Programmare giochi

"A mio fratello, che da anni cerca di convincermi a scrivergli un gioco online completo, cosi' puo' leggersi il libro e imparare a scriverselo da solo”

Programmare giochi

Indice generale

Introduzione

Questo libro vuole essere una introduzione alla programmazione di giochi su sistemi operativi linux o windows. La scelta di questi sistemi e' dovuta alla loro ampia diffusione, e alla disponibilita' di librerie e esempi, ma si cerchera' di mantenere una trattazione il piu' possibile indipendente dalla piattaforma, puntando il piu' possibile l' accento sulle metodologie di programmazione e sui concetti piuttosto che sul semplice sviluppo di codice.
Da quando il calcolatore e' diventato una macchina che comunica in maniera interattiva, con l' introduzione di una console composta solitamente da monitor e tastiera, i programmatori hanno iniziato a sviluppare, oltre ai programmi applicativi, giochi.
Per un programmatore sviluppare un gioco e' solitamente una sfida. Infatti, mentre un programma di calcolo richiede solitamente la descrizione di un algoritmo ben definito, che si sviluppa con un inserimento di dati iniziale, una elaborazione e in seguito la presentazione di una risposta, la programmazione di un gioco richiede lo sviluppo di schemi piu' complessi, considerando la natura interattiva che solitamente li caratterizza. In un gioco all' utente non e' solitamente richiesto un solo inserimento dati, ma la risposta, in tempi ragionevoli, a situazioni presentate dal calcolatore.
A seconda del tipo di gioco, al calcolatore possono essere richieste diverse funzioni che non sono normalmente richieste per programmi di calcolo, come ad esempio la possibilita' di mostrare oggetti grafici, spesso in movimento, di gestire suoni, di interagire con periferiche diverse dalla tastiera, di decidere una azione da compiere secondo il comportamento dell' utente, e

tutto in una situazione in cui i tempi di risposta sono critici.
Il gioco, su un calcolatore, richiede solitamente una quantita' di risorse molto piu' elevata dell' esecuzione di altri tipi di programma.
A partire dagli anni '70 il calcolatore ha visto una massiccia evoluzione come macchina di uso comune, specialmente dovuta al fatto che il costo dei suoi componenti fosse divenuto cosi' basso da permetterne l' acquisto ad una normale famiglia.
Questo malgrado il fatto che una normale famiglia non avesse in realta' bisogno di una macchina di calcolo cosi' complessa.
La molla che spinse le famiglie, e non solo gli hobbisti del settore, all' acquisto di questa nuova macchina fu allora proprio la presenza dei giochi.
I primi calcolatori domestici vennero presentati come "consoles", cioe' come macchine di utilizzo estremamente semplice. La macchina era semplicemente una scatola, con un cavo che poteva essere collegato ad un televisore e una o due "paddles". La paddle era un sistema molto semplice per interagire con il calcolatore: era composta solitamente da una piccola scatola contenente un pulsante e una manopola. Premendo il pulsante o ruotando la manopola a destra o a sinistra si poteva interagire con il calcolatore e giocare la partita.
In figura viene mostrato l' "atari pong", una delle prime console interattive. In questo caso il calcolatore era estremamente semplice, in grado solo di giocare a ping-pong, e le racchette del ping pong erano pilotate da due manopole inserite direttamente sulla scatola dell' apparato.
C'e' da dire che non si tratta della prima console: Il magnavox odissey fu realizzato e posto sul mercato prima dell' atari pong, ma era poco conosciuto e l'atari ebbe una maggiore distribuzione.

Questo perche' venne venduto prima nelle sale giochi, che al momento avevano solo flippers, e poi ne fu realizzata una versione per l'uso domestico.

In realta' gia' questa appicazione era complessa per i calcolatori dell' epoca. Il disegno delle racchette e della pallina doveva infatti essere presentato sullo schemo di un normale televisore, e questo richiedeva la capacita' di generare un segnale video composito.
Il segnale video doveva essere generato continuamente per poter presentare schermate intervallate solamente di un venticinquesimo di secondo l' una dall' altra. Mentre il microprocessore era impegnato a svolgere questo compito, lo stesso doveva essere in grado anche di gestire l' utente, muovendo la sua racchetta quando la manopola veniva ruotata, e

