

# Software Design Description

Group: "Patrik-o-Co"

## 1 Introduction

### 1.1 Design Overview

### 1.2 Build procedure

## 2 System Architectural Design

### 2.1 Chosen System Architecture

#### 2.1.1 MVC

#### 2.1.2 The GameLoop

#### 2.1.3 Strategy pattern

### 2.3 System Interface Description

## 3 Detail Description of Components

### 3.1 Component StartGameActivity

### 3.2 Component GameViewActivity

### 3.3 Component GameView

### 3.4 Component GameLoopThread

### 3.5 Component HighScoreActivity

### 3.6 Component HighScores

### 3.7 Component OptionsActivity

### 3.8 Component AboutUsActivity

### 3.9 Component HelpActivity

### 3.10 Component GameOverActivity

### 3.11 Component Level

### 3.12 Models

#### 3.12.1 Component GameObject

#### 3.12.2 Component MovingObject

#### 3.12.3 Component SpaceShip

#### 3.12.4 Component Enemy

#### 3.12.5 Component Asteroid

#### 3.12.6 Component AlienShip

#### 3.12.7 Component Projectile

#### 3.12.8 Component Explosion

#### 3.12.9 Component LifeBar

#### 3.12.10 Component ParallaxBackground

### 3.13 Audio

#### 3.13.1 Component GameMusic

#### 3.13.2 Component SoundEffects

## 4 User Interface Design

### 4.1 Description of the user interface

### 4.2 Screen Images

### 4.3 Objects and Actions



# 1 Introduction

This document specifies the requirements for the production and design of the Gradiuss software for the Google Android operating system. The product will be a shoot'em all game for a user to play on devices running the Android OS. The game Gradiuss lets the user steer a spaceship that shoots and avoids obstacles such as asteroids and aliens that attack the spaceship. The user interface will be easy to use and allow the user to manage the application with ease. It will use the cell phone operating system Android, which is on a lot of cell phones today. The android operating system uses a market place to sell applications for the phone.

## 1.1 Design Overview

See UML document in repository. (dokument/umlGradius.png)

## 1.2 Build procedure

see Gradius development guide. (dokument/Gradius development guide.pdf)

# 2 System Architectural Design

## 2.1 Chosen System Architecture

Our Android Project Gradiuss will be built with the Google Android Software Development Kit (SDK) for Java. It will be implemented with Java and XML. The base controlling classes will be Level'N'(LevelOne) meaning the class that extends Level(class) that is used in GameView which in turn is used by GameViewActivity. The GameViewActivity is the screen where the user plays the actual game and uses an GameView instance for creating the game. The class MainMenuActivity will be the upper level class called at the start of the program. The MainMenuActivity class will manage the user and guide him/her through out the outside-activities of the game such as GameViewActivity where the game is played, OptionsActivity where volume for sounds are decided, HelpActivity where instructions about how to play the game are described, AboutActivity where description about the team creating the game is presented, HighScores where the 10 highest scores can be seen, and GameOverActivity where the player ends up when killed.

