Bilkent University

Object Oriented Software Engineering Project

Space Despot

Final Report

Cihangir Mercan, Çağdaş Han Yılmaz, Fırat Sivrikaya, Gökçe Şakir Özyurt

Supervisor: Prof. Dr. Uğur Doğrusöz

Table of Contents

A Appendix

⍰

Introduction

1.1 Purpose of the system

Purpose of Space Despot is to develop a game that will entertain the player by using our knowledge that we learned from our department. We are inspired by the classics Space Impact and Asteroids [1] [2]. Developing this system will help us to understand the notion of object oriented software, enhance our current skills on Java and help us to practice the work we will be doing after we graduate.

1.2 Design goals

- Game and its development will include the principles we learned so far from our department.

- Optimizing the system will be one of our most important goals.

Because, players doesn't want to play a game which freezes even though it doesn't require server connection during the game play. Game elements should enter or exit the screen without any visible errors or freezes in the game. Every action that takes place in the system mustn't affect game play experience of player. We must also decrease the input lag as much as possible.

- We will try to design the best user interface and graphics possible with the current Java libraries. User interface will be user friendly and good looking. We will try to design the images of objects through Adobe Photoshop to increase the user's game play experience.

- Space Despot will be expendable, in terms of its content, mechanics, interface and graphics. Using Object Oriented approach it is easier to enhance the system over time by doing reverse engineering and then adding new features or enhancing the existing ones.

- Since we are developing a game, we should also handle the game design, not in computer science context, but in context of Psychology. While designing the game, we will try to obey the key aspects of the game design as much as possible. We will try to implement the four essential characteristics of the game design and

Flow Theory, to provide a better game play experience [3] [4].

Trade Offs

- Performance vs. Memory: In order to increase the game play experience, user must not see any kinds of graphical or mechanical (system's) errors. In order to prevent those errors and increase performance, we won't be concerning on keeping the memory requirements low. We must check for collisions, calculate damages and display the game efficiently and error-free, to do this we will use memory a lot more.

- Functionality vs. Usability: We are thinking to develop a game has a lot of functionalities. For beginners usability will be relatively low because of the game's functionalities. Player will find it difficult at first to understand how to do use these functionalities. However, after playing for a while player will get used to the game and hence its functionalities.

2.Software Architecture

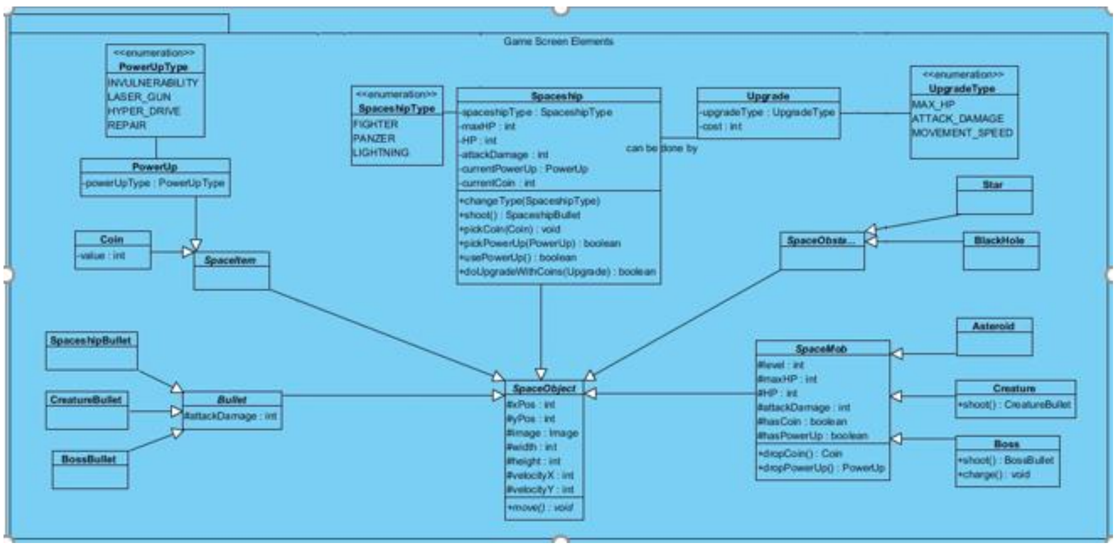
Our software is a single game with numerous object interactions. While we are designing the game, our duty will be to come up with a simple design that has the lowest coupling so that implementation will be easier.

