|  |
| --- |
| **BILKENT UNIVERSITY**    **COMPUTER ENGINEERING DEPARTMENT**    **CS 319 – OBJECT ORIENTED SOFTWARE ENGINEERING**  **SECTION 03 – GROUP 3L**    **PROJECT TITLE: Space Impact**    **DESIGN REPORT**  **Second Iteration**  **Team Members**  **Büşra Arabacı**  **Büşra Oğuzoğlu**  **Gerard Hysa**  **Alperen Kaya**  **Deniz Şen**  [**1.** **Introduction** 4](#_Toc500013496)  [1.1 Purpose of the system 4](#_Toc500013497)  [1.2 Design goals 4](#_Toc500013498)  [**2.** **Software Architecture** 5](#_Toc500013499)  [2.1 Subsystem Decomposition 5](#_Toc500013500)  [2.2 Hardware/Software Mapping 8](#_Toc500013501)  [2.3 Architectural Styles 8](#_Toc500013502)  [2.4 Persistent Data Management 8](#_Toc500013503)  [2.5 Access Control and Security 8](#_Toc500013504)  [2.6 Boundary Conditions 9](#_Toc500013505)  [**3.** **Subsystem Services** 9](#_Toc500013506)  [3.1 Detailed Object Design 9](#_Toc500013507)  [3.2 Design Patterns 9](#_Toc500013508)  [3.3 User Interface Subsystem 11](#_Toc500013509)  [3.3.1 MainMenu Activity 11](#_Toc500013510)  [3.3.2 LevelActivity Activity 11](#_Toc500013511)  [3.3.3 StoreActivity Activity 11](#_Toc500013512)  [3.3.4 StartLevelActivity Activity 11](#_Toc500013513)  [3.3.5 PauseMenu Activity 11](#_Toc500013514)  [3.3.6 GameActivity Activity 11](#_Toc500013515)  [3.3.7 GameOverActivity Activity 12](#_Toc500013516)  [3.4 Game Management Subsystem 12](#_Toc500013517)  [3.4.1GameView Class 12](#_Toc500013518)  [3.4.2 GameData Class 12](#_Toc500013519)  [3.4.3 LevelCreator Class 12](#_Toc500013520)  [3.4.4 Store Class 12](#_Toc500013521)  [3.4.5 GameEngine Class 12](#_Toc500013522)  [3.5 Game Elements Subsystem 13](#_Toc500013523)  [3.5.1 GameObject Interface 13](#_Toc500013524)  [3.5.2 Bullet Class 13](#_Toc500013525)  [3.5.3 Spaceship Abstract Class 13](#_Toc500013526)  [3.5.4 NPC Abstract Class 13](#_Toc500013527)  [3.5.5 UserSpaceShip Class 13](#_Toc500013528)  [3.5.6 EnemySpaceship Class 13](#_Toc500013529)  [3.5.7 BossSpaceship Class 13](#_Toc500013530)  [3.5.8 BehaviourPolicy Class 14](#_Toc500013531)  [3.5.9 EnemyBehaviour Class 14](#_Toc500013532)  [3.5.10 EasyBehaviour Class 14](#_Toc500013533)  [3.5.11 MediumBehaviour Class 14](#_Toc500013534)  [3.5.12 HardBehaviour Class 14](#_Toc500013535)  [3.5.13 Collectable Interface 14](#_Toc500013536)  [3.5.14 Coin Class 14](#_Toc500013537)  [3.5.15 PowerUp Class 14](#_Toc500013538)  [**4.** **Glossary** 14](#_Toc500013539) |

# **Introduction**

## Purpose of the system

Space Impact is an old arcade game which is based on shooting the enemies and completing levels. In our project, we are recreating this old classical game to make an appealing version of it and adapt it to the new technologies and styles. To reach this goal, we are planning to provide a system that keeps the old and loved features of the game and adds new features that enriches and supports them. Diverse game elements such as enemies, rewards, upgrades and power ups gives the novelty and excitement that the game needs. In addition, our system aims to provide comfortable user interface and glorified visual environment that consummates the overall gaming experience.

## Design goals

In order to decide the design of the overall system and create subsystems effectively, the importance of determining design goals cannot be overestimated. Therefore, before the system design, we examine and discuss non-functional requirements carefully and clarify the design goals. Design goals of our system is described in the following sections.

