|  |  |
| --- | --- |
|  | **2012** |
|  | **RMIT UNIVERSITY**  **SOFTWARE ENGINEERING 2**  **Team CYBERTECH** |

|  |
| --- |
| **[TECHICAL REPORT]** |
|  |

Table of Contents

[Design of the game DevFortress 2](#_Toc343587947)

[Difficulties encountered and overcome 2](#_Toc343587948)

[Applicability of the design patterns 4](#_Toc343587949)

[Applicability of Scrum project management 5](#_Toc343587950)

[Changes made to the specification and justification for the changes 5](#_Toc343587951)

[Additional constraints not listed in the specification 5](#_Toc343587952)

[Technology stack and tools chosen and justification 6](#_Toc343587953)

# Design of the game DevFortress

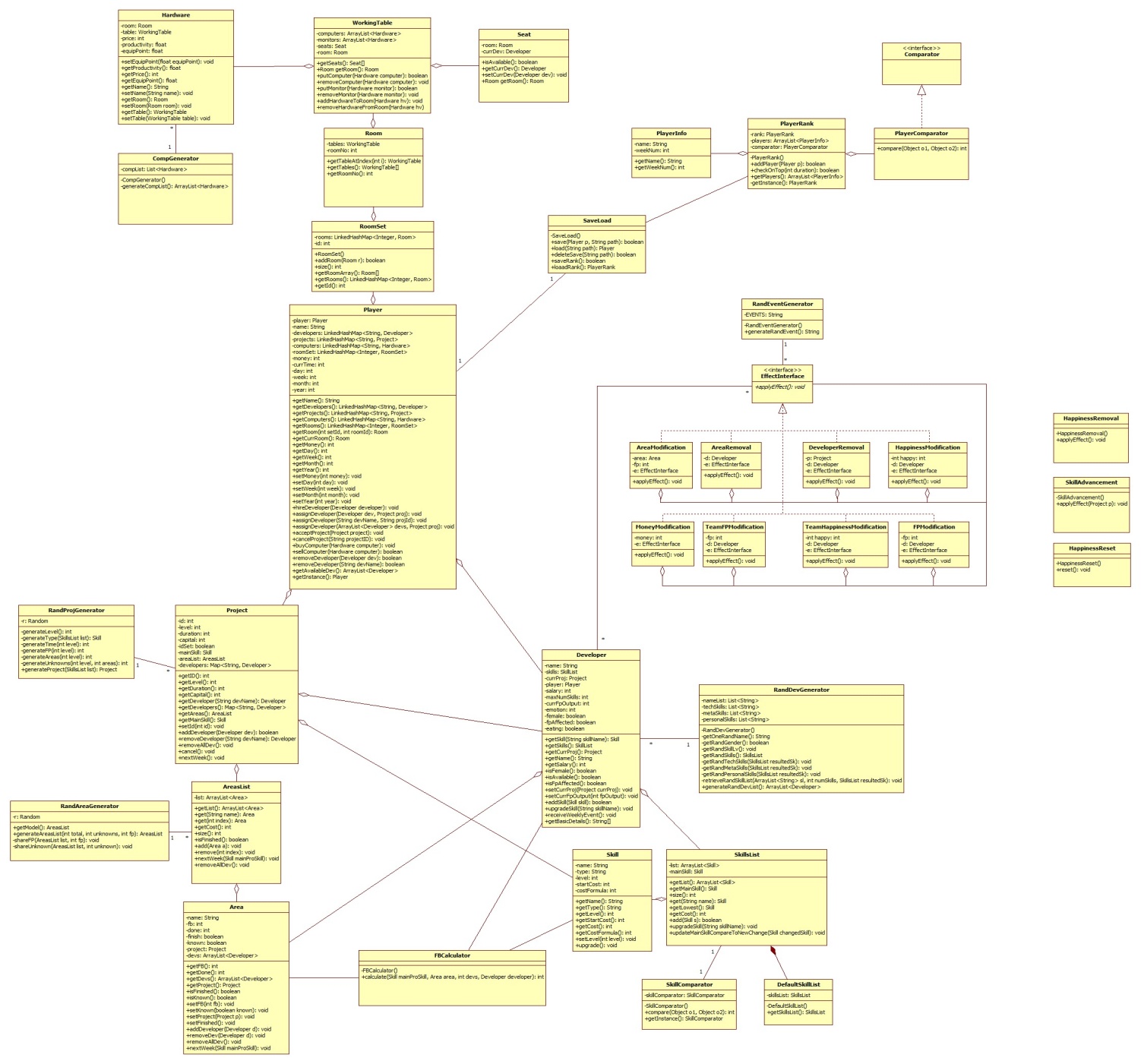
DevFortress is a management turn-base game, in the game; the players are responsible for managing developers and earn profit for the company.