2.1 Subsystem Decomposition

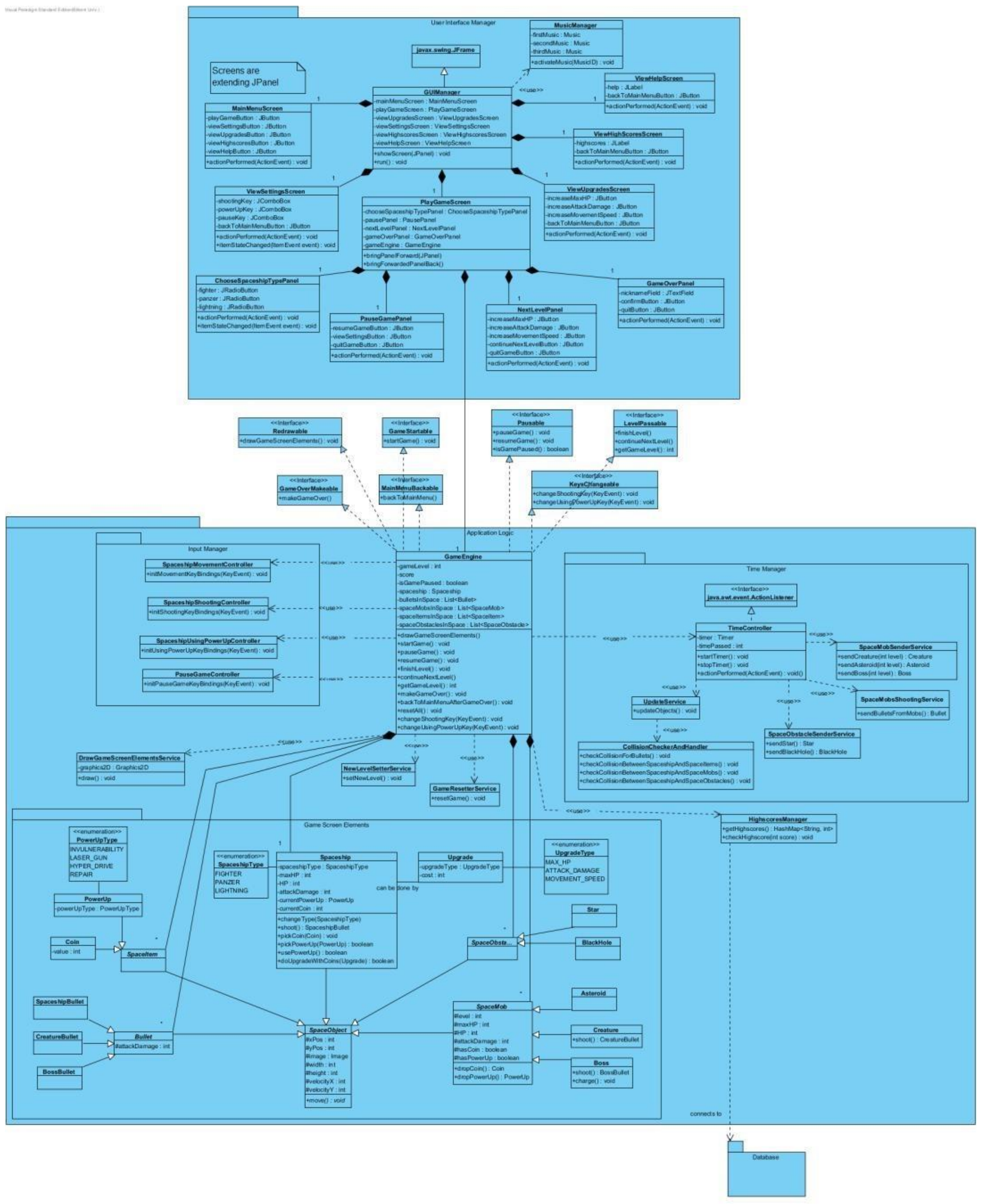
For Space Despot, we choose the three-tier architectural style which will organize our subsystems into three layers: GUI, Application Logic and Storage (Figure 1).
The GUI layer will have classes that are responsible for providing an interface between the user and the system.
The Application Logic layer will includes all control and entity objects.
The Storage layer will take care of the storage of information in the database.

