Bilkent University

Department of Computer Engineering

Object Oriented Software Engineering Project

Project Short-Name: Space Despot

Design Report – Part II

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

Supervisor: Prof. Dr. Uğur Doğrusöz

Progress Report
Nov 28, 2016

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Object Oriented Software Engineering Project, course CS319.

Contents

Design Report – Part II

Project short-name: Space Despot

4 Low-level design

4.1 Object design trade-offs

In object design, to maximize the performance and have a bug free program, we need to apply the principles of reusability, and encapsulation.

4.1.1 Reusability

Basic strategy to deal with complexity of the program is reducing a complex problem into subsystems and establishing proper connections using hierarchies between classes. Usage of existing components is the key element of reusability in other words "plug and play".

4.1.1.1 Inheritance

Using inheritance in object design directly promotes reusability. Extending applications functionality from super class prevents doubling the code and constitutes managing the complexity. For example: Asteroid, Creature and Boss classes have common attributes and characteristics. Grouping those classes under the super class SpaceMob provides more efficient design. Also, SpaceMob is extending SpaceObject which is more general than it (Figure 1).

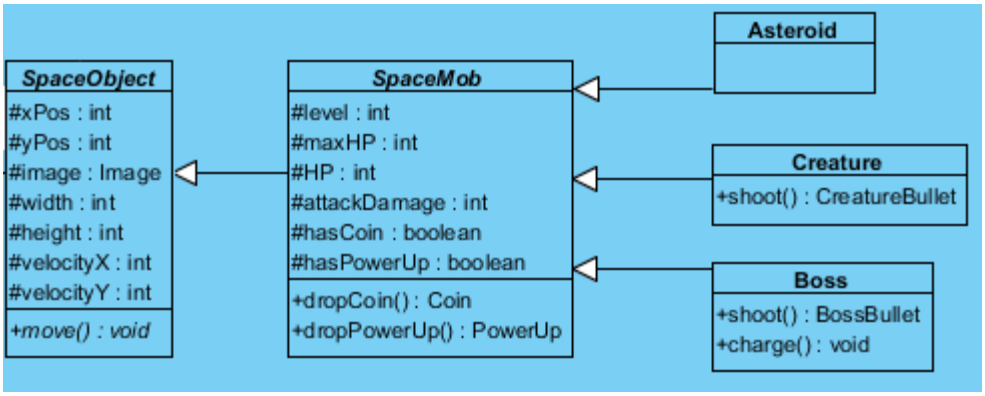

Figure 1- Inheritance

4.1.1.2 Off-the Shelf Components

Package software solutions such as frameworks are commonly used in commercial software. It makes not only the implementation easier but also saves time. On the other hand, it brings out obvious tradeoffs along with it. Cost is the first thing that comes to mind. If the cost of an external component will have a significant impact on the overall cost; designer should think twice before buying it.

In our game, requirements of our instructor and our budget did not let us use of any kind of external component. Thus, we did not use any premium asset package and commercial game engine. Instead we implemented all the components from scratch.

4.1.1.3 Design patterns

Identifying the objects during analysis stage or subsystem decomposition in the system design stage takes time. In addition, it is not guaranteed to have an optimized solution while keeping making modification in object model. In order to have a simple yet optimized design, designer should follow common techniques used in software development.

In our design we followed the standard KISS principle. We kept our design simple and understandable by everyone. We decide on not to use any special design patterns as we see any need of using them.

4.1.1.4 Using Interface

Using interface also addresses the reusability principle, as it promotes modularity and extensibility of design. Packing up the design into discrete units is a good design choice that minimizes the coupling and maximizes the cohesion between classes.

Our GameEngine subsystem is using many interfaces. Reason for that is to make use of its methods by other classes.

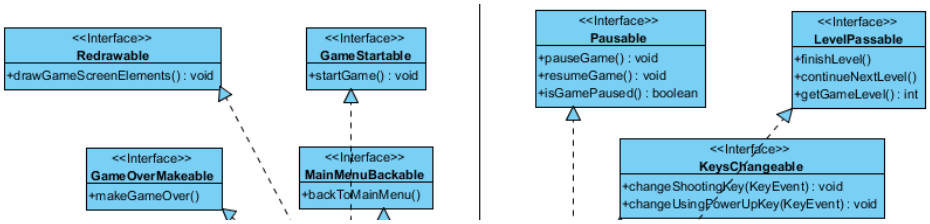

Figure 2

4.1.1.5 Using Polymorphism

Polymorphism is another aspect of reusability principle. Basically, an object is able to take on many forms. The most common usage of polymorphism is referring the child classes of a super class. To increase the performance and manage the complexity, we will use polymorphism in our implementation. For instance, while drawing the bullets on screen game engine will simply call "draw()" method once for all different types of bullets.