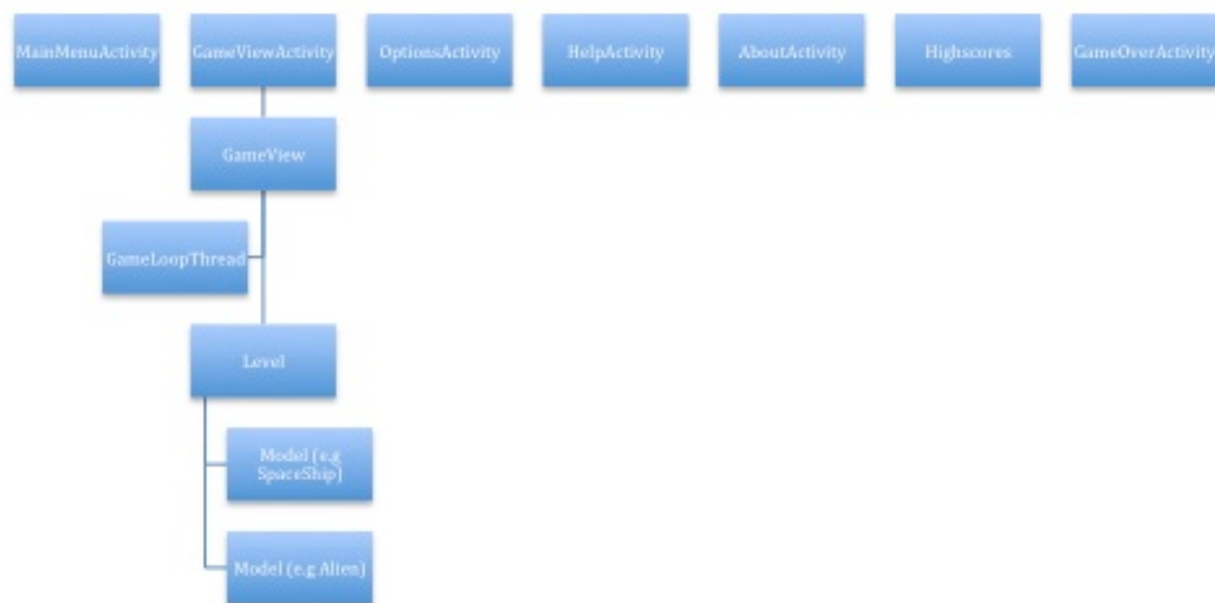


Fig. Architectural Design for Gradius

### 2.1.1 MVC

To create the GameView we have used the design pattern Model-View-Controller. The pattern is basically used for separating concerns for data management and is preferably used in designing games with user interfaces. The Model describes the objects used in the game and contains the behaviour of the specific object and how to transform data by the specific object. The View renders the data that has been transformed so the user sees the new state. The Controller in turn notifies the Model through the Observer pattern by implementing listeners and observers in the View. In our application the Model is represented by the models (e.g. SpaceShip) and the View by the GameView class. The Controller is represented by the GameViewActivity that interacts with the user through buttons. Some of the model objects (e.g. Aliens) in the application are controlled by time events and random variables.

### 2.1.2 The GameLoop

The GameView also uses a Gameloop. "The central component of any game, from a programming standpoint, is the game loop" (wikipedia). The GameLoop is used so the game is continuing regardless of the users input and is necessary for generating the environment for the game. The GameLoop runs as an thread and regenerates events that the GameView listens to and when a new event has occurred the event is notified to the models in the GameView and thus transforms the old data.

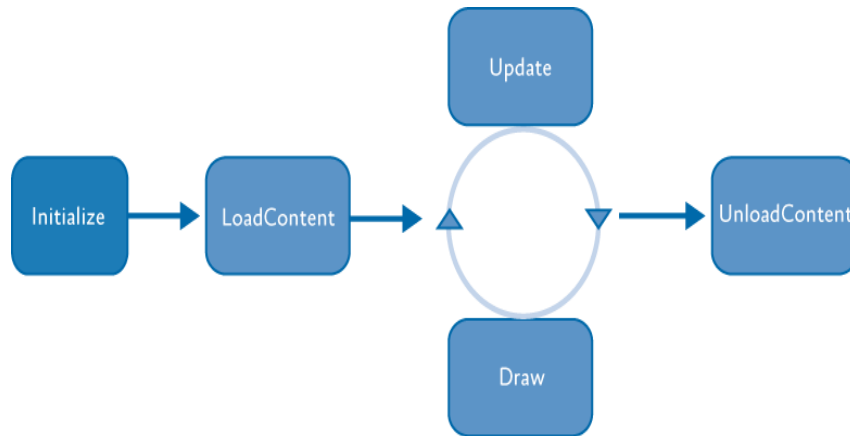


Fig. GameLoop Overview

(src: <http://iloveshaders.blogspot.se/2011/04/anatomy-of-xna-project.html>)

### 2.1.3 Strategy pattern

The State pattern is useful when objects need different behaviour depending on the internal state of the application. This pattern has been used to create the Levels of the game (currently 2012-05-17 we have only one level, i.e. LevelOne). The Levels of the game extend the abstract class Level that is referred to in the GameView so different levels are applicable in the Game. The different levels of the game represent both different difficulties and different environments in the game to give the user a heightened experience.

