Esercizio guidato

Settimana 11
13/12/2022

Si ringrazia il Dott. Giacomo Baruzzo per il materiale

Richiami: classi e oggetti

Una classe costituisce lo schema progettuale degli oggetti di una determinata categoria e **definisce un nuovo tipo di dato**

Una classe definisce le proprietà dei suoi oggetti (**attributi**) e le operazioni che possono applicarsi su di essi (**metodi**)

Passi per progettare una classe:

1. Definizione interfaccia pubblica
2. Definizione dello stato dell'oggetto
3. Definizione del costruttore
4. Definizione dei metodi

1) Interfaccia pubblica

self è una speciale variabile parametro che memorizza l'oggetto con cui il metodo è stato invocato

- La ***interfaccia pubblica*** è l'insieme dei metodi invocabili con oggetti che siano esemplari di tale classe
- L'interfaccia pubblica di una classe **definisce le funzionalità dei suoi esemplari**
- È costituita dai suoi metodi e, per ciascun metodo:
  - Nome
  - Elenco delle variabili parametro e del loro significato
  - Eventuale valore restituito e suo significato
  - Descrizione della funzionalità svolta (non necessariamente il codice del metodo)
- Tutti i metodi definiti all'interno di una classe devono avere, come **primo parametro**, la variabile **self**

2) Stato dell'oggetto

Le variabili di esemplare, per convenzione che seguiremo, hanno nomi che iniziano con _

- Lo ***stato dell'oggetto*** è l'insieme delle informazioni (proprietà, attributi) che lo caratterizzano, in funzione dei metodi che con esso verranno invocati
- Lo stato dell'oggetto viene memorizzato al suo interno tramite le **variabili di esemplare**
- Le variabili di esemplare di un oggetto non hanno **NESSUNA** relazione con le variabili di esemplare di un altro oggetto dello stesso tipo, anche se hanno gli stessi nomi

3) Costruttore

Il costruttore, in una classe, è un metodo che si chiama **__init__**

- Di norma, ***il costruttore*** è il primo **metodo** definito in una classe e il suo compito è quello di **assegnare un valore a tutte le variabili di esemplare** dell'oggetto appena creato
- Viene invocato **automaticamente** dall'interprete dopo aver creato un oggetto: non va **mai** invocato esplicitamente
  - Metodi Python il cui nome inizia con un doppio carattere di sottolineatura hanno uno scopo specifico, riservato all'interprete (nel caso del costruttore, inizializzare un oggetto appena creato).

4) Metodi

- La definizione di un metodo è molto simile alla definizione di una funzione
- IMPORTANTE: Tutti i metodi definiti all'interno di una classe devono avere, come primo parametro, la **variabile self**
  - Questo permette, dall'interno del corpo del metodo, di avere accesso alle variabili di esemplare dell'oggetto su cui si agisce
  - Attenzione: se dimentichiamo la variabile parametro **self** nella definizione del metodo, **non viene segnalato un errore di sintassi**
- Suggerimenti utili:
  - Tutte le variabili di esemplare dovrebbero essere private e quasi tutti i metodi dovrebbero essere pubblici.
  - Le variabili di esemplare vanno quindi manipolate soltanto attraverso metodi.
  - Occasionalmente, può aver senso progettare qualche metodo privato (dovrebbe avere un nome che inizia con un singolo carattere di sottolineatura).

Lab 3 – Es 9 (slide 1/3)

Il gioco di Nim. Si tratta di un gioco con un certo numero di varianti: utilizzeremo la versione seguente, che ha una strategia interessante per arrivare alla vittoria. Due giocatori prelevano a turno biglie da un mucchio. In ciascun turno, il giocatore sceglie quante biglie prendere: deve prenderne almeno una, ma non oltre la metà del mucchio. Perde chi è costretto a prendere l'ultima biglia.