4.1.2 Encapsulation

Developers hide information as encapsulation suggests. Main usage of it is to prevent clients accessing the data portion of program

when it is not needed. It basically keeps the program stable in its boundary that is useful to prevent possible bugs that can be occurring in execution. However, as a trade-off overall performance of the system might get affected due to the fact that program may need extra procedures to use the data.

We used interfaces and made our class attributes private to limit the boundaries of our program for stable execution.

4.2 Final object design

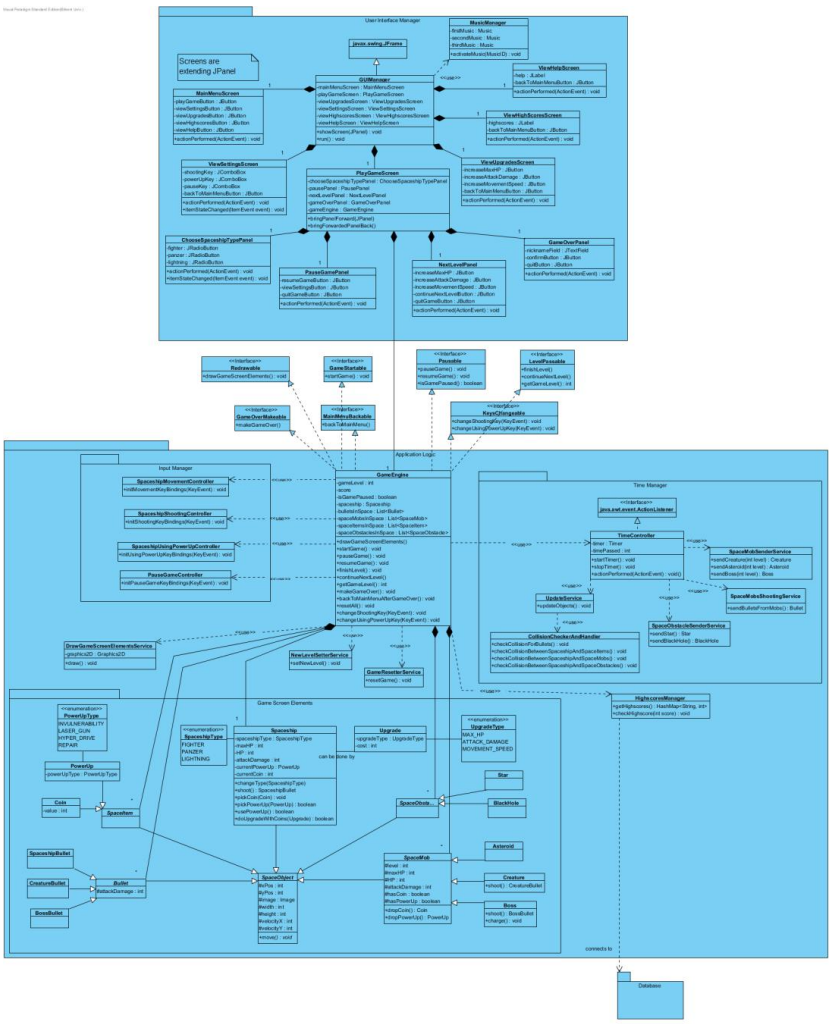

Figure 3 – Final Object Design - http://i.imgur.com/G04a5PA.png

4.3 Packages

4.3.1 User Interface Manager Package

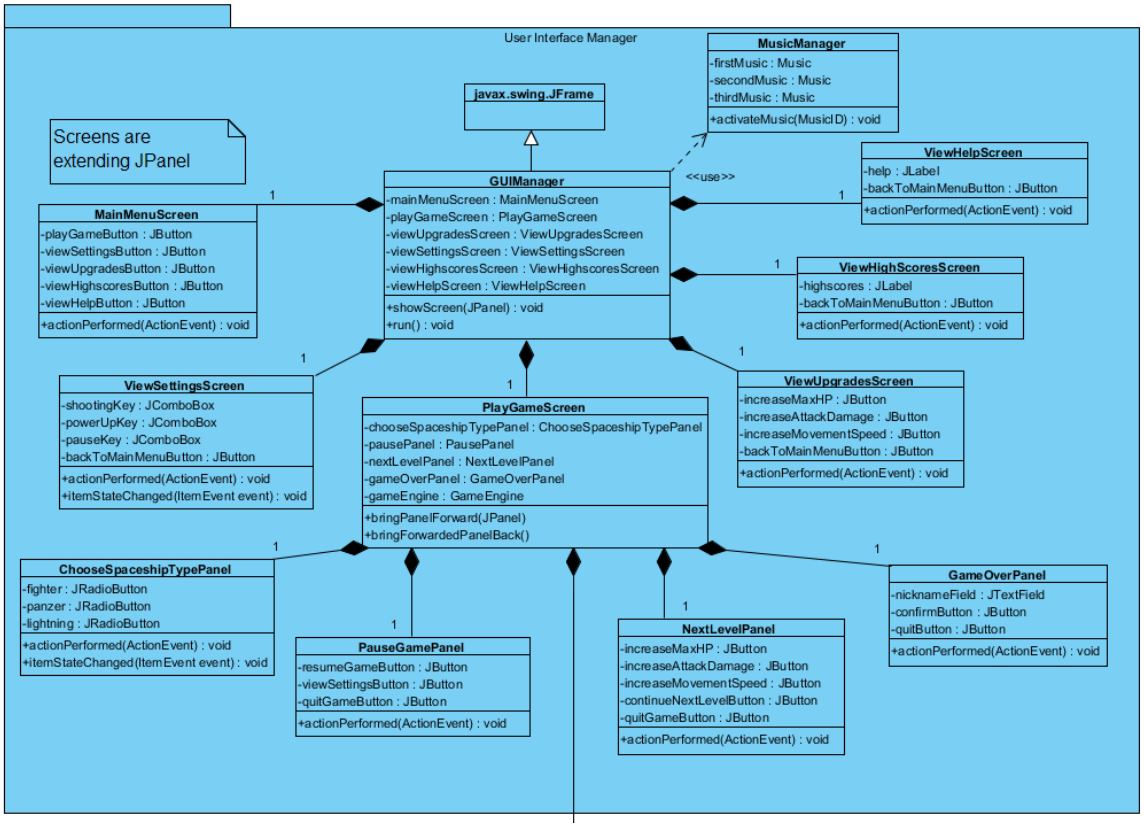

Figure 4 – User Interface Manager Package

This package will have classes that are responsible for providing an interface between the user and the system.

This package's core class is GUIManager. It regulates transition between main menu screen and other menu screens. These screens are: MainMenuScreen, ViewUpgradesScreen, ViewSettingsScreen, ViewHighScores Screen, ViewHelpScreen and PlayGameScreen.

When user chooses to play game, the job is transferred to Play Game Screen. Play Game Screen consists of game play map and

some panels. These panels are: ChooseSpaceshipTypePanel, PauseGamePanel, NextLevelPanel and GameOverPanel.

When the game first starts, ChooseSpaceshipTypePanel is brought forward. After user chooses spaceship type, this panel becomes invisible and user starts game. When user presses ESC while playing game, PauseGamePanel is brought forward. It will be gone after user chooses to resume game. Then, when one of levels is finished, user sees NextLevelPanel. After user is done with upgrades in this panel, he starts next level and this panel is brought back. When the game is over, GameOverPanel is activated. Here, user will be asked to give a name for the score he gained. User can quit this screen either directly or with giving a nickname. Then, user is brought back to main menu.