3.Subsystem Services

3.1



3.1 Detailed Object Design


User Interface Manager
Screens are extending JPanel
javax.swing.JFrame
MusicManager
-firstMusic : Music
-secondMusic : Music
-thirdMusic : Music
+activateMusic(MusicID) : void
<<use>>
GUIManager
-mainMenuScreen : MainMenuScreen
-playGameScreen : PlayGameScreen
-viewUpgradesScreen : ViewUpgradesScreen
-viewSettingsScreen : ViewSettingsScreen
-viewHighscoresScreen : ViewHighscoresScreen
-viewHelpScreen : ViewHelpScreen
+showScreen(JPanel) : void
+run() : void
MainMenuScreen
-playGameButton : JButton
-viewSettingsButton : JButton
-viewUpgradesButton : JButton
-viewHighscoresButton : JButton
-viewHelpButton : JButton
+actionPerformed(ActionEvent) : void
ViewHelpScreen
-help : JLabel
-backToMainMenuButton : JButton
+actionPerformed(ActionEvent) : void
ViewHighScoresScreen
-highscores : JLabel
-backToMainMenuButton : JButton
+actionPerformed(ActionEvent) : void
ViewSettingsScreen
-shootingKey : JComboBox
-powerUpKey : JComboBox
-pauseKey : JComboBox
-backToMainMenuButton : JButton
+actionPerformed(ActionEvent) : void
+itemStateChanged(ItemEvent event) : void
PlayGameScreen
-chooseSpaceshipTypePanel : ChooseSpaceshipTypePanel
-pausePanel : PausePanel
-nextLevelPanel : NextLevelPanel
-gameOverPanel : GameOverPanel
-gameEngine : GameEngine
+bringPanelForward(JPanel)
+bringForwardedPanelBack()
ViewUpgradesScreen
-increaseMaxHP : JButton
-increaseAttackDamage : JButton
-increaseMovementSpeed : JButton
-backToMainMenuButton : JButton
+actionPerformed(ActionEvent) : void
ChooseSpaceshipTypePanel
-fighter : JRadioButton
-panzer : JRadioButton
-lightning : JRadioButton
+actionPerformed(ActionEvent) : void
+itemStateChanged(ItemEvent event) : void
PauseGamePanel
-resumeGameButton : JButton
-viewSettingsButton : JButton
-quitGameButton : JButton
+actionPerformed(ActionEvent) : void
NextLevelPanel
-increaseMaxHP : JButton
-increaseAttackDamage : JButton
-increaseMovementSpeed : JButton
-continueNextLevelButton : JButton
-quitGameButton : JButton
+actionPerformed(ActionEvent) : void
GameOverPanel
-nicknameField : JTextField
-confirmButton : JButton
-quitButton : JButton
+actionPerformed(ActionEvent) : void
1
<<Interface>> Redrawable
+drawGameScreenElements() : void
<<Interface>> GameStartable
+startGame() : void
<<Interface>> Pausable
+pauseGame() : void
+resumeGame() : void
+isGamePaused() : boolean
<<Interface>> LevelPassable
+finishLevel()
+continueNextLevel()
+getGameLevel() : int
<<Interface>> GameOverMakeable
+makeGameOver()
<<Interface>> MainMenuBackable
+backToMainMenu()
<<Interface>> KeysChangeable
+changeShootingKey(KeyEvent) : void
+changeUsingPowerUpKey(KeyEvent) : void
Application Logic
Input Manager
SpaceshipMovementController
+initMovementKeyBindings(KeyEvent) : void
SpaceshipShootingController
+initShootingKeyBindings(KeyEvent) : void
SpaceshipUsingPowerUpController
+initUsingPowerUpKeyBindings(KeyEvent) : void
PauseGameController
+initPauseGameKeyBindings(KeyEvent) : void
GameEngine
-gameLevel : int
-score
-isGamePaused : boolean
-spaceship : Spaceship
-bulletsInSpace : List<Bullet>
-spaceMobsInSpace : List<SpaceMob>
-spaceItemsInSpace : List<SpaceItem>
-spaceObstaclesInSpace : List<SpaceObstacle>
+drawGameScreenElements()
+startGame() : void
+pauseGame() : void
+resumeGame() : void
+finishLevel() : void
+continueNextLevel()
+getGameLevel() : int
+makeGameOver() : void
+backToMainMenuAfterGameOver() : void
+resetAll() : void
+changeShootingKey(KeyEvent) : void
+changeUsingPowerUpKey(KeyEvent) : void
Time Manager
<<Interface>> java.awt.event.ActionListener
TimeController
-timer : Timer
-timePassed : int
+startTimer() : void
+stopTimer() : void
+actionPerformed(ActionEvent) : void()
SpaceMobSenderService
+sendCreature(int level) : Creature
+sendAsteroid(int level) : Asteroid
+sendBoss(int level) : Boss
SpaceMobsShootingService
+sendBulletsFromMobs() : Bullet
UpdateService
+updateObjects() : void
SpaceObstacleSenderService
+sendStar() : Star
+sendBlackHole() : BlackHole
CollisionCheckerAndHandler
+checkCollisionForBullets() : void
+checkCollisionBetweenSpaceshipAndSpaceItems() : void
+checkCollisionBetweenSpaceshipAndSpaceMobs() : void
+checkCollisionBetweenSpaceshipAndSpaceObstacles() : void
DrawGameScreenElementsService
-graphics2D : Graphics2D
+draw() : void
NewLevelSetterService
+setNewLevel() : void
GameResetterService
+resetGame() : void
HighscoresManager
+getHighscores() : HashMap<String, int>
+checkHighscore(int score) : void
Game Screen Elements
<<enumeration>> PowerUpType
INVULNERABILITY
LASER_GUN
HYPER_DRIVE
REPAIR
PowerUp
-powerUpType : PowerUpType
Coin
-value : int
SpaceItem
<<enumeration>> SpaceshipType
FIGHTER
PANZER
LIGHTNING
Spaceship
-spaceshipType : SpaceshipType
-maxHP : int
-HP : int
-attackDamage : int
-currentPowerUp : PowerUp
-currentCoin : int
+changeType(SpaceshipType)
+shoot() : SpaceshipBullet
+pickCoin(Coin) : void
+pickPowerUp(PowerUp) : boolean
+usePowerUp() : boolean
+doUpgradeWithCoins(Upgrade) : boolean
can be done by
Upgrade
-upgradeType : UpgradeType
-cost : int
<<enumeration>> UpgradeType
MAX_HP
ATTACK_DAMAGE
MOVEMENT_SPEED
Star
BlackHole
SpaceObsta...
Asteroid
SpaceMob
#level : int
#maxHP : int
#HP : int
#attackDamage : int
#hasCoin : boolean
#hasPowerUp : boolean
+dropCoin() : Coin
+dropPowerUp() : PowerUp
Creature
+shoot() : CreatureBullet
Boss
+shoot() : BossBullet
+charge() : void
SpaceshipBullet
CreatureBullet
BossBullet
Bullet
#attackDamage : int
SpaceObject
#xPos : int
#yPos : int
#image : Image
#width : int
#height : int
#velocityX : int
#velocityY : int
+move() : void
connects to
Database