The game is designed using MVC in order to separate the Model (game core/ business layer), the View (game UI/ presentation layer) and the Controller (application layer). As being demonstrated in the two demos in class, the design included some design patterns such as, Singleton, Factory Method patterns.

On top of that, the source code is grouped and presented in the isolated and meaningful packages to make it more readable and maintainable.

The Mock UI will be shown in the video clip that talk about how we build the game, difficulties and other things we encounter when we were making the game DevFortress so it will not be mention here.

The following is the class diagram of the game:



# Difficulties encountered and overcome

In this section, there were team’s difficulties and individuals’ problems that the whole team had been facing and trying to overcome.

Followings are the individuals’ hardship founded during the development process:

**Hung**:

## Problem 1

One of my main task is to implement the GUI and animation of the game, and I have problems in designing them. The reason is that my creativity skill is low such that it is difficult for me to design the UI.

### Solution

To solve this problem, an appropriate member in our team is assigned the task which is to design the UI and animation. After finishing a design, he sends it to me for implementation.

## Problem 2

Another problem that I encountered is the contradict between me and the member who designs the UI and animation. The problem occurs when the design is difficult for me to implement. In that case, I often ask the designer to redesign the GUI. As a result, more time is taken for redesigning.

### Solution

To reduce the probability of redesigning, the designer and I often work together so we can immediately detect and change the design. Also, during the redesign time, I try to debug so that the time is not wasted.

**Vinh**:

* The first and most important problem I had was to design the game, I wanted the game to be modern, sleek and clean. It must be easy to use, simple design, and appears attractive to the players, just like the Windows 8’s modern GUI apps. However, it was not easy for me to come up with the design of the game, since I have never designed for a game before.
* So I overcome the difficulty by making a lot of mockups, and to find the right people to give me ideas about the design as well as their opinion about the mockups I did, since I am not truly a designer to be fully confident about the game design. After observing ideas, opinions and adjust the design, I finally came up with the most suitable design for the game.
* Perhaps the one and biggest problem I have encountered during the process is to work with other people. Back then when I was doing Software Engineering Process & Tools, I was the one and only who designed and coded for the application’s GUI, so I tended to code things the way I wanted, and I needed to read only my code, everything was so clear and consistent in terms of coding style and ease of maintenance. However, when I started doing this project, one of my team member and I were allocated for doing the GUI of the game together, I could no longer do things on my own. Everything got worse when I only started working on the code after the first two weeks of designing the GUI on Photoshop. I ended up reading, analyzing codes that were not mine, trying to understand the concepts, the coding style that my partner put into the game. One of the reason was because his programming skills is far more advanced than mine, as he is known for having involved in many projects with many other people, on many programming languages. Another reason was that he used the graphic context g to form the GUI instead of using JPanel like I always do. Since the game involved a great deal of animation, and we had to make sure that whatever running under the hood will not affect the performance of the animation, so using Graphics g appeared to be the best choice over JPanel, though it means that we will have to deal with one of the Java’s nightmares: the null layout.
* There was a lot of times I thought I would never be able to understand my partner’s codes. My patience ran out that I thought I would never be able to help. But fortunately, after weeks analyzing his codes and learning new concepts, I was able to really get involved with the project. I could then easily manipulate the Graphics gthe way I wanted, and eventually I have come to create the GUI inspired by the Windows 8’s Modern UI that I wanted, thought it was not perfect as in my imagination. I have to admit, that without my partner I probably would not come this far.