4.3.2 Application Logic Package

4.3.2.1 Game Screen Elements Package

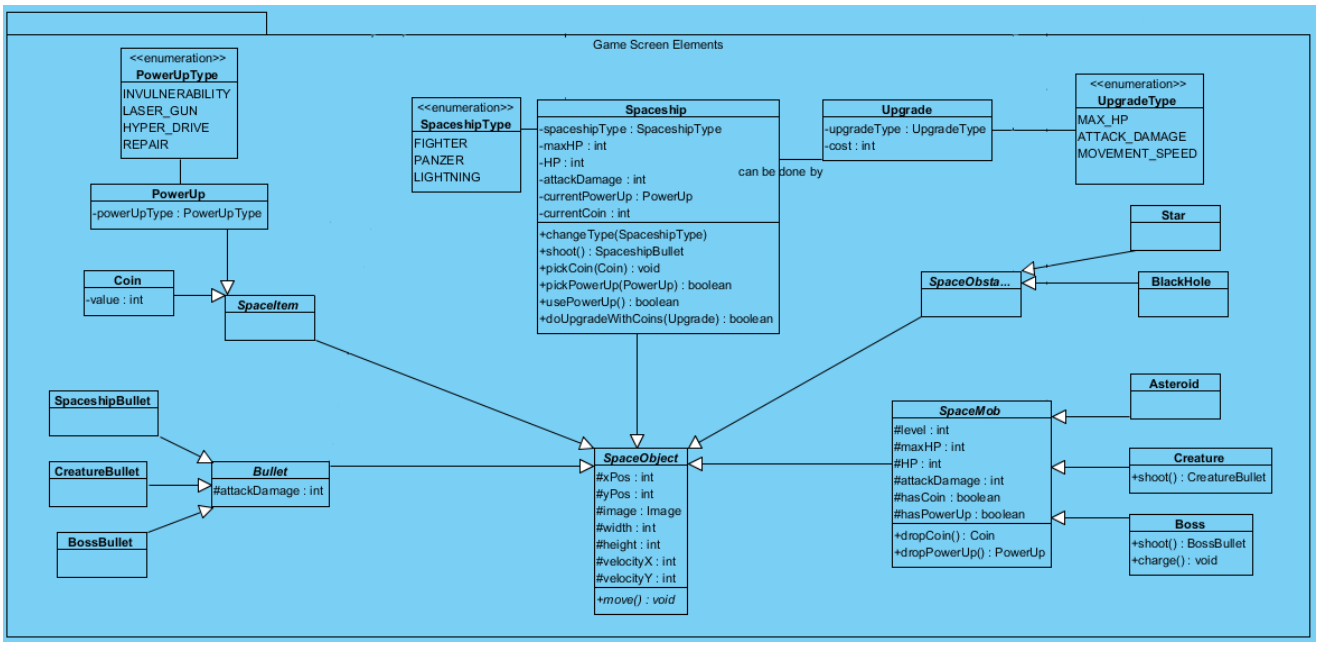

Figure 5 – Game Screen Elements

SpaceshipBullet Class

SpaceshipBullet class represents the bullets which are fired by spaceships (**re**presented as Spaceship class). SpaceshipBullet class extends to Bullet class.

CreatureBullet Class

CreatureBullet class represents the bullets which are fired by creatures (represented as Creature class). CreatureBullet class extends to Bullet class.

BossBullet Class

BossBullet class represents the bullets which are fired by bosses (represented as Boss class). BossBullet class extends to Bullet class.

Bullet Class

Bullet class represents bullets that are used by game objects. It holds property attackDamage as integer. attackDamage of bullets varies depending on the type of game object by which it is used.

Coin Class

Coin class represents the coin objects which are located in the game space. It holds property value as integer. Coin class extends to SpaceItem class.

PowerUp Class

PowerUp class represents the power up objects which are located in the game space. It uses enumerator PowerUpType, which defines power up types INVULNERABILITY, LASER_GUN, HYPER_DRIVE, REPAIR as integer values. PowerUp class extends to SpaceItem class.

SpaceItem Class

SpaceItem class represents the collectable objects which are located in the game space. SpaceItem class extends to SpaceObject class.

SpaceObject Class

SpaceObject class represents the objects which are located in the game space. It holds the coordinates of the object with xPos and yPos as integers, dimensions of the object with width and height as integers, velocity of the object with velocityX and velocityY as integers. It also has a function move(), which takes the previous location of the object as parameters and moves it by changing its position with respect to its velocity values and game time. Note that some space objects such as space items, space obstacles do not move and their velocity values are 0; whereas objects such as space mobs, bullets and spaceship can move.

SpaceMob Class

SpaceMob class represents the mobs which are located in the game space. The class has property level as integer, which changes

depending on the current level. level is used as a coefficient for calculating values of the properties maxHP, attackDamage. maxHP holds the maximum HP value of the mob as integer, and attackDamage holds the attack damage value of the mob as integer. SpaceMob class has two boolean properties which are named as hasCoin and hasPowerUp. hasCoin denotes whether the mob instance will drop coin or not, whereas hasPowerUp does the same for power up. This class has two functions named as dropCoin() and dropPowerUp(). When HP of an instance becomes lower or equal than zero, it gets destroyed and calls function dropCoin() or dropPowerUp(), depending on the states of corresponding boolean variables.

Asteroid Class

Asteroid class represents the asteroid mobs which are located in the game space. It extends to SpaceMob class.

Creature Class

Creature class represents the creature mobs which are located in the game space. It extends to SpaceMob class. It has function shoot(), which calculates the damage of current bullet from the property attackDamage, then returns an instance of CreatureBullet class.

Boss Class

Boss class represents the boss mobs which are located in the game space, at the end of each level. It has function shoot(), which

calculates the damage of current bullet using the property attackDamage, then returns an instance of BossBullet class. It also has function charge(), which represents “Charge” skill of the boss.

SpaceObstacle Class

SpaceObstacle class represents the space obstacles which are located in the game space. Space obstacles cannot move, so they don’t use function move(). Their properties velocityX and velocityY are 0.

Star Class

Star class represents the star obstacles which are located in the game space. Star class extends to SpaceObstacle class.

BlackHole Class

BlackHole class represents the black hole obstacles which are located in the game space. BlackHole class extends to SpaceObstacle class.