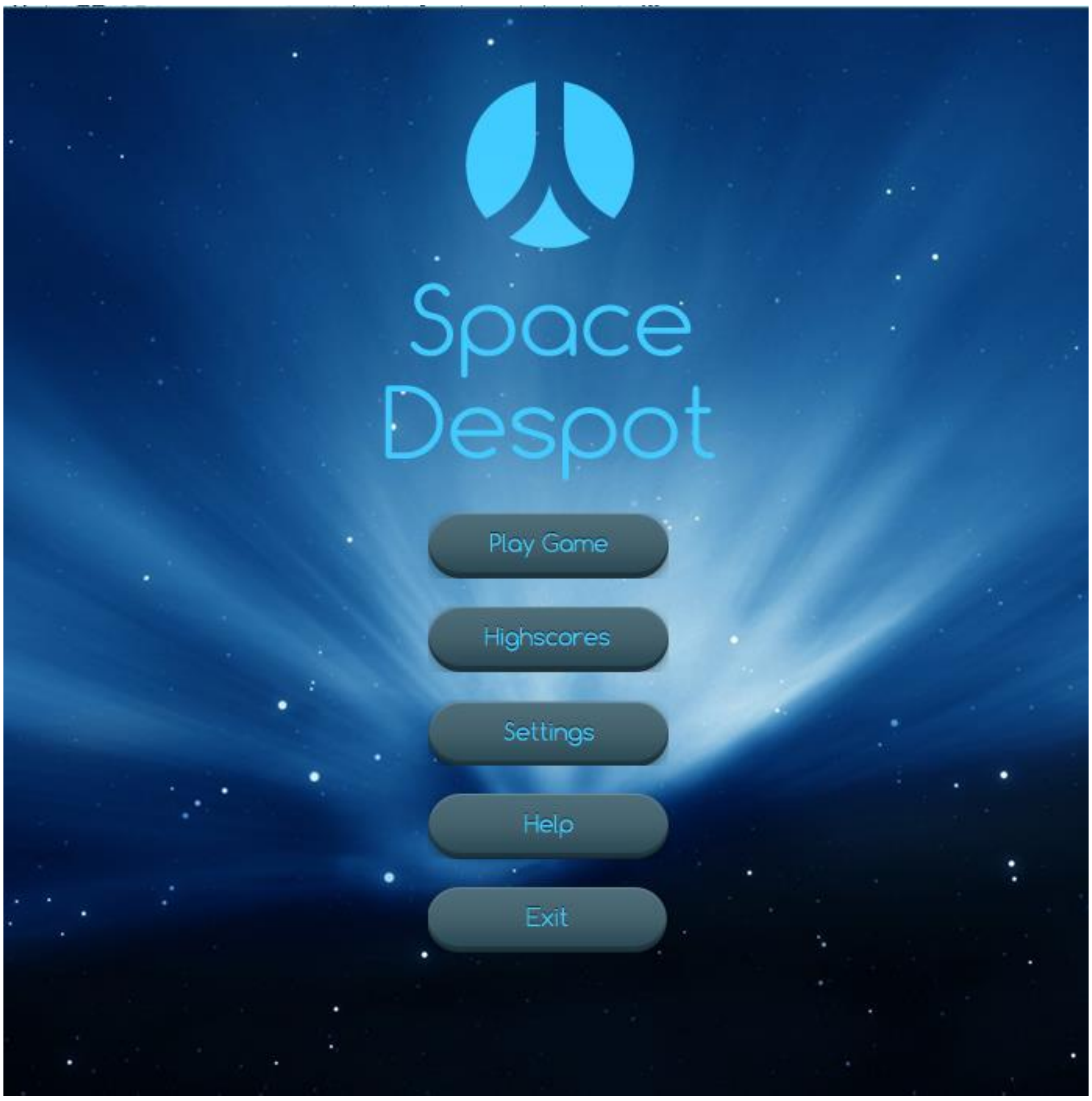
CLICK FOR HIGH RESOLUTION(1744x2300)

http://i.imgur.com/G04a5PA.png

//Old Design

4 User Interfaces

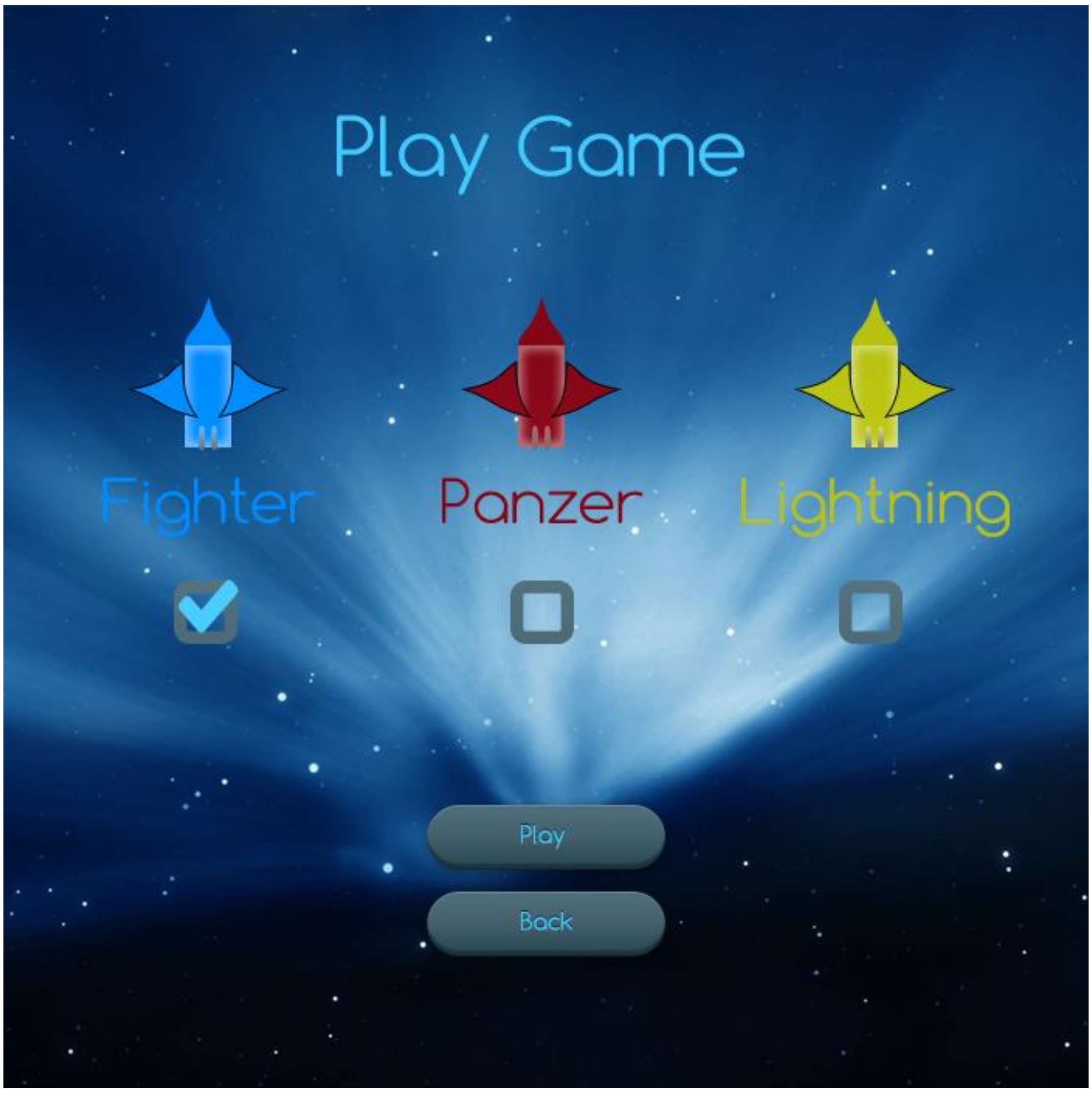
Main Menu Screen



MainMenuScreen is the first screen initialized when the game is started. Player can access Play Game screen, Highscores screen, Settings screen, Help screen and exit the game using buttons.

