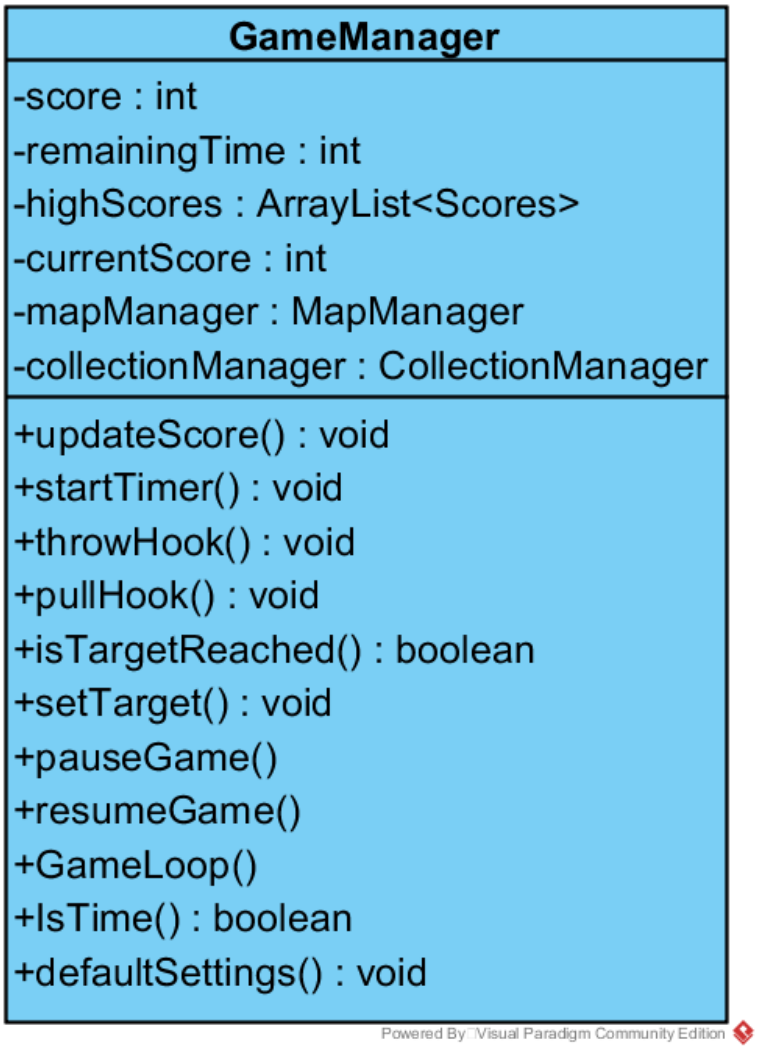


Figure 22 Singleton Pattern

Façade Pattern

Problem Gold Miner has Object subclasses which have different characteristic features like different point's weight and properties. These subsystems cannot be communicated by one interface.

Solution For solving this problem we used façade pattern which provides specified interface for the set of objects in a subsystem. This pattern makes the system easier to communicate with each other. We apply this system to collectibles subclasses

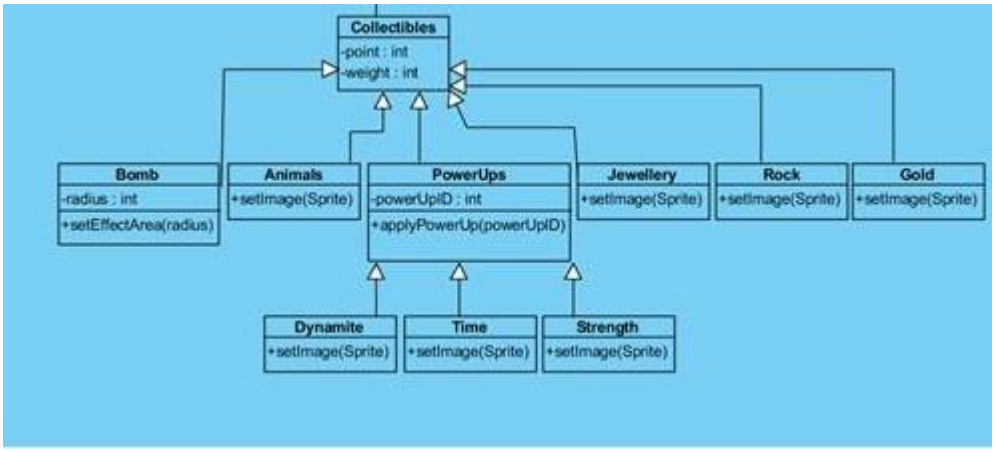


Figure 23 Façade Pattern

Bridge pattern

Problem During the creation of the classes stage, it is not feasible to implement all the classes, so we use temporary classes in the solve this problem and this temporary test classes which can provide basic functions are used until the real classes are implemented . However, replacing the real ones and temporary one's can be challenging

Solution For this problem we use Bridge pattern because it achieve to decoupling. In this way it is easy to interchange alternating implementation for different step.

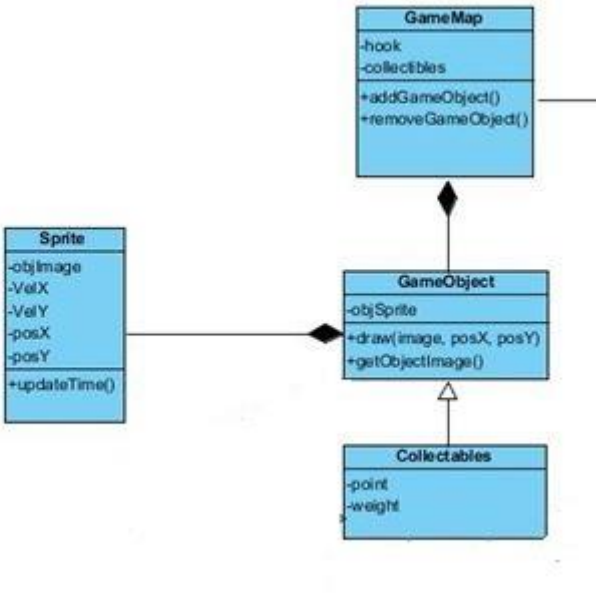


Figure 24 Bridge Pattern

Strategy Pattern

Problem The Gold Miner has different collectible object and power ups. According to collected items, the effect on the game can change differently. Some specific objects have different properties than others. Strategy pattern is used to differentiate these objects with different algorithms. For example in the Gold Miner, power ups inherit bomb, time and strength power ups .However, all of them behaves differently and this situation needs different algorithms in run time, but during execution, switching between algorithms can create problems.