## 2.3 System Interface Description

The application is specified and implemented on the Android platform. The game will run on two different machines. One is an emulator of a cell phone that is used for development and testing. The other is the user's cell phone that hosts the game. The game must be compatible with the Android operating system.

## 3 Detail Description of Components

### 3.1 Component StartGameActivity

This class is the main menu that pops up when the user clicks on the application icon. It contains several buttons that take the user to different screens, like options, highscores table and so on.

### 3.2 Component GameViewActivity

The GameViewActivity is the activity-class that creates a new GameView. It is the Controller part of the MVC-pattern, containing game pads and buttons that handle user interaction in the by registering touches from the user.

### **3.3 Component GameView**

This class, together with the GameLoopThread-class, can be viewed as the heart of the game. The class extends the SurfaceView-class from the Android Framework and handles everything that happens on the screen when the user is playing. It contains all the levels that the game is made up of and a game-loop (a GameLoopThread-object).

### **3.4 Component GameLoopThread**

The other really important class that handles the updating and drawing of all the models and bitmaps. The class also keeps track of how many frames the execution goes through per second, or the FPS (frames per second).

### **3.5 Component HighScoreActivity**

Is an activity that has assigned the buttons on the xml to so that the main menu can be accessed and the highscores can be viewed.

### **3.6 Component HighScores**

The highscore class creates a database to store the highscore values in. The class has an inner class to help in creating the database which is called DbHelper. The class helps in inserting new values in the table that has been created in the database. The ten most highest scores are represented when the information is fetched from the table.

### **3.7 Component OptionsActivity**

The OptionsActivity is the activity that shows the options that the user can choose from. It contains two seekbars that controll the volume of the music and the sound effects.

### **3.8 Component AboutUsActivity**

This activity presents the creators of the game and and the authors of this documentation.

### **3.9 Component HelpActivity**

The HelpActivity-class shows a description of how the game should be played. It describes what the different buttons do and what the standard objects on the screen represents.

### **3.10 Component GameOverActivity**

The GameOverActivity is a class that handles the input the player name and storing the higscore value which is sent from the class ScoreCounter. These values are inserted and then

stored into the db which is in the HighScores class. This class has buttons in it which can send the user to different screens e.g. the main menu or restart the game.

### **3.11 Component Level**

The Level-class is an abstract class that represents a level in the game. A new level is added to the game by extending this class and adding the specific level-functionality. The class contains three abstract methods: `initializeLevel()`, `updateLevel()` and `renderLevel()`. These methods are implemented by the extending class `LevelOne`. Future levels are `LevelTwo`, `LevelThree` etc. These classes are supposed to extend the Level-class and add their own functionality by these three methods.

### **3.12 Models**

The models in the game extend from the `GameObject`-class or `MovingObject`-class depending on if they are supposed to be static objects like lifebars and score-counters, or moving dynamic objects like enemies. Here is a summary-list of all the models.

#### **3.12.1 Component GameObject**

The `GameObject`-class is an abstract superclass for most of the models in the project. This class contains a bunch of set and get methods as well as three important methods: `initialize`, `updateState` and `draw`. The `initialize` method is used to initialize the model. `updateState` updates the model and gives it a specific behavior. The `draw` method takes a `Canvas` object as a paramter and draws the object on the `GameView`. This way, every object is responsible for it's own updating and rendering.

#### **3.12.2 Component MovingObject**

The `MovingObject`-class extends the `GameObject`-class with movement functionality, by extending it and adding methods. The class is an abstract class and thus it is not possible to create instances of it. It contains fields for speed and movement in certain directions as well as setters and getters for these.

#### **3.12.3 Component SpaceShip**

The `SpaceShip`-class extends from `MovingObject` and is the model that is controlled by the user by pressing the pad-buttons. It implements the methods `updateState()` and `draw()`, and extends them with the functionality of movement in the x and y directions. It also contains getters and setters for fields like *shooting* and *hit*, where shooting controlles if the spaceship should shoot projectiles and hit determines if the spaceship got hit by an enemy or an enemy projectile.