No significant change is made in this class.

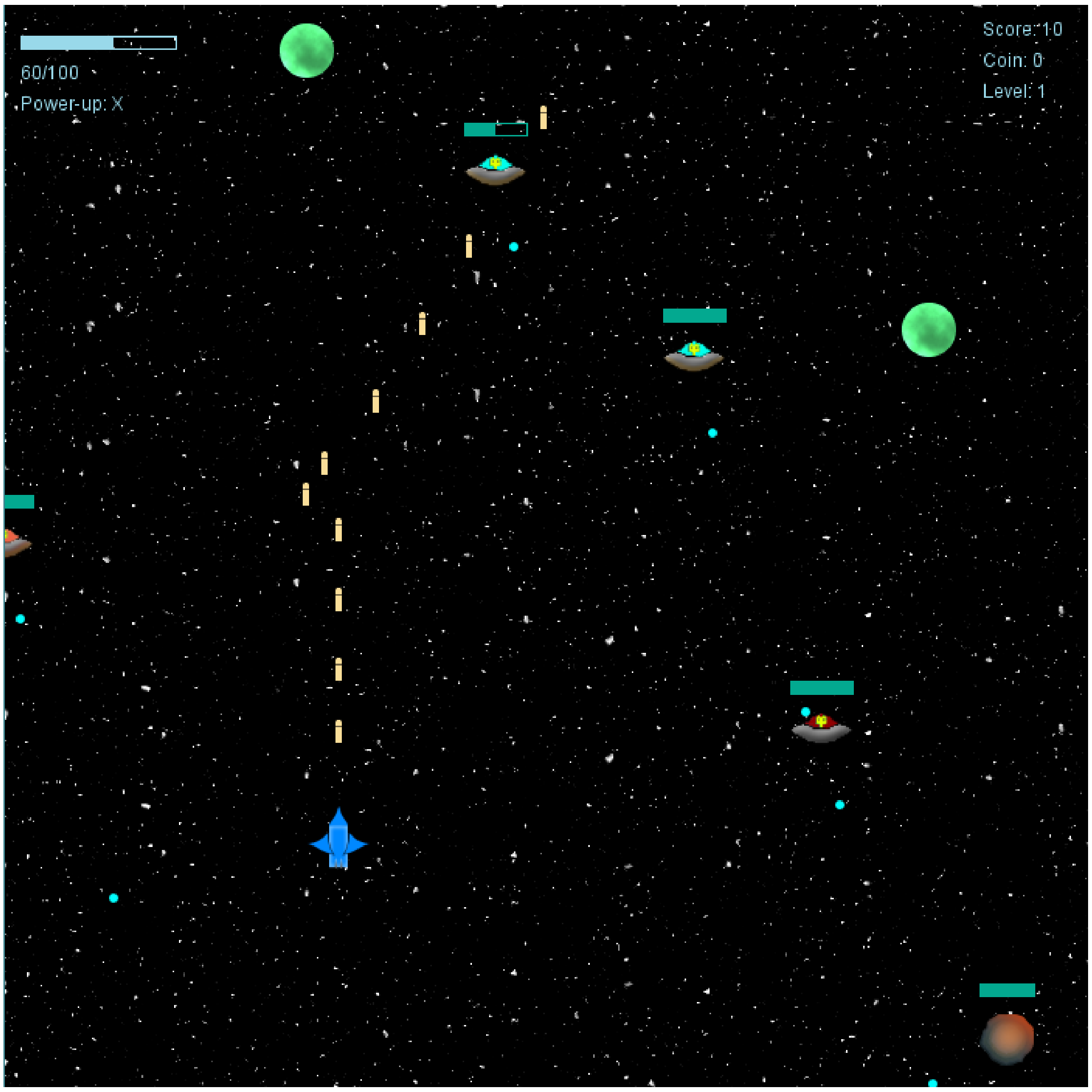
Play Game Screen



In order to play the game, player needs to access play game screen. In this screen, player chooses one spaceship among 3 types. Although the boxes under spaceship types look like a check box, actually they are radio buttons. Player can only select 1 type of spaceship before starting the game. Default selection is Fighter. When the selection is completed, player can proceed to the game by clicking Play.

No significant change is made in the final version of class.

