

Projeto Final de Orientação a Objetos com Python

1 Introdução

O contexto deste trabalho é o mundo dos jogos de RPG, onde os jogadores estão imersos em mundos de fantasia. O foco principal são os personagens, suas características e ações.

Foram separados dois grupos de personagens: os heróis (aqueles que o jogador controla) e os monstros (os inimigos). Representando os heróis, temos guerreiros e magos, enquanto os antagonistas são compostos por orcs e dragões.

Figura 1: Representação dos personagens do jogo.



Guerreiro

Mago

Orc

Dragão

Essencialmente, o jogo consiste em explorar o mapa do mundo e travar batalhas. A mecânica dos confrontos é baseada em turnos, onde cada personagem e inimigo faz uma ação a cada turno, podendo atacar, defender e realizar outras ações que serão melhor descritas mais à frente.

2 Hierarquia

A classe *Personagem* é abstrata e possui os atributos *nome* e *nível*. O nível inicial de um personagem é 1 e o máximo é 10. Além disso, possui alguns métodos abstratos, que devem ser implementados em suas subclasses: *atacar*, *defender*, *lutar* e *mover*.

A classe *Heroi* herda *personagem* e possui alguns atributos e métodos a mais (descritos a seguir). Além disso, possui os *getters* e *setters* dos atributos.

Tabela 1: Descrição da classe *Heroi*

Heroi	
Atributos	raça : indica a raça do herói, visto que esta é uma escolha comum neste tipo de jogo. Alguns exemplos são: humano, elfo, anão, mas este atributo aceita qualquer <i>string</i> , portanto o jogador é livre para decidir sua raça conforme preferências.
	atk (ataque): quantidade de dano que causará ao inimigo. Valor inicial é 1, para um herói nível 1. E aumenta 0.50 a cada evolução
	hp_max (saúde): quantidade de dano que o herói pode sofrer até ser derrotado em batalha. Valor inicial é 10 para um herói nível 1. E aumenta 1 a cada evolução.
	hp (saúde): é o estado atual da saúde do herói. O valor inicial é igual a hp_max e diminui a cada dano sofrido. Sempre que o herói evolui, todo o hp é restaurado.
	nome e nível são atributos que todo personagem tem. Já explanados na classe <i>Personagem</i> .

Métodos	atacar: ataca um inimigo se o sucesso do ataque estiver no intervalo de 6 a 10.
	O hp do inimigo diminui o valor do atk do herói.
	No jogo, o sucesso seria gerado aleatoriamente sempre que necessário, com <code>randint(0, 10)</code> , simulando um lançamento de dados.
	defender: defende-se de um ataque do inimigo se o sucesso da defesa estiver no intervalo de 7 a 10.
	O hp do herói diminui o valor do atk do inimigo.
	evoluir: evolui o herói, melhorando seus atributos.
	Nível é acrescido de 1; Ataque é acrescido de 0.50; Hp_max é acrescido de 1; Hp é restaurado até o novo valor máximo.
	fugir: o herói pode fugir da batalha se o sucesso da fuga estiver no intervalo entre o nível do inimigo e 10.
	Não é possível fugir de 'chefes', independente no nível.
	lutar: lutar chama os métodos atacar, defender e fugir dependendo da ação escolhida.
	A ação é escolhida pela entrada do usuário e pode ser 1, 2 ou 3. Caso seja diferente disso, o método lutar é chamado novamente.
	mover: move o personagem pelo mundo, através de pares de coordenadas.
	A ideia é que o par de coordenadas sejam obtidos por um click no local para onde deseja se deslocar.

A classe Monstro também herda personagem, mas possui algumas diferenças com relação a Heroi. Seus atributos são calculados de maneira diferente e ao invés de fugir, o monstro pode curar-se (restaurar saúde).

Tabela 2: Descrição da classe Monstro

Monstro	
Atributos	tipo: monstros podem ser inimigos comuns ou chefes.
	atk (ataque): quantidade de dano que causará ao inimigo.
	Valor inicial é $nível * 0.50$.
	hp_max (saúde): quantidade de dano que o herói pode sofrer até ser derrotado em batalha.
	Valor inicial é $10 + nível$.
	hp (saúde): é o estado atual da saúde do herói.
	O valor inicial é igual a hp_max e diminui a cada dano sofrido.
Métodos	nome e nível são atributos que todo personagem tem.
	Contudo, o nível de um monstro é especificado quando é instanciado, não sendo 1 por padrão.
	atacar: similar ao de herói, mas com intervalo de 5 a 10.
	defender: similar ao de herói, mas com intervalo de 6 a 10.
	curar: restaura a o HP do monstro em 25% do seu hp_max.
	Se o novo HP for maior que o máximo, é restaurado somente até o máximo.
	lutar: similar ao de herói, mas a ação seria gerada aleatoriamente com <code>randint(0, 10)</code> , pois não é possível controlar o monstro.
	mover: similar ao de herói, mas a escolha da coordenada seria calculada pelo computador, pois não é possível controlar o monstro.