#### **3.12.4 Component Enemy**

This is an abstract class that extends from `MovingObject`. It contains all the necessary information that an enemy should have in the game. The fields `damage` determines how much damage it should do to the spaceship if they collide, and the field `life` determines the life of the enemy.

### **3.12.5 Component Asteroid**

The class extends from the Enemy-class and represents an Asteroid-enemy. The asteroid implements the methods `updateState()` and `draw()` from the `GameObject`-class. The `updateState()`-method makes the size of the Asteroid's smaller if it gets hit by a projectile.

### **3.12.6 Component AlienShip**

This class represents an alien-spaceship that is a little more advanced than the Asteroid-enemy. It extends from the Enemy-class. It's `updateState()`-method updates its position so that it follows the `SpaceShip`. It also shoots projectiles when it's x-position is in line with the `SpaceShip`. These features are created to give some sense of AI to the enemies and to give a better and more challenging user-experience.

### **3.12.7 Component Projectile**

The abstract superclass for all the projectiles in the game. It's subclasses are `TypOneProjectile` and `AlienProjectile`. The `Projectile`-class extends from `MovingObject` to make the projectiles movable.

### **3.12.8 Component Explosion**

This class represents an explosion that occurs after an object has been destroyed. It's `updateState()`-method flips between 9 different explosion bitmaps to give the impression of an explosion.

### **3.12.9 Component LifeBar**

This class is a life bar that is drawn in the top-left corner of the screen. It keeps track of the life of the spaceship and has methods for decreasing and increasing the lifebar. The class extends from `GameObject`.

### **3.12.10 Component ParallaxBackground**

This class represents a background that draws several copies of bitmaps. It moves them down the screen to create an impression of constant movement through space. The class contains a list of arrays. Each array contains several `Background`-objects with copies of the same bitmap and the class automatically creates the right amount of copies. To add a background-layer to the list just call the `addBackground()`-method and pass a bitmap and the speed at which it should be moving.

## **3.13 Audio**

The game contains two classes that handles the audio.

### **3.13.1 Component GameMusic**

The `GameMusic`-class contains a `MediaPlayer`-object and some setters and getters for setting the resource-id for the audio-files to be played.



### 3.13.2 Component SoundEffects


The SoundEffects-class contains a SoundPool-object to handle short sounds representing explosions and firing weapons etc.

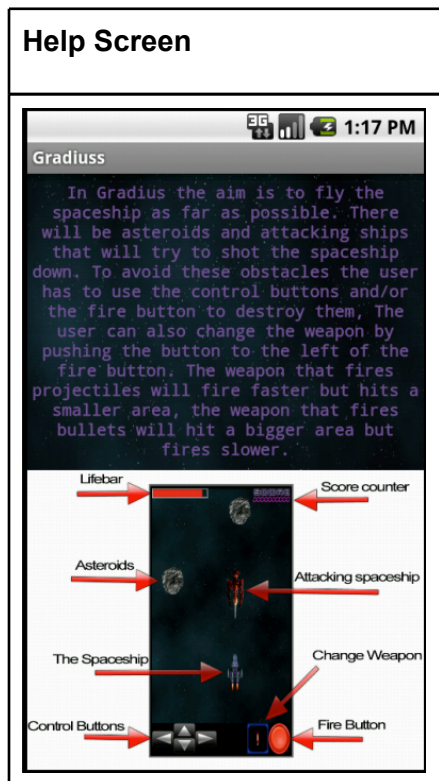
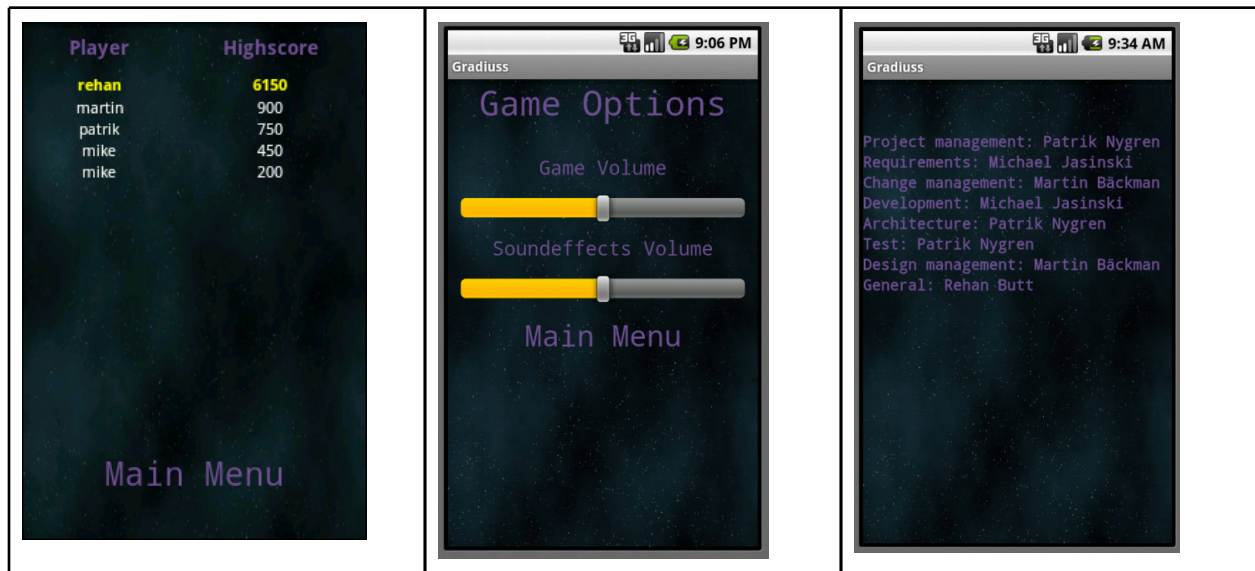
## 4 User Interface Design