Scrivere il (programma-)GIOCO **nim.py** con cui un giocatore umano possa giocare a Nim contro il computer. Si genera un numero intero casuale, compreso fra 10 e 100, per indicare il numero iniziale di biglie. Si decide casualmente se la prima mossa tocca al computer o al giocatore umano. Si decide casualmente se il computer giocherà in modo intelligente o stupido. Nel modo stupido, quando è il suo turno, il computer si limita a sottrarre dal mucchio un numero casuale di biglie, purché sia una quantità ammessa. Nel modo intelligente, il computer preleva il numero di biglie sufficiente affinché il numero di quelle rimanenti sia uguale a una potenza di due diminuita di un'unità, ovvero 1, 3, 7, 15, 31 o 63: ricordate però che la mossa è valida se il giocatore non prende più della metà delle biglie del mucchio. Si può dimostrare che, se il computer preleva il numero di biglie sufficiente affinché il numero di quelle rimanenti sia uguale a una potenza di due diminuita di un'unità (biglie rimanenti = 1, 3, 7, 15, 31 o 63), si tratta sempre di una mossa valida, eccetto quando il numero delle biglie è uguale a una potenza di due diminuita di un'unità. In caso di mossa non valida il computer preleverà una quantità di biglie casuale, purché ammessa

Lab 3 – Es 9 (slide 2/3)

Si noti che, nel modo intelligente, il computer non può essere sconfitto quando ha la prima mossa, a meno che il mucchio non contenga inizialmente 15, 31 o 63 biglie. Nelle medesime condizioni, un giocatore umano che abbia la prima mossa e che conosca tale strategia vincente, può ovviamente vincere contro il computer.

Il programma deve giocare una partita dopo l'altra, fino a quando l'utente non introduce un opportuno comando che interrompe il gioco. Ogni partita è completamente indipendente dalle precedenti, sia per numero di biglie iniziali, sia per stupidità/intelligenza del computer, sia per assegnazione della prima mossa.

Lab 3 – Es 9 (slide 3/3)

Per risolvere un problema articolato come questo, occorre procedere "per gradi", individuando soluzioni di problemi intermedi che si possano collaudare, in modo da aggiungere funzionalità a uno schema già funzionante, altrimenti, se si collauda il programma soltanto alla fine, diventa ESTREMAMENTE difficile diagnosticare i problemi che si manifestano (e che ci saranno senz'altro...).

Un possibile schema da seguire potrebbe essere questo:

1. impostare un ciclo (virtualmente) infinito, il cui corpo conterrà tutto ciò che serve per giocare una partita, ma che per il momento abbia soltanto il compito di chiedere all'utente se vuole giocare un'altra partita, gestendo accuratamente tutti i casi ("Vuoi giocare ancora?": risposta S, risposta N, risposta diversa); questa fase di input richiede, a sua volta, un ciclo, che termini soltanto quando l'utente risponde S o N; risolto il problema, collaudare il programma in tutti i casi possibili e verificare che il comportamento sia sempre corretto
2. arricchire il corpo del ciclo principale (quello impostato al passo precedente), aggiungendo la fase di inizializzazione dei parametri di gioco (chi giocherà per primo? il computer sarà "intelligente" oppure no? quante biglie saranno inizialmente in gioco?), con temporanea visualizzazione di tali parametri, procedendo a un loro collaudo con più esecuzioni, verificando anche che i parametri cambino di partita in partita se il giocatore chiede di giocare di nuovo (senza eseguire di nuovo il programma)
3. concludere la soluzione del problema inserendo nel corpo del ciclo principale, tra la fase di inizializzazione dei parametri e la fase di richiesta "Vuoi giocare ancora?", un ciclo che gestisca i turni (alternando i due giocatori, umano e computer), tenendo conto della quantità di biglie rimaste nel mucchio e decidendo quando la partita è finita, decretando il vincitore; collaudare il programma completo mediante molteplici esecuzioni

Es 10-2

Riprendere in esame il Gioco di Nim visto nei precedenti esercizi e scrivere il programma nim3.py con cui un giocatore umano possa giocare a Nim contro il computer, effettuando una progettazione orientata agli oggetti. Il comportamento del programma deve essere identico al comportamento di nim2.py.