Solution Strategy pattern is solution of this problem. This pattern can provide different algorithm in execution.

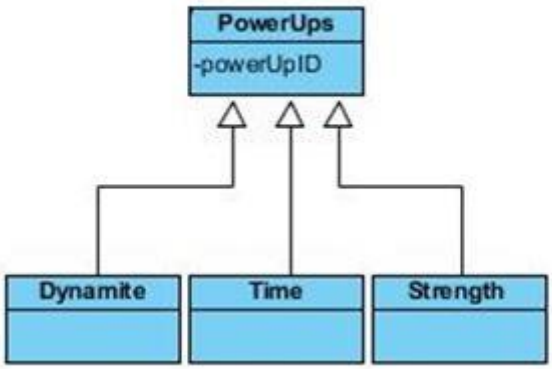


Figure 25 Strategy Pattern

5.2 Class Interfaces

Display Class

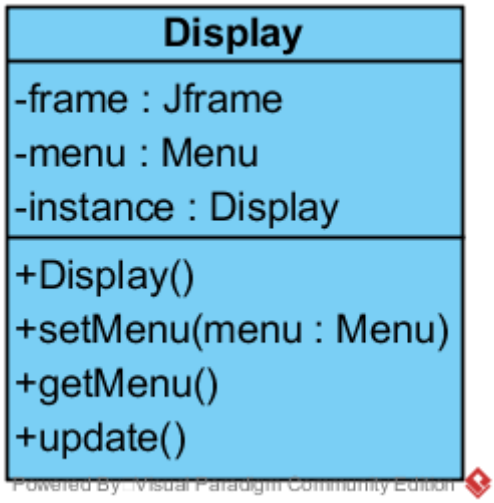


Figure 26 Display Class

This class is the part of the user interface subsystem, Main purpose of this class is the perform display action of the system.

Attributes

Frame: it is the main frame of the game. This frame shows all the visual context of the game.

Menu: This is the instance of the Menu object.

Instance: This is the instance of the Display object itself.

Constructor

Display(): It takes the default settings of the screen and construct the display object.

Operations

setMenu(menu): This operations sets the selected menu for displaying.

getMenu(): This operations get the selected menu which is set by setMenu(menu)

update(): It redraw components of the game.

Menu Class

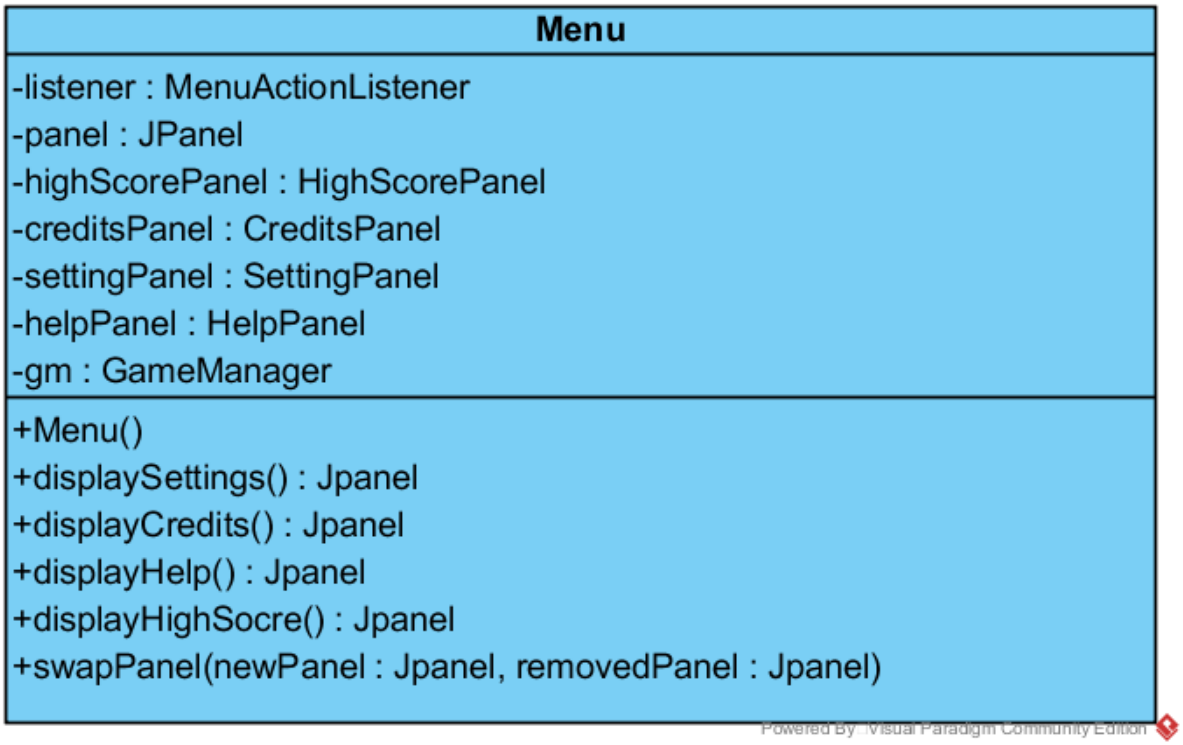


Figure 27 Menu Class

This class displays all visual contexts and gets the user inputs by action listener, and it holds all the panels that are belong to the panel classes. It also holds the game management subsystem.

Attributes

Listener: This attributes main purpose is to get the user input from the interface.

Panel: This attributes holds the entire panels which is property of the Jpanel class.

highScorePanel: This attributes holds the highscore panel.

creditsPanel: This attributes holds the credits panel.

settingPanel: This attributes holds the setting panel.

helpPanel: This attributes holds the help panel.

Gm: This attribute hold the game manager subsystem.

Constructor

Menu(): It is the constructor which initialize the all panels and game manager, settings and listener properties.