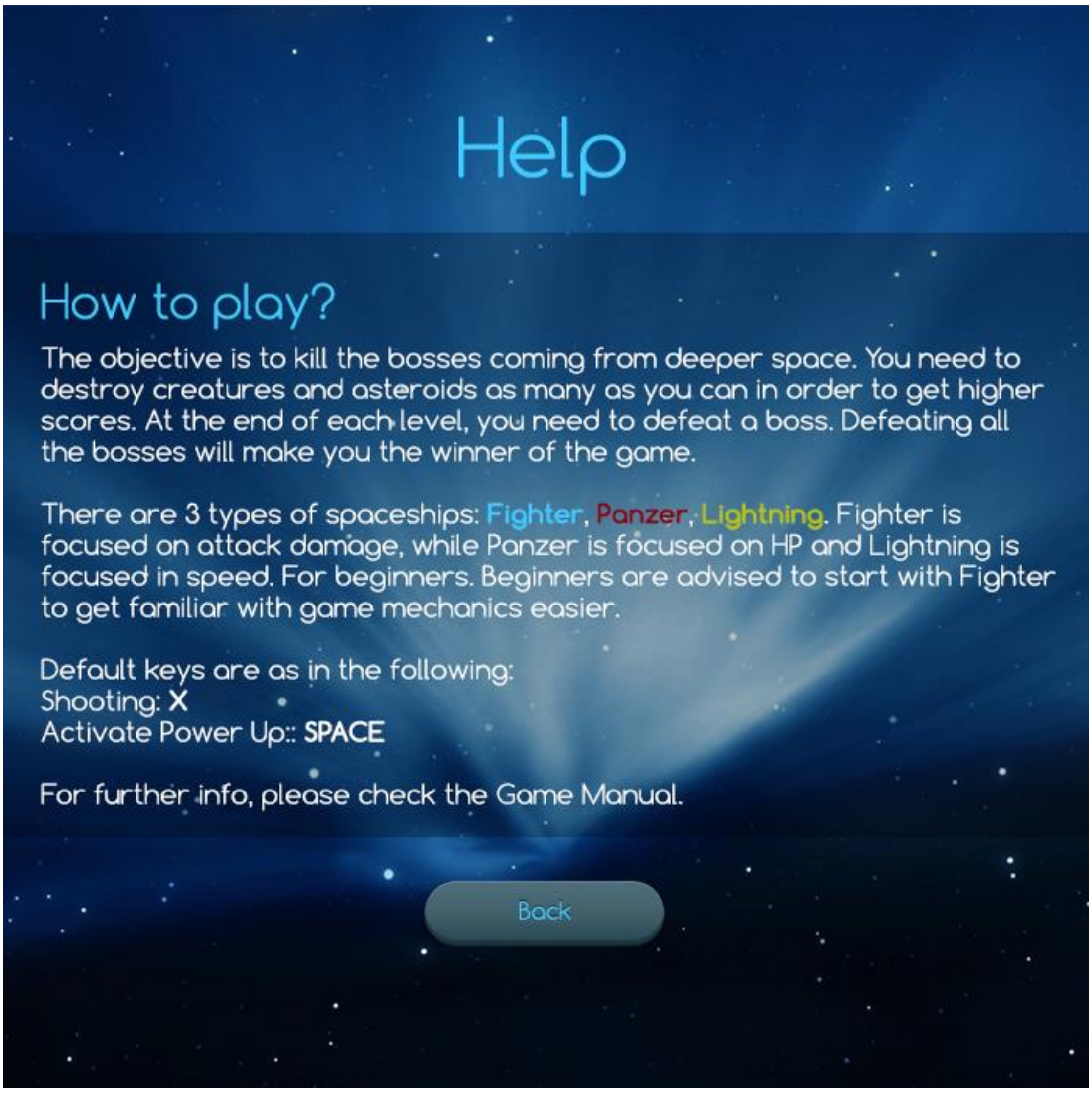
Game Screen



The screen above is the main screen of the game. Player can progress in the game until end state by using the functionalities of the spaceship in this screen. Pressing ESC during this screen opens up Pause Menu and freezes the game.

No significant change is made in the final version of class.

Help Screen



Help

How to play?

The objective is to kill the bosses coming from deeper space. You need to destroy creatures and asteroids as many as you can in order to get higher scores. At the end of each level, you need to defeat a boss. Defeating all the bosses will make you the winner of the game.

There are 3 types of spaceships: Fighter, Panzer, Lightning. Fighter is focused on attack damage, while Panzer is focused on HP and Lightning is focused in speed. For beginners. Beginners are advised to start with Fighter to get familiar with game mechanics easier.