**Ease of use:**

The main purpose of our system is to provide an enjoyable gaming experience to user so it is very important to make the gaming environment easy to use and not distracting. Since the complications in game controls, game environment or game logic can easily affect user and cause them to leave the system, we consider this goal important. Our system will allow user to easily move the spaceship with finger and use a well-designed spaceship control panel to use other features of the spaceship. Also, by providing clear menus and easy to access pause menu by lifting finger from screen, our system will ensure easy user interface.

**Ease of learning:**

Like ease of use goal, ease of learning is important to gain user’s interest and give them a good experience. Since the game has tutorials and explanations in it, it will be easier for user to learn the game. Also, in store, all upgrades and power ups will have descriptions, user can easily understand their purpose.

**Extendibility:**

Since keeping user’s attention and always enjoying them is a crucial aim for games, extendibility of the system gains importance. The system will be able to create new levels with different maps, enemy types and difficulties. Also, our system will be suitable for new game elements, new features and functions because it will separate subsystems carefully and creating new functionalities in a subsystem will not affect the overall game.

**Portability:**

Portability is an important goal for our system because we want to make a new version of a game that will draw attention to users that know and love old version. This requires a platform that can reach wide range of users. We decided to use Android environment which is a widely used platform.

**Efficiency:**

Since games are dynamic and gaining or losing points occur immediately, they should provide continuous and error-free experience. Our system aims to provide a gaming environment that responds to user’s actions quickly and runs smoothly to prevent user from losing points that arise from system errors. In order to achieve this state, our system will create continuous game screen scroll and uninterrupted spaceship control facility. Also, our system will have quick response time to user’s actions in order to keep the attention of users.

**Trade offs:**

**Ease Of Use and Ease of Learning vs. Functionality:**

The main concern of our system is to entertain user and keep user’s attention longer. To achieve this, it is important to make the system both easy to use and easy to learn. Complicated systems tend to decrease the motivation of user to continue the game which contradicts our system’s main goals. This means our system will avoid complicated and extra functionalities. We tried to carefully examine the user’s actions and avoid unnecessary functions and actions for both user and game.

**Performance vs. Memory:**

Our system cares creating impressive visual effects and animations which is a high cost for memory. For example, explosion effects, shooting effects and power ups effects are considered important for game and our system will try to make them realistic as much as possible. However, creating these effects may lower the performance which is an important concern for the system. So, we decided to increase the memory not to compromise performance.

# **2.** **Software Architecture**

## 2.1 Subsystem Decomposition

In order to see how the system is divided, subsystems are created. Subsystems interact with each other independently so to create a software that can be easily extended, modified and has great performance. In this way the division into subsystems makes it easier even for the team during the implementation stage. Everything will be ordered and have its own place.

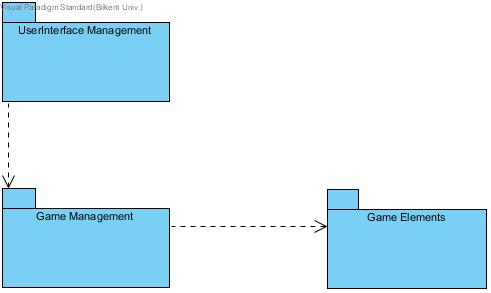


Figure 2.1 Subsystem Decomposition Representation

In Figure 2.1 the subsystem decomposition is represented which consists of three subsystems that interact with each other. User Interface Management interacts with Game Management which is closely related and interacts with the Game Elements. A division of this manner gives the opportunity to locate faster a bug without interfering with the other parts. As it will be seen in the deeper look below the classes that perform tasks related to each other are grouped in a subsystem. In order to implement this software MVC architectural pattern will be used which will give us the need organization of the project. In this way we will have a model, a view and a controller. The controller is the subsystem which will send and get data back and forth from the view and the model so to create an interaction between them.