di gestire il movimento e i rimbalzi della pallina.
Una seconda generazione di calcolatori furono gli "home computers". Queste macchine avevano, oltre alle paddles o ai "joystick", usati per i giochi, anche una tastiera, che consentiva finalmente di parlare con la macchina, la quale solitamente rispondeva ad istruzioni in linguaggio basic. Questo perche' il basic era un linguaggio estremamente semplice, e un interprete basic poteva essere inserito in pochi kilobytes di memoria. Cosa fondamentale per macchine il cui costo doveva essere bassissimo.
Molto interessante fu la famiglia delle macchine presentate dalla "commodore". Dal Vic20 agli Amiga.
Fondamentalmente si trattava di calcolatori con un processore molto lento anche per l' epoca, (vic20 e commodore 64 montavano un processore 6502, la cui velocita' era di circa un milione di istruzioni al secondo, ma in cui ogni istruzione doveva essere ancora scomposta in un insieme di molti passi elementari).
Era sufficiente per programmi di calcolo ma non per l' elaborazione di giochi e per la gestione dello schermo.
Per poter eseguire giochi, al microprocessore erano affiancati una serie di processori di supporto, come il SID, che era un integrato che lavorava come sintetizzatore audio, e il VIC, che si occupava di tutta la gestione della grafica. Il microprocessore, per presentare un disegno sullo schemo o per far sentire un suono, si limitava a collegarsi al VIC o al SID, a descrivere la richiesta, ed era l'integrato di supporto ad occuparsi effettivamente di mostrare il disegno o generare il suono. Il processore restava quindi fondamentalmente "libero" di occuparsi delle regole del gioco e dell' interazione con l'utente.
Fino agli anni '90, per i giochi, le macchine della commodore e gli home computers furono nettamente superiori ai PC che

usiamo oggi, i quali non offrivano ne' un processore grafico in grado di svolgere le funzioni del VIC ne' una scheda audio, ma offrivano il solo vantaggio di avere una velocita' di calcolo nettamente superiore.
Anche oggi l' acquisto di una macchina da utilizzare per eseguire giochi richiede una spesa maggiore rispetto all' acquisto di una macchina da utilizzare come calcolatore di ufficio. Questo perche' anche se le capacita' delle macchine che stiamo utilizzando sono aumentate enormemente (Oggi stiamo usando macchine in grado di lavorare con una velocita' di circa tre gigahertz, cioe' di circa tre miliardi di istruzioni parallele per secondo, ma per ogni ciclo di clock in parallelo possono essere eseguite in parallelo fino a quattro operazioni), le richieste di potenza dei giochi sono aumentate allo stesso modo.
Mentre una volta un gioco aveva poche animazioni e tutto era lasciato alla fantasia dell' utente, oggi un gioco presenta animazioni con qualita' equivalente a quelle dei films, personaggi definiti in tre dimensioni, permette l' interativita' con altri partecipanti, e in molti casi diventa uno strumenti di comunicazione utilizzando internet.
Se ci proproniamo di costruire un gioco che debba essere venduto oggi sul mercato, dobbiamo tenere presente che potremmo stare pianificando un compito estremamente complesso. Oggi le grandi compagnie che producono giochi sono spesso le stesse che producono films, e il cast di un gioco inizia a somigliare sempre di piu' al cast di un film, includendo nella maggior parte dei casi decine di persone, cioe' programmatori, grafici, doppiatori, progettisti per gi ambienti, consulenti per le musiche e cosi' via.
Avremo comunque il vantaggio di poter disporre di una piattaforma su cui non avremo i problemi che i programmatori di giochi avevano anni fa, cioe' quelli di dover limitare la

dimensione del gioco a causa dei limiti di memoria della macchina e di dover ottimizzare gli algoritmi in modo di ottenere la maggiore velocita' possibile.
Se il nostro scopo e' quello di imparare a programmare con una sfida interessante e risultati ben visibili, lo sviluppo di giochi puo' essere il campo giusto: sia progettare che scrivere giochi richiede una buona capacita', spesso molto maggiore di quella richiesta per la realizzazione di altri programmi, inoltre il risultato della programmazione di un gioco e' facilmente visibile a schermo, e quindi solitamente non c'e' necessita' di utilizzare complessi strumenti di debug per valutare la qualita' del risultato del nostro lavoro.
Oggi, comunque, lo sviluppo di giochi anche semplici per calcolatore sta ritornando ad essere una attivita' interessante, in quanto l'uso di internet e la cresciuta interattivita' delle pagine web stanno rendendo popolari molti piccoli giochi realizzati per girare da un server di internet, senza alcuna installazione, solitamente scritti in java. E in molti casi si tratta di giochi anche molto semplici, senza alcuna pretesa di avere una grafica accattivante o di presentarsi con musiche di alto livello, ma dotati di una buona giocabilita, e soprattutto con un aspetto simpatico o "gag" divertenti, in modo che il loro uso si espanda per passaparola da un utente agli altri.

Tipi di giochi

Iniziamo ora con un breve studio su alcuni tipi di giochi disponibili prima sugli home computers e ora sui computers moderni, e facciamo alcune considerazioni per stabilire quali siano gli strumenti che ci servano per realizzare un gioco dello stesso genere. E' davvero impossibile elencare tutte le tipologie di giochi realizzate per il calcolatore. Per fare un esempio wikipedia alla voce "videogiochi" ne elenca al momento circa quaranta diversi tipi, alcuni suddivisi in sottocategorie. Per cui prenderemo come esempio alcuni giochi classici per stabilire cosa ci serva per la loro realizzazione.