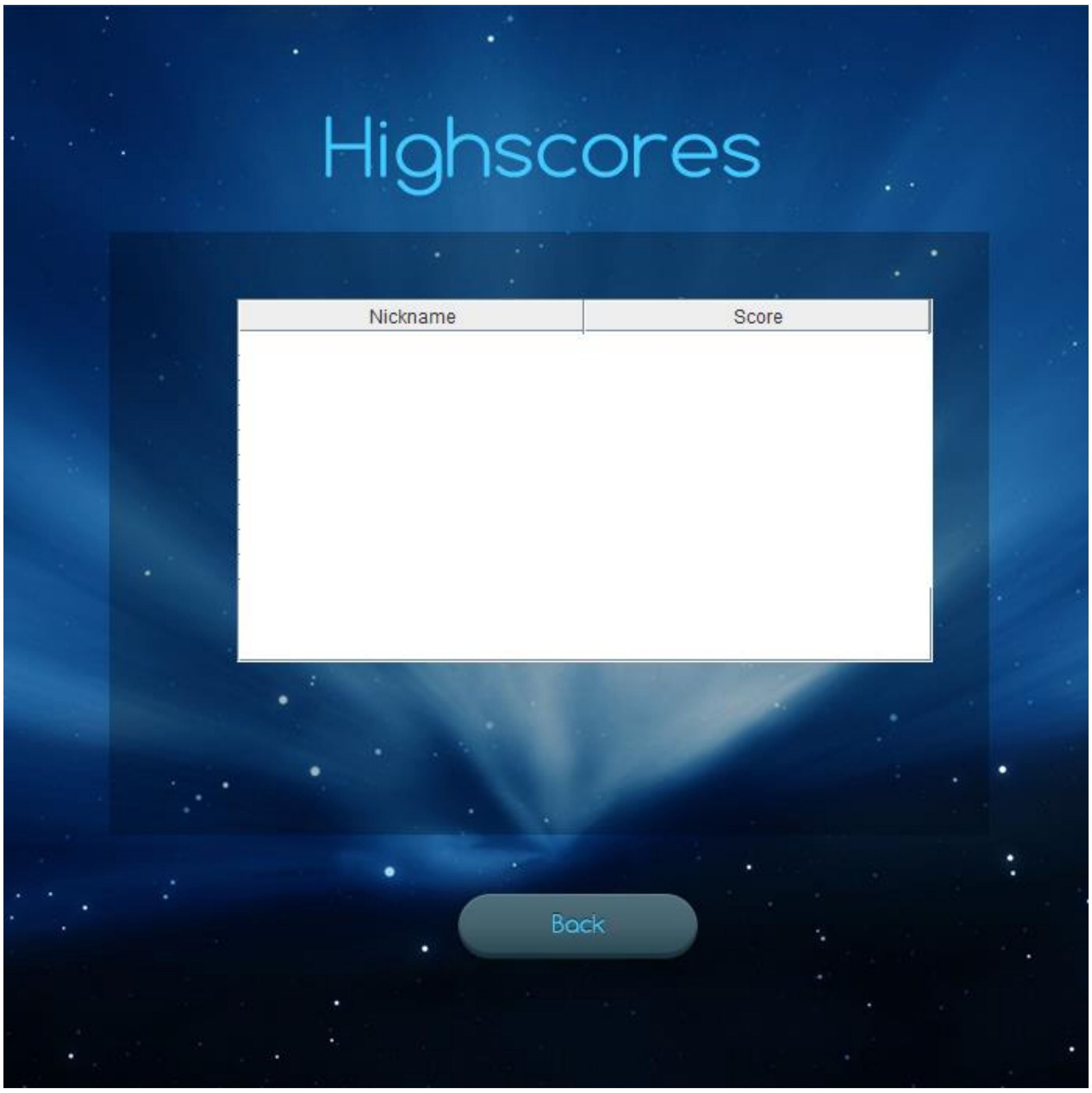
Default keys are as in the following:
Shooting: **X**
Activate Power Up:: **SPACE**

For further info, please check the Game Manual.

Help Screen tells the user necessary information about the game. It consists of a static image and a back button, which lets user return back to main menu.

We first planned to use components of Java Swing to create this screen; however, in order to enhance the visual quality of the screen, we designed a static image file using Adobe Photoshop, then added it to the screen.

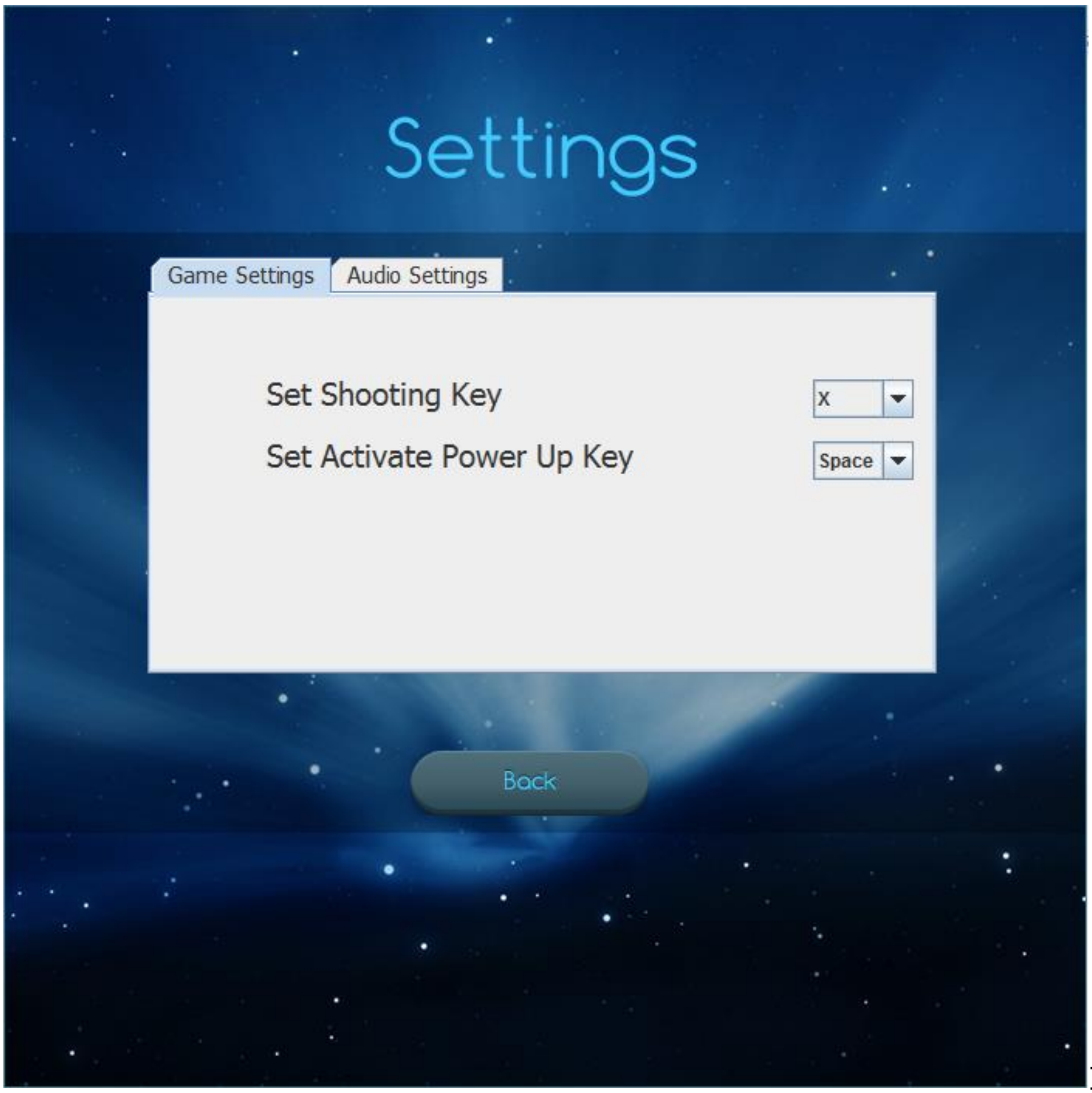
Highscores Screen



Highscore Screen displays the top scores achieved by the players. Scores are kept in a remote database, which means that the highscore table is global. PHP is used for database connection purposes.