Operations

displaySettings() : It displays the settings panel by using the swapPanel() method which purpose is to swap the current panel with the selected panel.

displayCredits() : It displays the credits panel by using the swapPanel() method which purpose is to swap the current panel with the selected panel.

displayCredits() : It displays the credits panel by using the swapPanel() method which purpose is to swap the current panel with the selected panel.

displayHelp() : It displays the help panel by using the swapPanel() method which purpose is to swap the current panel with the selected panel.

displayHighScore() : It displays the credits panel by using the swapPanel() method which purpose is to swap the current panel with the selected panel.

swapPanel(newPanel,removedPanel): This methods swap current panel with the new panel that the user choose.

MenuActionListener Class

Figure 28 MenuActionListener Class

This class purpose is to make the user actions performed. Class uses the overridden actionPerformed method which is created to perform this systems action.

MainMenu Class

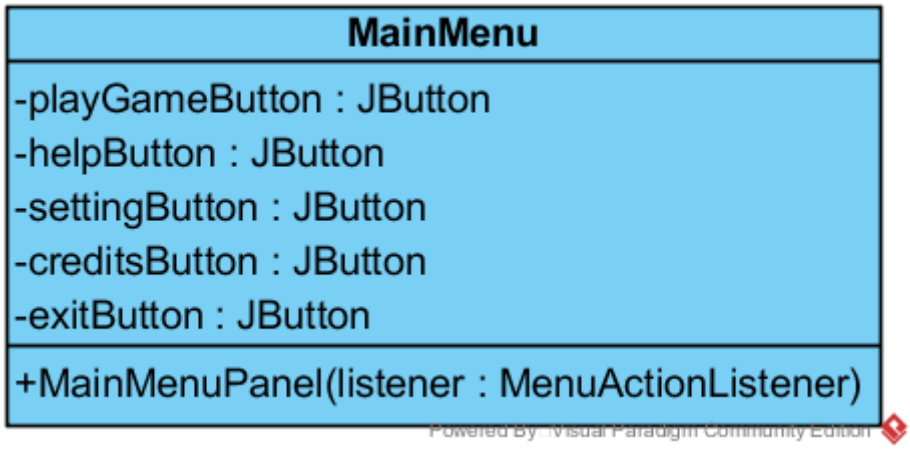


Figure 29 MainMenu Class

This class is the panel of the main menu class.

Attributes

playGameButton: This is the attribute of the play game button

helpButton: This is the attribute of the help button

settingButton: This is the attribute of the setting button

creditsButton: This is the attribute of the credits button

exitButton: This is the attribute of the exits button

Operation

MainMenuPanel(menuaActionListener): It is the property of the menu action listener. It holds all the button which connects the main menu panel with the others menus panel

PauseMenu Class

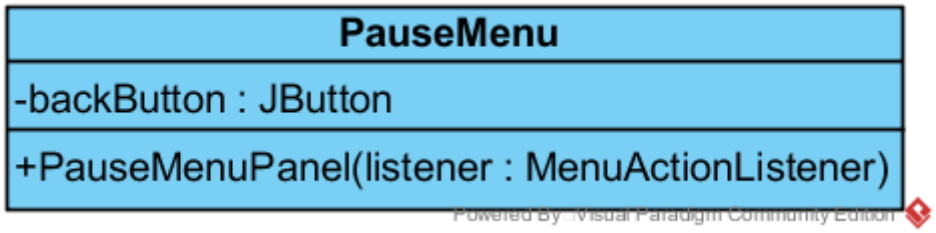


Figure 30 PauseMenu Class

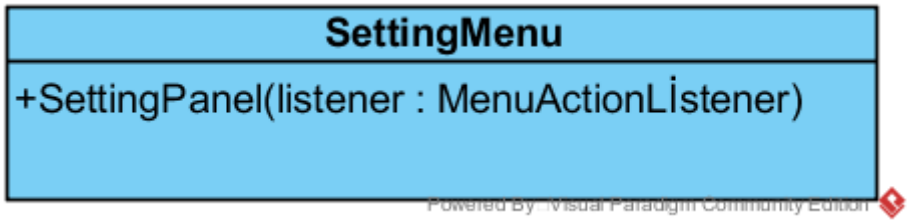
Attribute

backButton: This is the attribute of the back button.

Operation

PauseMenuPanel(MenuActionListener): It is the property of the menu action listener. It holds the entire buttons which connects the pause menu panel with the game.

SettingMenu Class



Operations

SettingPanel(MenuActionListener): It is the property of the menu action listener. It holds the entire buttons which connects the setting menu panel with the main menu.

CreditsMenu Class

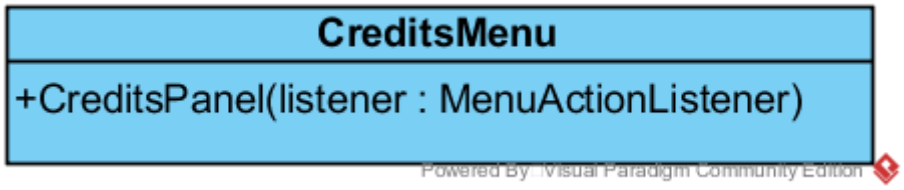


Figure 31 CreditsMenu Class

Operations

CreditsPanel(MenuActionListener): It is the property of the menu action listener. It holds the entire buttons which connects the credits menu panel with the main menu.

HelpMenu Class

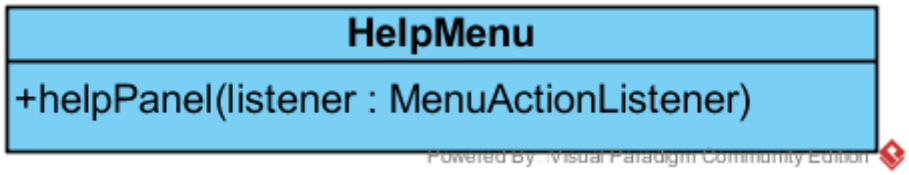


Figure 32 HelpMenu Class

HelpPanel(MenuActionListener): It is the property of the menu action listener. It holds the entire buttons which connects the helps menu panel with the main menu.

