|  |  |  |  |
| --- | --- | --- | --- |
| **CS102** | **Spring 2015/2016** | Project Group | 5H |
| Instructor: | **Özcan Öztürk** |
| Assistant: | Dorukhan Arslan |

|  |  |  |
| --- | --- | --- |
| **Criteria** | **TA/Grader** | **Instructor** |
| Presentation |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
| Overall |  |  |

~ 2D Rocket Game ~

Rocket Fool

**Ömer Ekmekcioğlu, Alperen Erkek, Levent Berk Güngen,**

**Hakan Yaman Sözer, Hakan Türkmenoğlu**

|  |
| --- |
| Detailed Design Report  ( First Draft ) |

# Introduction

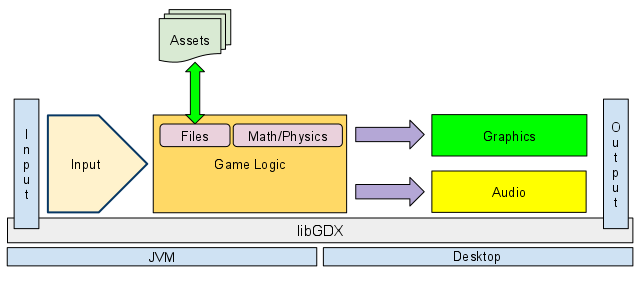
The game, currently titled “Rocket Fool” as with the group name, is a computer game that is created with the aim of making various principles of space travel, such as orbital mechanics, easy to understand for a general audience in an entertaining way. The game is primarily about piloting a rocket from one location to another in a two-dimensional space environment with realistic gravitation but reduced distances while managing additional concerns like fuel consumption and avoiding obstacles.

This report explains the project group’s proposal for the design of the game, disclosing details like the design patterns to be used and the existing software to integrate. It also provides a UML diagram of the core classes of the game. Finally, the report outlines the distribution of tasks to the group members.

# System Overview

## Application Form

* *Rocket Fool* will use the Model-View-Controller (MVC) pattern in its design (See Figure 1). This pattern is preferable for its flexibility and clarity. It allows various viewing schemes to apply to the same model. This makes it easier to try different GUI ideas and would help particularly if the game is made modifiable in the future. It also makes it easier during debugging, because it is easier to tell if a problem is due to the graphics or the model itself. The separation of the controller-related code makes it much easier to test different control ideas and makes the game much easier to port to Android, if we decide to do that in the future. As with views, separating controllers also helps during debugging.
* A singleton is used to store/retrieve persistent data, such as save data and preferences.
* The game is to be a desktop application packed into a single *.jar* file. This application format is good for the game because the game is thus stand-alone and can easily be played or distributed offline after it is downloaded once from a hosting website. The format also allows the game to be more immersive as it has its own window, which can also be made full screen. Also, it doesn’t have the setbacks/limitations (like with memory shortage) that might occur in applets.



(View)

P

l

a

y

e

r

Assets

& Saves

(Model)

(Controls)

Figure 1: System Architecture Diagram

(Modified version of a typical libGDX programs’ diagram,

Source  <https://libgdx.googlecode.com/svn/wiki/img/modules_overview_diagram.png> )

## Technologies used

* While *Rocket Fool* is written in Java, it primarily employs the Java development framework libGDX because of its handy features like its extensive API, and efficient garbage collection. It is a popular framework that is used and improved by many developers.
* The Box2D physics engine, compatible with libGDX thanks to the latter’s Java native interface wrapper, serves as an efficient physics engine to create the core of the game.
* The game uses serialization for saving scores and game progress.
* Figure 2 depicts the screens of the game. The \*\*\*

# Core Design Details

The following notes supplement Figure 2 (on the next page).

