CS319 Object-Oriented Software Engineering

Project Analysis Report

Ferhat Serdar Atalay

Aylin Çakal

Ali Bulut

Ismail Serdar Taskafa

1. Introduction

PlanetTrip is a Java game we have chosen as our project. The game is inspired by by Javanoid type arcade games. In such games, there is a map of meteors, a paddle and a ball. Using paddle, player is supposed to give direction to ball and destroy all the meteors by hitting them. Those bricks have different stiffness and some of them are harder to destroy than others.

All of our group members are huge fans of SpaceX and we are all fascinated with their latest launch of Falcon 9. Because of that, we decided to shape our scenario around space travel. In scenario, player will travel across the solar system, starting from Neptune to the Sun. Each planet is a different level and player can see the respective planet in the background. When player clears all meteors on the map, they can see the planet in its full entirety and continue with the next level. As the player progresses with game, levels will be more challenging to keep the sense of engagement. Currently game will have the following features:

- Single Player Mode
- Level Selection [from Neptune to Sun]

We have chosen Java as our language for development because all group members are experienced with that and it is a powerful language that supports principles of object oriented programming. Using these principles will make our software easier to maintain and to add new features without disrupting all parts of the program. We believe this project will be a good exercise to learn and practice these principles. Moreover, developing a desktop application with an UI component using Java is more straightforward compared to C/C++.

2. Game Overview

This section will give a brief overview of our game. It is composed of descriptions of gameplay, levels and different bonus types. Levels and different bonuses are mechanisms to change level of challenge and keep user in a state of flow. The last section will explain how user can modify game settings.

2.1 Gameplay

“PlanetTrip” is a single player game. After selecting Play Game in main menu, user can decide to play with any of nine levels and they are ordered with respect to their challenges. In each level, user is presented with a map and meteors in it. They will move the paddle by keyboard to direct the ball and not let it fall through the void. If ball falls, user will lose one life and game will end if user has exhausted all their lives. Level will conclude and next level will be unlocked if user hits and destroys all meteors.

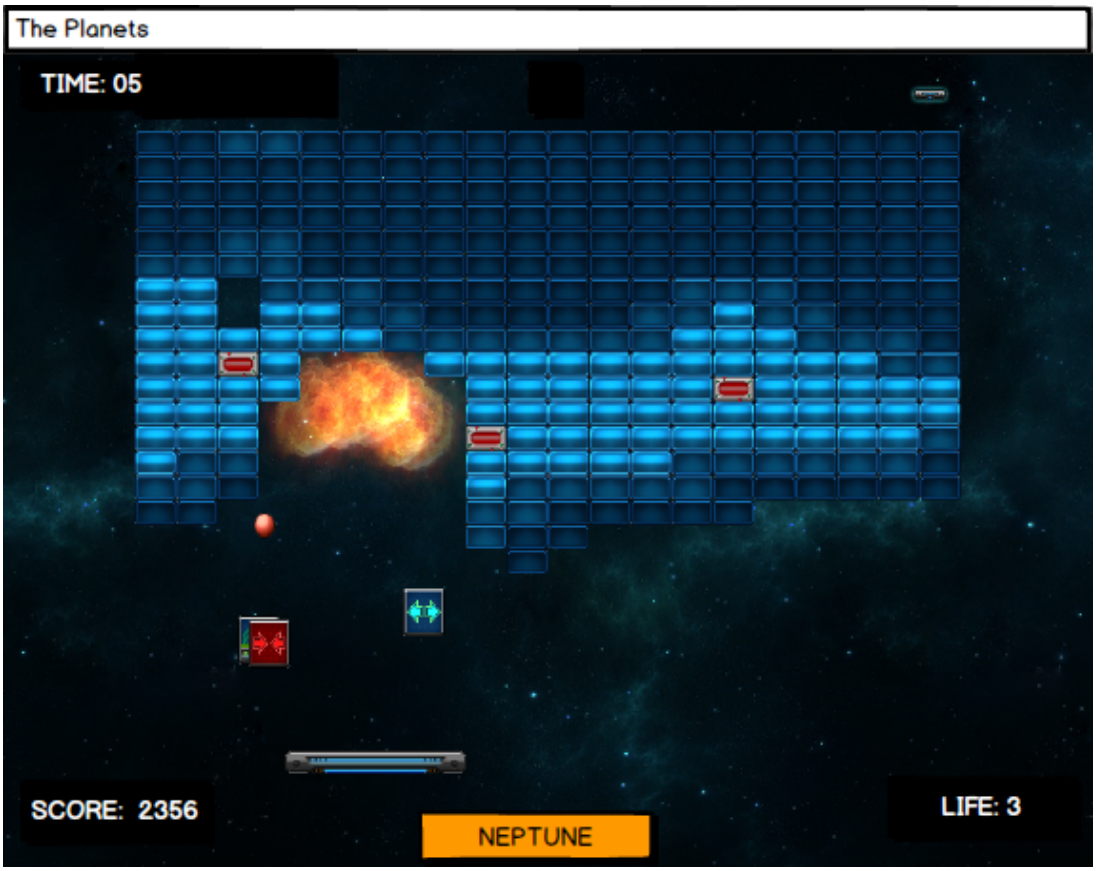


Figure 1. Gameplay

2.2. Levels

In order to adjust difficulty level according to the user's abilities, difficulty will increase with each level. For that purpose, there are two mechanisms and they are different types of meteors and change velocity of the ball.

There are four main types of meteors, which are Alpha, Beta, Gamma and Radioactive. First three are ordered with respect to their stiffness. With each level, ratio of the latter will increase while the former will decrease. That change will make the game more challenging.

- Alpha: It is the softest meteor. It will be destroyed after single hit but it will still deflect the ball.
- Beta: It is stiffer than alpha but softer than Gamma. Two hits are needed to destroy it.

- Gamma: It is the stiffest meteor in our game. It will require three hits to be destroyed. Its proportion to other meteors will gradually increase.
- Radioactive: It destroys surrounding meteors if user hits that. It may or may not appear in each level.

The second mechanism to make the game more difficult is to change speed of ball. It will be faster by 10% with each level. In the last level, the ball will be roughly two times as fast as in the first level. Such an increase will make the game considerably more difficult.

In addition to classic levels, there is also different type of level called bonus level. This level can be activated via getting bonus-level bonuses. In this level player tries to shot gegls (kind of aliens) from outer space with a laser gun. Also gels will try to shoot user. This level ends when either all gegls die or user dies three times in this level. This level helps player to get more points in a game.

2.3. Bonuses

Another mechanism to adjust difficulty and add novelty to gameplay is bonuses. There are four different types of bonuses and they can make the game either easier or more difficult for player.

- Earn/Lose a Life: This bonus will either earn or lose a life to user. If user had only one life left and they get “Lose a Life” bonus, game will conclude and user fails.
- Replace Meteors: It will replace all destroyed meteors on the map.
- Upsize/Downsize the Paddle: It will either increase or decrease size of user’s paddle. Its increase will make receiving the ball easier for user while that will get harder with the decrease.
- Increase Ball’s Speed: Normally, the ball will get faster only with each level. However, this bonus will make it faster within a level for a limited time period.

- Activate Enemy Paddle: When a player gets this bonus, an enemy paddle and an additional ball will be activated. Enemy paddle will be controlled by the system and it tries to challenge the player.

2.4. Settings

User can modify sound settings. They can increase or decrease sound level or can decide to mute/unmute sound altogether.

3. Functional Requirements

3.1 Play Game

The game is single player. When the player runs the game, Main Menu comes first. Main Menu consists of Play Game, How to Play, Options (Settings) and Credits buttons. The game is mouse controlled. To select the buttons, the player should click on it. When the player selects “Play Game” button by clicking on, Level Selection Panel comes. After selecting the level, the game starts with an initiated game map of objects to be destroyed. Play Game button is a tool for level selection.

3.2 Select the Level

After clicking on the Play Game, Level Selection Panel appears. The player should select the level of the game before the category comes. There are nine possible levels, which are inspired from planets in the solar system and the sun itself. “Neptune” is the easiest and “Sun” is the last and hardest level. From that panel, user can choose any of them but not the ones that they have not played before (obviously except the first one).

3.3 How to Play

The player should know about some information of the game before starting. How to Play button which is located in the Main Menu informs the player about the rules, levels and mouse controlling levels of the game and types of different bonuses in the game. The player may access How to Play part from the Main Menu by clicking on the How to Play button.

3.4 Options (Settings)

Options part includes some changeable settings of the game. The player may access this button from the Main Menu by clicking on the Options button.

Changeable settings of the game are as follows:

- Adjusting the volume
- Mute / unmute the volume

3.5 Credits

Credits part includes the information about the game's developers and their email addresses. This button is located in the Main Menu which is the initial screen. The player may access the Credits part from the Main Menu by clicking on the Credits button.

4. Non Functional Requirements

4.1 Game Performance

As with any game, this app needs to achieve a decent performance and it can be measured with frame-per-second (30 FPS) rate. Any user would be annoyed if their computer cannot run the game with high performance. Our game will be lightweight compared to most modern PC games and it should run with high performance on most machines, if not all. Also, the game will be compatible with all computers.

4.2 User-Friendly Interface

The User Interface is directly connected with the player's game approach. Graphical User Interface will be tried to be prepared in the best way. In the game, number of lives, total score and game map along with the user’s paddle will be put to frame in an user friendly manner. The game will be fully screened and the player will feel a real competition environment.

4.3 Challenge of the Game

The game should include some challenge for the player to feel the competition air. There are so many elements that make it easier and also harder for the player. This is exemplified by the fact that user may have more than one lifeline and it can be increased by bonuses. There are also another bonuses; which can change the size of user’s paddle or

replace the objects to be destroyed. Between levels, game speed or velocity of ball and proportion of harder objects to softer ones will increase. These two will increase difficulty and ensure a sense of challenge for player.

4.4 Extendibility

The game can be expanded so that other bonuses can be added at any time, a multiplayer version or an option to play against computer are also other options. Moreover, ability to save the game and be able to continue from there would also be a useful addition. Additionally, new levels and scenarios can be added. Our scenario revolves around space travel across the solar system but different routes and stories are also possible additions.