HighScore Menu

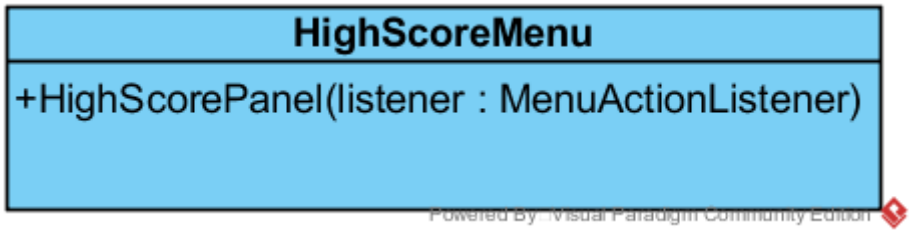


Figure 33 HighScoreMenu Class

HighScorePanel(MenuActionListener): It is the property of the menu action listener. It holds the entire buttons which connects the HighScore menu panel with the main menu.

GameLogic Sub System

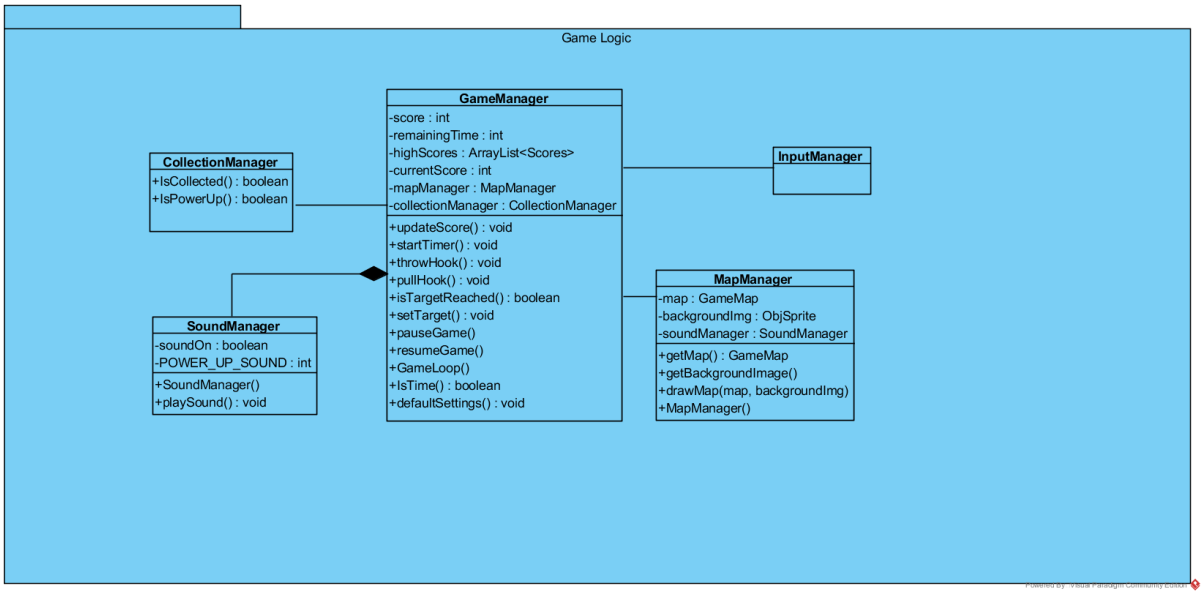


Figure 34 GameLogic Subsytem

GameManager Class

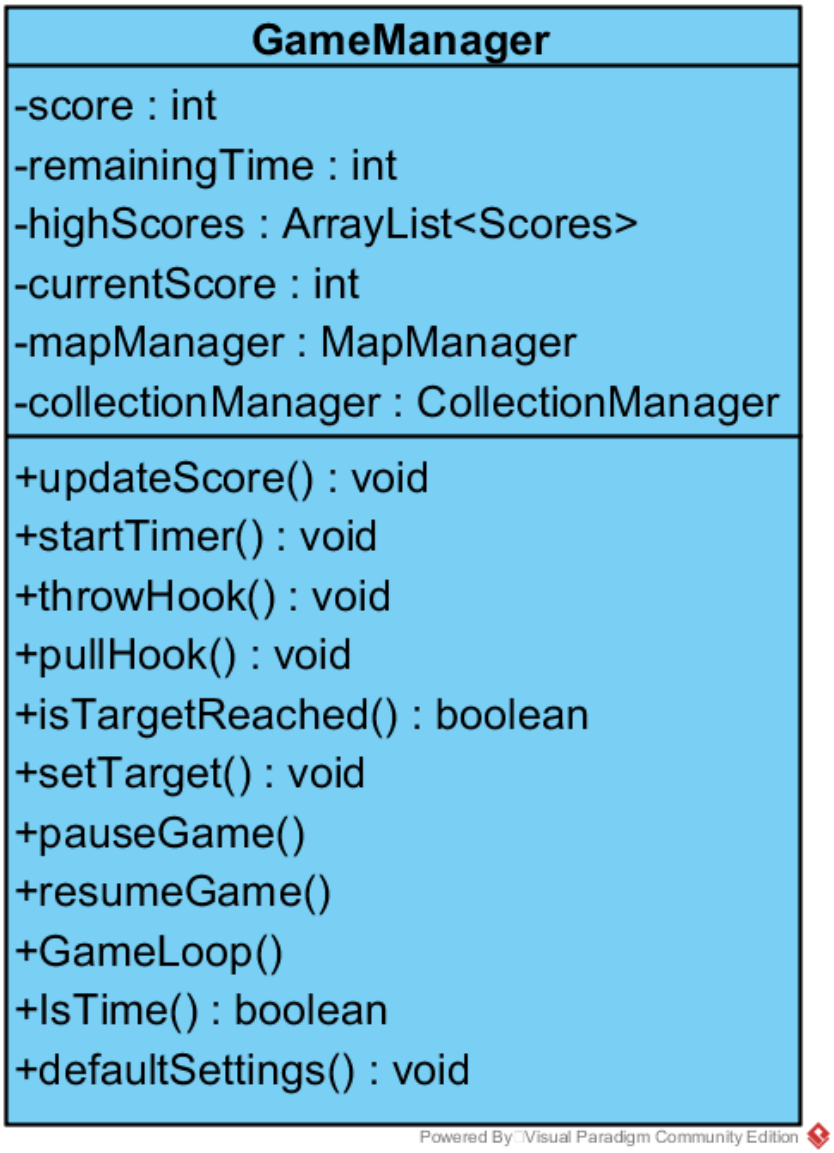


Figure 35 GameManager Class

This is a façade class of the GameLogic subsystem. Main purpose of this class is perform the game loop by using request of the user view subsystems

Attributes

Score: This is the users' points' integer value that comes from the collected items.

