



ft_arena

Introductory Object Oriented Programming

Michael Lu mlu@student.42.fr

Summary: This project is about creating fighters who will battle it out in the arena to teach object oriented programming concepts.

Contents

I	Foreword	2
II	Introduction	3
III	Goals	4
IV	General instructions	5
V	Mandatory part	6
VI	Bonus part	8
VII	Turn-in and peer-evaluation	9

Chapter I

Foreword

Have you ever wondered how to write and structure your own game?

One game company infamous for having a very talented team of programmers but failure to execute is IMC Games, producers of the 2016 MMORPG Tree of Saviors. This team contained developers from Gravity Co which was famous for Ragnarok Online, a old-school 2.5d MMORPG that has an emphasis on social play. So what went wrong with Tree of Saviors?

IMC Games had previewed a lot of crazy cool stuff like having four main classes that can branch into multitude of classes. Boasting eight new classes (two for each of the four main classes) at each rank (out of nine ranks), you had 76 different classes! Imagine being able to build any kind of class you wanted with those combinations! Sadly, this promise turned out to be only but a dream.

In fact the IMC Games team succeeded in implementing this cool idea, but they had a lot of optimization and balancing issues. There were constant server problems, latency and imbalanced classes that slowly dwindled the player community. By the time the developers had fixed the majority of the optimization issues, balancing issues, and server issues, the damage had already been done. The game's hype was gone and many gamers had moved onto other MMORPGs. Was it their development process that killed it? Maybe it was their communication practices? Who knows.

As someone who still plays Tree of Savior I hope this tidbit gives you some interesting insight.



Chapter II

Introduction

The goal of this project is to create a fun arena simulator where your own characters who will fight for the death! Let's be real, it's exciting to make a game and learn the skills to do so, and at the same time it's important to start with the basics. The basics, i.e.:

- **Classes** are templates for **Objects**
- **Objects** are instances of **Classes**
- **Objects** have **variables** and **methods**
- **Classes** can **inherit** from other classes

Ruby and modern Python are both inherently object-oriented programming languages. They have built-in syntax for creating objects and classes, and in fact, every data type you use - from arrays/lists to integers to strings - objects themselves.



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

So how do you even start making an arena simulator? Simple. First, you need to provide information about your fighters. Second, you need to make every fighter unique. It'd be boring if they all did the same thing. Lastly, you need to make them fight. And guess what? All of this can be done using object oriented programming. After you learn classes you'll never go back. The skills you learn from this can be applied to any of your other projects, thanks to the usefulness of classes. So let's start with the basics first.

Chapter III

Goals

The goal of ft_arena is to introduce you into basic object oriented programming. By the end of this project you should know how to:

- Design classes (parent, children, etc)
- Utilize inheritance
- Utilize multiple classes
- Create class interactions
- Be awesome

You will be exploring a fundamental topic of object oriented programming so take advantage of all the resources including the video, your neighbor 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.
- You 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 child and which is the parent).
- A container class is optional, but highly recommended.



It will help you a lot to practice writing your parent and child class first. After that you can focus on writing your container or main and storing your classes. I recommend a container with an array to store your classes. You can see this demo in the tutorial video. There are other ways to store your classes so feel free to choose what you feel most comfortable with.

- You must create at least three unique child classes all inheriting from same parent.
- You must have a menu/selection screen with proper loop handling
- You must allow the ability for players to pick which fighter they want to use in the arena
- You must use at least two variables for the parent class: health and attack. You can shorten these names or rename them, but you must have some sort of health and attack variables
- Ask your peers, mentor, slack or anywhere else if you need help, and feel free to get creative with this project.

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 program that will take two fighters and have them fight each other.
- All fights will be one on one, so do not worry about having more than two fighters fighting each other at the same time.
- At the beginning of the program the users should have a menu that allows them to create a fighter, start a fight in the arena, and to see all the created fighters available (any extra stuff is great).

```
Menu (Please enter the numeric value to access)
1. Player Types
2. Add Players
3. Play Arena
4. View Players
> 
```

- Only health and attack is required variable for the fighters, feel free to add any other variables if needed.
- When the user chooses to have fighters fight in the arena they must be able to select two unique fighters. Both fighters cannot be the same fighter.
- You can choose to reset the fighter after the fight, or leave them weakened/incapacitated from their previous fight. It is not required to have all the fighters fully heal for the next fight.
- You must display all the fights with printout to the terminal.

```
BATTLE WILL COMMENCE BETWEEN SELKIE AND PILO
3...
2...
1...
0...
Selkie attacks the enemy!
Pilo took 5 amount of damage. Current health: 495
Pilo strikes twice, healing for 18 points of damage. Current health: 513
Selkie took 14 amount of damage. Current health: 436
```

- It should be clear in the terminal who is attacking and who is taking damage.
- When a fighter's health reaches zero the fight is over and the victor must be declared in the terminal.
- When the fight is over you must have the user redirected back to the main menu where they can create more fighters or start another fight.



How you structure your while loops, and if/else, case switch, hash table, etc can make or break your menu. Consider designing something simple so that it is easy to repeatedly call the menu and to add stuff to the menu.

- You should try to make the user experience as enjoyable as possible. For this reason the user should be able to see all created fighters so they can pick from them for the arena fights.
- Because you are simulating a simple arena fight make sure you delay your terminal output. Having the entire battle post immediately to the terminal instantly is not okay.
- You must handle any kind of user error to the best of your ability. The most vital one is to make sure there is no error in creating a character, and there is at least two viable fighters to put into the arena together.

Chapter VI

Bonus part

Remember this is a simple self automated simulator, kind of like a game. For this reason there are many things you can do for bonus. Feel free to add whatever you'd like to spice up your program as long as you meet all the requirements above. Some things you could try out for bonuses are listed below:

- Sound effects
- Additional classes
- Cool text/color effects in terminal
- Any visualizer
- An amazing game container that holds all their classes
- Unique skills, attributes or stats
- Additional mechanics, or randomness to the fight
- Any other cool features you can come up with to enhance your simulator!

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!