5. System Models

5.1 Use-Case Model

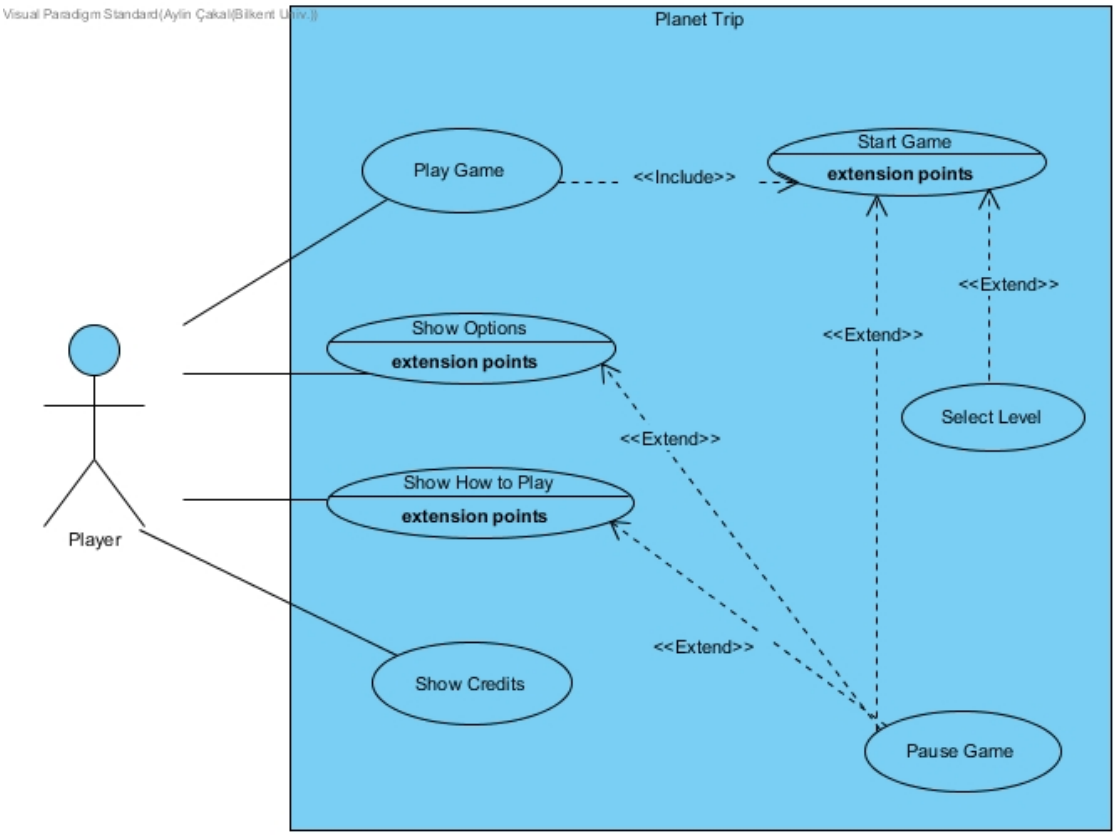


Figure 2: The Use Case Diagram

Use Case #1

Use Case: Play Game

Primary Actor: Player

Stakeholders and Interests:

- Player selects play the game.
- System creates the game and starts the game.

Pre-conditions:

- Player must be in the main menu.

Entry conditions:

- Player clicks the “Play Game” button on the main menu.

Exit conditions:

- Player chooses a level and starts playing the game.

Success Scenario Event Flow:

- Player chooses a valid level.
- The game is initialized by placing the ball, paddle and meteors on the screen.
- By giving a direction to the ball, player hits and destroys all the objects to be destroyed (e.g. stones), and does not miss the ball in this process.
- The level successfully finishes and the player moves to the next one and increase the difficulty (by increasing speed of ball and making the stones to be broken harder).

Alternative Event Flows:

- If player pauses the game:

  a. The player returns to level selection panel again by continue button.

  b. The player selects Options by options button.

  c. The player selects How to Play by how to play button.

  d. The player exits the game.

- If the player selects the start game button and current level is the last one, the results panel appears.

Use Case #2

Use Case: Pause Game

Primary Actor: Player

Stakeholders and Interests:

- Player pauses playing the game to exit the game, to get a description of how to play it, to change game settings or to continue playing it after a temporary break.

Pre-conditions:

- Player must be playing the game

Entry conditions:

- Player presses “P” button on the keyboard.

Exit conditions:

- Player clicks “How To Play” on the pause menu.
- Player clicks “Options” on the pause menu.
- Player continues playing the game by pressing “Esc” button on the keyboard.
- Player continues playing the game by pressing “Continue” button.

Success Scenario Event Flow:

- Player presses “P” button on keyboard.
- System displays pause menu and the game pauses.

Alternative Event Flows:

- If player wants to return playing the game, player presses “Esc” and system continues the game.
- If player wants to change the game options, player clicks “Options” button on pause menu and system opens options menu.
- If player wants to view how to play the game , player clicks “How to Play” button on pause menu and system opens How to Play menu.
- If player wants to exit the game , player clicks “Exit” button on pause menu and the system closes.

Use Case #3

Use Case: Show Options

Primary Actor: Player

Stakeholders and Interests:

- Player changes game settings.

Pre-conditions:

- Player must be in the main menu.
- Player must pause the game and must be in pause menu.

Entry conditions:

- Player clicks “Options” button on pause or main menu.

Exit conditions:

- Player returns to the main menu.
- Player resumes playing the game.

Success Scenario Event Flow:

- Player presses “Options” key on pause menu and system opens the options panel.

Alternative Event Flows:

- Player is in the main menu and presses the “Options” key there. Then, system opens the options panel.
- If player is in the options panel and they press the “Main menu” key on that menu, the system will go to the main menu.

Use Case #4

Use Case: Show How To Play

Primary Actor: Player

Stakeholders and Interests:

- Player gets a description of the game and instructions for how to play it by entering the “How To Play” section.

Pre-conditions:

- Player must pause the game.
- Player must be already in the main menu.

Entry conditions:

- Player clicks the “How To Play” button on the pause or main menu.

Exit conditions:

- Player returns to the main menu.
- Player resumes playing the game.

Success Scenario Event Flow:

- Player clicks the “How To Play” button on the pause menu. Consequently, system will go to the “How To Play” panel.

Alternative Event Flows:

- Player clicks the “How To Play” button on the main menu, instead of pause menu. Consequently, system will go to the “How To Play” panel.
- Player presses “Main menu” button and system will open the main menu

Use Case #5

Use Case: Show Credits

Primary Actor: Player

Stakeholders and Interests:

- Player gets information about the developers of the game.

Pre-conditions:

- Player is already on the main menu.

Entry conditions:

- Player clicks the “Credits” button on the main menu.

Exit conditions:

- Player clicks the “Main Menu” button.

Success Scenario Event Flow:

- If player wants to see the credits, user clicks “Credits” button on the main menu and system displays credits.

Alternative Event Flows:

- If player wants to return to the main menu, user clicks the “Main menu” and system goes to the main menu.

5.2 Dynamic Models

5.2.1 Starting a New Game

Scenario Name: Starting a New Game

Scenario: This process prepares the app to play a game when the player wanted to do so. First, a selection of available levels is shown to the user. If the user has never played the game before, only the first level can be chosen. Otherwise, they can choose any level up to the last level they have played before, including the latest one. After the user has chosen one of them, the game is loaded to the frame.

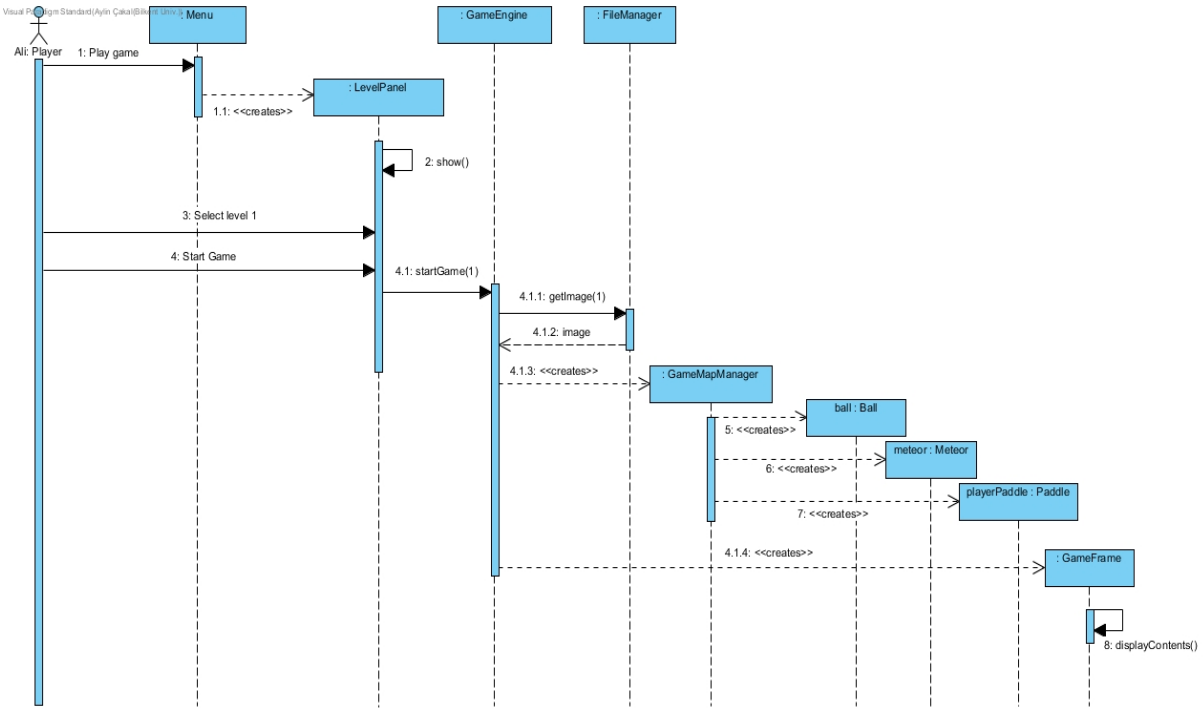


Figure 3: Starting a New Game

Description: Player presses play game button screen on the main menu. Then, available levels are shown to the user on LevelPanel. User chooses one of the level and background image, objects and game map is loaded to the game engine, according to the level chosen. The GameMapManager creates the objects of the map that are Paddle, Ball and Meteor. Then, the GameEngine creates GameFrame that provides frame. GameFrame class calls self-method which is displayContents places the ball, paddle and meteor on the screen. Background image will be loaded from file manager.

5.2.2 Gameplay Mechanism

Scenario Name: Gameplay Mechanism

Scenario: Player already selected a level and started the game. Player presses to spacebar to move the ball. After that process, ball starts to move and hits a meteor. After collision, ball reflects to a new direction. When the ball starts to fall player moves the paddle and reflects the ball moves to a new direction. This loop continues until there is no meteor left.