A classe Guerreiro herda Heroi e implementa 1 atributo novo, seus *getters* e *setters* e alguns métodos.

Tabela 3: Descrição da classe Guerreiro

guerreiro = Guerreiro('Humano', 'Lancelot')	
Atributos	arma: guerreiros usam armas para atacar. A arma padrão de um guerreiro são seus punhos.
	É possível equipar mais 3 armas: adaga, arco e flecha e espada. Cada uma delas melhora os pontos de ataque de forma distinta.
	Os demais atributos são os mesmos de Heroi.
Métodos	atacar: modifica o método atacar de Heroi para mostrar com que arma o ataque foi realizado.
	equipar: altera a arma do guerreiro.
	Adaga aumenta atk em 10%, arco e flecha aumenta atk em 20% e espada aumenta atk em 30%

A classe Mago também herda Heroi e implementa 1 atributo novo, seus *getters* e *setters* e alguns métodos.

Tabela 4: Descrição da classe Mago

mago = Mago('elfo', 'Aldriel')	
Atributos	<p>poder: magos utilizam poderes para atacar. O padrão é 'disparar raios'.</p> <p>É possível utilizar mais 3 poderes: parar o tempo, 'bola de fogo' e 'encantamento'.</p> <p>Os demais atributos são os mesmos de Heroi.</p>
Métodos	<p>atacar: modifica o método atacar de Heroi para mostrar com que poder o ataque foi realizado.</p> <p>usar_poder: altera o poder do mago.</p> <p>Cada poder aumenta o atk do mago, similar ao que as armas fazem com o guerreiro.</p>

A classe Orc herda Monstro e implementa um novo método – emboscar – que potencializa o ataque do orc, causando danos maiores ao inimigo.

Tabela 5: Descrição da classe Orc

orc = Orc('comum', 'Baldur', 5)	
Atributos	<p>funcao: orcs costumam ter funções definidas dentro do exército.</p> <p>Os demais atributos são os mesmos de Monstro.</p>
Métodos	<p>emboscar: modifica o método o ataque do orc e causa danos graves ao inimigo.</p> <p>atk = atk*1.50</p>

A classe Dragao também herda Monstro e implementa alguns métodos novos, além de modificar alguns.

Tabela 6: Descrição da classe Dragao

dragao = Dragao('Spyro', 20)	
Atributos	<p>Os atributos são os mesmos de Monstro, exceto que todo dragão é, por padrão, um chefe.</p> <p>Logo, o herói não pode fugir de um dragão.</p>
Métodos	<p>cuspir_fogo: é o ataque especial do dragão, se utilizado, o herói perde a batalha (hp = 0).</p> <p>mover: modificado para chamar o método voar.</p> <p>voar: dragões se movem pelo mapa voando para as coordenadas desejadas.</p>

Durante o desenvolvimento do código foi possível colocar em prática os pilares do paradigma de programação orientada a objetos.

A herança foi aplicada cada vez que se criava uma classe de um novo nível:

