CSE 351 Design Project

Term project, Statement of Work

20190808042 - Yener Karaca

The Corrupted

An amateur indie game group called Victorious Games decided to make a turn-based story-telling game called “The Corrupted”. First, they came up with an idea of how the game is going to be like. So, they started documenting:

Game Summary (Story line)

This game will take place in the Gatholla Kingdom, where 3 tribes exist: humans as knights, orcs as warriors and wizards as sorcerers.

Humans live in the glorious city of Castlemouth which stands in the middle of Gatholla.

Wizards live in Winddale, a small city above the clouds, highest point of the mountain Smiheim which stands in the northside of Gatholla.

Orcs live in Mudwood, a mysterious town deep inside the Forest Arnguay which stands in the southside of Gatholla.

Each tribe used to live in peace, until corruption started happening. It caused people to go hostile towards others, it was infectious. Our player is destined to find the person who’s responsible for this chaos and bring back peace… Or is he?

Notes from one of the developers before starting

- We must prompt the user to select a class.
- We should write something in our game so that once a user selected a class, we can decorate it with its class stats. I've thought about having and "if-else" structure but that would get too messy for us. Discuss this with Yener. (Yener was developer's close friend who recently took a Design Patterns course)
- Instead of typing in all the lore inside of the main class, we should create a class to keep all the strings.
- Fights must be turn based, and attack chances must be random for fair play.
- Each time player must level up so that it doesn't feel too repetitive.
- We should have a plot twist at the end so that it doesn't end boring.
- ...

With that in hand, they started working on it…
Developer discussed about how he could implement this feature in their program with Yener and he said:

- Ah, you can easily achieve this goal using a Decorator Pattern design. You should have something like this:

```
abstract class Player {
//code goes here
}
class DifferentClass extends Player {
//Modify the code accordingly
}

Class AnotherDifferentClass extends Player {
//Modify the code accordingly
}
```

That way you can change the variable values at compiling time! said Yener. Developer was glad to discuss this problem with him.

After a couple of days later, they came up with a working program. They tested the game and fixed bugs. After finishing up their work. Developer wanted to share his finished code with Yener for final thoughts from him. He examined the program for a bit, and said:

Why didn't you use the Singleton Pattern in your program? It could have been the perfect opportunity since it's a single player game. You are going to have only one Player instance either way. Developer replied, "Well, since our last discussion I went online and searched for useful design patterns I can use in my program and saw Singleton pattern. I tried implementing it, but it didn't work the way I thought it would, first, Singleton constructor takes no parameters, but in my code, I need parameters to pass the values to the player class. Singleton just seemed not viable for me. Oh, and don't forget that my player class is abstract since I used Decorator Pattern. Don't think I can create an abstract Singleton class that can take parameters." Yener said "fair enough! It really is pointless for this situation. Well, how about the way you handled your game states? Shouldn't it be a cleaner code for you if you've used State pattern?" Developer replied, "I actually didn't know such pattern existed, let me read the documents really quick." As he reads the documents and looks at the code examples and says "I don't think it could have done much difference for me, you see; my program uses those states as integer numbers to let program know the current state of the fight. If he should continue or not, or how to continue depending on if its resting time or action time. It is nothing more than that, after creating a State Structure, I'm still going to have to use the 'if-else' cases to let program know how to act. So, I don't think it's practical use for me." Yener, seemed convinced, and said "Well, I guess you can't force a Design Pattern in your code just for the sake of using Design Patterns".

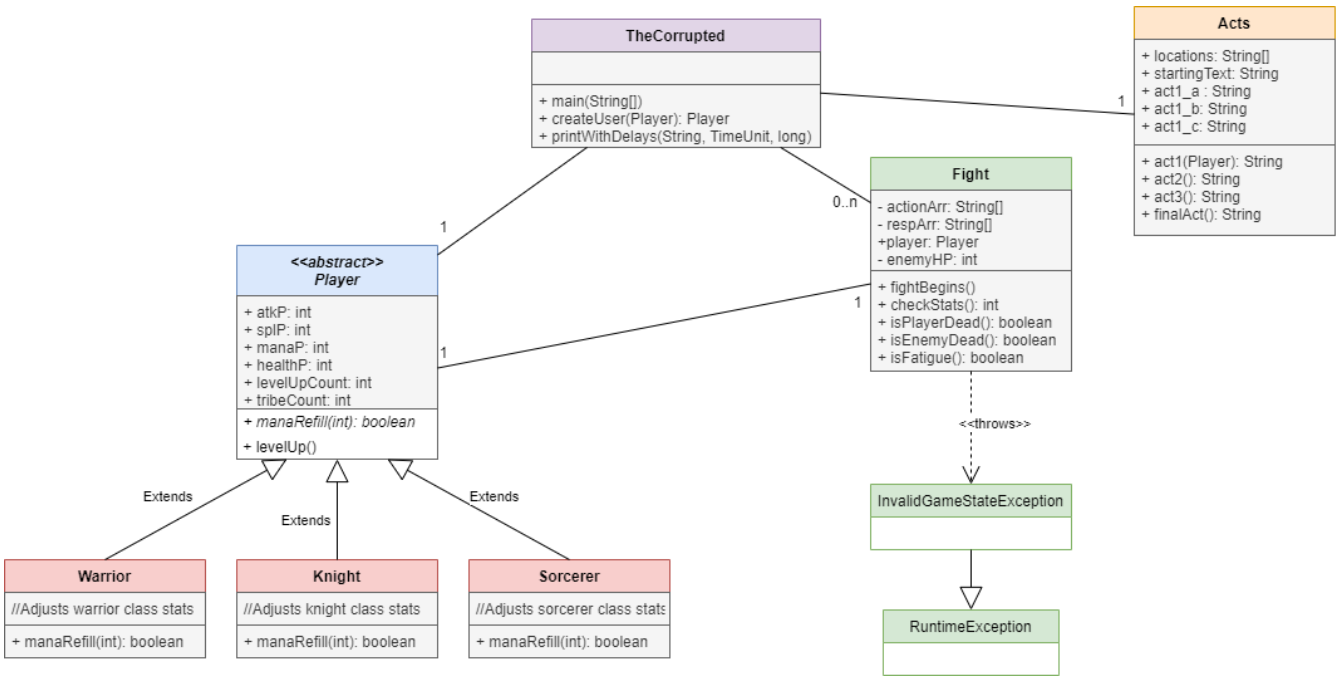
After all this Design Patterns discussions this developer had with Yener, he and his team published their game on amateur developer websites, and they mostly received positive feedback. A couple professional game companies reached out to some of the developers involved in this project and offered an internship continued with a job. And thus, their journey to the life of a professional game developer started. As for Yener though, he's still busy with school.

*My video presentation YouTube link: https://youtu.be/3A780OwgqU0

Yener Karaca

20190808042

He also sent Yener his UML Diagram as a memory.



He thought it was pretty clear and very well detailed, a solid 9.2 out of 10.,

The end ~