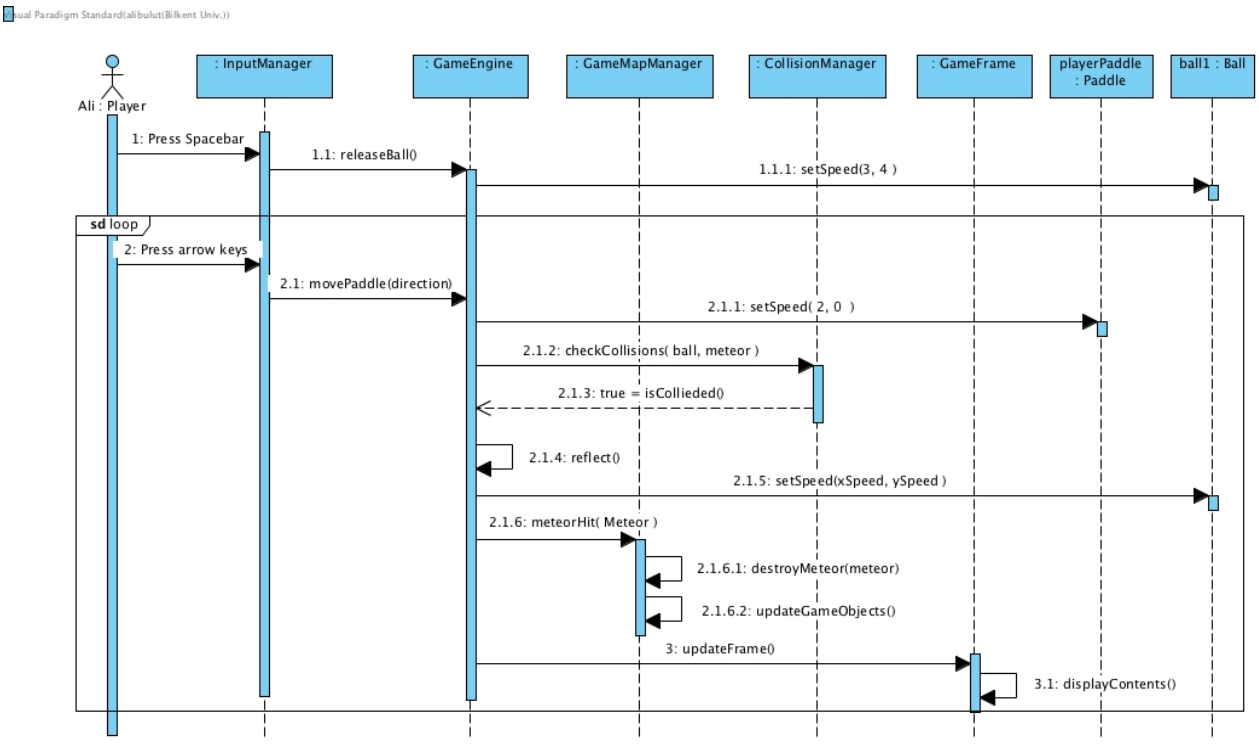


Figure 4: Gameplay

Description: Initially, level objects already loaded such as paddle, ball and meteors. Game waits for player's spacebar command. After that, GameEngine gives initial speed to ball and ball starts to move. When ball hits a meteor, ball reflects according to collision side of the meteor. When the ball started fall, player moves the paddle by pressing the arrow keys on the keyboard. After the ball hits the paddle, it reflects. The direction is determined according to angle between the middle of the paddle and ball. When the all destructible meteors hit by the ball, the level finishes successfully.

5.2.3 Acquiring a bonus which is refreshing game map

Scenario Name: Acquiring a Bonus which is refreshing game map.

Scenario: User hits a bonus and an object representing the bonus will fall down. They catch that object and the respective bonus will be triggered.

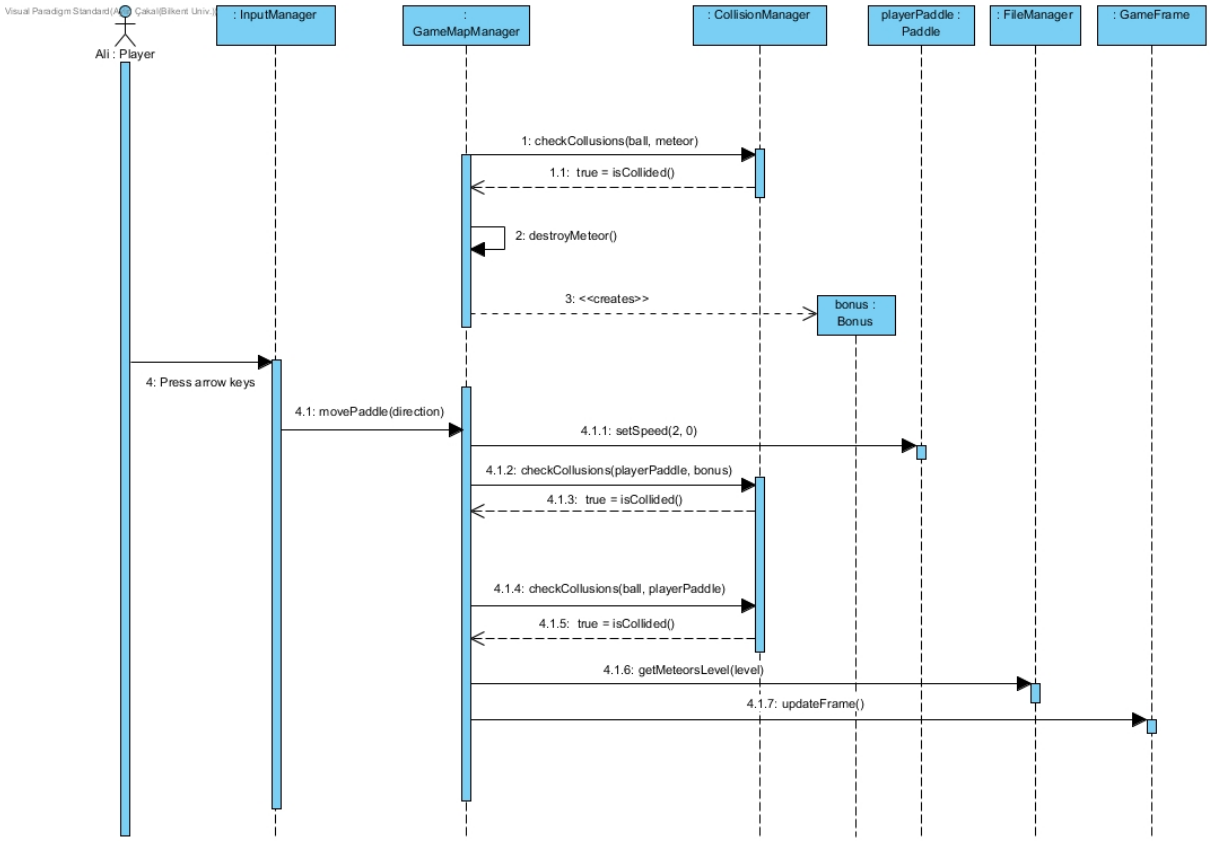


Figure 5: Acquiring a bonus which is refreshing game map

Description: In course of gameplay, CheckCollision method called constantly by GameMapManager. When there is a collision with a ball and meteor, CollisionManager notifies the GameEngine GameMapManager creates a random Bonus, in this case Refresh, which resets all of the meteors.

After bonus is created, the player moves the paddle in order to catch the bonus. InputManager carries the direction to GameMapManager. When paddle and bonus collides, CollisionManager notifies the GameEngine and starts the Bonus procedure. For Refresh bonus to work, paddle and ball needs to collide again. When GameEngine is notified, it calls for the initial map of the current level. As GameMapManager transfers the information of meteors, GameMapManager updates the GameFrame.

5.2.4 Display of How to Play / Credits

Scenario Name: Displaying How to Play / Credits

Scenario: Player is already on the main menu or has paused the game while playing. Then, they open up the how to play or credits panels by pressing the respective buttons on the screen. There, user can see a description of game rules and a brief tutorial on how to play it or credits information for the game. After that, user can return back to the main menu by pressing the “main menu” button on the screen.

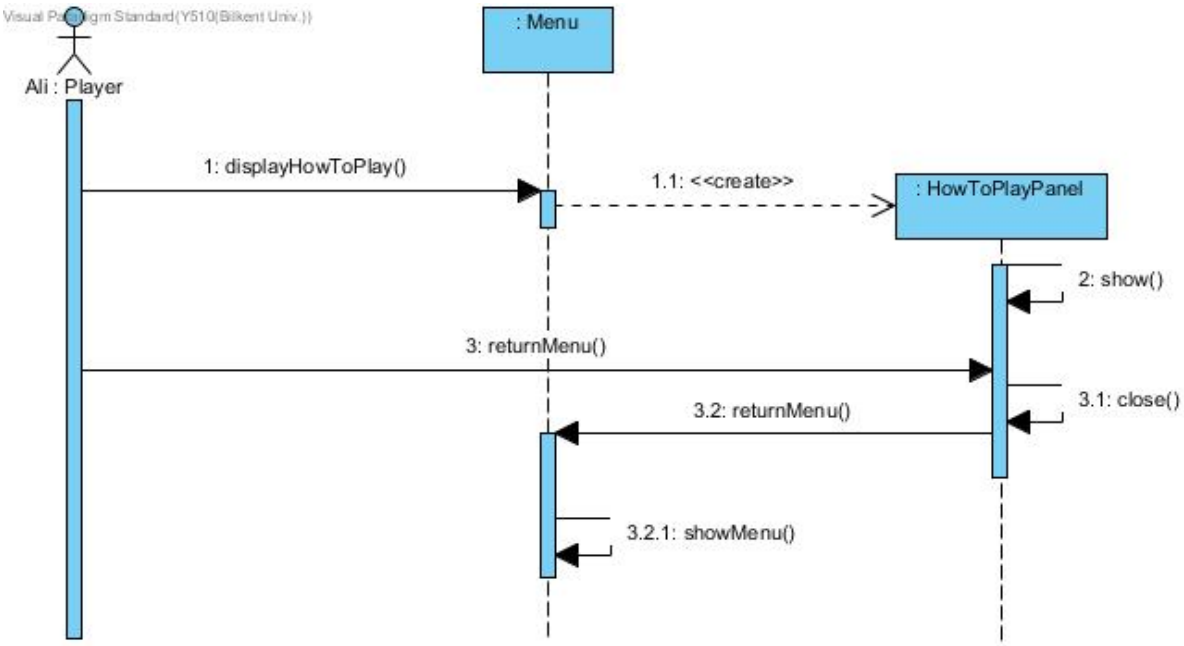


Figure 6: How to Play

Description: When user presses the "How to Play" section, a new panel, which is HowToPlayPanel, is created and loaded. To that frame, a text describing the game rules and an introductory tutorial is loaded and shown to player. Then, user can decide to return back to the main menu. If so, user can press the main menu button. This action will enact returnMenu and showMenu methods, respectively. These two methods will show the main menu, instead of the how to play section, on screen. Also, HowToPlayPanel is closed.

How To Play and Credits are represented under the same diagram because the underlying processes are quite similar. The only difference is in credits, only credits text is loaded to the frame instead of a description of game rules.

5.2.5 Options

Scenario Name: Changing sound settings

Scenario: User wants to adjust sound settings of the game. For that, they open the options panel either from pause menu or from the main menu. There, user display current settings and modify them. One of the available modifications is muting the sound altogether. After that, they apply changes and return to the main menu.