We first planned to make the highscore table local, but after some discussion, we concluded that making it global will enhance the competitiveness of the game.

Settings Screen



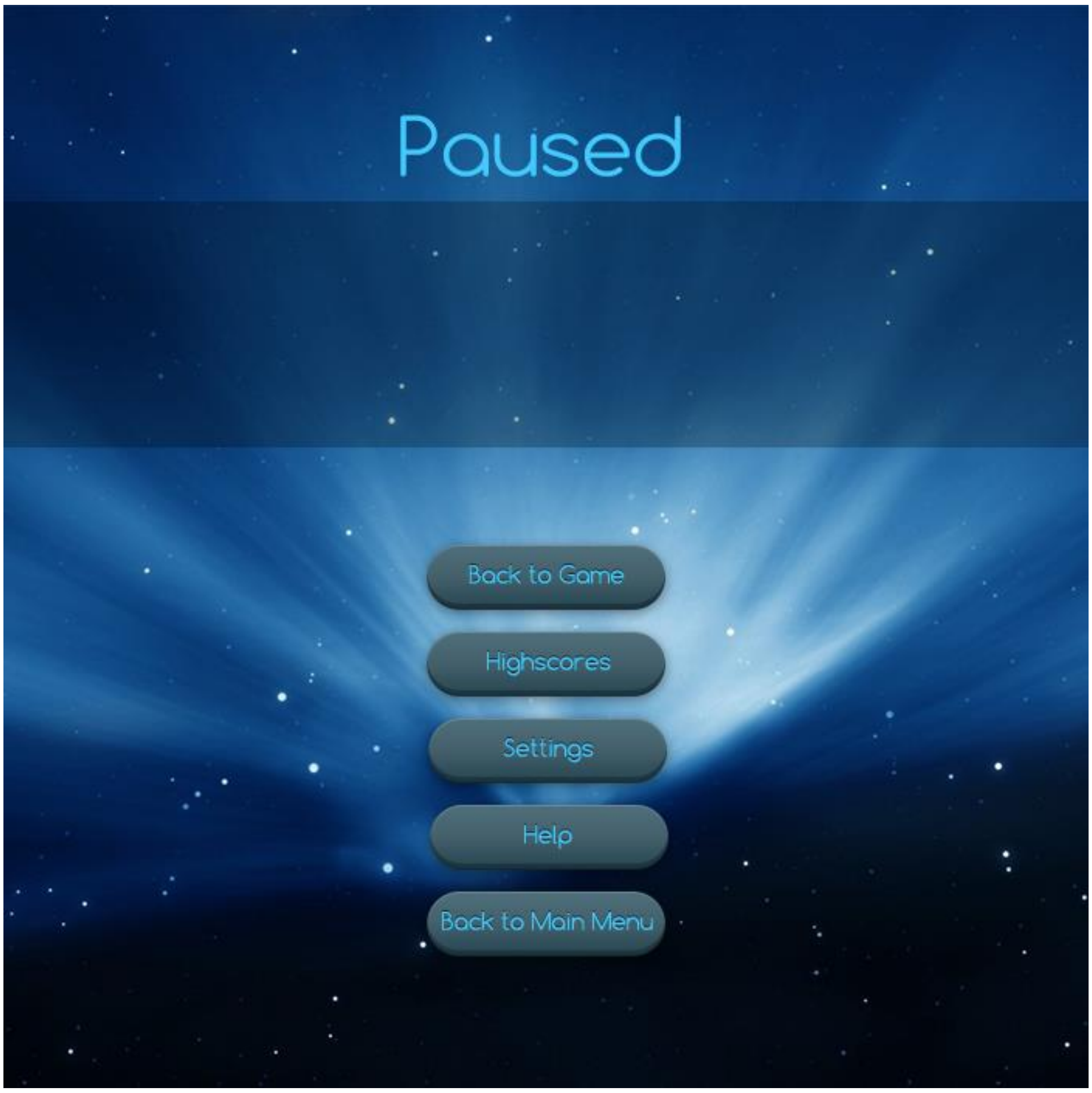
Set Shooting Key: X

Set Activate Power Up Key: Space

In Settings screen, player can change shooting key and power-up key, also he can change the volumes of FX and Music.

No significant change is made in the final version of class.

Pause Menu



When the player presses ESC button during game, Pause Screen pops up and freezes the game. While in this screen, clicking Back to Game or pressing ESC again open the game screen from the last state. Player can display highscores, settings and help as in main menu.

No significant change is made in the final version of class.

Parts that are not implemented

We implemented the graphical components in Adobe Photoshop, but due to the time limit, we could not integrate some of the graphics to Java Swing. Other than this, we successfully implemented all the core classes, game logic, and logics of user interfaces. At the point we submit the report, the game is in playable state and all the game logic works fine as far as we play-tested the game. We also handled all the design by ourselves and composed the game musics. Adobe Photoshop is used for user interface and game components designs, Logic Pro X is used for composing musics.

A- Appendix

1 - Game Manual - https://github.com/DjCedrics/cs319_group19_project/blob/master/reports/GameManual.pdf)

2 - Javadoc API - https://github.com/DjCedrics/cs319_group19_project/blob/master/space_despot_api.zip