Progettare le **classi** seguenti:

- **NimPile** che rappresenta il mucchio di biglie usato durante una partita. Deve avere:
  - il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio
  - il metodo **getTotal** che restituisce il numero di biglie presenti nel mucchio
  - il metodo **take** che riceve il numero di biglie da eliminare dal mucchio in seguito a una mossa di uno dei giocatori
- **NimHumanPlayer** che rappresenta il giocatore umano. Deve avere:
  - il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e chiede all'utente di fare una mossa (ripetendo la richiesta finché non viene indicata una mossa valida), dopodiché mette in atto la mossa (invocando il metodo **take** del mucchio)
  - dato che gli esemplari di tale classe non hanno bisogno di variabili di esemplare, non hanno bisogno di un costruttore
- **NimComputerPlayer** che rappresenta il computer che gioca. Deve avere:
  - il costruttore che riceve un valore booleano, **True** se e solo se il computer deve giocare in modo *intelligente*
  - il metodo **isExpert** che restituisce **True** se e solo se il computer sta giocando in modo *intelligente*
  - il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e calcola la mossa del computer sulla base della modalità in cui sta operando, dopodiché mette in atto la mossa (invocando il metodo take del mucchio)
- **NimGame** che rappresenta un'intera partita. Deve avere:
  - il costruttore che non riceve parametri
  - il metodo **play** che gioca un'intera partita e visualizza il vincitore; se il vincitore è il computer, visualizza anche la modalità di gioco (*intelligente* o *stupido*)

Il programma principale è semplicemente costituito da un ciclo "infinito" che, ad ogni iterazione, crea un esemplare di NimGame, ne invoca il metodo play e, poi, chiede all'utente se vuole fare un'altra partita oppure no.

Es 10-2

Riprendere in esame il Gioco di Nim visto nei precedenti esercizi e scrivere il programma nim3.py con cui un giocatore umano possa giocare a Nim contro il computer, effettuando una progettazione orientata agli oggetti. Il comportamento del programma deve essere identico al comportamento di nim2.py.

Progettare le **classi** seguenti:

- **NimPile** che rappresenta il mucchio di biglie usato durante una partita. Deve avere:
  - il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio
  - il metodo **getTotal** che restituisce il numero di biglie presenti nel mucchio
  - il metodo **take** che riceve il numero di biglie da eliminare dal mucchio in seguito a una mossa di uno dei giocatori
- **NimHuman**
  - il metodo … una mossa (ripeten… invocando il metodo …
  - dato che… costruttore
- **NimCompu…**
  - il costrut… *te*
  - il metodo **isExpert** che restituisce **True** se e solo se il computer sta giocando in modo *intelligente*
  - il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e calcola la mossa del computer sulla base della modalità in cui sta operando, dopodiché mette in atto la mossa (invocando il metodo take del mucchio)

Per ognuna delle classi, seguiremo i 4 passi della progettazione:

1. **Definizione interfaccia pubblica**
2. **Definizione dello stato dell'oggetto**
3. **Definizione del costruttore**
4. **Definizione dei metodi**

- **NimGame** che rappresenta un'intera partita. Deve avere:
  - il costruttore che non riceve parametri
  - il metodo **play** che gioca un'intera partita e visualizza il vincitore; se il vincitore è il computer, visualizza anche la modalità di gioco (*intelligente* o *stupido*)

Il programma principale è semplicemente costituito da un ciclo "infinito" che, ad ogni iterazione, crea un esemplare di NimGame, ne invoca il metodo play e, poi, chiede all'utente se vuole fare un'altra partita oppure no.

1) Classe NimPile

Classe **NimPile** rappresenta il mucchio di biglie usato durante una partita.

Deve avere:

- il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio
- il metodo **getTotal** che restituisce il numero di biglie presenti nel mucchio
- il metodo **take** che riceve il numero di biglie da eliminare dal mucchio in seguito a una mossa di uno dei giocatori

Classe NimPile – Definizione interfaccia pubblica

```
class NimPile :


 ## Restituisce il numero di biglie
 # @return numero di biglie
 def getTotal(self) :
  # codice metodo


 ## Elimina "tot" biglie
 # @param takenBalls N. biglie da
eliminare
 def take(self, takenBalls) :
  # codice metodo
```

Classe **NimPile** rappresenta il mucchio di biglie usato durante una partita.
Deve avere:

- il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio
- il **metodo getTotal** che **restituisce il numero di biglie** presenti nel mucchio
- il **metodo take** che **riceve il numero di biglie da eliminare** dal mucchio in seguito a una mossa di uno dei giocatori

Classe NimPile – Definizione stato dell'oggetto

```
class NimPile :


 ## Restituisce il numero di biglie
 # @return numero di biglie
 def getTotal(self) :
  # codice metodo


 ## Elimina "tot" biglie
 # @param takenBalls N. biglie da
eliminare
 def take(self, takenBalls) :
  # codice metodo
```

Classe **NimPile rappresenta il mucchio di biglie usato durante una partita**.