RemainingTime: This attributes shows the time that remain to reach the target point.

highScore: This attributes holds the users highest score in an araylist.

currentScore: This attributes holds the users current point that comes from collected items.

mapManager: This is the instance of the mapManager which creates the map of the whole game.

collectionManager: This is the instance of the collectionManager which controls whole collected items.

Operations

updateScore(): This methods update the current score after user collects an item.

startTimer(): This methods start the timer after the user start a game also after every newly generated level timer will be updated and start again.

throwHook(): This methods will be activated after user clicks the down button and it will be throw the hook.

pullHook(): This method will be activated after the throwHook completed.

istargetReached(): This method check the given target is reached or not reached.

setTarget(): This methods will be updated the current target after the current level is completed successfully.

pauseGame(): This method pauses the game.

resumeGame(): This method resumes the game.

gameLoop(): This method is the main game loop and all the other actions going to be inside this loop.

IsTime(): This method checks the remaining time to continue the game.

DefaultSettings(): This methods set the settings to the default mode.

Collection Manager Class

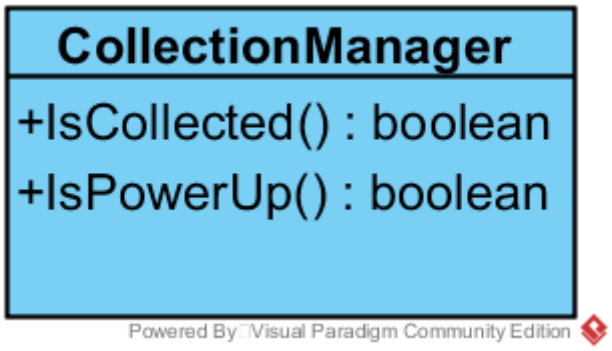


Figure 36 CollectionManager Class

This class is the part of the game logic subsystem and its checks the collected object and decide the collected objects is power up or not.

Operations

IsCollected(): this method checks if the hook collide with a collectible item.

IsPowerUp(): This method checks if the collected item is a power up or not.

MapManager Class

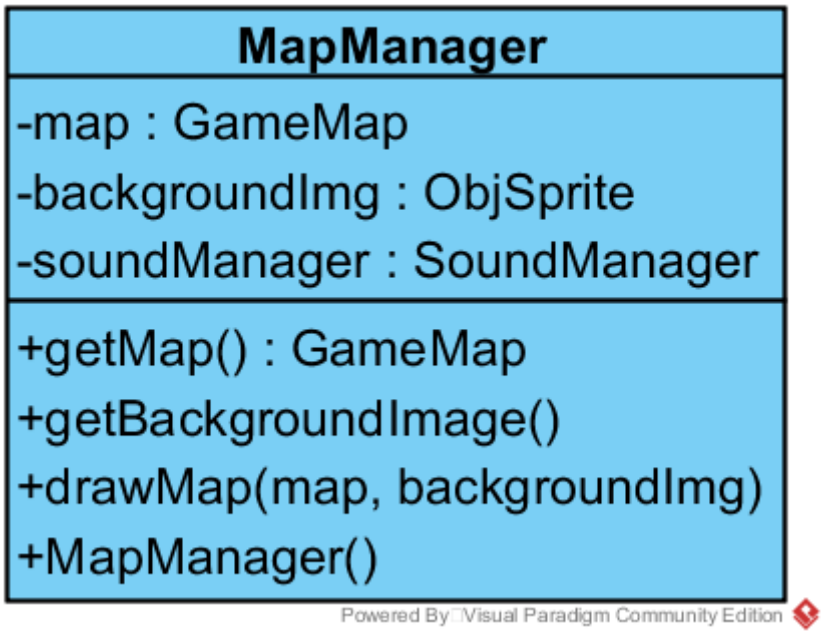


Figure 37 MapManager Class

This class is the part of the logic subsystem and it's managing the maps of the level.

Attribute:

Map: This is the instance of the GameMap object.

backgroundImg: This is the attribute that hold the image of the background.

soundManager: this is the instance that hold the sound manager object.

Operations

getMap(): This method gets the map of the game when the level is completed.

getBackgroundImage(): This method gets the background image.

drawMap(map,backgroundImg): This method create the game map by taking the map object and background image.

mapManager(): This method runs the map managing functions.

SoundManager Class

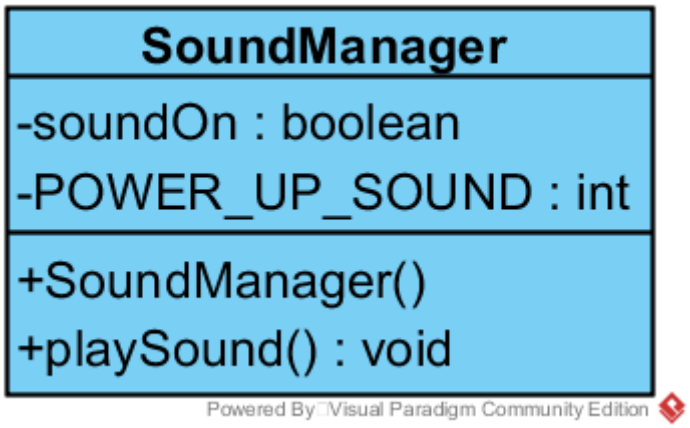


Figure 38 SoundManager Class

This class is the part of the game logic subsystem and manages the sound options.

Attributes

soundOn: This is the control instance that controls sound is on or off.

POWER_UP_SOUND: this is the predefined sound of the power up.