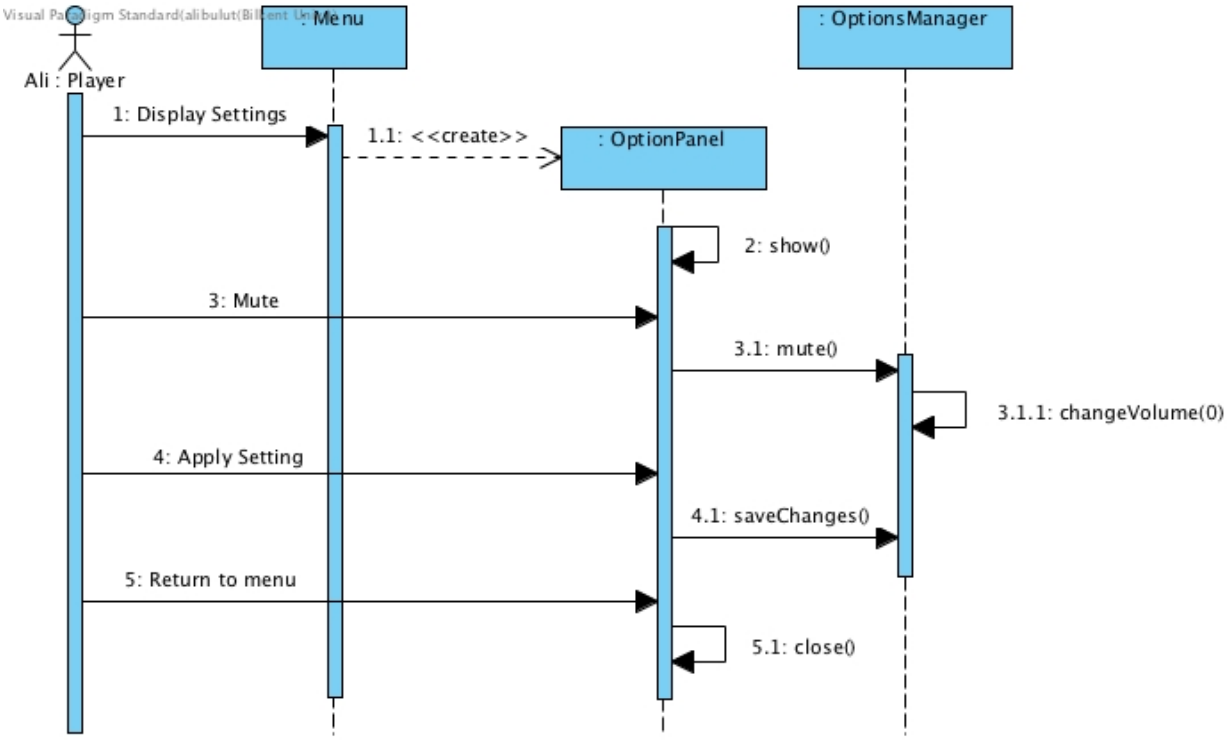


Figure 7: Options

Description: User decides to show the settings and a new frame to display them is created. Current settings are loaded to that frame, OptionPanel. Then, user adjusts the sound level. OptionsManager adjusts the sound accordingly. After that, user saves the settings. When they are finished with changing sound settings, user returns back to the main menu. For that, OptionPanel is closed and main menu is displayed instead.

5.2.2 Activity Diagram

The following diagram describes the flow of gameplay.

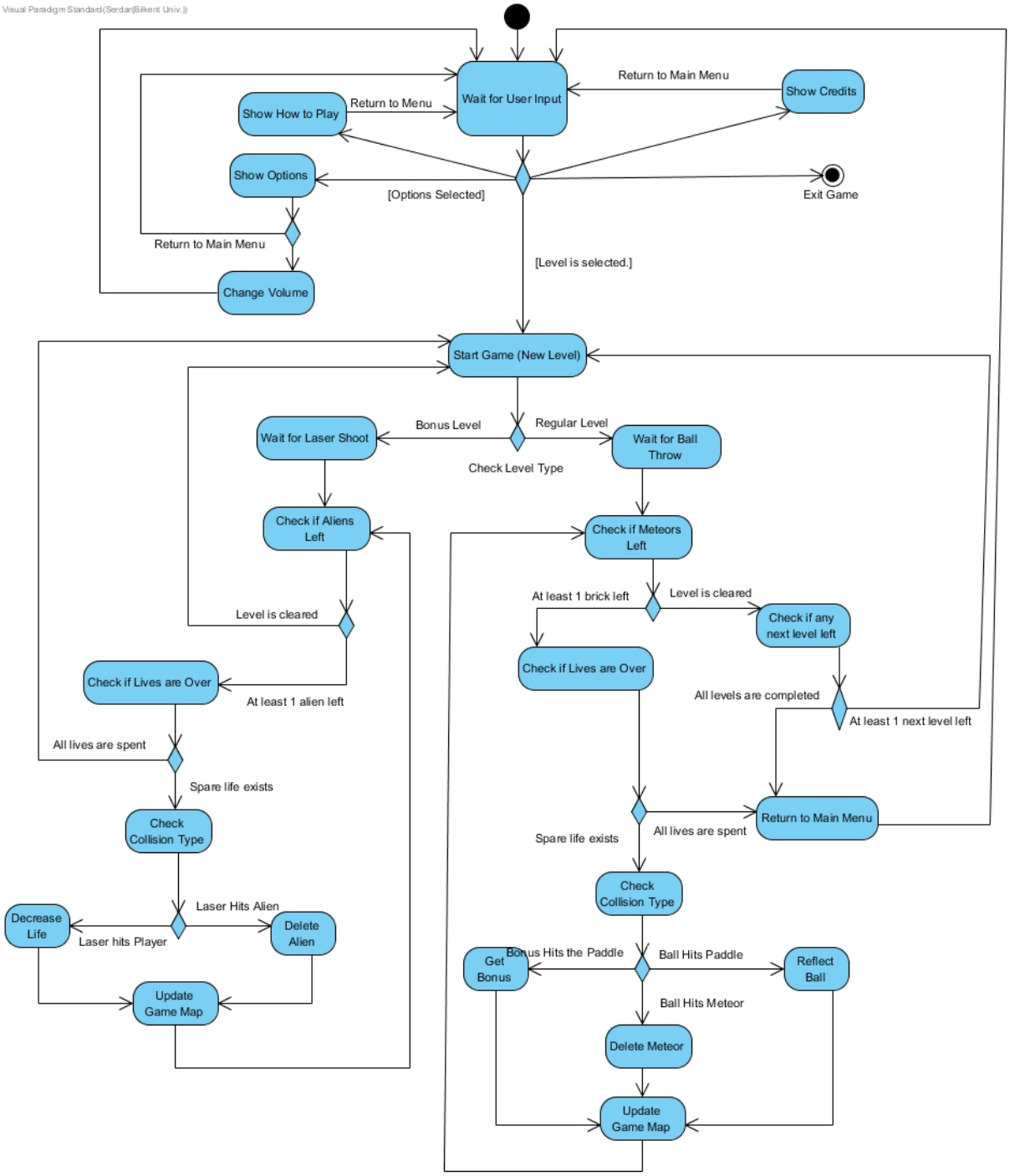


Figure 8: Activity Diagram for Planet Trip Game

At the beginning, the system waits the user to input. At this point, the user (player) may start the game, show credits, change options, show how to play, or exit the game. If the

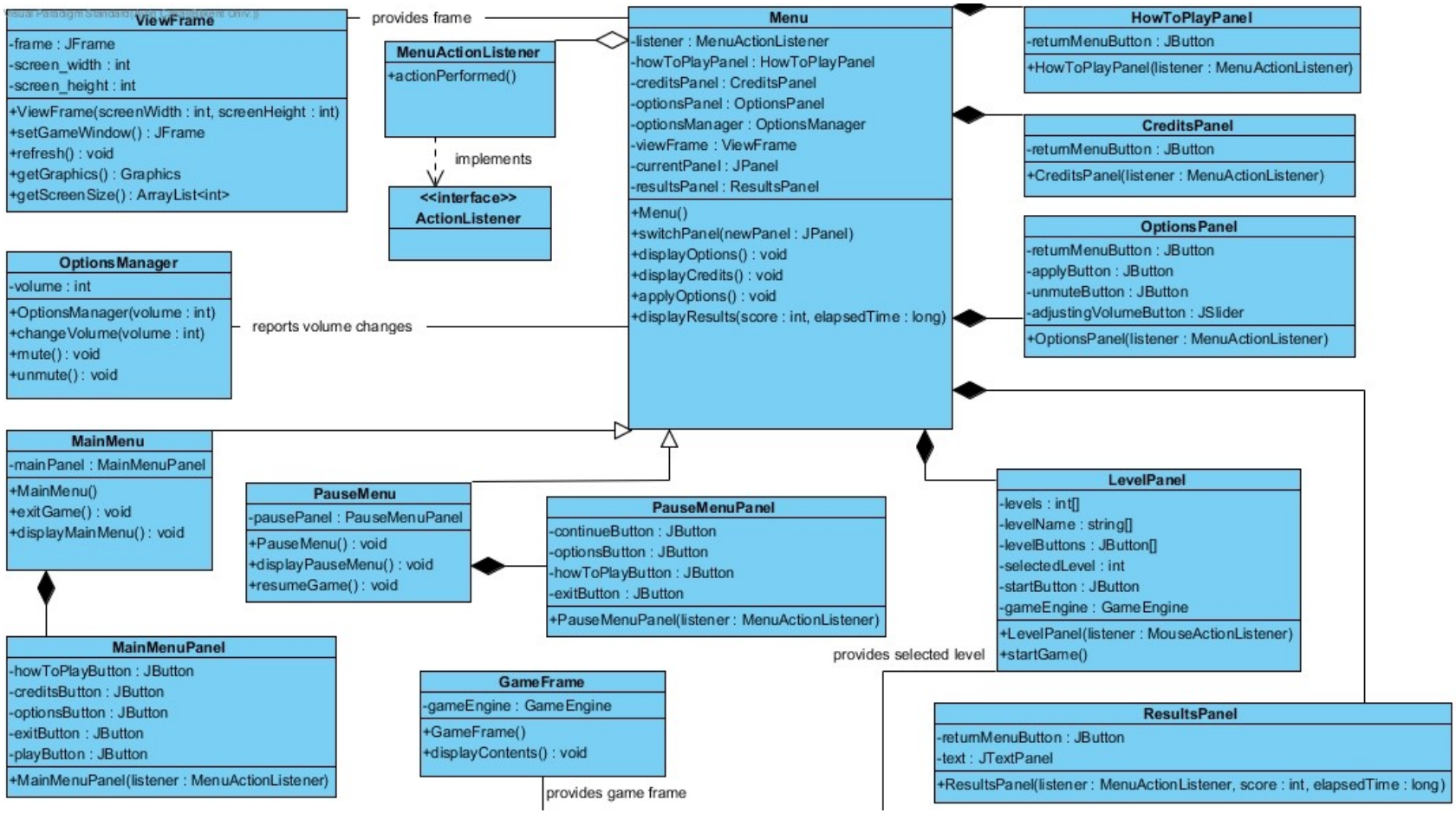
user starts the game, the system waits the user for the level selection. For each selection, the next thing is setting selected level of the game. Then, the system initializes the game.
When the game starts, the system will wait for user to throw the ball. Throughout the game, system will check whether the map is cleared and player has lives left. If there is no meteor left, next level will be opened until the last one. In that case, game is won and player return to main menu. If player has no lives left, game is lost player returns to main menu.