Deve avere:

- il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio
- il metodo getTotal che restituisce il numero di biglie presenti nel mucchio
- il metodo take che riceve il numero di biglie da eliminare dal mucchio in seguito a una mossa di uno dei giocatori

Classe NimPile – Definizione costruttore

```
class NimPile :

  def __init__(self, numBalls) :
    if numBalls < 1 :
      raise ValueError("Error in NimPile constructor: " + str(numBalls))
    self._numBalls = numBalls

  ## Restituisce il numero di biglie
  # @return numero di biglie
  def getTotal(self) :
    # codice metodo

  ## Elimina "tot" biglie
  # @param takenBalls num. Biglie da eliminare
  def take(self, takenBalls) :
    # codice metodo
```

Classe **NimPile** rappresenta il mucchio di biglie usato durante una partita.
Deve avere:

- **il costruttore che riceve il numero di biglie da inserire inizialmente nel mucchio**
- il metodo getTotal che restituisce il numero di biglie presenti nel mucchio
- il metodo take che riceve il numero di biglie da eliminare dal mucchio in seguito a una mossa di uno dei giocatori

Classe NimPile – Definizione metodi

```
class NimPile :

  def __init__(self, numBalls) :
    if numBalls < 1 :
      raise ValueError("Error in NimPile constructor: " + str(numBalls))
    self._numBalls = numBalls

  ## Restituisce il numero di biglie
  # @return numero di biglie
  def getTotal(self) :
    return self._numBalls

  ## Elimina «tot» biglie
  # @param takenBalls num. Biglie da eliminare
  def take(self, takenBalls) :
    if takenBalls < 1 or takenBalls > self._numBalls // 2 :
      raise ValueError("Error in NimPile take method: " + str(takenBalls))
    self._numBalls -= takenBalls
```

2) Classe NimHumanPlayer

Classe **NimHumanPlayer** rappresenta il giocatore umano.

Deve avere:

- il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e chiede all'utente di fare una mossa (ripetendo la richiesta finché non viene indicata una mossa valida), dopodiché mette in atto la mossa (invocando il metodo **take** del mucchio)
- dato che gli esemplari di tale classe non hanno bisogno di variabili di esemplare, non hanno neanche bisogno di un costruttore

Classe NimHumanPlayer - Definizione interfaccia pubblica

```
class NimHumanPlayer :

 ## Dato in input il mucchio di
 #  biglie, chiede all'utente di
 #  fare una mossa e la attua
 # @param pile Mucchio di biglie
 def move(self, pile) :
  # codice metodo
```

Classe **NimHumanPlayer** che rappresenta il giocatore umano.

Deve avere:

- il metodo **move** che **riceve il mucchio di biglie** (cioè un oggetto di tipo NimPile) e chiede all'utente di fare una mossa (ripetendo la richiesta finché non viene indicata una mossa valida), dopodiché mette in atto la mossa (invocando il metodo **take** del mucchio)
- dato che gli esemplari di tale classe non hanno bisogno di variabili di esemplare, non hanno neanche bisogno di un costruttore

Classe NimHumanPlayer – Definizione stato oggetto e costruttore

Classe **NimHumanPlayer** che rappresenta il giocatore umano.

Deve avere:

- il metodo move che riceve il mucchio di biglie (cioè un oggetto di tipo NimPile) e chiede all'utente di fare una mossa (ripetendo la richiesta finché non viene indicata una mossa valida), dopodiché mette in atto la mossa (invocando il metodo take del mucchio)
- dato che gli esemplari di tale classe non hanno bisogno di variabili di esemplare, non hanno neanche bisogno di un costruttore

Dal testo risulta chiaro che:

- **Non ci sono variabili di esemplare**
- **Non è necessario definire un costruttore**

Classe NimHumanPlayer - Definizione metodi

```
class NimHumanPlayer :

  ## Dato in input il mucchio di biglie, chiede all'utente la mossa e la attua
  # @param pile Mucchio di biglie
  def move(self, pile) :
    while True :
      numBalls = pile.getTotal()
      print("Biglie presenti nel mucchio:", numBalls)
      taken = input("Quante ne vuoi prendere? ")
      try :
        taken = int(taken)
        if taken >= 1 and taken <= (numBalls // 2) :
          pile.take(taken) # eseguo la mossa
          return
        print("Mossa errata: riprova!")
      except ValueError :
        print("Mossa errata: riprova!")
```

3) Classe NimComputerPlayer

Classe **NimComputerPlayer** rappresenta il computer che gioca.