* The Level class does most of the work in the program including updating forces acting on the playable spacecraft.
* All game levels are extended from the Level class.
* The map class…
* All <particles/objects…> are extended from the abstract parent
* Level-trigger relation
* Body is a class..\*
* We use floats because
* updateSolidObjects()
* Playable objects include satellites, ships…\*
* Getters and setters were skipped on purpose, //\*\*\*
* Controller
* <başka?>

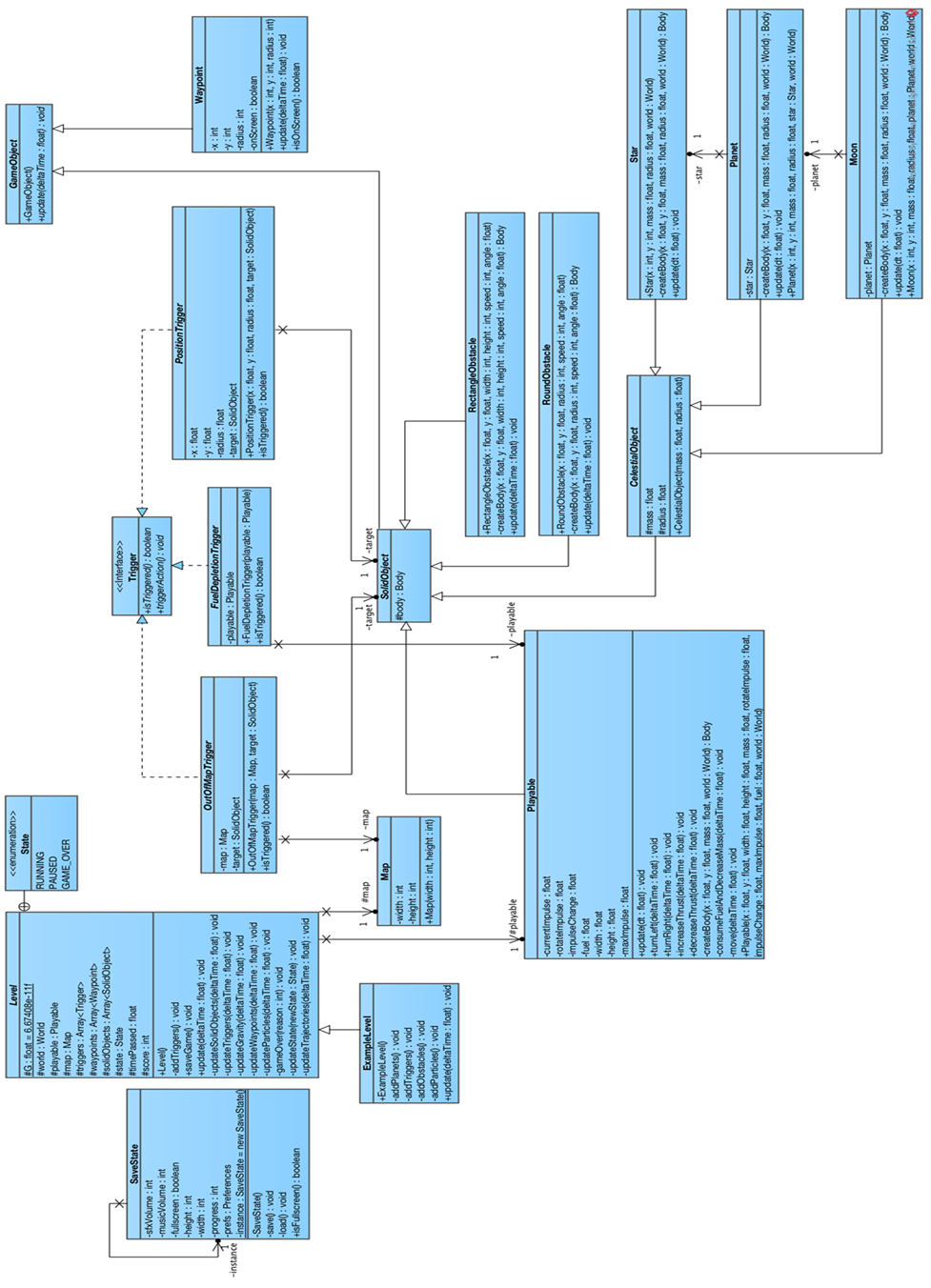


Figure 2: UML classes of model part of game (also available at https://turkmenog.lu/static/cd8.png)

# Task Assignment

## Ömer

* a

## Alperen

* a

## Levent

* a

## Yaman

* a

## Hakan

* a