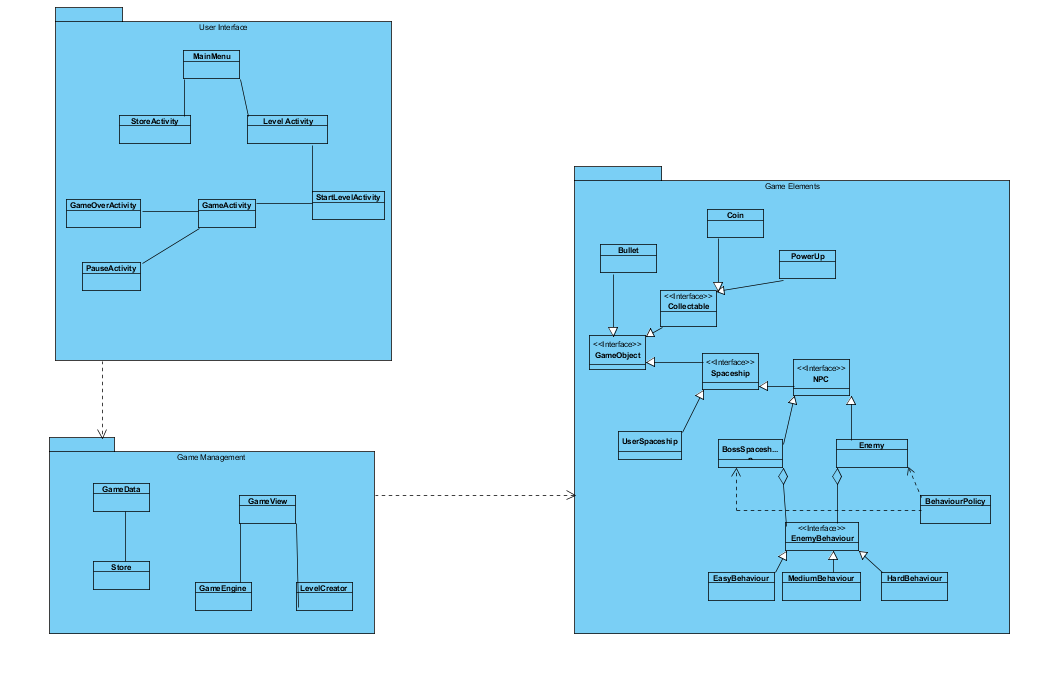


Figure 2.2 Subsystems detailed

In Figure 2.2 a deeper look into the subsystems composition is shown. The User Interface Management will contain the “MainMenu” class from where the user can select from the different options displayed. “MainMenu” class has extended classes such as “StoreActivity”, “LevelActivity”. “LevelActivity” class has extended class called “StartLevelActivity”. Additionally, “StartLevelActivity” is a parent class of “GameActivityClass”. Finally, the classes that extend “GameActivityClass” are “GameOverActivity” and “PauseActivity”. From the User Interface Management different requests from the user are sent to the Game Management subsystem which contains the “GameEngine” class which will control all the requests and get all the necessaries from the Game Elements subsystem. Furthermore, “GameData” class deals with the information based operation such as highscore, store operations by the help of the Game Elements subsystem. In Game Management subsystem all the inputs from the user will be received and the state of the game will be updated. In Game Elements subsystem are located all the needed classes for the objects of the game. One of the special design decision is that in Game Elements subsystem, we decided to use strategy design pattern for selection of algorithms called “BehaviourPolicy” class and a “EnemyBehaviour” class which is a generalization for different behaviour algorithms. This design pattern is beneficial for our game because different algorithms for different enemy types can be added easily and without affecting any other classes and subsystems.

## 2.2 Hardware/Software Mapping

Our game will be implemented in Java and Android, since it will be a game played in mobile phones. In order to implement this in android we will need all the SDK tools and a Java Runtime Environment. The game will be played in phone, so the phone itself will be the hardware needed to play the game. The player by using touch can control the game so no extra hardware needed for this game. Android has a custom 2D graphics library which will help us to draw our animations. The program will not contain heavy graphics but still we will keep a balance between nice gameplay and usability so almost everyone can play it in their phones with different android versions. The program will not require any connection to the internet to store, but everything will be stored in the phone’s memory

## 2.3 Architectural Styles

For this game we will use the MVC architectural style which consists of three subsystems that interact with each other, the model, the view and the controller. By using this style the view part which is the User Interface Management is put apart and forms a separate subsystem. The model of the MVC consists of our Game Elements subsystem where all the objects of the game are divided into classes and interact with each other. In order to provide the connection between the model and the view, the controller stands between them sending and receiving requests. The controller part consists of our Game Management subsystem, where the requests from the view are handled from the manager classes and the needed data is sent to the model. MVC is a great choice for games since it provides a smooth work during the implementation and mostly all of the changes that happen in a part do not affect the other parts.

