Universidade do Minho	Escola de Engenharia da Universidade do Minho **Mestrado Integrado em Eng. Electrónica Industrial e Computadores** **Complementos de Programação de Computadores**	**MIEEIC** **(1º Ano)** **2º Sem**
Luís Paulo Reis		
Aula Prática 3b: Exercícios de Introdução à Programação Orientada a Objectos		

Objectivos:

Esta Folha de Exercícios destina-se a:

• Compreender os conceitos de classes e encapsulamento, dados público e dados privados.

• Compreender os conceitos de operador e sobrecarga de operadores.

Os exercícios aqui propostos deverão ser realizados no mais simples ambiente de desenvolvimento possível para a linguagem C: editor de texto, editor DevC++, codeblocks e/ou ferramentas da GCC (GNU Compiler Collection) e afins.

Exercício 3b

Pretende-se escrever um programa em C++, denominado "JogoMedieval". O nosso jogo vai ter dois tipos de personagens: os jogadores e os monstros. O programa deve então conter uma classe denominada `CJogoMedieval` com a seguinte estrutura:

```
class CHARACTER {
   public:
      string tipo; // “monster” or “player”
      int health;
      int strength;
      double x;
      double y;
      bool operator==(CHARACTER c2);
};

class CJogoMedieval {
  private:
    vector<CHARACTER> characters;
    int activeCharacter;

  public:
    CJogoMedieval (int nPlayers, int nMonsters);
    CJogoMedieval ();

    void saveGame (const char *filename);
    void loadGame (const char *filename);
    void show_monsters_players();

    CHARACTER findWeakestMonster(int maxHealth);
};
```

Execute as tarefas seguintes, tendo sempre o cuidado de testar no programa principal (função `main`), todas as funções criadas. O código fonte do programa deve ser escrito em ficheiros com o nome "main_JogoMedieval.cpp", "JogoMedieval.h".e "JogoMedieval.cpp".

a) Implemente o construtor

```
CJogoMedieval (int nPlayers, int nMonsters);
```

que gera para o vector `characters` um conjunto de `nPlayers` e `nMonsters`. A posição de cada um deve ser aleatória entre 0 e 1, a `strength` gerada aleatoriamente entre 40 e 100, e a `health` inicializada a 100. Deve ainda colocar o índice do personagem activo (`activeCharacter`) a 0.

O construtor `CJogoMedieval();` apenas coloca o índice do personagem activo a 0.

b) Implemente a função membro pública:

```
void show_monsters_players();
```

que mostra no écran todos os players e monsters do jogo e as respetivas características.

c) Implemente os membros-função

```
saveGame (const char * filename);
loadGame (const char * filename);
```

para guardar e carregar o estado do jogo a partir de um ficheiro.

O ficheiro está organizado em linhas, cada linha contendo o tipo de personagem, a saúde, força e a posição x e y, separados por espaços, por exemplo:

```
player 98 56 0.4 0.7
monster 77 34 0.1 0.4
monster 9 18 0.7 0.2
```

d) Implemente o membro-função

```
CHARACTER findWeakestMonster(int maxHealth);
```

que retorna o monstro mais fraco, tendo no máximo `maxHealth` de `health`.

e) Implemente o operador

```
bool CHARACTER::operator==(CHARACTER& character);
```

que determina se dois personagens são iguais.

f) Implemente o operador

```
istream& operator>> (istream& is, CJogoMedieval::CHARACTER& character);
```

Altere a função-membro `loadGame()` para usar este operador.