Deve avere:

- il costruttore che riceve un valore booleano, **True** se e solo se il computer deve giocare in modo *intelligente*
- il metodo **isExpert** che restituisce **True** se e solo se il computer sta giocando in modo *intelligente*
- il metodo **move** che riceve il mucchio di biglie (cioè un oggetto di tipo **NimPile**) e calcola la mossa del computer sulla base della modalità in cui sta operando, dopodiché mette in atto la mossa (invocando il metodo **take** del mucchio)

Classe NimComputerPlayer - Definizione interfaccia pubblica

```
class NimComputerPlayer :

  ## Restituisce se gioco in
modalità intelligente
  # @return modalità intelligente
  def isExpert(self) :
    # codice metodo

  ## Esegue la mossa
  # @param pile Mucchio di biglie
  def move(self, pile) :
    # codice metodo
```

Classe **NimComputerPlayer** rappresenta il computer che gioca. Deve avere:

- il costruttore che riceve un valore booleano, True se e solo se il computer deve giocare in modo intelligente
- **il metodo isExpert che restituisce** True se e solo **se il computer sta giocando in modo intelligente**
- **il metodo move** che **riceve il mucchio di biglie** (cioè un oggetto di tipo NimPile) e **calcola la mossa** del computer sulla base della modalità in cui sta operando, dopodiché **mette in atto la mossa** (invocando il metodo take del mucchio)

Classe NimComputerPlayer - Definizione stato dell'oggetto

```
class NimComputerPlayer :


  ## Restituisce se gioco in
modalità intelligente
  # @return modalità intelligente
  def isExpert(self) :
    # codice metodo


  ## Esegue la mossa
  # @param pile Mucchio di biglie
  def move(self, pile) :
    # codice metodo
```

Classe **NimComputerPlayer** rappresenta il computer che gioca. Deve avere:

- il costruttore che riceve **un valore booleano, True se e solo se il computer deve giocare in modo intelligente**
- il metodo isExpert che restituisce True se e solo se il computer sta giocando in modo intelligente
- il metodo move che riceve il mucchio di biglie (cioè un oggetto di tipo NimPile) e calcola la mossa del computer sulla base della modalità in cui sta operando, dopodiché mette in atto la mossa (invocando il metodo take del mucchio)

Classe NimComputerPlayer - Definizione costruttore

```
class NimComputerPlayer :

  def __init__(self, isExpert):
    self._isExpert = isExpert

  ## Restituisce se gioco in
modalità intelligente
  # @return modalità intelligente
  def isExpert(self) :
    # codice metodo

  ## Esegue la mossa
  # @param pile Mucchio di biglie
  def move(self, pile) :
    # codice metodo
```

Classe **NimComputerPlayer** rappresenta il computer che gioca. Deve avere:

- **il costruttore che riceve un valore booleano, True se e solo se il computer deve giocare in modo intelligente**
- il metodo isExpert che restituisce True se e solo se il computer sta giocando in modo intelligente
- il metodo move che riceve il mucchio di biglie (cioè un oggetto di tipo NimPile) e calcola la mossa del computer sulla base della modalità in cui sta operando, dopodiché mette in atto la mossa (invocando il metodo take del mucchio)

Classe NimComputerPlayer - Definizione metodi

```
class NimComputerPlayer :

  def __init__(self, isExpert):
    self._isExpert = isExpert

  ## Restituisce se gioco in modalità intelligente
  # @return modalità intelligente
  def isExpert(self) :
    return self._isExpert

  ## Esegue la mossa
  # @param pile Mucchio di biglie
  def move(self, pile) :
    # codice metodo
```

```
class NimComputerPlayer :

  def __init__(self, isExpert):
    self._isExpert = isExpert

  def isExpert(self) :
    return self._isExpert

  def move(self, pile) :
    n = pile.getTotal()
    if self._isExpert :
      if n == 3 or n == 7 or n == 15 or n == 31 or n == 63 :
        taken = randint(1, n // 2)
      elif n > 63 :
        taken = n - 63
      elif n > 31 :
        taken = n - 31
      elif n > 15 :
        taken = n - 15
      elif n > 7 :
        taken = n - 7
      elif n > 3 :
        taken = n - 3
      else :
        taken = 1
    else :
      taken = randint(1, n // 2)
    print("Biglie nel mucchio:", n, ". Il computer ne ha prese:", taken)
    pile.take(taken)
```

Nel modo intelligente, se il numero delle biglie è una potenza di due diminuita di un'unità (1, 3, 7, 15, 31 o 63), il computer preleva una quantità di biglie casuale.