## 2.4 Persistent Data Management

Space Impact will only use the internal hard drive of the device. The game will not use any kind of cloud or server storage to keep the data. The data will be saved before and after the game time since the saving data on the storage is an expensive process. Saving the data during the game might cause stuttering issues for the game. The game will load the background and figure images, sprites and sounds during the gaming process as expected. These data will be stored in the internal data as well.

## 2.5 Access Control and Security

Space Impact does not require any kind of authentication, therefore the game does not require any kind of personal data to play the game. For that, the security is not a big concern for this game. Thus, the game data will not be encrypted. The game data will be only accessible through the “Game Management” package, which not only ensures the security of the data, but also make the data more easy to manipulate.

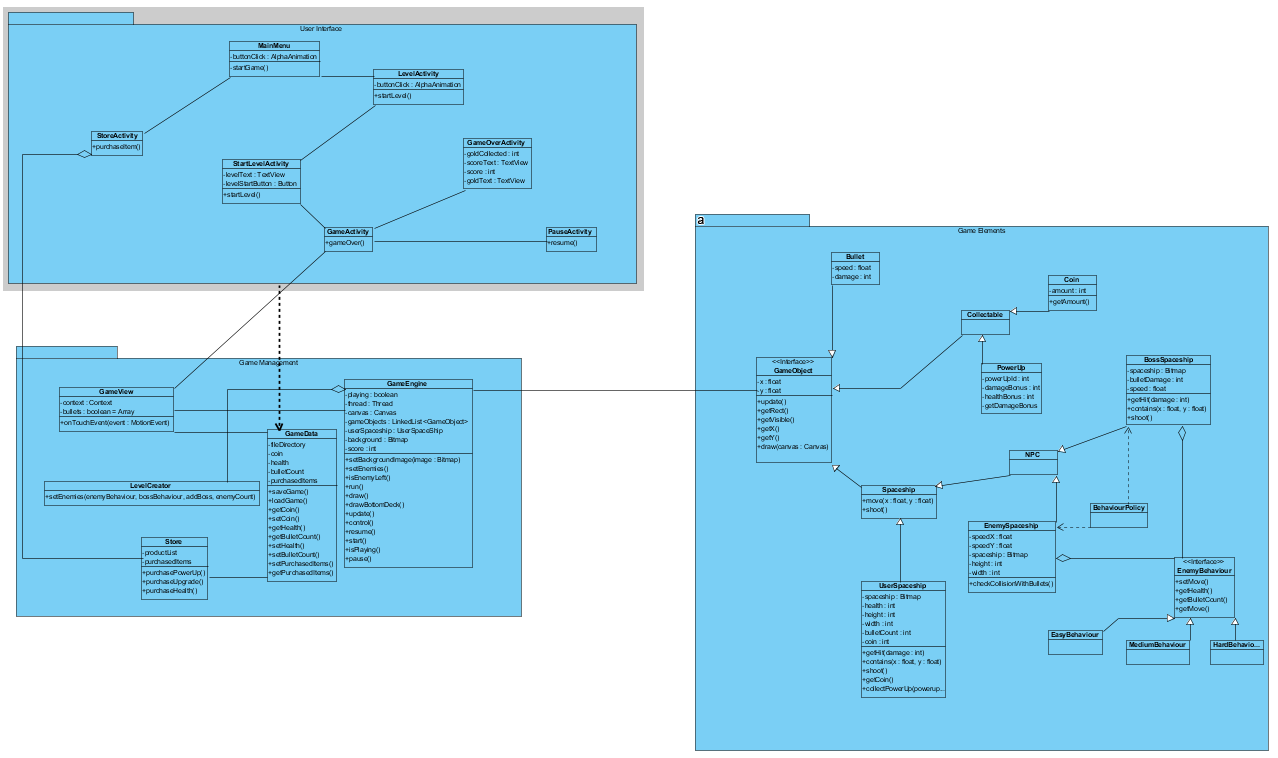
## 2.6 Boundary Conditions

The game will check for a user data file at the start of the program. If such file does not exist, the game will automatically create a new one and put it under usage. If the game ends either by the player completing the level or losing all the health, the game will open the result screen and in the case of gaining the highest score of the level, the data file will update the highest score of the level.