Spaceship Class

Spaceship class represents the spaceship object which is located in the game space and directly controlled by user. Its spaceshipType property defines the type of the spaceship. The class uses enumerator SpaceshipType, which defines the spaceship types FIGHTER, PANZER and LIGHTNING and maps them as integers. It has properties maxHP, HP, attackDamage, currentCoin as integers. Also, it has functions changeType(SpaceshipType), shoot(),

pickCoin(Coin), pickPowerUp(PowerUp), usePowerUp(), doUpgradeWithCoins(Upgrade). changeType function is called when it is required to change the type of the spaceship, which is done by the user in play game menu screen. shoot function calculates the damage of current bullet using the property attackDamage, then returns an instance of SpaceshipBullet class. pickCoin function is called when the spaceship contacts with a coin object in the game space. It uses value property of the coin instance, then adds it to currentCoin, and then destroys the coin object. pickPowerUp function is called when the spaceship contacts with a power up object in the game space. It uses powerUpType property of the power up instance, then copies it to currentPowerUp property of the spaceship, and then destroys the power up instance. It returns the result as boolean. usePowerUp() activates the current power up. Precondition of this function is currentPowerUp != null. It returns the result as boolean and changes the value of currentPowerUp property to null. doUpgradeWithCoins function takes Upgrade as parameter, then applies the update, if the precondition is satisfied. The precondition is that the player should have enough coins to afford the upgrade. It returns the result as boolean.

Upgrade Class

Upgrade class represents the upgrades which can be done by spaceship. It is used by Spaceship class. It has properties upgradeType, which defines the type of the upgrade by using the enumerator UpgradeType, and cost, which stores the cost of the

upgrade as integer value. Upgrade class uses enumerator UpgradeType, which defines upgrade types MAX_HP, ATTACK_DAMAGE and MOVEMENT_SPEED as integers.

4.3.2.2 GameEngine Class

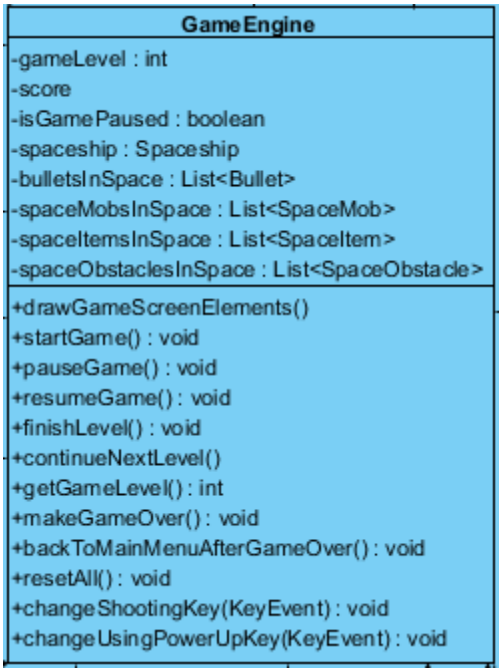

Figure 6 - GameEngine

The core thing of this Application Logic package is GameEngine class. When user chooses to play game, at first, GameEngine will construct the game.

Game Engine is the main synchronizer class of game objects. In our game there are numerous types of space objects. At first, there will be spaceship for player. Then, there will be space mobs (List<SpaceMob> spaceMobsInSpaces): creatures, asteroids, and bosses. Also, there will be space obstacles (List<SpaceObstacle> spaceObstaclesInSpace): stars and black holes. Space mobs may drop space items (List<SpaceItem> spaceItemsInSpace): power-

ups and coins. Spaceship can shoot spaceship bullets and space mobs can shoot creature bullets and boss bullets, depending on the mob type (List<Bullet> bulletsInSpace).

Besides, it will hold gameLevel and score of the current game as integers. Then, it will hold the info about whether or not the game is paused as a boolean.

The first called method of this class is drawGameScreenElements(). Actually, it will be looped along the game as objects are updated. This method will use DrawGameScreenElements service as a helper. This service class will draw objects and also their HP bars.

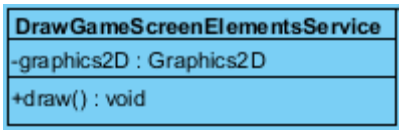

Figure 7 – DrawGameScreenElementsService

Next method startGame() will be called after user chooses spaceship type and play the game.

pauseGame() will be called when user pressed ESC. Then, everything will stop and PauseGamePanel will be brought forward. isGamePaused becomes true after pause. On the other hand, after user chooses to resume game, resumeGame() will do the work.