**Tuan:**

Problem

At first, each time happiness of a certain developer is set by a method ***setEmotion(int emotion)*** in ***Developer*** class, the game engine will check the degree of that happiness. If the developer is unhappy, there will be a small change (about 30%) that he will leave the company. This solution seems to work; however, after several test play, it is found out that there is a bug happened when removing the unhappy developer.

Solution

Because the bug is considerably rare, therefore, in order to test that bug, it is decided that several changes will be applied to the game. Firstly, after each week, every developer will have his happy set to unhappy. Secondly, the change of leaving is increased to 100% in order to considerhow removing developer could generate bugs.

At the first test, after the first week, the bug occurs and its name is ***ConcurrentModificationException***. Therefore, it seems that in general, when traversing through the collection of developers, if the happiness of a certain one is set to unhappy and the change that he will leave company occur; then removing him will result in breaking the loop and result in ***Concurrent Modification Exception***.

In order to solve that problem, it is decided to create another class called ***HappinessRemoval***. This class will traverse through the list of developers and check the leaving chance of every unhappy developer. Provided that a developer is indicated to leave, his name will be stored in an array. After all developers are checked, those whose names are stored in the array will be removed. As a result, the bug of***Concurrent Modification Exception*** has been eliminated.

**Dinh:**

Problems: at first, after Tuan has finished the dwarf model, I started writing test. However, because it is not the final model so every time he changed something, I have to rewrite the test. This is going against the idea of writing test in the first place, test are written first in order to test the model but I was doing the opposite, writing test so that the model can pass.

Solution: after a while, I recognize the problem and so I decide to use the Mockito to write test cases. This really helps me much, many simple tests are not written again and again any more. However, this does not solve everything. With higher components that call many other components and operate that make change in the called components; I cannot test to see how the called component changed. If the model used the Spring framework I can do the test even easier. If I know this I would have made Tuan to write the model using Spring in the first place.

**Team problem:**

The team’s problems are about the conflicts in design, the debates while discussing the features chosen to be demonstrated. Fortunately, because the each member in the team behaved in professional ways so that the problems were soon solved by discussing peacefully and understanding others in an objective, profestional way.

At first, we were using Dropbox as the source control, however, later, we changed to Github. In the beginning, it was new, but because Hung has some experience using it, we manage to adapt to is very quickly. However, in a beautiful day, someone mess up Git and many code is conflicted. We were panicked at that time, but Vinh and Dinh was able to fix it in the day and recover almost everything to normal although some of the code Tuan wrote has been lost.

In the first Sprint, when we synchronize everything together, the GUI, the model and controller, there were many unexpected bugs occurred. We have to fix many things from the GUI to the model and we decided to have all the team members join in the process. We have 4 people so we divided into 2 pairs in order to attempt pair programming. It was very efficient because when one ask another about how the logic of the code works, the other one had to explain things and in the explanation process, he himself may recognize the bug. It took a while to resolve all the bugs and our plan was running slow because of this, but we managed to finish almost all of the functions in the 1st Sprint Backlog.