# **3.** **Subsystem Services**

## 3.1 Detailed Object Design

To understand the subsystems, their behaviours and relationships with other subsystems, detailed object diagram is provided. This diagram also shows classes in subsystems which is abstracted in subsystem decomposition part for simplicity. While creating this model, associations and behaviours of classes examined carefully and subsystems from closely related classes.



## 3.2 Design Patterns

We used several design patterns in our subsystem decomposition in order to provide neat solutions for problems related to communication and coupling between classes or implementation. These patterns are facade design pattern and strategy design pattern.

3.2.1 Facade Design Pattern

Facade design pattern is a solution for reducing coupling between subsystems. In our system, we have three subsystems which communicates with each other: User Interface, Game Management and Game Elements. By providing facade classes to these three subsystems which implements high-level interfaces of subsystems, we simplified the interactions between them. With facade design pattern, subsystems’ communications are based on the high-level methods that interfaces provide and this shields the low-level classes.

3.2.2 Strategy Design Pattern

Encapsulating different algorithms and mechanishms is a problem we faced when designing enemy classes. Enemies in our game behaves differently which requires a solution for selecting among different enemy behaviours. Strategy design pattern is used for providing different algorithms or strategies and selecting them. In our system, we created a policy class for selection of algorithms called BehaviourPolicy class and a EnemyBehaviour class which is a generalization for different behaviour algorithms. This design pattern is beneficial for our game because different algorithms for different enemy types can be added easily and without affecting any other classes and subsystems.

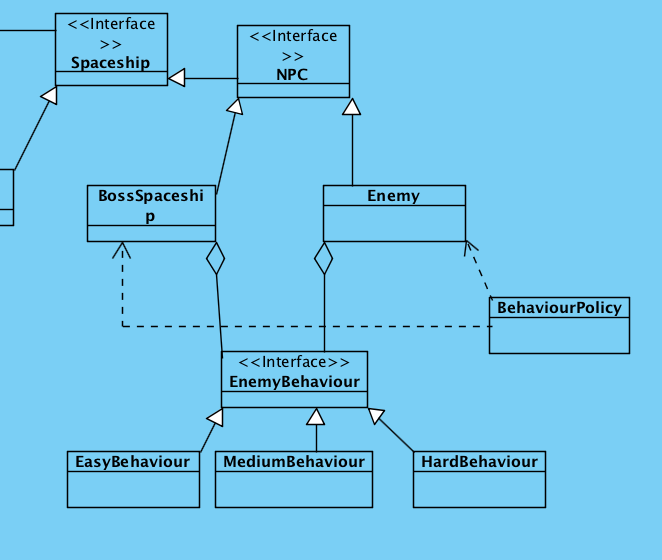


Figure: Strategy desing pattern application

## 3.3 User Interface Subsystem

Providing user interface for playing game and using features of the game is an important part of the overall project. User Interface Subsystem is responsible for presenting well-designed, easy-to-use interface to user, collecting actions and informations from user and transmit them to Game Management Subsystem. There are Menu classes to receive user’s preferences for application such as starting game or going to store. Also, another important interface is needed when playing game in order to control spaceship which is handled by SpaceshipControlPanel. This class is in charge of providing interface to user for power-ups management, shooting and checking remaining lifetime, bullets or power-ups.

### 3.3.1 MainMenu Activity

This activity holds the first screen that the user will interact with. The user can either proceed to level selection screen or go to the store.

### 3.3.2 LevelActivity Activity

This menu will appear before the user starts the game. Here, the level selection will be made as well as the starting power-up selection. This menu has an array of buttons all of which indicate the number of a certain level. Also the power-up selector will determine the power-up selection of the user.

### 3.3.3 StoreActivity Activity

The user can buy upgrades, health and powerups in this menu. The menu will also check the availability of the coins.

### 3.3.4 StartLevelActivity Activity

This activity pops up when the user selects a level to play. There is a confirmation message for the game to begin.

### 3.3.5 PauseMenu Activity

This menu appears whenever the user releases their finger from the game screen while the GameState is on. The user can either continue gaming or terminate the game and go to the main menu.

### 3.3.6 GameActivity Activity

The surface holder of the game screen. The canvas of the game will be drawn on this activity.

### 3.3.7 GameOverActivity Activity

This screen will pop up when the game is resulted in either defeat of victory. It will display th score, the number of enemies killed and coins collected. The user will then go to the main menu.