Operations

SoundManager(): This methods runs the sound managing system.

playSound(): This methods play the predefined sound

GameEntity Sub system

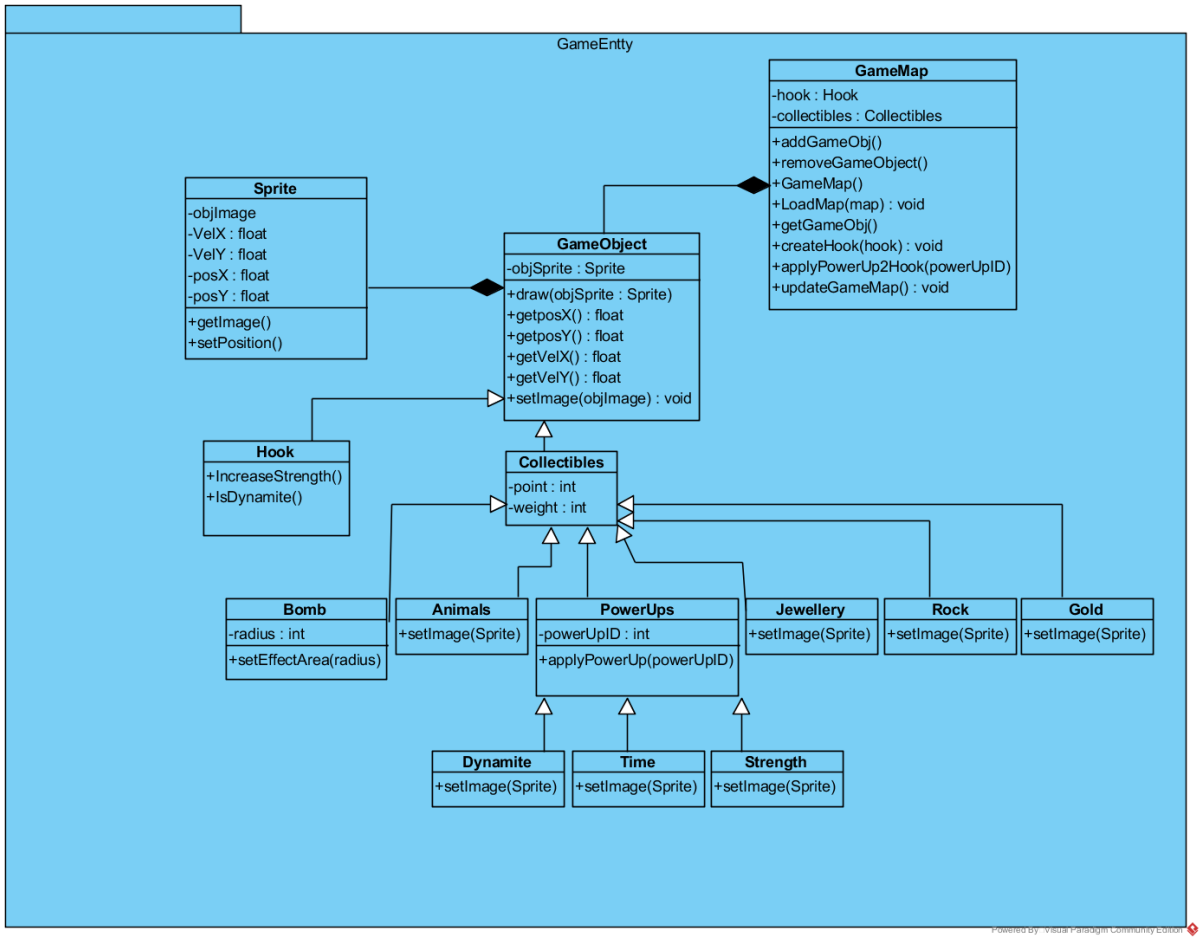


Figure 39 GameEntity Subsystem

GameMap Class

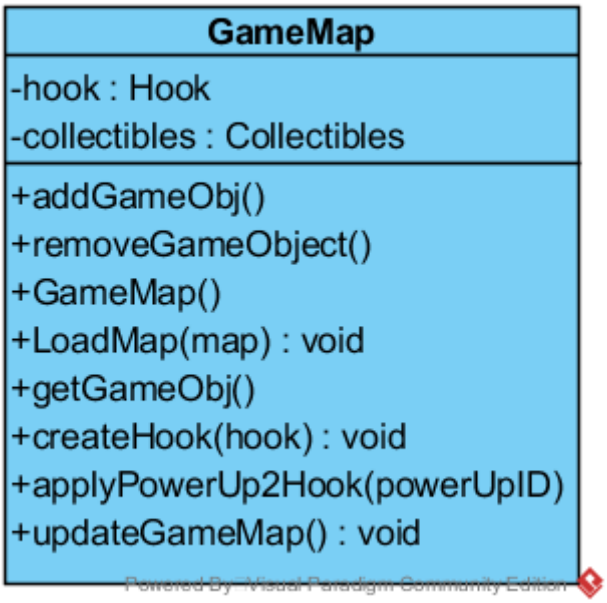


Figure 40 GameMap Class

GameMap class is the part of the GameEntity subsystem and it provides communication with the other components of the GameEntity subsystem therefore it indicates that it has methods which are related to create and change game entities.

Attributes:

Hook: this attribute represents the one of the main objects of the game. As the name indicates it shape is like a hook which provides to collect collectible items.

Collectibles: This attribute represent the other main objects of the game which are the collectibles of the game.

Constructor:

GameMap(): Construct game map with default attribute.

Operations:

addGameObj(): It adds the game objects to the game map.

removeGameObj(): it removes the game objects from the game map.

LoadMap(GameMap): This method main purpose is the creation of the game map with game objects.

getGameObject(): This methods main purpose is retrieve the objects to the map.

createHook(hook): This methods main purpose is the creation of the hook.

applyPowerUp2Hook(): This methods main purpose is applying power ups to the hook.

updateGameMap(): Main purpose of this method is updating map after the changes of the map.