Altrimenti il computer preleva il numero di biglie sufficiente affinché il numero delle rimanenti sia una potenza di due diminuita di un'unità (1, 3, 7, 15, 31 o 63).

Nel modo stupido, quando è il suo turno, il computer sottrae un numero casuale di biglie.

4) Classe NimGame

Classe **NimGame** rappresenta un'intera partita.

Deve avere:

- il costruttore che non riceve parametri
- il metodo **play** che gioca un'intera partita e visualizza il vincitore; se il vincitore è il computer, visualizza anche la modalità di gioco (*intelligente* o *stupido*)

Classe NimGame - Definizione interfaccia pubblica

```
class NimGame :

  ## Gioca partita
  def play(self) :
    # codice metodo
```

Classe **NimGame** rappresenta un'intera partita. Deve avere:

- il costruttore che non riceve parametri
- il **metodo play** che **gioca un'intera partita e visualizza il vincitore**; se il vincitore è il computer, visualizza anche la modalità di gioco (intelligente o stupido)

Classe NimGame - Definizione stato dell'oggetto

```
class NimGame :

  ## Gioca partita
  def play(self) :
    # codice metodo
```

Classe **NimGame** rappresenta un'intera partita. Deve avere:

- il costruttore che non riceve parametri
- il **metodo play** che **gioca un'intera partita e visualizza il vincitore**; se il vincitore è il computer, visualizza anche la modalità di gioco (intelligente o stupido)

Che cosa caratterizza una partita di Nim?

- Mucchio di biglie
- Giocatore umano
- Giocatore computer
- Turno di gioco

Classe NimGame – Definizione costruttore

```
class NimGame :

  def __init__(self) :
    self._pile = NimPile(randint(10, 100))
    self._humanPlayer = NimHumanPlayer()
    self._computerPlayer = NimComputerPlay
    if random() < 0.5 :
      self._nextPlayer = self._humanPlayer
    else :
      self._nextPlayer = self._computerPlayer

  ## Gioca partita
  def play(self) :
    # codice metodo
```

Classe **NimGame** rappresenta un'intera partita. Deve avere:

- **il costruttore che non riceve parametri**
- il metodo play che gioca un'intera partita e visualizza il vincitore; se il vincitore è il computer, visualizza anche la modalità di gioco (intelligente o stupido)

Classe NimGame – Definizione costruttore

```
class NimGame :

  def __init__(self) :
    self._pile = NimPile(randint(10, 100))
    self._humanPlayer = NimHumanPlayer()
    self._computerPlayer = NimComputerPlayer(random() < 0.5)
    if random() < 0.5 :
      self._nextPlayer = self._humanPlayer
    else :
      self._nextPlayer = self._computerPlayer

  ## Gioca partita
  def play(self) :
    # codice metodo
```

Che cosa caratterizza una partita di Nim?

- Mucchio di biglie
- Giocatore umano
- Giocatore computer
- Turno di gioco

Classe NimGame – Definizione metodi

```
class NimGame :

  def __init__(self) :
    self._pile = NimPile(randint(10, 100))
    self._humanPlayer = NimHumanPlayer()
    self._computerPlayer = NimComputerPlayer(random() < 0.5)
    if random() < 0.5 :
      self._nextPlayer = self._humanPlayer
    else :
      self._nextPlayer = self._computerPlayer

  def play(self) :
    while self._pile.getTotal() > 1 :
      self._nextPlayer.move(self._pile)
      if self._nextPlayer == self._computerPlayer :
         self._nextPlayer = self._humanPlayer
      else :
         self._nextPlayer = self._computerPlayer
    if self._nextPlayer == self._computerPlayer :
      print("Hai vinto!")
    else :
      print("Il computer ha vinto: era ", end="")
      if self._computerPlayer.isExpert() :
        print("intelligente")
      else :
        print("stupido")
```