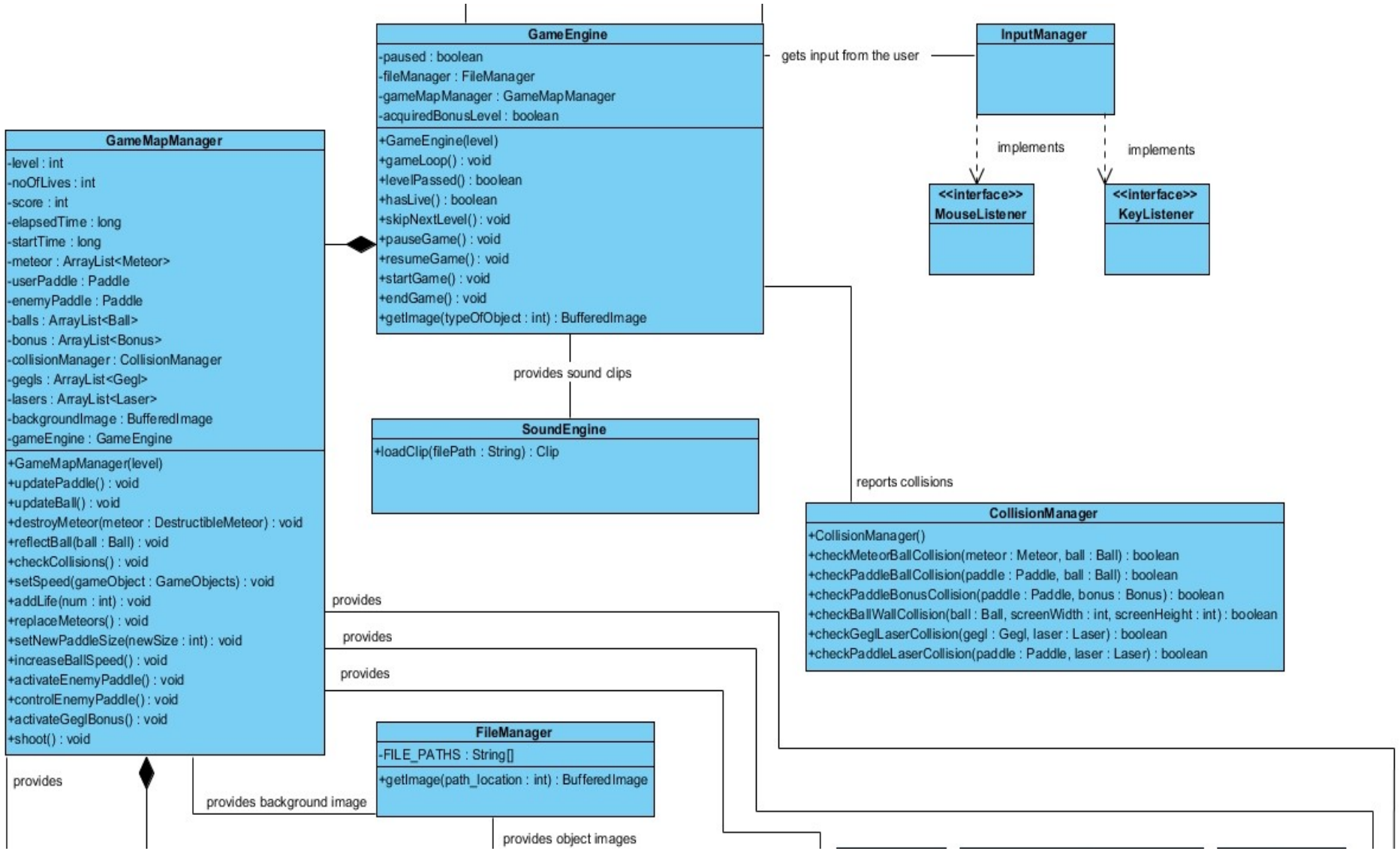
There are two types of levels. According to the bonus get on the previous level, user can unlock the bonus level which influenced by the Space Invaders game. Since this game has different mechanics, there will be no reflections. User can lose a life if got hit by a laser shot. In original levels, a ball hit to our paddle will be reflected instead.

In ordinary levels the system will check for collisions if a bonus hits the paddle or ball hits to meteors or to the paddle. If there is a ball collision, ball will be reflected and unless it hit the paddle, the collided meteor will be destroyed. If it is a bonus and paddle collision, player will get the corresponding bonus. The game map is updated if one of the meteors is destroyed as result of collision.

5.3 Object and Class Model

The following diagram describes our app's object and class model.


ViewFrame
-frame : JFrame
-screen_width : int
-screen_height : int
+ViewFrame(screenWidth : int, screenHeight : int)
+setGameWindow() : JFrame
+refresh() : void
+getGraphics() : Graphics
+getScreenSize() : ArrayList<int>
provides frame
MenuActionListener
+actionPerformed()
implements
<<interface>>
ActionListener
Menu
-listener : MenuActionListener
-howToPlayPanel : HowToPlayPanel
-creditsPanel : CreditsPanel
-optionsPanel : OptionsPanel
-optionsManager : OptionsManager
-viewFrame : ViewFrame
-currentPanel : JPanel
-resultsPanel : ResultsPanel
+Menu()
+switchPanel(newPanel : JPanel)
+displayOptions() : void
+displayCredits() : void
+applyOptions() : void
+displayResults(score : int, elapsedTime : long)
HowToPlayPanel
-returnMenuButton : JButton
+HowToPlayPanel(listener : MenuActionListener)
CreditsPanel
-returnMenuButton : JButton
+CreditsPanel(listener : MenuActionListener)
OptionsPanel
-returnMenuButton : JButton
-applyButton : JButton
-unmuteButton : JButton
-adjustingVolumeButton : JSlider
+OptionsPanel(listener : MenuActionListener)
OptionsManager
-volume : int
+OptionsManager(volume : int)
+changeVolume(volume : int)
+mute() : void
+unmute() : void
reports volume changes
MainMenu
-mainPanel : MainMenuPanel
+MainMenu()
+exitGame() : void
+displayMainMenu() : void
PauseMenu
-pausePanel : PauseMenuPanel
+PauseMenu() : void
+displayPauseMenu() : void
+resumeGame() : void
PauseMenuPanel
-continueButton : JButton
-optionsButton : JButton
-howToPlayButton : JButton
-exitButton : JButton
+PauseMenuPanel(listener : MenuActionListener)
LevelPanel
-levels : int[]
-levelName : string[]
-levelButtons : JButton[]
-selectedLevel : int
-startButton : JButton
-gameEngine : GameEngine
+LevelPanel(listener : MouseActionListener)
+startGame()
provides selected level
MainMenuPanel
-howToPlayButton : JButton
-creditsButton : JButton
-optionsButton : JButton
-exitButton : JButton
-playButton : JButton
+MainMenuPanel(listener : MenuActionListener)
GameFrame
-gameEngine : GameEngine
+GameFrame()
+displayContents() : void
provides game frame
ResultsPanel
-returnMenuButton : JButton
-text : JTextPanel
+ResultsPanel(listener : MenuActionListener, score : int, elapsedTime : long)


GameEngine
-paused : boolean
-fileManager : FileManager
-gameMapManager : GameMapManager
-acquiredBonusLevel : boolean
+GameEngine(level)
+gameLoop() : void
+levelPassed() : boolean
+hasLive() : boolean
+skipNextLevel() : void
+pauseGame() : void
+resumeGame() : void
+startGame() : void
+endGame() : void
+getImage(typeOfObject : int) : BufferedImage
gets input from the user
InputManager
implements
implements
<<interface>>
MouseListener
<<interface>>
KeyListener
GameMapManager
-level : int
-noOfLives : int
-score : int
-elapsedTime : long
-startTime : long
-meteor : ArrayList<Meteor>
-userPaddle : Paddle
-enemyPaddle : Paddle
-balls : ArrayList<Ball>
-bonus : ArrayList<Bonus>
-collisionManager : CollisionManager
-gegls : ArrayList<Gegl>
-lasers : ArrayList<Laser>
-backgroundImage : BufferedImage
-gameEngine : GameEngine
+GameMapManager(level)
+updatePaddle() : void
+updateBall() : void
+destroyMeteor(meteor : DestructibleMeteor) : void
+reflectBall(ball : Ball) : void
+checkCollisions() : void
+setSpeed(gameObject : GameObjects) : void
+addLife(num : int) : void
+replaceMeteors() : void
+setNewPaddleSize(newSize : int) : void
+increaseBallSpeed() : void
+activateEnemyPaddle() : void
+controlEnemyPaddle() : void
+activateGeglBonus() : void
+shoot() : void
provides sound clips
SoundEngine
+loadClip(filePath : String) : Clip
reports collisions
CollisionManager
+CollisionManager()
+checkMeteorBallCollision(meteor : Meteor, ball : Ball) : boolean
+checkPaddleBallCollision(paddle : Paddle, ball : Ball) : boolean
+checkPaddleBonusCollision(paddle : Paddle, bonus : Bonus) : boolean
+checkBallWallCollision(ball : Ball, screenWidth : int, screenHeight : int) : boolean
+checkGeglLaserCollision(gegl : Gegl, laser : Laser) : boolean
+checkPaddleLaserCollision(paddle : Paddle, laser : Laser) : boolean
provides
provides
provides
provides
FileManager
-FILE_PATHS : String[]
+getImage(path_location : int) : BufferedImage
provides background image
provides object images

