Mini Project Vikings

DSMLFT
Eric, Isis, Marius
Jan 16, 2026

Mini Project: Vikings vs Saxon War Game

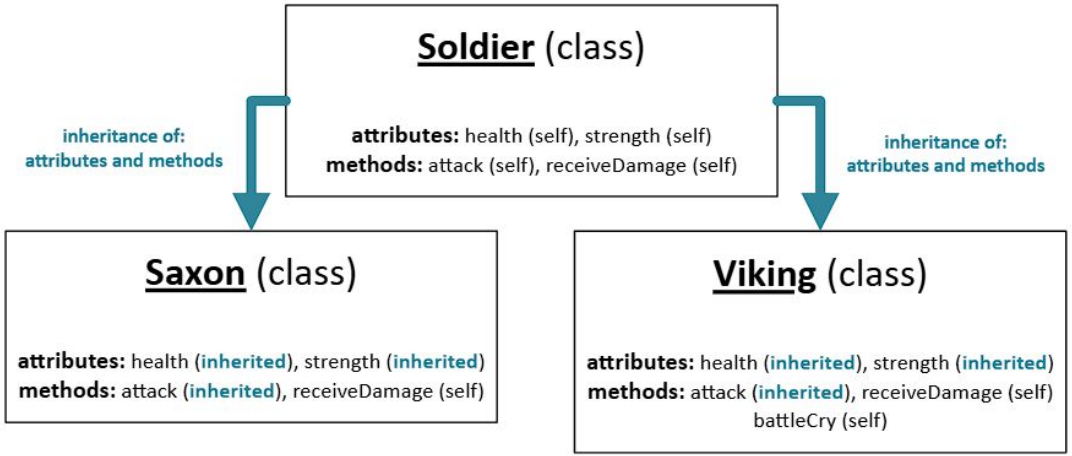
Challenge: Apply lessons learned about Object Oriented Programming, including concepts of classes and inheritance.

1. Create the Class Soldier and its arguments “health” and “strength”
2. Create the Classes Viking and Saxon
    a. these classes inherit “health” and “strength” attributes from Soldier
    b. they can “receive damage” via method, which impacts their “health”
        i. Viking soldiers also have a “name” and a “battle cry”
3. Create the Class War, which consists of
    a. the Viking and Saxon Armies
    b. Viking and Saxon Attacks: adding soldiers via appending
    c. the war status, that shows the outcome of the war, using an if/else statement

Goal: A file with all classes correctly defined that can be used to create a war game

The use of Object Oriented Programming (OOP)

- defining classes -> changing it will change all instances and all classes that contain inheritance
  - attributes and methods -> using the function super().__init__(attribute1, attribute2)
    - defined once -> can be used multiple times without copying the whole code
  - reusable and easy to maintain



Errors

Error that took the most time to solve:

```
======================================================================
FAIL: testSaxonHealth (__main__.TestWar2)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/isishassan/Documents/Ironhack/Week1/Projects/mini-project-vikings-en/4-testsWar.py", line 75, in testSaxonHealth
    self.assertEqual(self.saxon.health, oldHealt - self.viking.strength)
AssertionError: 60 != -90
```

Cause: Vikings were attacking themselves
Solution: Review variables, write down solution in pseudocode before writing actual code

Most Annoying Error

```
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/isishassan/Documents/Ironhack/Week1/Projects/mini-project-vikings-en/4-testsWar.py", line 39, in setUp
    cls.war.addSaxon(cls.saxon)
  File "/Users/isishassan/Documents/Ironhack/Week1/Projects/mini-project-vikings-en/vikingsClasses.py", line 70, in addSaxon
    self.saxonArmy.append(saxon)
AttributeError: 'War' object has no attribute 'saxonArmy'
```

Cause: Inconsistent capitalisation between **S**axon**A**rmy and **s**axon**A**rmy led to the missing attribute in the class “War”
Solution: Consistency

Lessons Learnt:

- You don’t always need return()/print() in your methods ***

  Remembering that methods are essentially functions within a class/object so return()/print() is necessary if you need an output. But they can still alter existing objects without return. My attack methods had unnecessary returns as all I require them to do is alter existing objects.

