



UNIVERSITÀ
DEGLI STUDI
FIRENZE



Esercitazione

Class exercise

Obiettivo

- Il progetto CLion fornito contiene classi e scheletri di classi che implementano un gioco in stile Rogue (<https://it.wikipedia.org/wiki/Roguelike>)
- Scopo dell'esercitazione è:
 - Completare le classi GameCharacter e Weapon, scrivendo codice di diversi metodi
 - Correggere il comportamento del gioco nel file main.cpp per fare in modo che il combattimento tra l'eroe ed un mostro sia più realistico

Obiettivo

```
-----  
|....|          #####  
|....|          #          #  
|.$.+.#####          #  
|....|          #          ----+----  
-----          #          |.....|  
          #          |.!....|  
          #          |.....|  
          #          |..@..|  
          #          |.....|  
-----          #          |.....|  
|. .|          #####+.D..|  
|<.+###          #          |.....|  
-----          #          |.?....|  
          #####          -----
```

- Muro
- # Corridoio buio
- . Zona illuminata
- \$ Oro
- + Porta
- | Muro
- ! Pozione magica
- @ Il protagonista
- D Drago rosso
- < Scale per livello sup.
- ? Pergamena magica

Schema del codice

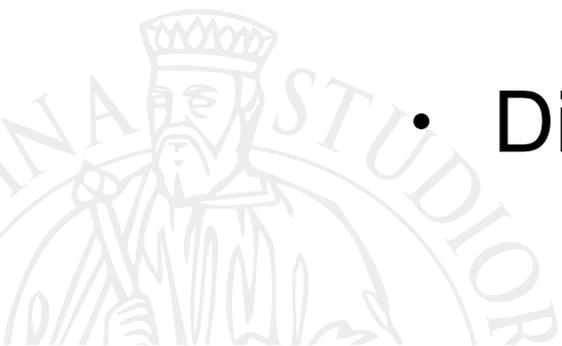
- Il programma è composto da 3 classi:
- Dungeon crea mappe casuali con stanze, corridoi, scale, porte, etc.
- Weapon rappresenta un arma con forza e magia
- GameCharacter rappresenta un personaggio del gioco, ed è composto con Weapon, sfruttandone il metodo `use()` per calcolare l'effetto del combattimento contro un altro GameCharacter.

Schema del codice

- In main sono presenti diverse funzioni che implementano elementi fondamentali del gioco:
 - `getEvent()` interagisce col giocatore tramite la tastiera, ricevendo i comandi
 - `isLegalCell()` controlla che una cella della mappa del Dungeon possa ospitare un personaggio
 - `setupCharacterCell()` cerca delle coordinate nella mappa dove disporre un personaggio all'avvio del gioco ("spawn")
 - `isLegalMove()` controlla che il movimento di un personaggio sia consentito, ovvero si muova su una cella che lo può contenere, controllando che questa cella già non contenga un altro personaggio.
 - `updateGame()` aggiorna lo stato del gioco secondo i comandi del giocatore
 - `renderHUD()` disegna sullo schermo le caratteristiche del giocatore (head up display)
 - `renderGame()` disegna sullo schermo la mappa ed i personaggi (operazione di "compositing")