## 3.4 Game Management Subsystem

Game Management Subsytem is the main place where all actions from user is processed and all game elements are controlled accordingly. The gameloop, transactions, data management and level creation will all be done in this subsystem.

### 3.4.1GameView Class

This is a custom view which extends built-in canvas class that Android echosystem offers. It is used by the GameEngine which handles most of the game actions. One other major function of this class is that it handles the user inputs because only the canvas itself can handle the inputs of the user. Hence it sends the user inputs to GameEngine.

### 3.4.2 GameData Class

This class contain static variables and classes since only one file is sufficient for the game. It saves and restores the current state and attributes of the user’s spaceship.

### 3.4.3 LevelCreator Class

A standalone engine which will create the enemies and collectable items of the game surface. It will be called when the GameEngine is constructed.

### 3.4.4 Store Class

This class will be instantiated when the user goes to StoreActivity. It will hold the items taht can be bought and manipultes the transactions. It will work with GameData since it will often update the items that the user has.

### 3.4.5 GameEngine Class

This class is the main class which will handle the changes on the game surface, as well as the changes on the screen. This class gathers changes of every single game element of the game. This class sets the canvas up, manipulates the data structures of GameObjects, such as the list of GameObjects and Bullet list. It contains the game loop. It will be instantiated inside GameView as GameEngine must have a canvas ready. The game loop will stop when the game is finished and paused.

## 3.5 Game Elements Subsystem

### 3.5.1 GameObject Interface

This interface is very important for the simplicity of the system. Everything that the user sees on the gameplay, must be a GameObject which has an x-coordinate and a y-coordinate. Also, every object should draw itself on the screen with the render() method, update its position with update() method and remove itself from the memory when it is no longer used with dispose() method.

### 3.5.2 Bullet Class

This object is used to indicate the bullets that come from the spaceships. Each bullet has an owner and a damage amount. These values are hold as properties of each object. Since this object will be created many times during the game, its manipulation is very important or it might cause some severe stuttering issues.

### 3.5.3 Spaceship Abstract Class

The subjects of the game are the spaceships and there can be several types of spaceships. Every spaceship must have some commonalities, which this abstract class puts as obligation. Every spaceship must have an amount of maximum health, have a move() method, have a shoot() method and have a getHit(damageAmount) method.

### 3.5.4 NPC Abstract Class

This class is for the non-player characters that are the enemy spaceships. These spaceships have a fixed intelligence level and a weapon level to indicate their strength. The escape() method is used when an allie bullet hits the spaceship, so it will not take any damage.

### 3.5.5 UserSpaceShip Class

This the spaceship which the user controls. It has a limited amount of ammo and an inventory which holds the collected things. It might regain health during the game and activate power-ups.

### 3.5.6 EnemySpaceship Class

This class does not have a special method to operate because the methods that NPC abstract class obligates are enough.

### 3.5.7 BossSpaceship Class

This kind of spaceship has some different kinds of shooting styles which can be changed using the changeAttackStyle(style) method. Every boss appear at different levels, so it is indicated by the level attribute of the object.

### 3.5.8 BehaviourPolicy Class

This class is part of the strategy design pattern that we applied for selection of enemy behaviours. This class determines which type of enemy needed and informs the Enemy and BossEnemy classes.

### 3.5.9 EnemyBehaviour Class

This is an interface which is part of the strategy design pattern. It unites EasyBehaviour, MediumBehaviour and HardBehaviour classes and provides the high-level methods.

### 3.5.10 EasyBehaviour Class

This class is part of the strategy design pattern that represents the algorithms for enemy behaviours in easier levels.

### 3.5.11 MediumBehaviour Class

This class is part of the strategy design pattern that represents the algorithms for enemy behaviours in medium levels.

### 3.5.12 HardBehaviour Class

This class is part of the strategy design pattern that represents the algorithms for enemy behaviours in harder levels.

### 3.5.13 Collectable Interface

This is an interface which covers game objects that can be collected during the gameplay. It is only needed for easy manipulation purposes.

### 3.5.14 Coin Class

This object has only 1 attribute that is its value since as the game proceeds, the values of the coins might increase.

### 3.5.15 PowerUp Class

This object is used during the gameplay but can only be received from the store by purchase. The power-ups have a particular type which is a property of the object.

# **Glossary**