Software Design Document

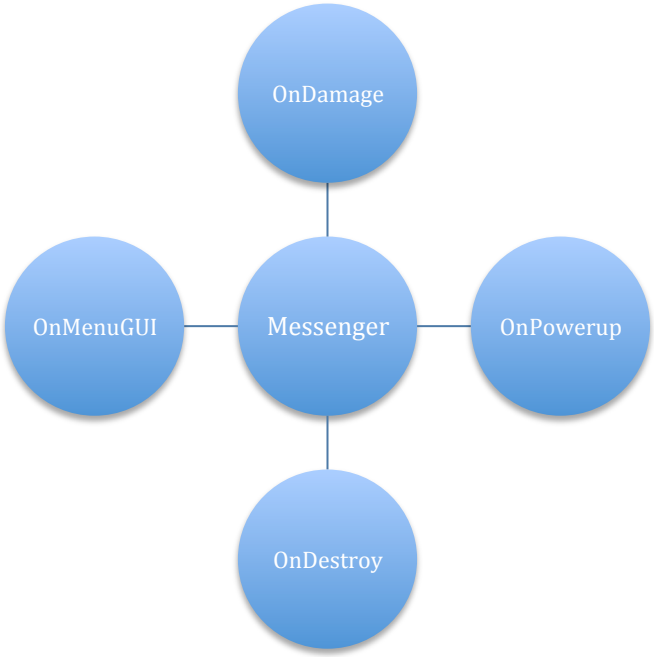
Alpha Apps
Research and Development project
Alper Aslan, Rafael Alejandro Imamgiller, Martijn Bonajo, Paul Dubbelman

System Overview

The game will be a spaceshooter provided by 3D graphics on a mobile device. It uses the features of touchscreen to control gameplay. And an additional menu screen when the end-user rotates it's device to landscape mode. This will not be an ordinary retro spaceshooter. Despite the 3D graphics it will support, there is also an ingame real-time advanced point system that tracks time and the end-users activity. Based on skill and the amount of time the user hasn't played this game, points will be provided to spent ingame. These points can be used to upgrade ingame feature. So the player can advance through levels and achieve more difficult levels. With this feature we hope to provide the end-user much time of pleasure and addiction to play a simple but powerfull gameconcept.

System Architecture

Architectural Design (AD)

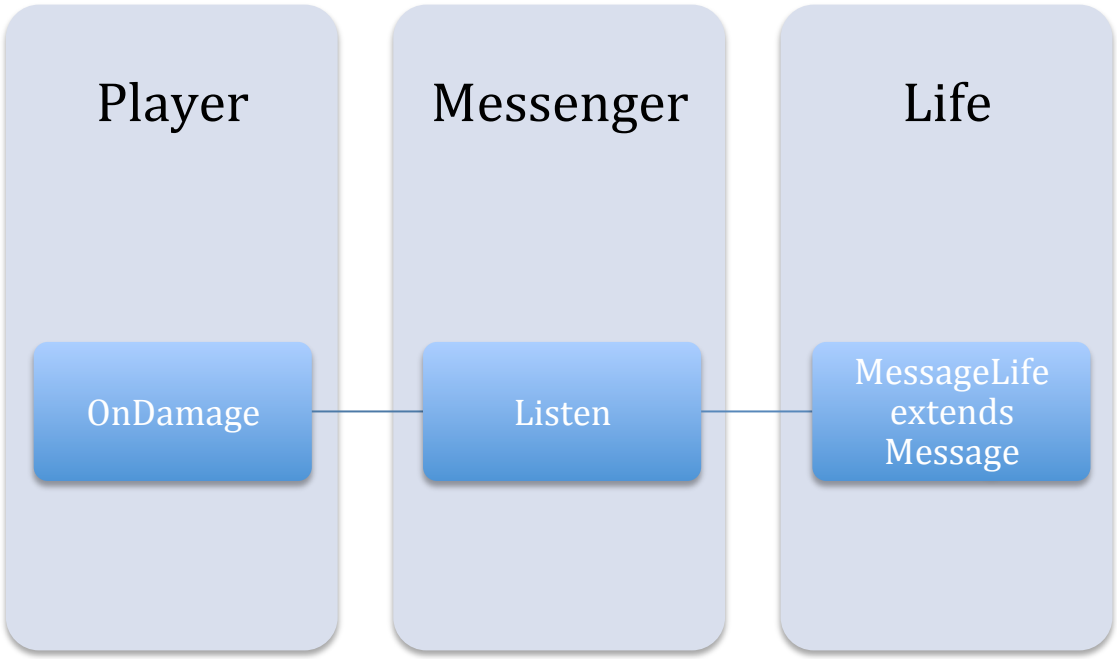


This is the Main Structure of our in game design. We are provided a Messenger script written in Javascript that is compatible with Unity. The Messenger behaves like a event-handler like in Java. There are listeners that listens to some events happening in the game like OnDamage, when a player gets hit by the enemy. The OnDamage method is called and tells te Messenger that something has happened. Every script that has subscribed for the OnDamage method will receive this

particular event and you can assign what to do next, e.g. remove one life unit from the life bar. This is a handy feature to composite the main structure of our game. The chart above gives a global overview how this works. Not all the methods are provided here, and the subscribers are not visible.

Decomposition Description (DD)

In the following diagram you can see an example how the flow works between the player and it's life. When on object hits the enemy, the OnDamage method will be called. The Messengers Listen method will listen to this call and it just passes this method to anyone that is subscribed for this particular call. Life can be subscribed to the Player's OnDamage method by extending Message and creating a MessageLife that subscribes to the Messenger's Listen. By this sequence, Life can substract one life which just will be visible on the screen. This is a very simple but powerfull technique where multiple functions can assign to one single call, i.e. an Enemy can also have an OnDamage method. When he calls this, Score can subscribe to this and can additionally add +10 score to the score counter on the screen.