Game Object Class

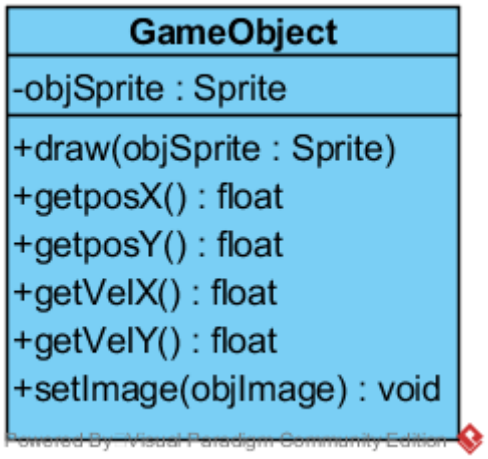


Figure 41 GameObject Class

Game Object class is the part of the GameEntity subsystem. Its main purpose is the creation of the game objects. This is the generalization of the creation of game object. We can also change the current state of the created object with this class.

Attributes

objSprite: This is the instance of the Sprite class.

Operations

draw(objSprite): This method enables game objects to draw itself.

getPosX(): This methods gets the horizontal axis value of the object.

getPosY(): This methods gets the vertical axis value of the object.

getVelX(): This methods gets the horizontal value of the hook velocity

getVelY(): This methods gets the vertical value of the hook velocity

setImage(objImage): This methods provides the image of the object.

Sprite Class

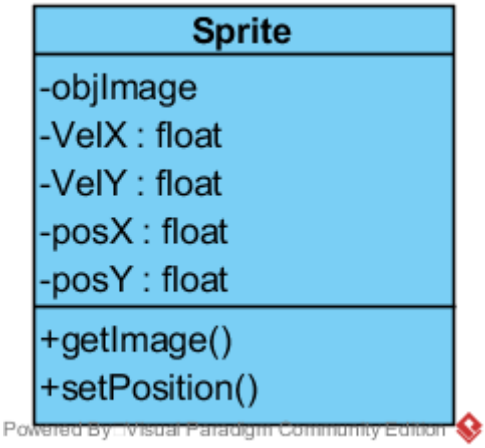


Figure 42Sprite Class

Main purpose of this class is updating objects position and image by providing position and velocity values by horizontal and vertical values..

Attributes

objImage: This attribute provide image of the object whose going to be updated by attribute of the class.

VelX: Horizontal velocity attribute of the object.

VelY: Vertical velocity attribute of the object.

PosX: Horizontal positional attribute of the object.

PosY: Vertical positional attribute of the object.

Operations

getImage(): This methods provides the image of the object.

setPosition(): This method set the position of the object.

Hook Class

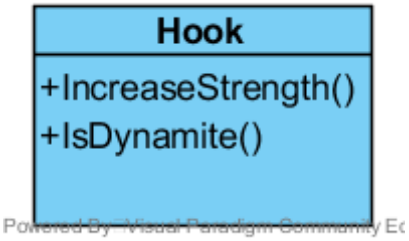


Figure 43 Hook Class

This class represents the main objects of the game. It provides changes on the hook according to collected power ups.

Operations

IncreaseStrength(): This methods indicates the increasing of the strength of the hook. the meaning of this statement is after the collecting of the strength power up, the speed of the pulling hook is increased even if the heaviest and biggest item is collected.

IsDynamite(): This method purpose is the checking if any dynamite power up is taken or not taken. If there are dynamite then the method notify the user that there is a dynamite

Collectibles Class

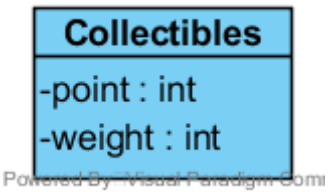


Figure 44 Collectbles CLass

Collectibles class indicates the all collectible item in the game. There are sub classes in the collectibles class.

Attributes

Point: this attribute shows the all collectible item's points

Weight: this attribute shows the all collectible item's weight

Bomb Class

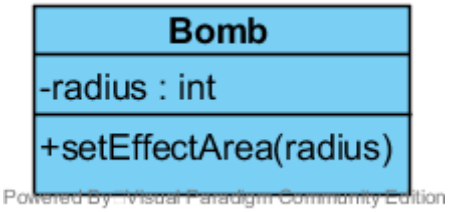


Figure 45 Bomb Class

This is the one of the collectible class and its main purpose is to destroy the nearby objects according to attribute of the radius

Attribute

Radius: this attribute is the bomb's effective radius area.

Operation

setEfectArea(radius): This method set the effective area of the bomb by given radius.

Gold Class

Figure 46 Gold Class

This class is the one of the collectible items of the game. These objects are the main source of the points.

Operation

setImage(Sprite): This method set the sprite of the object.

Rock Class

Figure 47 Rock Class

This class is the one of the collectible items of the game. These objects are the one of the obstacle of the game

Operation

setImage(Sprite): This method set the sprite of the object.

Jewelry Class

Figure 48 Jewelry Class

This class is the one of the collectible items of the game. These objects is the most valuable item in the game

Operation

setImage(Sprite): This method set the sprite of the object.

Animal Class

Figure 49 Animals Class

This class is the one of the collectible items of the game. These objects are the one of the obstacle of the game

Operation

setImage(Sprite): This method set the sprite of the object.

Power Ups Class

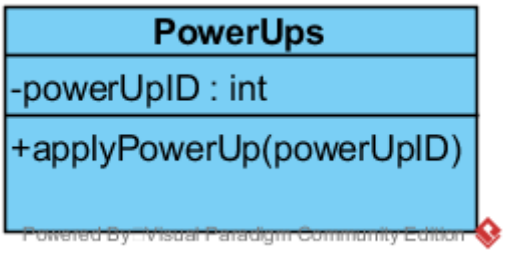


Figure 50 PowerUps Class

It is the one of the object class of the game and it inherits all the power ups classes. Power ups meant to make the game easier by collecting the power ups.

Attributes

powerUpID: This attribute indicates the ID of the power ups. this attribute is used to filter the power ups by their ID.

Operation

applyPowerUp(powerUpID): This methods purpose is the apply the collected power ups on the hook.

Dynamite Class

Figure 51 Dynamite Class

This class is the one of the power ups classes. Dynamite is the object that destroys the wrongly collected object.

Operation

setImage(Sprite): This method set the sprite of the object.

Time Class

Figure 52 Time Class

This class is the one of the power ups classes. Time is the object that increases the remaining time by predefined amount.

Operation

setImage(Sprite): This method set the sprite of the object.

Strength Class

Figure 53 Strength Class

This class is the one of the power ups classes. Strength class increase the power of the hook when collected and it decreases the time of the pulling the hook.