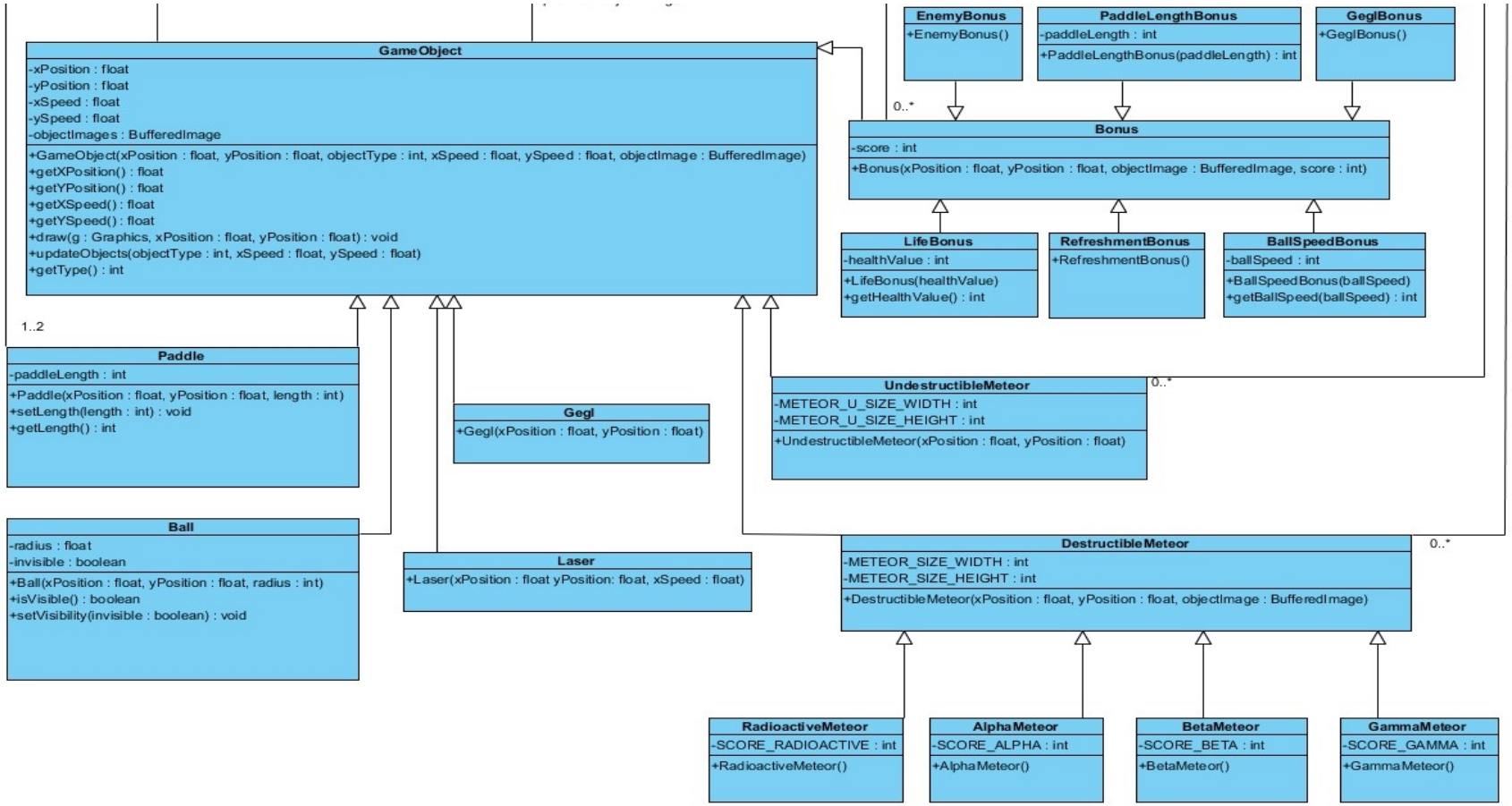


Figure 9: Object and Class Diagram for Planet Trip

ViewFrame:

This class controls the frames of the game. According to situation of the game, this class calls related subclass and then gives user a frame for a moment.

Menu:

This class manages its subclasses MainMenu and PauseMenu.

HowToPlayPanel:

HowToPlayPanel class controls the button of the how to play menu. This class listens to the button.

CreditsPanel:

CreditsPanel class controls the button of the credits menu. This class listens to the button.

OptionsPanel:

OptionsPanel class controls the buttons of the options menu. This class listens to the buttons.

OptionsManager:

OptionsManager class controls the volume of the sounds in the game. The user can adjust the volume by this class.

MainMenu:

MainMenu class provides MainMenuPanel. By this class, the user can exit the game and display main menu.

MainMenuPanel:

MainMenuPanel class controls the buttons of the main menu. This class listens to the buttons.

PauseMenu:

PauseMenu class provides PauseMenuPanel. By this class, the user can pause game and display pause menu.

PauseMenuPanel:

PauseMenuPanel class controls the buttons of the pause menu. This class listens to the buttons.

LevelPanel:

LevelPanel class provides all level selections to the user and listens selection of the user.

GameFrame:

GameFrame class provides game frame to the game engine.

ResultsPanel:

ResultsPanel provides information about the game result. It shows elapsedTime and score.

GameEngine Class:

This class is our main class to handle the game. This class controls events that happened in the gameplay. Game objects, controls, frames, user actions and collisions be controlled by this class.

InputManager Class:

Purpose of this class gets inputs from the player and then reports this actions to the GameEngine class.

FileManager Class:

This class gets needed files from the computer's hard drive and sends back to the GameEngine class. The files will contain game objects' and levels' background images.

CollisionManager Class:

This class controls if whether any two different type object has collided or not. If there is a collision this class reports collision and its type.

OptionsManager Class:

This class keeps sound settings and manages game sounds according to the adjustments.

GameMapManager Class:

This class keeps level of the game. According to the level it gets meteors and during the game it deletes destructed meteors. Also this class generates random bonuses and puts them into random meteors.

GameObject Class:

GameObject class is the main class which gathers all the common attributes of the game objects. This class keeps the position and speed of the objects. This class also changes position of the objects according to objects speed.

Paddle:

This is a subclass of GameObject class. This class keeps length of the paddles and changes paddle length when bonus has received.

Ball:

This is a subclass of GameObject class. This class keeps radius of the ball.

Bonus:

This class has 6 subclasses, also keeps track of how many bonuses used in level.

Enemy Bonus:

Enemy Bonus creates an enemy paddle when collected.

Paddle Length Bonus:

Length Bonus for expanding the paddle.

Life Bonus:

Gives player an extra life.

Gegl Bonus:

Unlocks the bonus level to be played next.

Refreshment Bonus:

Refreshes all of the meteors in the level map.

Ball Speed Bonus:

Increases the ball speed, causes it to move faster.

Gegl:

Enemy objects in the bonus level. Shoots lasers and dies immediately when got hit by lasers.

Laser:

Our weapon in the bonus level, also can be used by enemies and hit us.

DestructibleMeteor:

This class keeps destructible meteor types and if a collision occurs between ball and a meteor, this class whether changes type of meteor or destroy meteor(s).

Alpha Meteor:

Alpha meteors can be destroyed by getting 1 hit.

Beta Meteor:

This type of meteors require 2 hits to be destroyed.

Gamma Meteor:

Gamma meteors will be destroyed after getting 3 hits.

Radioactive Meteor:

Radioactive meteors will blow off the surrounding meteors when destroyed. Requires 1 hit.

UndesructibleMeteor:

This class keeps undestructable meteors. Collisions does not affect this class.

5.5 User Interface: Navigational Path and Screen Mock-ups

5.5.1 Navigational Path

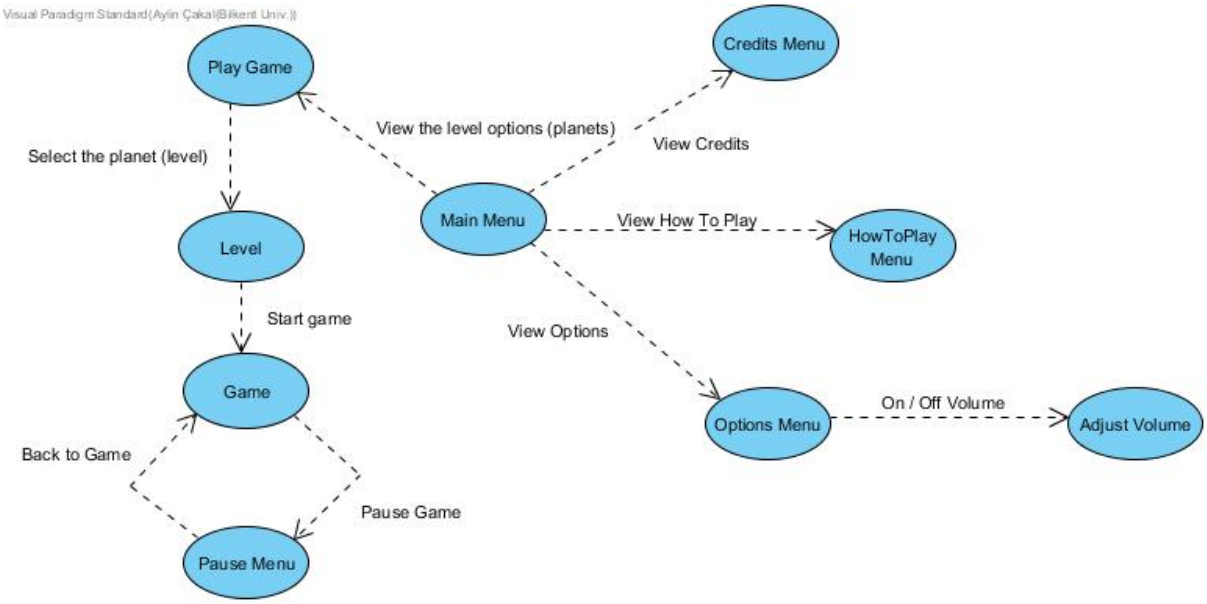
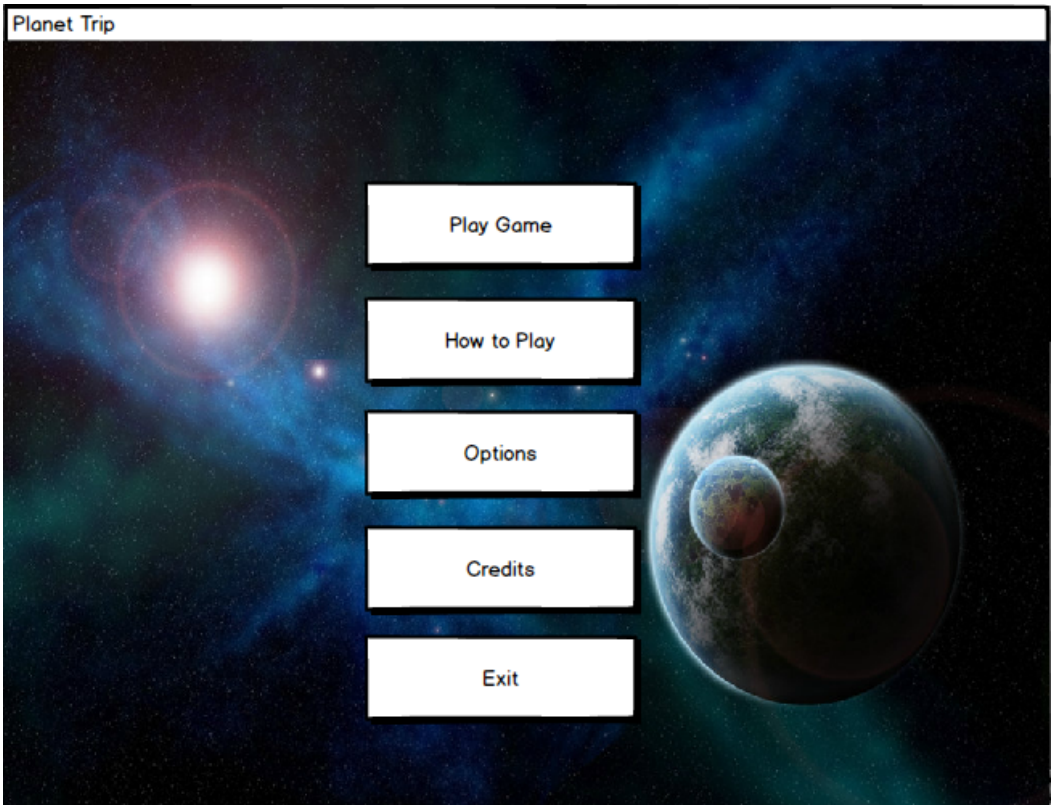