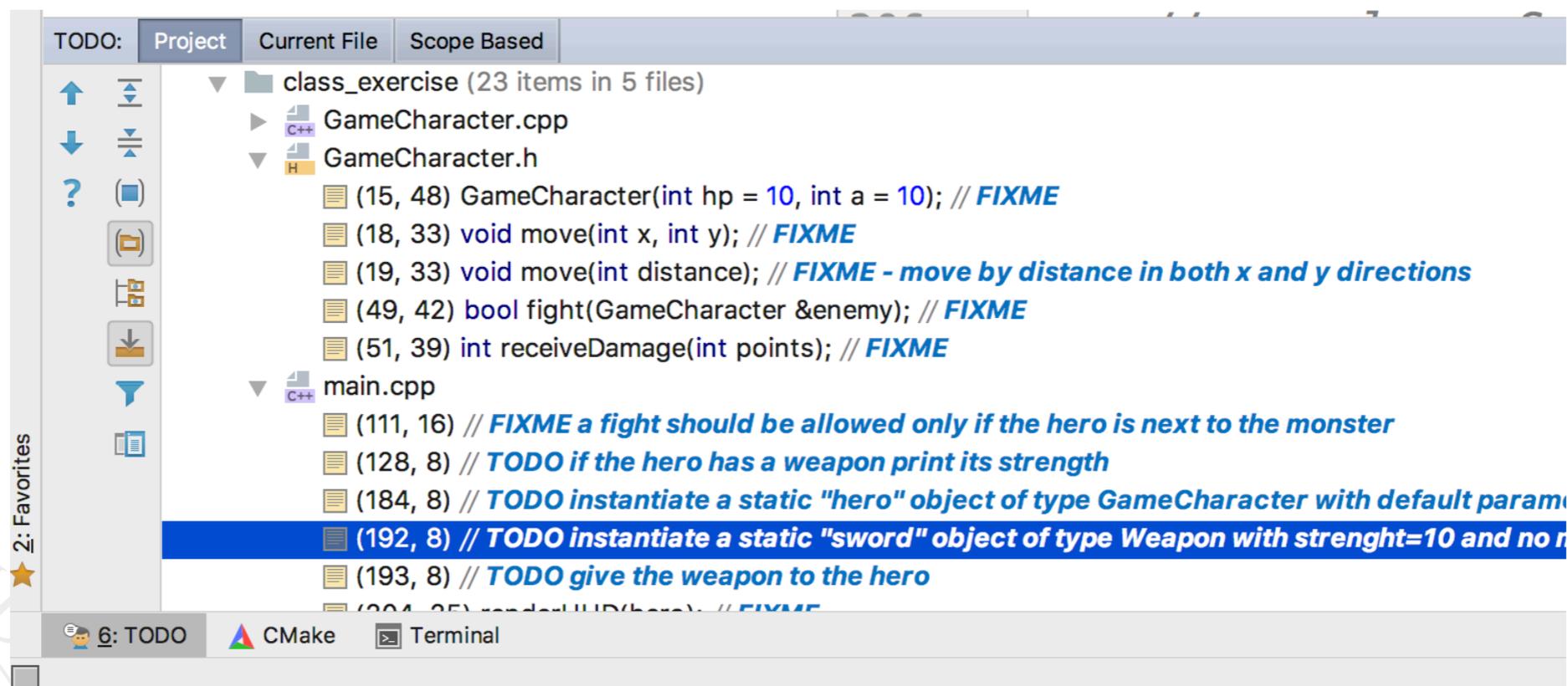
Schema del codice

- La funzione `main()` provvede ad istanziare personaggi, mappa ed armi. Il cuore è un classico “game loop”, ovvero un ciclo infinito (a meno di richiesta di fine gioco) in cui:
 - Si ricevono gli eventi del gioco (in questo caso i comandi del giocatore)
 - Si aggiorna lo stato del gioco sulla base degli eventi
 - Si disegna (rendering) il tutto:
 - Disegnando lo HUD
 - Disegnando la schermata del gioco



Dove modificare il codice

- Le indicazioni precise sul codice da modificare sono fornite come commenti indicati con TODO e FIXME
- Per vedere tutti questi commenti selezionare la finestra TODO di CLion



Dove modificare il codice

The screenshot shows the CLion IDE interface. The main editor displays C++ code in `main.cpp` with several TODO comments. The code is as follows:

```
185 // find a legal start position
186 int startX = 0;
187 int startY = 0;
188 setupCharacterCell(startX, startY, map);
189 hero.setPosX(startX);
190 hero.setPosY(startY);
191 // create a weapon and give it to hero
192 // TODO instantiate a static "sword" object of type Weapon with strenght=10 and
193 // TODO give the weapon to the hero
194 // create an enemy with a low grade armor
195 GameCharacter enemy(20, 2);
196 // find monster position not too far from hero position
197 startX += 5;
198 startY += 3;
199 setupCharacterCell(startX, startY, map);
200 enemy.setPosX(startX);
201 enemy.setPosY(startY);
202
203 // render
204 renderHUD(hero); // FIXME
205 renderGame(map, hero, enemy);
```

The TODO list at the bottom of the IDE shows the following items:

- (15, 48) `GameCharacter(int hp = 10, int a = 10);` // FIXME
- (18, 33) `void move(int x, int y);` // FIXME
- (19, 33) `void move(int distance);` // FIXME - move by distance in both x and y directions
- (49, 42) `bool fight(GameCharacter &enemy);` // FIXME
- (51, 39) `int receiveDamage(int points);` // FIXME
- (111, 16) // FIXME a fight should be allowed only if the hero is next to the monster
- (128, 8) // TODO if the hero has a weapon print its strength
- (184, 8) // TODO instantiate a static "hero" object of type GameCharacter with default parameters
- (192, 8) // TODO instantiate a static "sword" object of type Weapon with strenght=10 and no magic
- (193, 8) // TODO give the weapon to the hero
- (204, 25) `renderHUD(hero);` // FIXME



Classe Weapon

- Implementare il costruttore
- Implementare getter/setter per i due attributi
- Implementare il metodo `use()` secondo le specifiche nei commenti



Classe GameCharacter

- Implementare il costruttore (la signature è fornita)
- Implementare il metodo `move()` con due overload (signature fornite)
- Completare l'implementazione dei metodi `fight()` e `receiveDamage()`
- Notare come la classe `GameCharacter` sia composta con `Weapon`. Attenzione al fatto che `Weapon` è un puntatore e deve essere inizializzato e controllato prima del suo uso.

Main

- Istanziare oggetti: hero di tipo GameCharacter, sword di tipo Weapon, e enemy di tipo GameCharacter
- Completare la funzione renderHUD() per fare in modo che se hero ha un'arma a disposizione ne venga stampata la forza
- EXTRA: modificare la funzione updateGame(), eventualmente aggiungendo nuove funzioni di aiuto per fare in modo che il giocatore possa attaccare il mostro solo quando è in una posizione adiacente.



Classe Dungeon

- Questa classe non deve essere toccata

