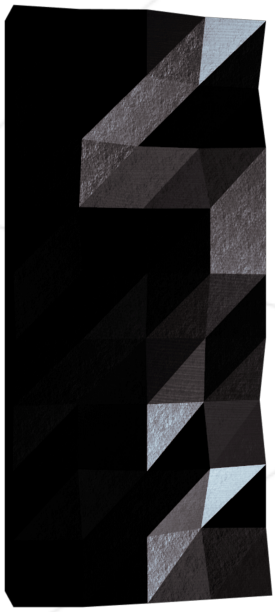

ft_boardgame

Reinforcing Object Oriented Programming

Michael Lu mlu@student.42.fr

Summary: This project is about creating a board game utilizing basic object oriented programming concepts and class interactions.

Contents

Chapter I

Foreword

Have you ever wondered what to do if your game fails?

Here's some more interesting information my favorite failed video games. Let's talk about Ragnarok Online, one of the most popular 2.5d MMORPGs. Yes, this is the same development team called Gravity whose core members moved to IMC to make Tree of Saviors (as you know if you read the foreword in the previous project). Why would we use Ragnarok Online as an example for a failed game if the game was so successful?

The main failure from Ragnarok is its sequel MMORPG, called Ragnarok Online 2 (RO2 for short). It was a very difficult game compared to RO1 (Ragnarok Online, the successful version) since it went from a 2.5d MMORPG to a full- fledged 3d MMORPG.

First, because the first game had been so successful the level of hype was incredible. Everyone was expecting a MMO to blow every other competitor out of the water. Second, it still had the same core team as the previous RO1 so people were hoping for what made RO1 special would also be present in RO2. Lastly, the team had promised a lot of unique mechanics that no other MMORPG had implemented. This was all that the RO1 MMORPG players had dreamt of: a successful sequel that held the charm of the first game and introduced new mechanics that no other game could match.

As you might guess they did succeed in implementing these new mechanics as promised. A combat system never seen in any other MMORPG, a unique world and leveling progession. Weapon and armor system was completely new, including how quest and rewards were handed out. So if this was the case, why did it fail so hard? The overly ambitious developers were driven to work so hard on creating a new experiences that they forgot

the most important factor of Ragnarok Online 2: Gates of the New World - that it was supposed to build on the legacy of RO1. RO2 ended up nothing like the first iteration of the Ragnarok game, which left many fans disappointed in the world, the game and everything about it. The backlash was so harsh that the company was forced to make drastic changes. This including scrapping the ENTIRE GAME, replacing MOST of the developers and creating a new Ragnarok Online 2 called Legend of the Second.

Needless to say, RO2: Legend of the Second is still alive today but it's just a generic MMORPG clone that focused more on bringing in the Ragnarok Online 1 theme. The game is nowhere as successful as RO1 and most of the developers went to work on Tree of Saviors which is also nowhere as successful as RO1. In this case, the developers replaced their team and even remade the game to try and appease fans. It's quite a difficult time for Ragnarok Online fans like myself because the odds of these talented developers ever making a true Ragnarok MMORPG sequel are unfortunately very slim after their first attempt experience.

Hopefully you learned a bit more about MMORPG games and the business of building them. There is much more information about this drama out there, and being a Ragnarok Online 1 fan, I followed everything on the RO2 development very closely. Needless to say the only thing that could fill that empty fanboy void is Tree of Saviors, or playing Ragnarok Online 1 again. Sadly, RO2 is not worth it and does not live up to anyone's expectations. (I personally enjoyed the first Ragnarok Online 2: Gates of the New World MUCH MORE than the remade version).

Another interesting fact, if you want to see another example of what happens when a game fails but the team succeeded in fixing it, check out the history behind Final Fantasy 14 MMORPG.

Chapter II

Introduction

The goal of this project is to create a boardgame with some simple mechanics for you and your friends to play. One of the main things you will be doing is continuing to improve on your object oriented skills, and working on reinforcing those concepts as well as learning about new concepts to help you make this project even better. We will explore a bit more about entities/containers, modules and extending class methods which Python and Ruby both utilize in their object oriented programming.

