#TODO Finish the part.

Part 3: Prise en main.

> @Romain It might be interesting to do that by groups What to you think ? Again you can copy all files from part2. But it's starting to be a mess here, So we are gonna organize our code.

Let's create a package `fighter` and put `warrior.py defender.py ...` Let's create a package `battle` and put `fight.py battle.py` in it.

Weapons

In a package `weapon` Create an Abstract Class Weapon(health attack defense vampirism heal_power) his constructor should be an abstractmethod which will allow equipping your soldiers with subclass of weapon. Every Weapon object will have the parameters that will show how the soldier's characteristics change while he uses this weapon. Assume that if the soldier doesn't have some a characteristic (for example, defense or vampirism), but the weapon has a bonus to those, these parameters won't be added to the soldier.

The parameters list: health - add this modifier to the current health and the maximum health of the soldier; attack - add this modifier to the soldier's attack ; defense - add this modifier to the soldier's defense ; vampirism - increase the soldier's vampirism to this number (in %. So vampirism = 20 means +20%); heal_power - increase the amount of health which the healer restores for allied units.

All parameters can be positive or negative, so when a negative modificator is being added to the soldier's stats, they are decreased. But none of them can be less than 0.

Let's look at this example: vampire (basic parameters: health = 40, attack = 4, vampirism = 50%) equips the FakeWeapon(20, 5, 2, -60, -1). The vampire has the health and attack characteristics, so they will change - health will increase to 60 (40 + 20), attack will become 9 (4 + 5). The vampire doesn't have defense or heal_power, so these weapon modificators will be ignored. The weapon's vampirism modificator -60% will work as well. The standard vampirism value is only 50%, so we'll end up with -10%. But, as we mentionned before, parameters can't be less than 0, so the vampirism after all modifiers will be just 0%.

Also you should create a few standard weapons classes, which should be the subclasses of the Weapon. Here's the list: Sword - health +5, attack +2, vampirism -10% Shield - health +20, attack -1, defense +2, vampirism -20% GreatAxe - health -15, attack +5, defense -2, vampirism +10% Katana - health -20, attack +6, defense -5, vampirism +40% MagicWand - health +30, attack +3, heal_power +3

And finally, you should add an `equipWeapon(weapon)` method to all of the soldiers classes. It should equip the chosen soldier with the given weapon. exemple: `my_army[0].equipWeapon(Sword())` - equip the first soldier with the sword. If your soldier has already a weapon equiped it should change his weapon and return the old one.

Notes: While healing (both vampirism and health restored by the healer), total health can't be greater than the maximum amount of health for this unit (taking into account all of the weapons' modifiers). If the healing from vampirism is float (for example 3.6, 1.1, 2.945), round it down in any case. So 3.6 = 3, 1.1 = 1, 2.945 = 2.

Preparing a Game

Races

Let's create 3 packages `dwarf elve human`.

You are the developer of the new strategy game and you need to add the soldier creation process to it. All `fighter` class are available But there is some new classes to create, specialized by races. Each army has unique names unit type.

Specialized Units are defined in stats.md - Magician - CavalryMaster - Sword-Master

TrainingCamp

Each Race has his `TrainingCamp` Class which let him recruit warrior and Train them under some condition.

A Training camp has limited time to train warrior, and has a limited number of warrior that can be recruted. `TrainingCamp(warriors_available, years_available, month_train_time)`

a getter `time_available` which return training time available in month. recruit_warrior() - train_vampire(warrior) - create an instance of the Vampire ... train_elite(trained_unit, specialisations)

This training camp will be given to your constructor Army. `Army(training_camp)`.

You can remove the `addUnits` methods

Your Army should have now 2 new methods. `recruitWarriors(num)` which recruits from the training camp num warrior and add them to the army. `trainUnit(type)` which train one unit to the type if possible. do nothing.

The `move_unit` method is now let to your appreciation.

> @Romain au debut je voulais fait une Abstract Factory avec `army`
> Finalement le besoins sans est pas fait ressentir, Mais au travers de

ca je pense que les notions de ArrayFunc et DictFunc seront mieux et c'est un debut de machine a etat aussi. Qu'en pense tu ? @lumy : je sais pas si tout le monde sera au point, ton objectif est bon je conseille de te laisser le temps de voir s'ils comprennent des design pattern basiques et introduire AbstractFactory si tu les sens prêts (quitte à le proposer de manière ciblée à certains)