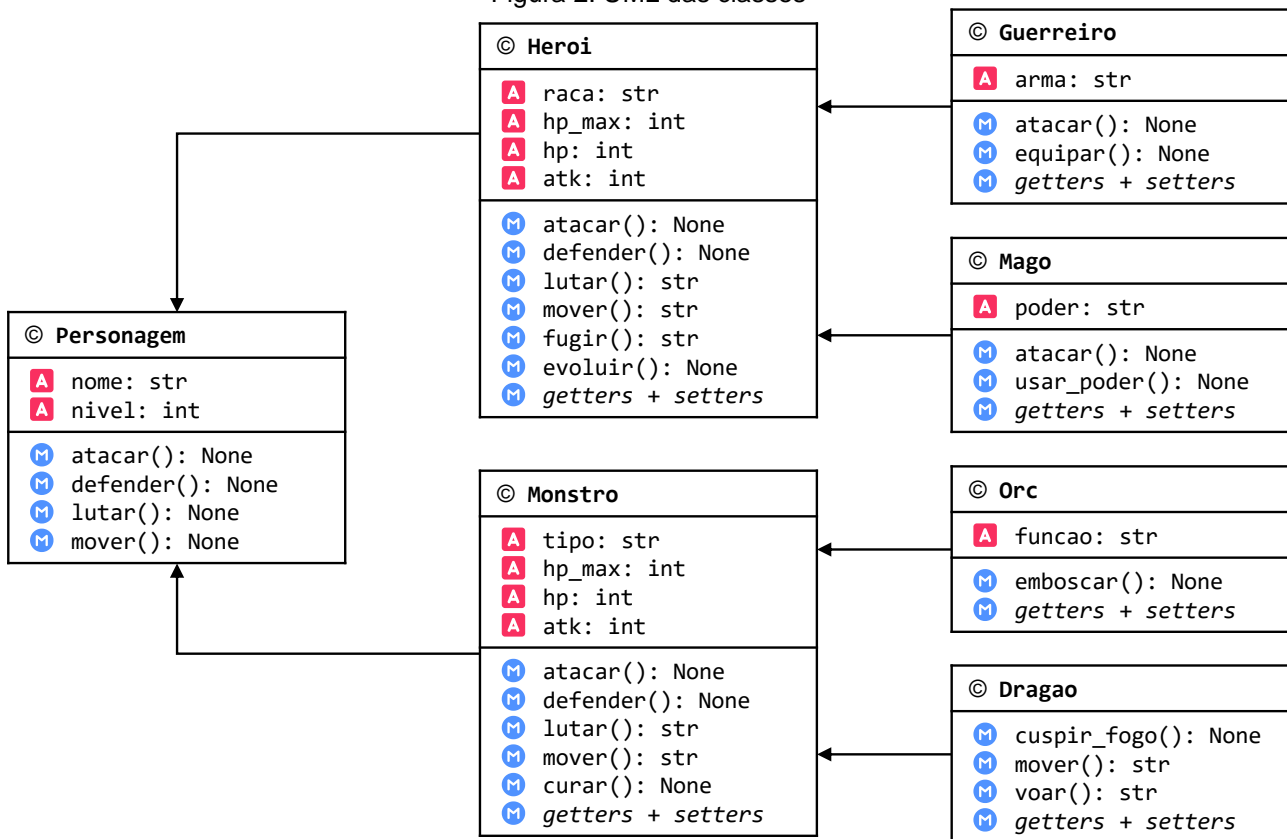
- Heroi e Monstro herdam atributos e métodos de Personagem
- Guerreiro e Mago herdam atributos e métodos de Heroi
- Orc e Dragao herdam atributos e métodos de Monstro

O encapsulamento foi empregado por meio dos *getters* e *setters*, utilizados para acessar e manipular atributos.

O polimorfismo foi utilizado nos casos em que alguns métodos foram sobrescritos para adaptar-se a nova classe. A exemplo: o método atacar é sobrescrito em diversas classes, assim como a forma de lutar de Heroi se distingue de Monstro.

O diagrama UML das classes é mostrado a seguir e nele é possível identificar o que foi descrito previamente nas tabelas. Para mais detalhes a respeito das classes, consulte o código em: https://github.com/mcalheiro/iartes-modulo03/tree/master/projeto_final.

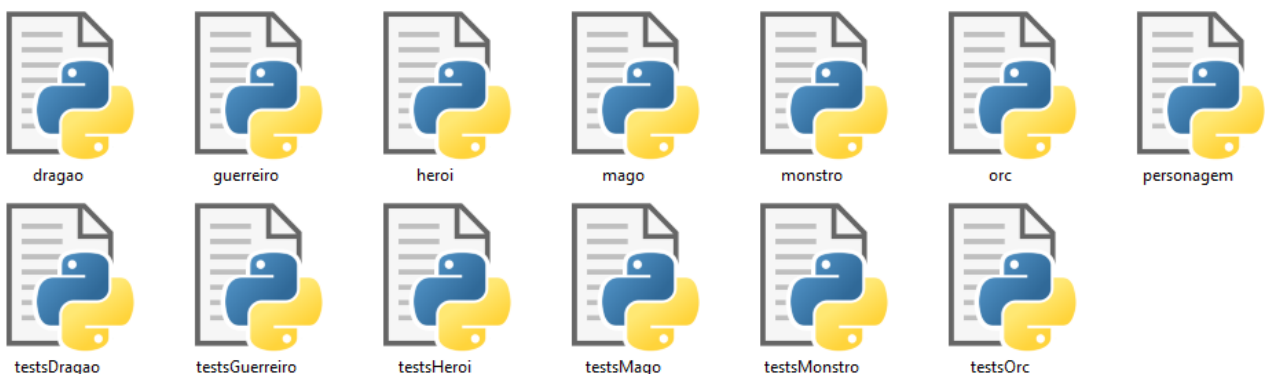
Figura 2: UML das classes



3 Cobertura do código

O código foi escrito em módulos, de modo que cada classe esteja em um arquivo. A metodologia aplicada foi TDD (*test-driven development*). Contudo, para a classe abstrata Personagem não foram escritos testes, devido a seus métodos serem abstratos.