  However print() was useful as it allowed the attack methods to be verbose.

- Don’t refer to the class name when defining it

  That when defining methods you are still in the process of defining the class, so self.method() makes far more sense than naming the class explicitly: class.method(self)

  War.showStatus(self) :(

  self.showStatus() :)

Lessons Learnt continued:

- Naming Conventions
  - Classes are capitalised in **CamelCase**
  - Methods should be in **snake_case**
    - The inconsistency of the naming scheme in “vikingsclasses.py” caused numerous errors. The methods appear to be a mix of lowercase and camel case.
      - self.exampleMethod()
- The Benefits of Good Comments
    - Clear and consistently formed comments make the process of reading and presenting code far less chaotic. Your team members and your future self will thank you.
    - Thank you Marius

Lessons Learnt continued yet again:

- “Do not always follow the first idea that comes to your mind”
  - Consider two or three approaches and then choose the best one
  - …or at least be willing to re-evaluate your approach

- *“If at first you don’t succeed: Try, try, try again”* - Maybe not the best advice
  - If you get stuck, consider trying to solve another part of the project and continue later
  - Sometimes you gain new insights and a new perspective on the problem by stepping away

Wargame Demo: Key Points ,Demo Output and Repo Link

- import from vikingsClasses and import random
- **war_1** object is created.
- User **input()**
- **addViking()** & **addSaxon()** methods on **war_1**
- Battle is conducted in either a while or for loop depending on user input
- Army health obtained from elements of **war_1.vikingArmy/saxonArmy** attributes of **War()** class
- Number of survivors obtained from respective **war_1.vikingArmy/saxonArmy** attributes
- New graveyard attribute (**War.valhalla**)
- Finally time to **print()** with an fstring

```
5 Saxon(s) versus 5 Viking(s) called: Jim,Mike,Rick,Rick's identical brother and Ted
Odin Owns You All!
A Saxon has died in combat
Jim has received 2 points of damage
Vikings and Saxons are still in the thick of battle. after 1 rounds!
Odin Owns You All!
A Saxon has died in combat
Jim has died in act of combat
Vikings and Saxons are still in the thick of battle. after 2 rounds!
Odin Owns You All!
A Saxon has died in combat
Ted has received 2 points of damage
Vikings and Saxons are still in the thick of battle. after 3 rounds!
Odin Owns You All!
A Saxon has died in combat
Ted has died in act of combat
Vikings and Saxons are still in the thick of battle. after 4 rounds!
Odin Owns You All!
A Saxon has died in combat
Vikings have won the war of the century! after 5 rounds!
Game Over,
0 Saxon(s) survive with 0 total health remaining,
while 3 Viking(s) survive with 12 total health remaining.
Never forget: Jim and Ted
```

https://github.com/Esalsac/mini-project-vikings-en


Q&A

Notes Eric

- Inheritance of attributes

  Used super().__init__(inherited_attribute_1,inherited_attribute_2,...) whilst defining the __init__ function for the Viking class so that it could inherit the health and strength attributes of the soldier class.

- Inheritance of methods

  Inheritance of the methods of the Soldier class to the Viking class was achieved during its definition: Class Viking(Soldier)

Notes Marius

- problems
  - using random.randrange instead of random.choice for selecting the attacker/defender
    - it’s more difficult afterwards:
      - random.randrange:

```
attacker_viking_nr = random.randrange(0,len(self.vikingArmy))
viking_strength = self.vikingArmy[attacker_viking_nr].strength
```

      - random.choice:

```
attacker_viking = random.choice(self.vikingArmy)
viking_strength = attacker_viking.strength
```

  - Trying to access the properties “health”/”strength” of a Saxon/Viking in a list of lists:
    - not possible -> it’s a list of instances from class “Viking”/”Saxon”
  - How to delete the Saxons/Vikings from the list before giving back the result?
    - storing the result of the method “.receiveDamage“
      - delete Saxon/Viking
        - giving back the stored result as the result of the method “vikingAttack”/”saxonAttack”

Notes Marius

- learnings
  - do not always follow the first idea that comes to your mind
    - Consider two or three approaches and then choose the best one
  - if you get stuck, try to solve another part of the project and continue later
    - sometimes you get a new perspective on the problem