Figure 10: The Navigational Path

5.5.2 Screen Mockups and Icons

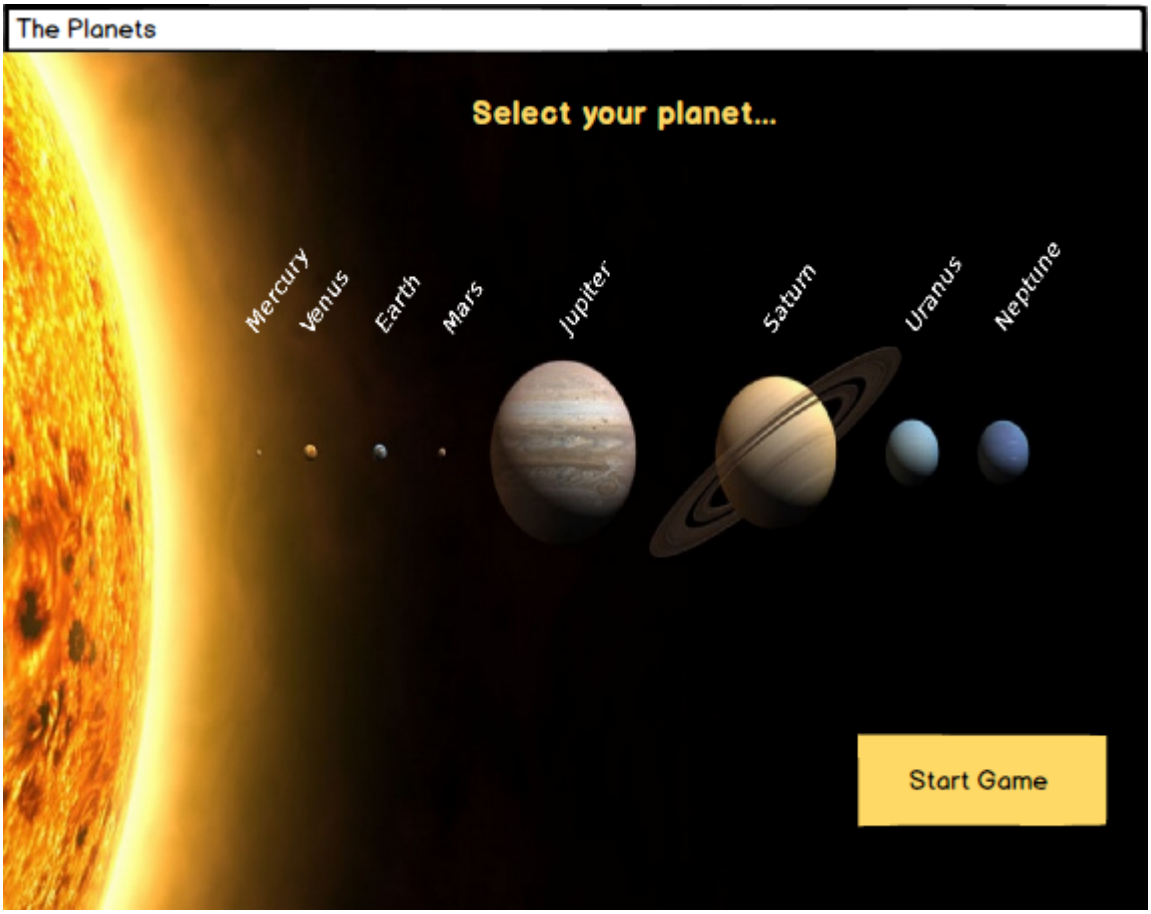
Main Menu Mockup Screen

The game will have simple game menu with a basic background.



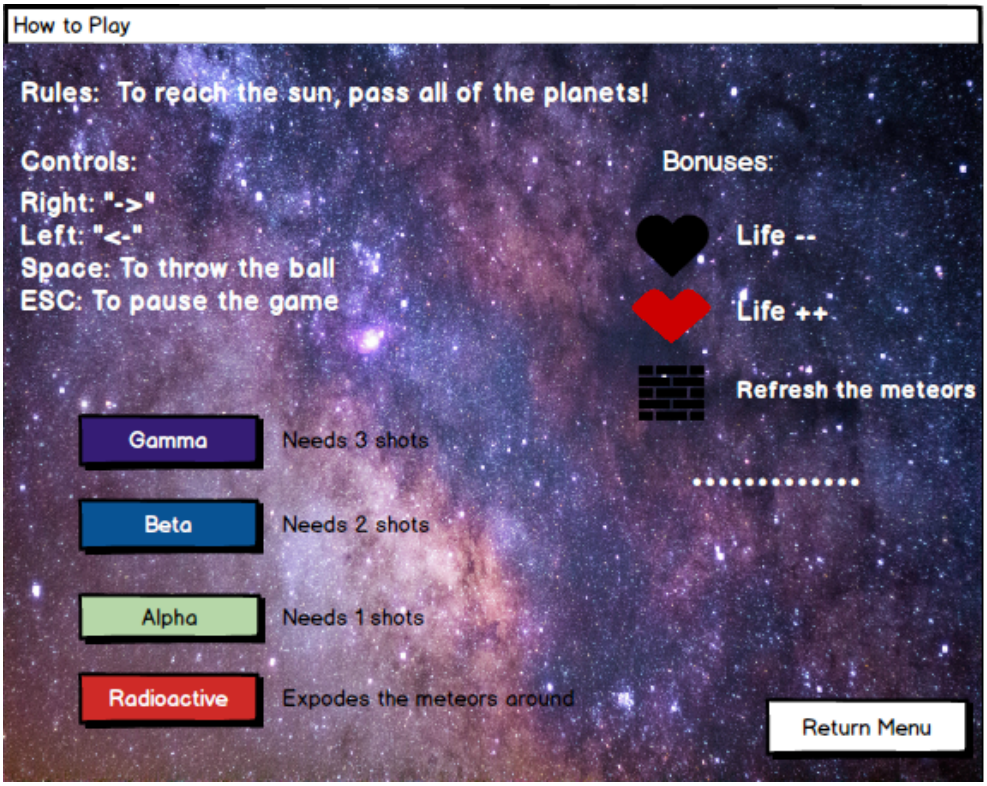
Level Selection Mockup Screen

This screen provides the number of the planets for level selection to the player. Player clicks the name of the planet and then clicks “Start Game”.



How to Play Mockup Screen

This screen provides the information about how to play “Planet Trip” and some icons of the game.

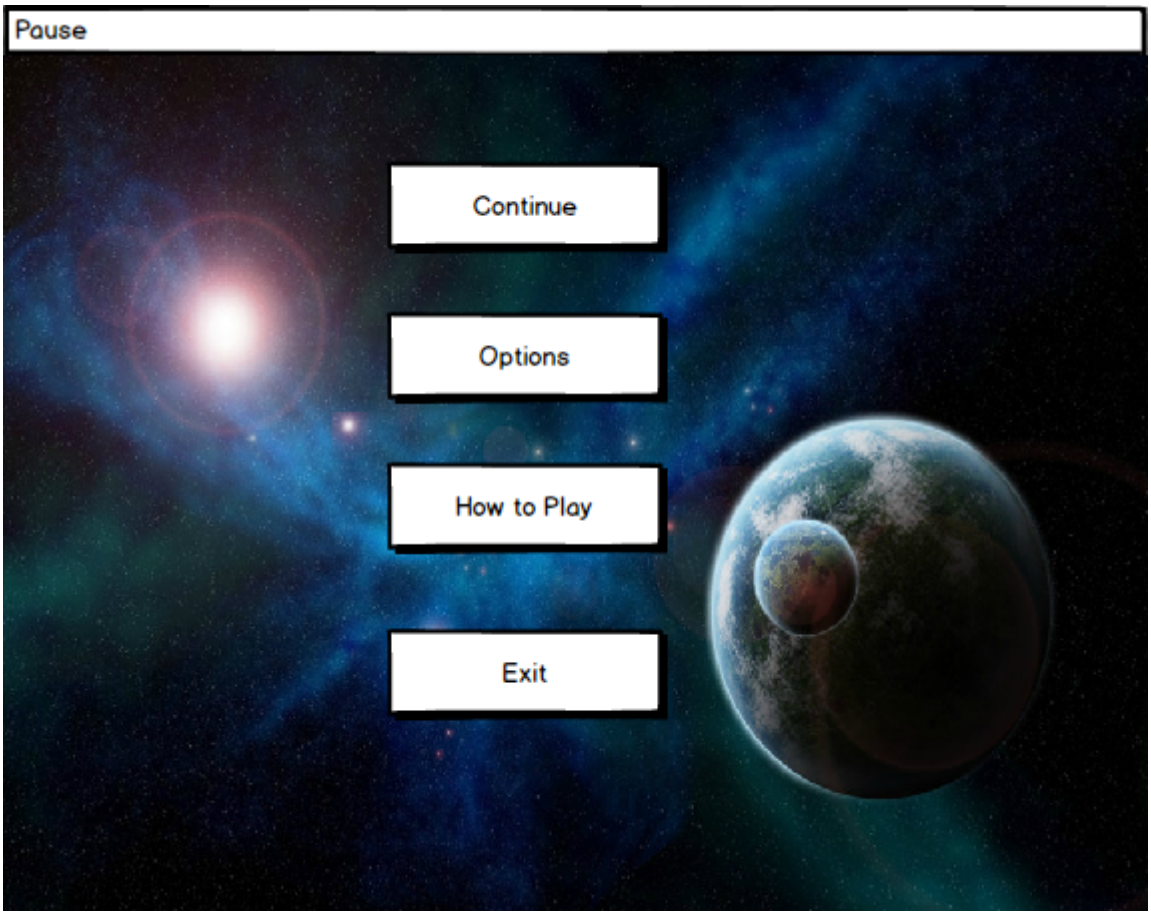


Game Mockup Screen

Game screen includes the name of the current planet at the bottom, elapsed time on the top left-hand corner, the score panel at the left bottom, the number of life at the right bottom and the paddle and the meteor map as shown in Game overview part (Figure 1).