Figura 3: Módulos do projeto



3.1 Cobertura em 50% do projeto

Visto que temos 7 classes, considerou-se que a metade do projeto seria uma vez que 3 delas estivessem prontas. Sendo assim, nesta etapa haviam sido escritas as classes Personagem, Herói e Monstro. Para obter a cobertura fez-se:

```
coverage run -m pytest testsHerói.py testsMonstro.py
coverage report -m
```

A essa altura, a classe Heroi possuía 12 testes prontos e a classe Monstro possuía 10 testes. Esta diferença se dá porque Heroi possui 1 método a mais. O resultado foi:

Name	Stmts	Miss	Cover	Missing

heroi.py	86	3	97%	17, 85, 117
monstro.py	80	0	100%	
personagem.py	13	0	100%	
testsHeroi.py	57	0	100%	
testsMonstro.py	45	0	100%	

TOTAL	281	3	99%	

3.2 Cobertura em 80% do projeto

Ao escrever mais 2 classes (totalizando 5), analisou-se novamente a cobertura. Agora, as classes Guerreiro e Mago também estavam prontas, com 4 testes cada.

```
coverage run -m pytest testsHeroi.py testsMonstro.py testsGuerreiro.py
testsMago.py
coverage report -m
```

E obteve-se máxima cobertura em todas as novas classes (Guerreiro e Mago).

Name	Stmts	Miss	Cover	Missing

guerreiro.py	25	0	100%	
heroi.py	86	3	97%	17, 85, 117
mago.py	25	0	100%	
monstro.py	80	0	100%	
personagem.py	13	0	100%	
testsGuerreiro.py	25	0	100%	
testsHeroi.py	57	0	100%	
testsMago.py	25	0	100%	
testsMonstro.py	45	0	100%	

TOTAL	381	3	99%	

3.3 Cobertura em 100% do projeto

Ao escrever mais as ultimas 2 classes (totalizando 7), atingiu-se 100% do desenvolvimento. Agora, as classes Orc e Dragao também estavam prontas, com 1 e 3 testes respectivamente. Executou-se mais uma vez o pytest com coverage

```
coverage run -m pytest testsHeroi.py testsMonstro.py testsGuerreiro.py
testsMago.py testsDragao.py testsOrc.py
coverage report -m
```

Mais uma vez obteve-se máxima cobertura em todas as novas classes (Orc e Dragao), e o projeto inteiro apresentou cobertura de 99% de cobertura dos testes. Os pontos indicados

como *missing* no arquivo **heroi.py** foram analisados e não aparentam gerar problemas, portanto a equipe finalizou o desenvolvimento do projeto.

Name	Stmts	Miss	Cover	Missing

dragao.py	14	0	100%	
guerreiro.py	25	0	100%	
heroi.py	86	3	97%	17, 85, 117
mago.py	25	0	100%	
monstro.py	80	0	100%	
orc.py	15	1	93%	12
personagem.py	13	0	100%	
testsDragao.py	21	0	100%	
testsGuerreiro.py	25	0	100%	
testsHeroi.py	57	0	100%	
testsMago.py	25	0	100%	
testsMonstro.py	45	0	100%	
testsOrc.py	15	0	100%	

TOTAL	446	4	99%	

4 Conclusões

O desenvolvimento deste projeto trouxe benefícios a equipe, pois alguns tiveram pouco ou nenhum contato prévio com TDD. Notou-se que utilizar esta metodologia para escrever programas torna a validação do código eficiente e garante alta cobertura, uma vez que quase tudo que será programado já está contemplado por testes.

A dificuldade inicial da equipe foi definir o contexto da abstração, pois todos tinham ideias muito distintas e pouco claras, mas o orientador da disciplina foi muito colaborativo em ajudar o time a definir um ponto de partida. Durante o desenvolvimento, a dificuldade foi o retrabalho, pois em alguns momentos a equipe decidiu mudar atributos e métodos de algumas classes a pós criar os testes, precisando refatorá-los. Contudo, tais decisões foram tomadas para que as classes fizessem mais sentido.

Notou-se que conforme o desenvolvimento avançava, menos testes eram necessários, pois as novas classes estavam reaproveitando código já previamente validado de suas superclasses. Vale destacar também que a modularização dos arquivos permitiu com que os membros da equipe trabalhassem ao mesmo tempo, acelerando o desenvolvimento sem o perigo de gerar conflitos no gerenciador de versões.