After one of levels is finished, finishLevel() will be called. This method will stop the game, it will bring NextLevelPanel forward. Also, inside, it will use NewLevelSetterService for preparing next level. This will clear game screen elements from old level such as

bullets. Also, it will make spaceship's HP full again and will bring it to the middle of space again.

Figure 8 – NewLevelSetterService

continueNextLevel() will increase gameLevel property. It will brought NextLevelPanel back again. Then, the new level will start.

makeGameOver() will be called when the game is over, which is controlled by some other classes that will be explained later (Time Manager Package). For example, when spaceship's HP drops to 0 or user passes all levels, the game will be over. Then this method will be called. This method will bring GameOverPanel forward.

After user enters a nickname for the score if he wants to and presses quit, then, backToMainMenuAfterGameOver() will be called. This will bring user back to the main menu and then it will call resetAll() method. This method will use GameResetterService class for next games. This class will set gameLevel to 1 again. Then it will reset spaceship's properties and it will clear every object from the space.

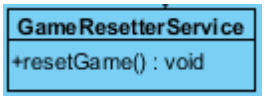

Figure 9 – GameResetterService

changeShootingKey(KeyEvent) and changeUsingPowerUpKey(KeyEvent) might be called from the settings menu. They will set new key for these actions by calling Input Manager Package's classes.

4.3.2.3 Input Manager Package

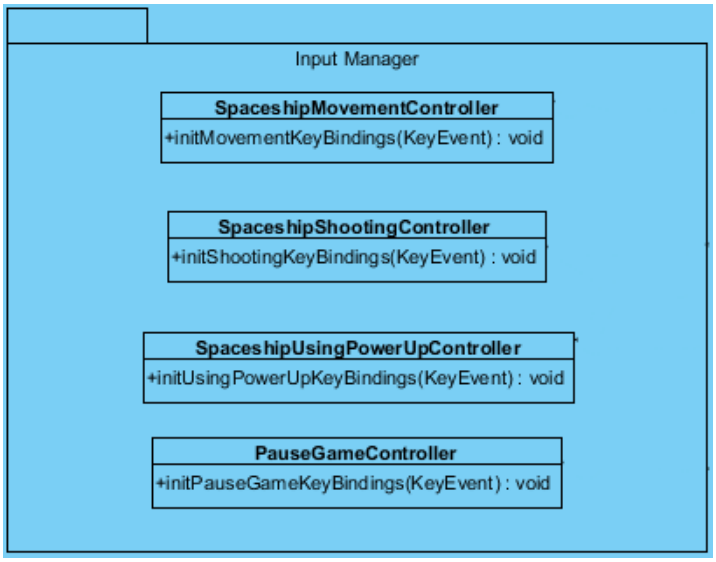

Figure 10 – Input Manager Package

This package will control whether or not UP, DOWN, LEFT, RIGHT(Movement), X (Shooting), Space (UsingPowerUp) and ESC (PauseGame) keys are pressed or released. Therefore, it will regulate the spaceship's movement. This package's classes' method can be called from GameEngine to change key for that event.

4.3.2.4 Time Manager Package

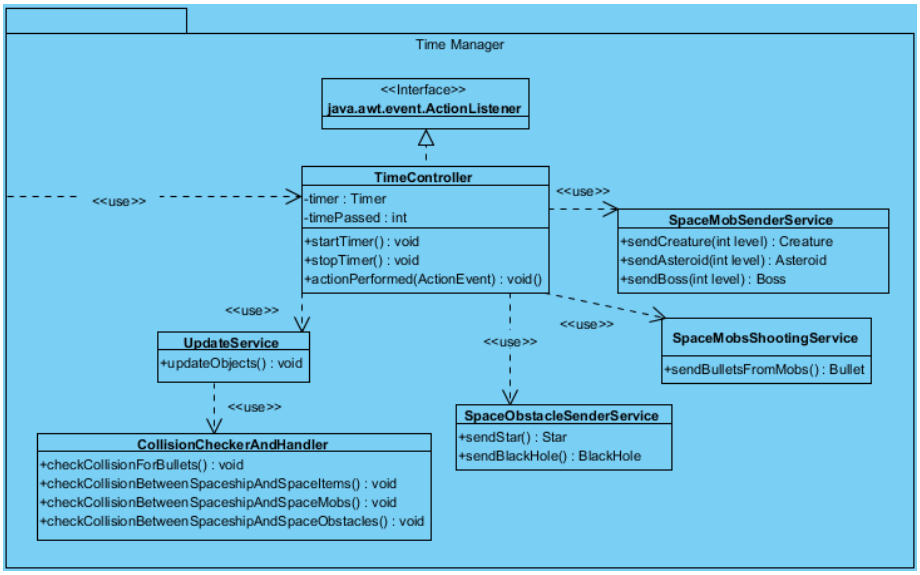

Figure 11 – Time Manager Package

TimeController

This class is the slave of GameEngine. It will control lots of things by using proper service classes. First of all, it has Timer on it. It will do the work in each time interval. To start with, it will control start game, end game, pause and resume actions by starting or stopping timer.