Design Rationale (DR)

The structure we chose provides some great features. Also this Messenger script was available as free source on the web by FlashBang Studios. It is reusable in any game project and may be edited. This provides us to use an easy but powerful script to set up the Architecture of our game, which can be seen as the skeleton.

Data Design

Data Description

Now the game provides an real-time points system which updates while the end-user is not using the app. These information must be stored in the database with timestamps. So the system can save data when the app quits, and check how

much time has passed when the user starts the app again. With this, the system can calculate how much points can be given to the current game of the end-user. At the moment, this is the most important feature our game has to provide. And what makes it unique.

Data Dictionary

At the moment there isn't a schematic design how we are likely to set up a Data System. This will folow soon.

Component Design

There is a lot of code provided in our system, so we will not briefly show them all, but we'll give some examples.

```
class MessageLife extends Message{
    var life:int;

    function MessageLife(life:int){
        this.life = life;
        super("life");
    }

}
```

This is like in Java an object. The object MessageLife extends our Message, so it can listen to "life".

Now OnDamage can make an object of this instance:

```
new MessageLife(1);
```

So Life can take a look at how much life there has to be substracted:

```
// track our total life value
private var totalLife:int = 8;

function Start()
{
    // tell the messenger that we want to listen to messages
of type "life"
    // (remember 'super("life")' in MessageLife ?
    Messenger.instance.Listen("life", this);
}

// function for displaying life on screen
function OnGUI () {  }

function FillLife(){
    totalLife = 8;
    print("LIFE HAS BEEN FILLED!");
}

// syntax for writing a message receive function:
// take the name of the message and replace 'Message' with '_'
```

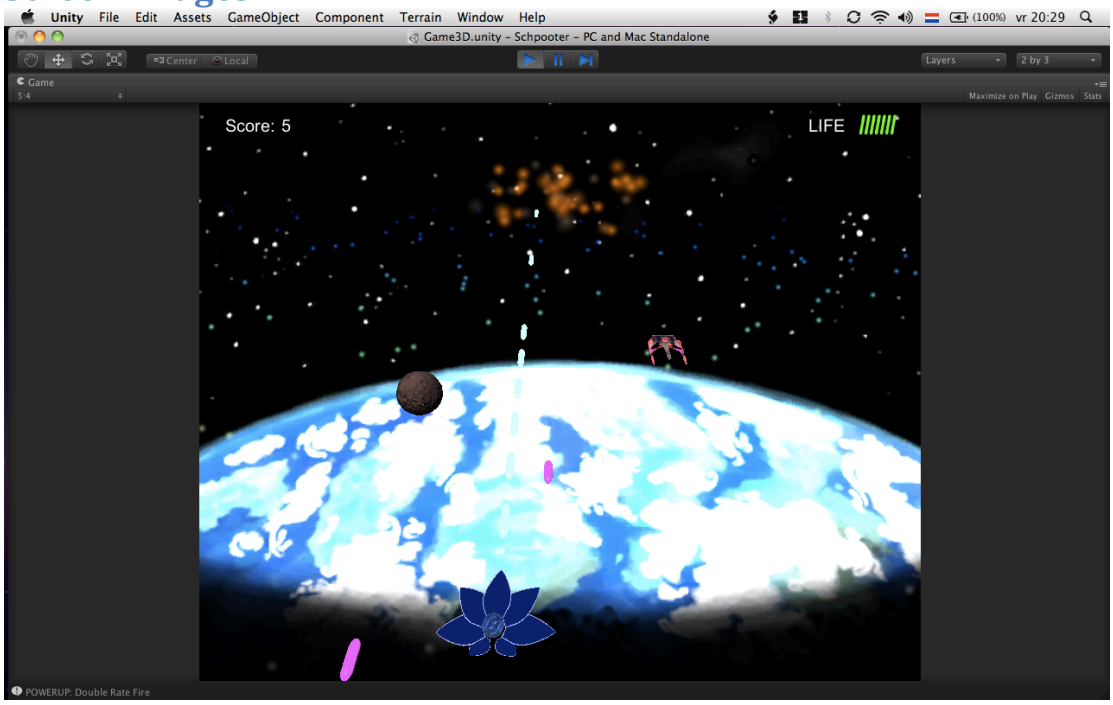
```
// e.g. 'MessageLife' becomes '_Life'
// the function should take the type of message we want to
receive
// e.g. MessageLife
function _Life(msg:MessageLife)
{
    // remember we created a variable to store the points in
the message?
    // now we can use it without needing to know where it came
from!
    totalLife -= msg.life;
}
```

Human Interface Design

Overview of User Interface

The user will be able to play the game on a mobile device supporting touch and the accelerometer. The aim device will be an iPhone. But additionally android devices will also be able to play the game. We are providing a touch based gameplay. The user has to touch the left or right side of the screen to move his ship. Within the gameplay the user has to destroy and avoid obstacles that comes toward him. In advance the user will gain points to upgrade weapons and unlock new features. The user can pause the game by turning the device into landscape mode so a Menu will popup and the game will be paused. In this Menu the user can trade points for ingame features. While ingame the user is provided by 3D graphics and on top of the screen a score of his points to spend, his life meter and a time counter. While the user is playing the game he may counter ingame features like powerups. The effect of this features will be shown on screen temporarely to notice the user of what additional feature he has gotten.

Screen Images



Screen Objects and Actions

As you can see at the bottom left corner of the screenshot above, the player just gained a Powerup that has doubled it's fire rate. The Score and Life images are visible. And the player has to dodge or destroy objects that he will counter. Like enemy's or astroids.