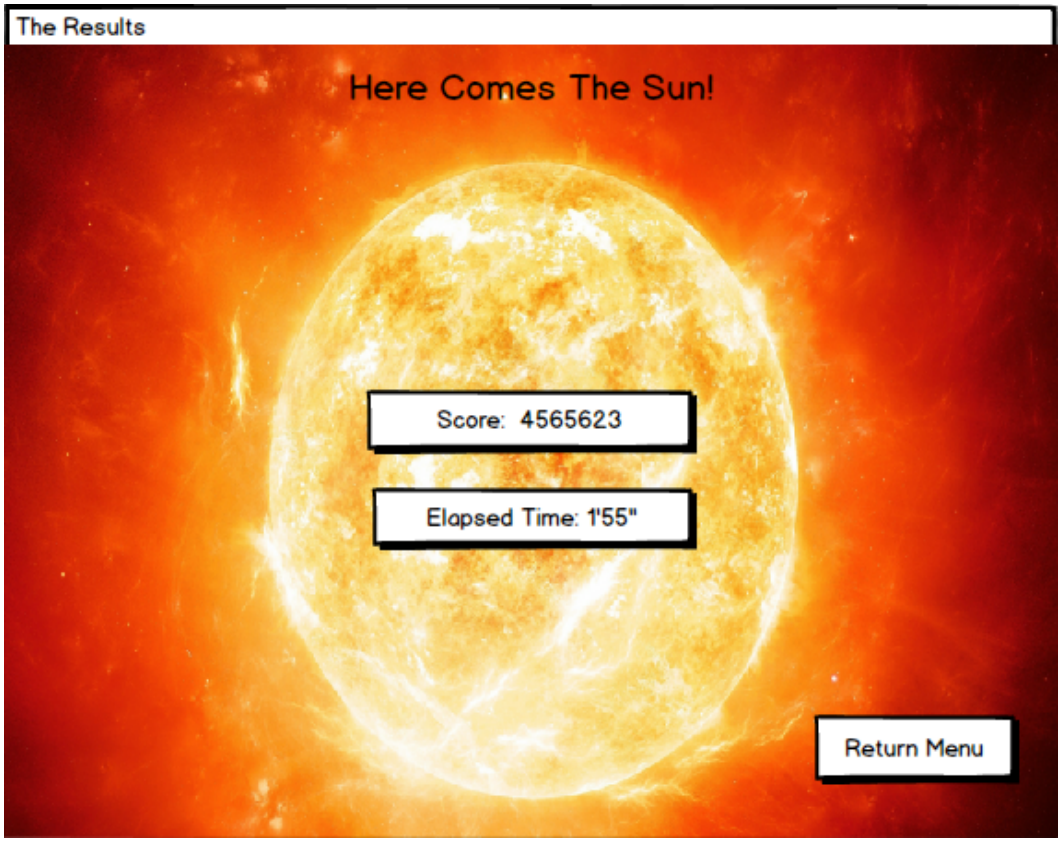
Pause Panel Mockup Screen

This screen provides the pause menu when the player pause the game. To pause game, the player must press “esc” button.



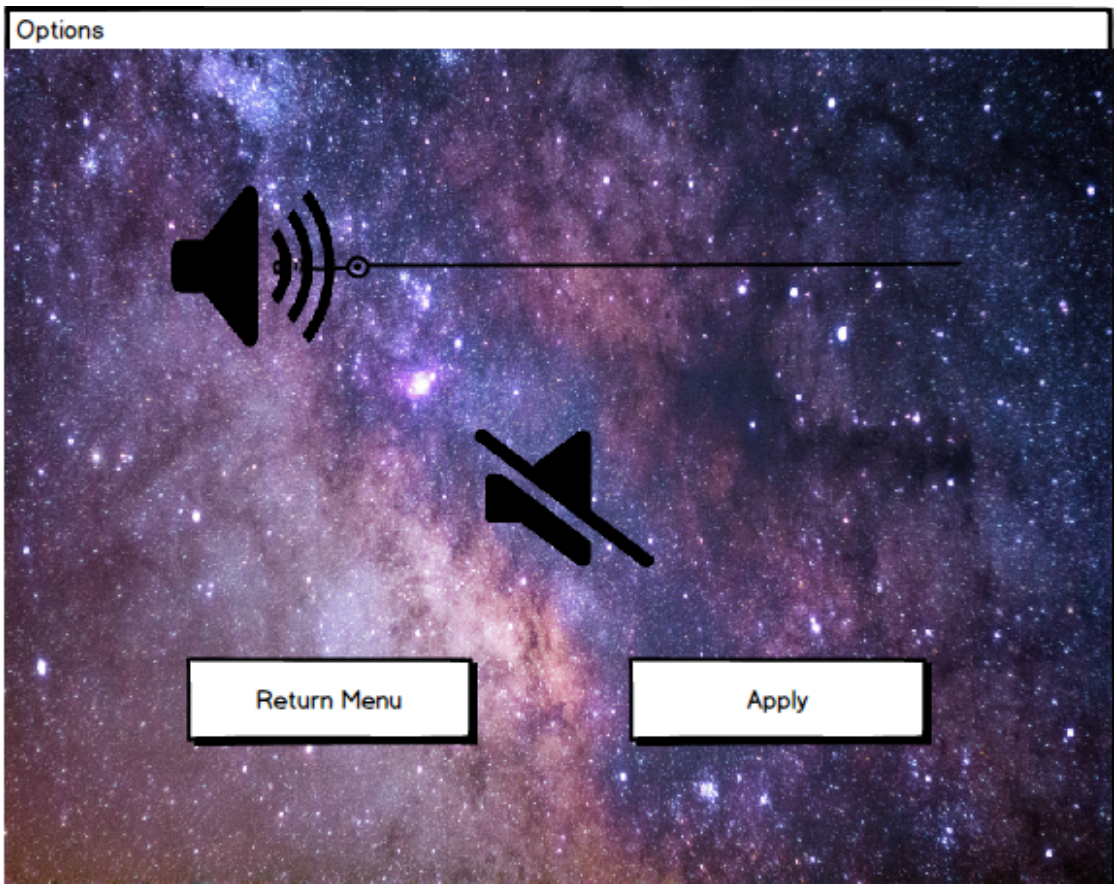
Results Panel Mockup Screen

This screen includes the total score, elapsed time during the game and also return menu button to return main menu. This screen is displayed after finishing all planets (levels).



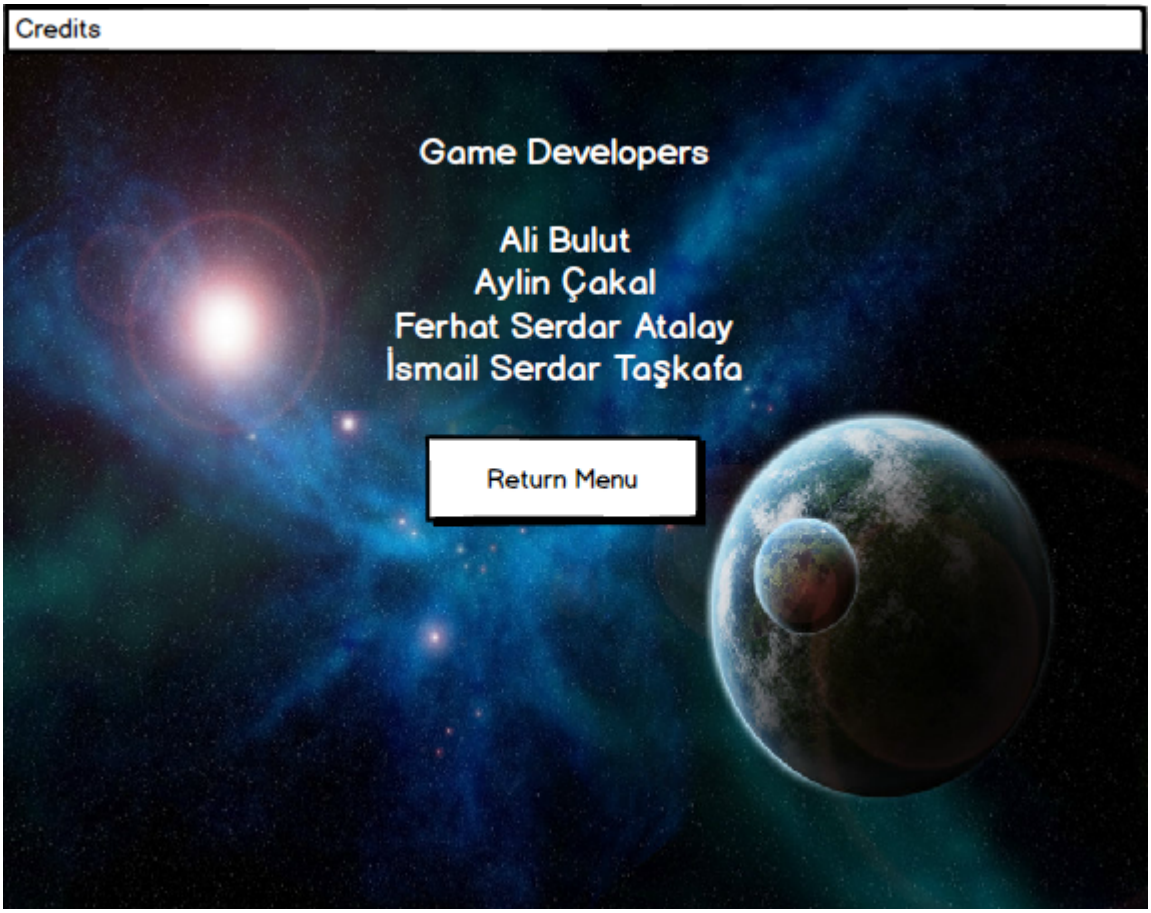
Options Mockup Screen

This screen allows the player to adjust the volume down or up and mute or unmute.



Credits Mockup Screen

This screen has information about this game developers.



6. Improvement Summary

In the second iteration, the main improvement is adding a new feature. That is introducing a new bonus level. In order to accomplish that, we added a different type of meteor, which is bonus meteor. Also, we fixed the problems with our diagrams and with our report in general.

- Our new functional requirement is during gameplay, if player hits one of these meteors, the bonus level is unlocked. That level differs from our original levels by its gameplay. Instead of moving and giving direction to a ball, the player will fire bullets. They will try to hit the meteors using these bullets and avoid getting hit by them in return.
- In our Use Case diagrams, we fixed the naming problems.
- Like in our UC diagrams, object models also had naming problems and they are fixed. Also, we added proper associations and cardinalities. Finally, we modified the object model to add the classes necessary for adding our new functional requirement, i.e. the bonus level.
- We modified the sequence diagrams to describe the actual scenarios in the game, rather than using an arbitrary ordering. Also, we fixed the problems with flow of data in our sequence diagrams.
-

7. Glossary & References

http://ww7.freearcade.com/Javanoid.jav/Javanoid.html?s=popular

https://upload.wikimedia.org/wikipedia/commons/thumb/c/cb/Planets2013.svg/2000px-Planets2013.svg.png

https://www.youtube.com/watch?v=Qwe79VJ3e30

https://www.youtube.com/watch?v=UCOlSsLp0EE