Mini-project Vikings

Comments on implementation

Ekaterina Kuznetsova   Georg F.K. Höhn

Sporty Spice

9 May 2025

Overview

Section 1

vikingClasses.py

Implementation

- class Soldier(health, strength)
  - attack() – returning strength
  - receiveDamage(damage) – reducing health
- class Viking(name, health, strength) – inherited from Soldier
  - modified receiveDamage(damage) to print damage received, name and possible death
- class Saxon(health, strength) – inherited from Soldier
  - modified receiveDamage(damage) to print damage received and possible death
- class War()
  - addViking(Viking) and addSaxon(Saxon) to fill up lists
  - [viking|saxon]Attack() to simulate effects of attack actions
  - showStatus() to test whether any army has been depleted

Deviations from the task

- dealing with death
- utility of battle cries

Dealing with death

- test suite expects health values below < 0 in `receiveDamage(damage)` methods

Failing 4-testsWar.py

```
self.assertEqual(self.saxon.health,
     oldHealt - self.viking.strength)
AssertionError: 0 != -90
```

- we consider this unrealistic and treat 0 health as absolute minimum
  - `receiveDamage(damage)`: health set to 0 if damage>health
  - checking for health==0 in cleanup of deceased in `War.[saxon|viking]Attack()`

Utility of battle cries

Project requirements

- battle cry is purely symbolic (simple shout)
- no effective impact on battle
- not used by default

Our implementation

- probabilistic use in `War.vikingAttack()`
- use increases strength/damage dealt by fixed value (3)
- adds a further random element, strengthening `Viking` instances

Section 2

Simulation logic

Overall approach

Design goals

- interactive choice of size for each army
- random property allocation of health and strength of Soldier objects within pre-defined ranges
- non-deterministic attack order

Structure

1. importing necessary packages (random, vikingClasses)
2. instantiating variables
3. defining auxiliary function for army creation
4. main program
   - greeting
   - creation of War instance
   - input logic for army sizes, army creation
   - action simulation loop
   - final result output

Variables

- `namelist`: list of names for Viking instantiation
- `valuedict`: dictionary of dictionaries for the following values for Saxons and Vikings
  - `num`: number of soldiers to create, instantiated as `None`
  - `health`: tuple of range for random health values
  - `strength`: tuple of range for random strength values
- `groups`: auxiliary list of ["Saxon","Viking"] for simplication of army raising

Auxiliary function

`raise_army(num, group, war)`

Raises an army of type {group} with {num} members in {war}

- num: number of members of army to be raised
- group: the group the army belongs to
  - 'Saxon' for Saxon
  - 'Viking' for Viking
- war: a War object

- each soldier is created with random values based on the constraints defined in `valuedict`

Main loop

Input logic

- loop through `groups`
- request size of each army
- `while`-loop to ensure valid input of integer
- call `raise_army()` function for each group

Action simulation

- `while`-loop until `showStatus()` returns a value other than the `elsewhere` case
- use condition of `randrange(10)<6` for 60% likelihood of triggering `saxonAttack()`, otherwise `vikingAttack()`
- output the return strings of those methods for transparency (or visual clutter?)

Section 3

Conclusions

Possible extensions

- more detailed or less cluttered output
- more options for interactive parameter setting (possibly in menu)
  - property range brackets (health, strength) of of Viking/Saxon instances
  - probabilities for attack actions in main cycle
  - probability and strength of battle cry in Viking class
- game-like direction:
  - defense values
  - upgradability
  - healing
  - two-player mode (semi-round-based?)
- simulation-like direction:
  - introduce meta-iterations of game loops to generate larger datasets of the effects of property settings
  - reduce outputs

Difficulties, lessons learned?

- use the tests and grow beyond them

Thanks for your attention!