Operation

setImage(Sprite): This method set the sprite of the object.

5.3 Specifying Contracts

Display:

1. **Context** Display::setMenu(menu:Menu):void

   Post: self.menu = menu

   After the setMenu(menu:Menu) method the parameter must equal the menu

2. **Context** Display::getMenu():Menu

   Post: menu = self.menu

   After the getMenu() method, menu name is returned

Menu:

3. **Context** Menu::displaySettings():Jpanel

   Pre: Menu()

   After displaySetting() method the setting menu will be displayed.

4. **Context** Menu::displayCredits():Jpanel

   Pre: Menu()

   After displayCredits() method the credits menu will be displayed.

5. **Context** Menu::displayHelps():Jpanel

   Pre: Menu()

   After displayHelps() method the helps menu will be displayed.

6. **Context** Menu::displayHighScore():Jpanel

   Pre: Menu()

   After displayHighScore () method the high score menu will be displayed.

GameManager

7. **Context** GameManager::updateScore():void

Pre GameLoop()

After updateScore() method , the score in the gameLoop will be updated.

8. **Context** GameManager::startTimer():void

   Pre GameLoop()

   After startTimer() method , the timer in the gameLoop will be started.

9. **Context** GameManager::throwHook():void

   Pre GameLoop()

   After throwHook() method , the hook in the gameLoop will be thrown

10. **Context** GameManager::pullHook():void

    Pre GameLoop()

    After pullHook() method , the hook in the gameLoop will be pulled

11. **Context** GameManager::isTargetReached():boolean

    Pre updateScore()

    Post result =TargetReached

    After isTargetReached() method , the score will be checked and return the Boolean value.

12. **Context** GameManager::isTime():boolean

    Pre updateScore()

    Post result = isTime

    After isTime() method , the time will be checked and return the Boolean value.

13. **Context** GameManager::setTarget():void

    Pre GameLoop()

    Post result = Target

    After setTarget() method , the time target will be set after every new level

CollectionManager

14. **Context** CollectionManager:: isCollected():Boolean

    Pre ThrowHook()

Post UpdateScore()

After isCollected() method, system check the hook for any collectible item.

15. **Context** CollectionManager:: isPowerUp():Boolean

Pre isCollected()

Post applyPowerUpToHook()

After isPowerUp() method, system check the collected item for any upgrade to apply.

GameMap

16. **Context** GameMap::addGameObj():void

Pre GameMap()

After addGameObj() method, system add game object to the map.

17. **Context** GameMap::removeGameObject():void

Pre GameMap()

After removeGameObject() method, system remove the game object from the map.

18. **Context** GameMap::loadMap(map):void

Pre GameMap()

Post map = Map

After loadMap() method, system load the map of the game.

19. **Context** GameMap::getGameObj():

Pre GameMap()

Post result = Object

After getGameObj() method, game object is returned

20. **Context** GameMap::createHook():void

    Pre GameMap()

        After createHook() method, hook will be created.

21. **Context** GameMap::applyPowerUpToHook(powerUpID):void

    Pre isPowerUp()

    Post result = powerUpID

        After applyPowerUpToHook (powerUpID) method, power up will be upgraded.

GameObject

22. **Context** GameObject::getPosX():void

    Post result = posX

        After getPosX() method, position X will be returned

23. **Context** GameObject::getPosY():void

    Post result = posY

        After getPosY() method, position Y will be returned

24. **Context** GameObject::getVelX():void

    Post result = velX

        After getVelX() method, velocity value of X will be returned

25. **Context** GameObject::getVelX():void

    Post result = velY

        After getVelY() method, velocity value of Y will be returned

26. **Context** GameObject::setImage(objImage):void

    Pre GameObject()

    Post Image = ObjImage

        After setImage() method, image will be set.

Sprite

27. **Context** Sprite::getImage():Sprite

    Post result = Image

        After getImage() method, image will be returned

28. **Context** Sprite::setPosition():void

    Post result = position

        After setPosition() method, position will be set randomly

Hook

29. **Context** Hook:: IncreaseStrength():void

    Pre isPowerUp()

    Post applyPowerUpToHook()

        After increaseStrength() method, the strength power up will be apply to the hook

30. **Context** Hook:: isDynamite():void

    Pre isPowerUp()

    Post applyPowerUpToHook()

        After isDynamite() method, the dynamite power up will be apply to the hook.

6. Conclusion

6.1 **Conclusion**

In analysis report, we designed analysis report of “Gold Miner”. We tried to explain what the game does, what functions it has. This analysis report has two parts which are requirement specifications and analysis part. In analysis part, we designed object model including class diagrams and dynamic model including state chart and sequence diagram. In requirement specification part, we explained all requirements of game instructions.

In design report, we explained the design goals of our project and shaped Subsystem Decomposition. Then we showed Architectural Patterns and Hardware and Software Mapping. Lastly, we examined Addressing Key Concerns.

In final report, we demonstrated Pattern applications, Class Interfaces and Specifying Contracts. After conclusion, we have deduced Lessons Learned that explains what we learn from class and what we practice with the project.

We used Java for our project. Also we tried to make our game is extendible for updates, reliable, simple and available for most type of computers which has its own flash player.

To sum up the game, our game is consist of 10 levels with various objects such as, bonus items (powerUp bag, bomb, dynamite, time penalty and extra time) , hook, time and collectible items (bomb, Stones, jewellery, gold, animals and bonus items). Player of all ages can play easily our game because only one button controls the hook. Aim of the game is collecting the jewellery, gold and bonus items with the hook in specific time interval and raising the point. There will be 2 ways to game over ; first one is if players’ point is less than target point and other one is exit level button which player clicks with his/her desire.

6.2 Lesson Learned

With Object-Oriented Software Engineering lesson, we have learned many things about principles of object-oriented software development. With this project, we performed

essential things what we learned in class and practiced for our future business lives. We have gained experience about team work and learned how to manage project instructions and models step by step. In the Analysis and Design Report, we designed the structure of our system. With final project, we finished the report parts of project. The project requires lots of thinking to examine every last detail of the instructions. In conclusion, we have tried to our best for designing reports, at the same time; we have gained experience for coming projects and our business lives.