If you are using Python or another language make sure to research into language equivalent concepts on your own. This project can be completed in any object oriented language, however the tutorial and video guides will be in Ruby.

So let's talk about how to make a boardgame. First, you need to create the spaces/tiles that make up the board. Second, you need to setup how you will display your board. Third, you need to work on a win condition. Afterwards, you will have to set up the players and any unique mechanics. In this project, you will be making cards/event cards for players to utilize as they progress around the board trying to win. It won't be as easy as the previous project, but you can do this!

Chapter III

Goals

The goal of ft_boardgame is to help you reinforce your object oriented programming skills. By the end of this project you should know how to:

- Extend class methods
- Utilize containers/entities
- Utilize modules
- Create multiple interactions between objects and classes
- Be even more awesome extra.

You will be exploring a fundamental topic, object oriented programming, so take advantage of all available resources including the video, fellow students in the lab and Google. There are many tutorials on classes and inheritance.

Chapter IV

General instructions

- This project will only be corrected by actual human beings. You are therefore free to organize and name your files as you wish, although you must respect some requirements listed below
- Your must have a parent class and a child class (they do not need to be named parent/child, but it should be easy to see which is the childand which is the parent)
- A container/entity class is required for your project

`It will help you a lot to practice writing your entity/container first since they help you store multiple instances of object/classes. Create a container, store your classes inside and practice creating the behaviors inside.`

- All child classes must be unique from one another
- You must have a menu/selection screen with proper loop handling
- You must allow the ability for people to create a player for the board game
- You can decide what variables you need for your board game
- You must display your boardgame in a GUI
- Your game must support at least two players
- Ask your peers, mentor, slack or anywhere else if you need any help, and enjoy customizing the design of the game according to your own style.

Chapter V

Mandatory part

The image example on this PDF are only examples, not what you need to replicate. You can design your output, your menu, etc. however you would like to design it. If you need help with design please visit the tutorial video guides as there is a demo there for you to see how the project should behave

- The goal of this project is to create a simple boardgame project
- At the beginning of the program the users should be able to create a character or see a menu to do so
- One container is required to hold your multiple classes
- The container must be able to store your objects/classes, and delete them as well as access them.
- Your boardgame should be made up of tiles/spaces that are child classes of a parent class. Your tiles/spaces could represent anything from doing nothing, to giving players some kind of event or item (ex. move forward X number of spaces, or give a player a card that allows them to steal points)
- The players must be given an option to roll their dice, or play an item/card from their hand.
- You can choose to end the game immediately after someone wins, or have players keep playing.
- You must display the boardgame in some sort of GUI. This means you can decide on what graphic library you want to use for this. Below is an example of a simple terminal GUI output

- You can design the player class freely as you'd like.
- You must have the player class equip/hold/contain the items/cards/etc that they can earn through the game.
- When a player uses an item/card/etc you must delete that item/card/etc instance from your player or container.
- To really emphasize class interaction you will be required to do the following:
  - A minimum of three items/cards/etc that players can earn from playing
  - All cards/items/etc are unique child classes that inherits from the same parent class
  - A minimum of three unique tiles/spaces that trigger some sort of event
  - All tiles/spaces are unique child classes that inherit from the same parent class
  - A container that holds all instances of your classes

Below is a sketch of what you should considering trying out if you are stuck and unsure of how to implement your classes in your container. You do not need to follow this outline, it is just here as a push in the right direction.

- You should try to make the user experience as enjoyable as possible. For this reason the GUI should be easy to understand.

- You must handle any kind of user error to the best of your ability.

Chapter VI

Bonus part

This is a creative project so you have many opportunities to earn bonus points. Make this games yours, be unique and come up with any crazy ideas you have! Below are some bonus suggestions, you can add any features that you think are cool too. However, one part of the bonus will be reserved for use of modules and class method extensions.

- Sound effects
- More event/items
- Cool effects on the GUI
- Additional mechanics, or randomness to the game
- Any other cool features you can come up with to enhance your boardgame!

Chapter VII

Turn-in and peer-evaluation

Turn your work in using your `GiT` repository, as usual. Only work present on your repository will be graded in defense.

Good luck and remember to have fun!