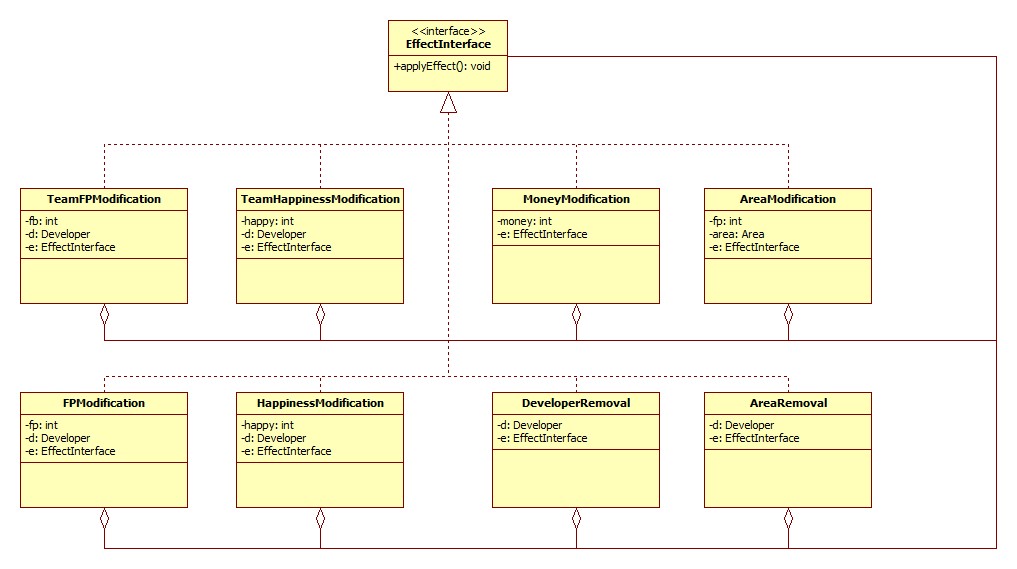
Another problem is that Hung does not have many comments in the GUI and controller. Every time the others need to do something involve with GUI or controller, we had to ask Hung to help, which really time consuming. Later, we were going to have Hung make a Façade but in the end, that was not successful because he has many other functions need to complete. We decide to have him explain the important methods that are often used and try to remember them. When we have difficulties, we meet online and resolve the problems.

# Applicability of the design patterns

1. Factory: used to create Sound Effects, View,… the reason we use this because it hide how we create the stuffs, we don’t have to deal with each of them but through their factories. It makes adding new stuff easier.
2. Singleton: because this game is single thread so we think it doesn’t have any problems when apply this. We need this for calling player, and front controller object. Using singleton will help us easy to invoke them everywhere in the system without reference everywhere needed.



1. Decorator: Using decorator allow us to code the Effects really easily. With this design pattern, we were able to add more effects in the game assign the original ones. It is also very good for future changes.



# Applicability of Scrum project management

* First of all, we read the specification carefully and asked for more from the customer (lecturer). After specified all the requirements, we had a Sprint planning meeting. In the meeting, we talked about which features are important and came up with a product backlog, then we decided which we should implements in the 1st and 2nd Sprints. After that, we assigned tasks to the members. Although everyone do a little of everything, the main model programmer is Tuan, GUI programmer is Hung, Designer & Art Work is Vinh and Dinh do the testing, debugging and manage the Scrum process.
* Every weekday, we tried to have a Scrum daily meeting so that everyone knows how the project is going and how should we change roles is there are anything that requires more efforts.
* Based on the Product Backlog, we create 2 Sprint Backlogs and Burn down charts. Based on the Sprint Backlog and Burn down chart, we were able to see if we were in the right direction. When we are going too slow or too fast, when we focus on the wrong priority of the functions, we immediately adjust to fit with the situation.
* After every Sprint, we have a Customer meeting with the lecturer in order to decide if the functions are “done”.

# Changes made to the specification and justification for the changes

1. We decide not to implements the Difficulties because this kind of game is suppose to be really hard and the player is going to lose I a couple of turns. It also because that we do not have enough time to implements all the features.
2. We also do not implements the manually feed developers, each turn, the developers are feed automatically and the we have to pay for the food of course. We made it this way mainly because we do not have enough time and we also don’t have enough space in the UI in order to implement this function.
3. However, we decide to implements one more very cool function that is add room function. Because the game design is based on the inspiration of windows 8, the metro UI so we decide to impress the customer with the way we can drag the room down and change the working room.

# Additional constraints not listed in the specification

None

# Technology stack and tools chosen and justification

1. Mockito: used to make the temporary and mock object for testing so that the testing can be done more cohesive without changing the other parts of the source code.
2. JUnit: used to implements the testing process for the business logic of the game.
3. Trident: used to implements the time line to deal with animation.
4. JMF: used to implements sound effects.