### 4.1 Description of the user interface

The user interface are divided into 7 major activities: AboutActivity, GameOverActivity, GameViewActivity, HelpActivity, OptionsActivity, HighScoreActivity and the StartGameActivity. The user will be able to navigate through these screens, except the GameOverActivity which will come up automatically when you “die” and the game is over.

### 4.2 Screen Images

Start Screen	Game Screen	GameOver Screen
		
HighScore Screen	Options Screen	About Screen



## 4.3 Objects and Actions

The screen interfaces will all have buttons using a `OnClickListener()` which help to navigate between menus. The hardware button “Return” will allow the user to go back to the previous menu and since there are only one level of menus the user will always come back to the main menu. The Start Game Screen are the main menu and from this screen the user can make five choices. Depending on what the user choose, different screens will come up. The Help Screen

will help the user to know how to play the game. In this screen there is a text which describes what the goal of the game is. There is also a capture of the play screen and the and an explanation of what happens when they are pushed. In the About Screen there is a list of the people involved in the project and what roles they had in the project. In the Options Screen the user can change the volume of the music that is played when the game starts. The user can also change the the volume of the sounds effects that could be heard when shoots are fired and when the opponents explode. The HighScore Screen displays a list of the highest scored achieved in the game. When the user “dies” (i.e when the Spaceship gets hit and has no more life) the GameOver Screen comes up and the user can write its name. This name and the result will be saved in a HighScore, that could be reached when pushing the HighScore button in the main menu. In the Game Screen the user can push six different buttons. Four of these buttons, placed in the left corner of the screen, will make the Spaceship move in different directions. Pushing the left button will make the spaceship move to the left, pushing the upper button will make the Spaceship move upwards and so on. The red button to the right of the screen will make the Spaceship fire shoots. The button to the left of the fire button will change the weapon that is used by the Spaceship. The user can choose to shoot either projectiles or bullets. The projectiles will be fired with shorter intervals but will hit a smaller area. The bullets will hit a bigger area but are fired with longer intervals compared to the projectiles. In the left top corner of the game screen there is a “lifebar” which displays how much more damage the spaceship can take before it destroys. When the spaceship gets hit by the asteroids or the attacking spaceship or by shoots from the attacking spaceship this bar gets reduced. When the bar goes down to the bottom the game is over and the user comes to the game over screen. In the right corner of the game screen there is a score counter. When asteroids and the attacking spaceships are shot down the user receives points. When the game is over, then this points are saved as the highscore.