At certain intervals, first, it will increase user's score as time passes. Second, it will send SpaceMob objects to the space such as Asteroid and Creature. Third, it will make these SpaceMobs shoot at certain intervals. Third, it will send SpaceObstacle objects such as Star and BlackHole. It will do that by using these services.

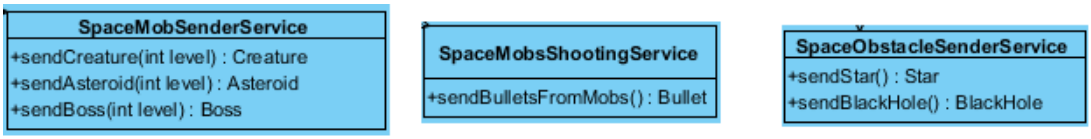

Figure 12 – TimeController Services

Then, after a certain time, it will stop sending these small mobs and obstacles. It is time for boss so it will send end-level boss.

So now, there are some objects in the map that needs to be updated in every time interval. This class needs to do updates in each time intervals. However, it will not do that directly. It will use UpdateService to do this.

UpdateService

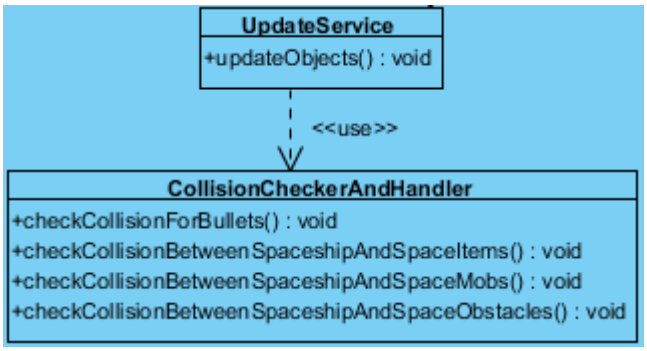

Figure 13 – UpdateService

UpdateService is the most important class of this whole project.

The first thing it does is, call game screen element's move() methods in each time interval. Some of them has zero velocities but still they will be called because of the polymorphism.

The second thing is, it won't let spaceship go out of map. In other words, Spaceship's x and y coordinates cannot be zero and cannot be more than width and height of the map.

The third thing is, spaceship's HP might drop to zero, then it will call makeGameOver() from the GameEngine by using GameOverMakable interface.

The fourth thing is, some of space objects might go out of map. Therefore, this class will identify them and remove from the List at the GameEngine.

The fifth thing is, some of the mobs may be destroyed by SpaceshipBullets, by user in other words. Hence, if this SpaceMob is boss, the level will finish, finishLevel() will be called from the GameEngine by using LevelPassable interface. If this boss is final

level (3 for now), again makeGameOver() will be called. However, if this boss is Creature or Asteroid, they might drop power-ups or coins by chance according to their hasPowerUp() and hasCoin() methods. Thus, this class will control them so that it has a chance to add new SpaceItem (PowerUp or Coin) to the space.

Then, what about collisions?

CollisionCheckerAndHandler

This is the service class that is used by UpdateService. In each time interval, it will check collisions between objects and handle them accordingly.

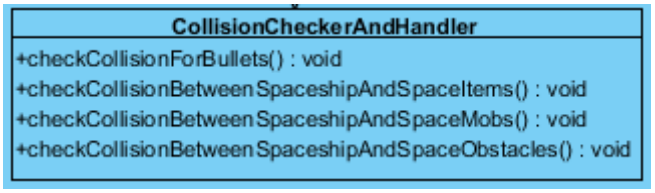

Figure 14 - CollisionCheckerAndHandler

- checkCollisionForBullets()

When SpaceshipBullet hits to SpaceMob, it will drop their HPs by attack damage. When SpaceMob's bullets hit to Spaceship, it will drop Spaceship's HP by bullet's attack damage.

- checkCollisionBetweenSpaceshipAndSpaceItems()

If spaceship collides with Coin, Coin will be removed from the map and spaceship's Coin value will increase. If spaceship collides with PowerUp and spaceship.pickPowerUp(PowerUp) returns true,

spaceship will collect this PowerUp and PowerUp will be deleted from map.

- checkCollisionBetweenSpaceshipAndSpaceMobs()

In this case, SpaceMob will explode and does damage to Spaceship.

- checkCollisionBetweenSpaceshipAndSpaceObstacles()

In this case, game will be over.

4.3.2.5 Highscores Manager Package

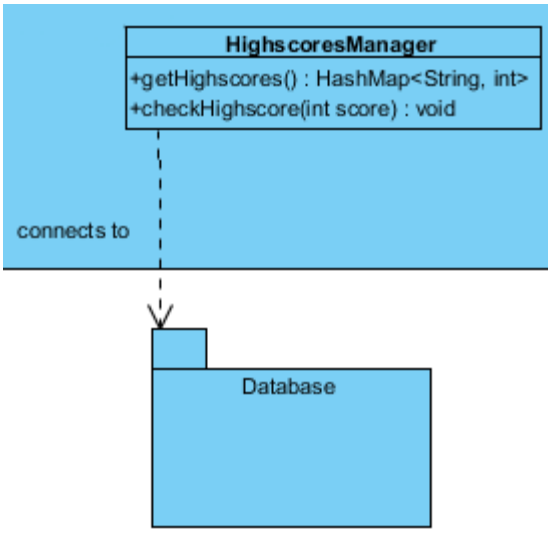

Figure 15 – Highscores Management

HighscoresManager class is responsible for connecting to database and saves or loads high scores. It has two functions, first one is getHighScore() which gets the high scores from database and returns them inside a hash map. Second one is checkHighScore(int score), which checks score attribute that comes from GameEngine class by comparing it with the high scores in the database. If the score is a new high score, it is saved into the database.

4.4 Class Interfaces

These interfaces are implemented by GameEngine. We used interfaces for methods in order to make use of GameEngine's methods by other classes.

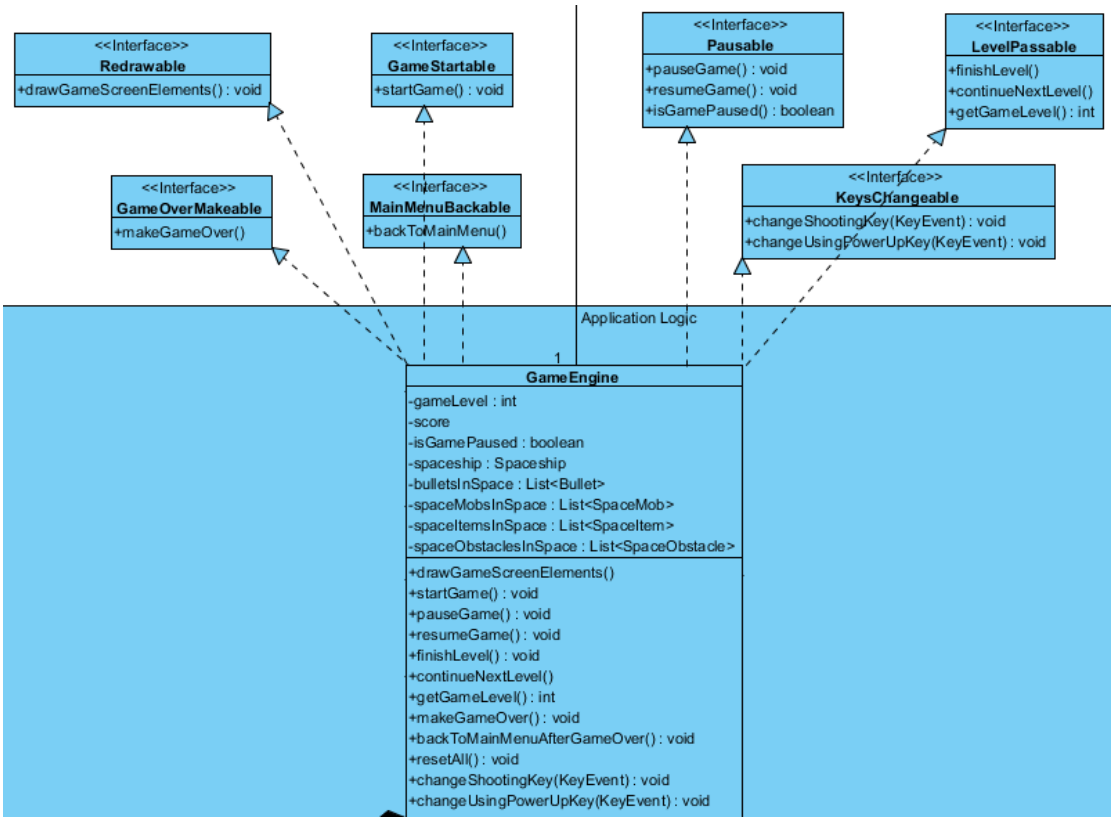

Figure 16 - Interfaces

5 REFERENCES

-