"Shoot-em all"

Gli "Shootemall", letteralmente "spara a tutti", come ad esempio "space invaders", erano molto comuni sui primi home computers. Il meccanismo di gioco e' piuttosto semplice: il giocatore e' rappresentato da una navetta che si trova solitamente nella parte inferiore dello schermo. i nemici sono un insieme di navette che riempiono l' altra meta' dello stesso schermo e scendono lentamente, e per vincere la mano di gioco il giocatore deve poter abbattere tutti gli alieni che si trovano sullo schermo prima che questi arrivino ad un certo livello di altezza (e atterrino), eventualmente evitando colpi sparati dagli alieni stessi. Nella versione originale di space invaders delle barriere proteggevano la navetta, e le barriere potevano essere danneggiate dai colpi degli alieni.

Vediamo che cosa richiede la realizzazione di un gioco di questo tipo:
Per prima cosa e' necessario poter disegnare sullo schermo gli alieni e la navetta, ed e' necessario poterli spostare.
Quests sembra una operazione semplice, e lo era su una macchina come il commodore 64, ma su un PC comporta delle complicazioni. Infatti il VIC del commodore 64 offriva come astrazione gli sprites (letteralmente folletti).
Uno sprite era un disegno di dimensione fissa che veniva controllato direttamente dal VIC, e poteva essere spostato sullo schermo del computer senza "sporcarne" il contenuto, cioe senza cambiare il disegno che si trovava sotto. Uno sprite poteva essere spento o acceso, per farlo sparire dallo schermo, ed era possibile rilevare la collisione tra sprites e sfondo e tra sprite e sprite. Questo significa che era possibile chiedere al VIC se uno sprite stesse toccando un altro sprite o lo sfondo, e agire di conseguenza.
Ottenere gli sprite su un PC sara' un po' piu' complesso, dato che il PC ha uno schermo composto da un unica griglia di punti, e il nostro unico modo di disegnare qualcosa sullo schermo sara' di sovrascrivere questi punti. In pratica ogni volta che disegneremo un oggetto sullo schermo saremo costretti a cancellate tutto quello che c'e' sotto. Quindi, per poter ripristinare lo sfondo quando sposteremo l'oggetto saremo costretti a memorizzare da qualche parte quello che c'era sotto prima della cancellazione. Dovremo inolte sempre sapere se il nostro sprite al momento e' visibile o meno,e dove si trova.
Molti linguaggi, come ad esempio visual basic, danno a disposizione dell'utente degli oggetti, come le immagini, che sono in realta' degli sprites, ma sono solitamente piuttosto lenti, e quindi un gioco realizzato in questo modo presenterebbe dei problemi a girare. Se gli sprites sono troppo lenti, quando uno di

questi si sposta, diventa percettibile l' intervallo tra la cancellazione e il successivo ridisegno, e quindi il movimento non risulta fluido. L' effetto e' che lo spostamento sia causato da una miriade di cancellazioni e ricomparse dello stesso oggetto, e viene comunemente chiamato "flickering", cioe' sfarfallio. Dovremo quindi realizzare i nostri sprites nella maniera migliore possibile, in modo da poter evitare il piu' possibile il problema del flickering.
Come seconda cosa e' necessario poter comunicare con l'utente, che deve essere libero di spostare la propria navetta in ogni momento (o almeno averne l'impressione), e di sparare colpi agli alieni.
Anche questa cosa richiede un po' di lavoro: e' necessario infatti poter accedere direttamente al driver di tastiera per chiedere se un tasto e' stato premuto, e in questo caso leggere il tasto, mentre le funzioni standard per avere informazioni dall'utente richiedono che il programma si fermi finche' l'utente non abbia digitato un intera riga terminata da un caporiga, per poi proseguire ad elaborare quello che l'utente ha scritto.
Ancora, se vogliamo che il gioco abbia un audio dovremo poter comunicare con la scheda audio, passsare a questa dei suoni e suonarli quando necessario, ad esempio quando uno degli alieni viene colpito.
Quindi, tanto per cominciare avremo bisogno degli sprites, di un gestore di tastiera e di un player audio. Dopodiche' potremo iniziare a descrivere l' algoritmo di gioco.
Logicamente, in giochi di questo tipo, ci sono vari agenti, ciascuno dei quali dovra' svolgere delle azioni indpiendentemente dagli altri.
La navetta dell' utente dovra' controllare lo stato della tastiera e muoversi si conseguenza. Se viene premuto sparo dovra' generare dei missili.

Gli alieni dovranno muoversi, orizzontalmente o verticalmente, per avvicinarsi gradualmente alla navetta dell'utente.
I missili dovranno muoversi gradualmente verso l' alto, e se toccano un alieno dovranno cancellarlo.
Qualcosa dovra' controllare se gli alieni sono troppo in basso,e quindi hanno vinto, o se gli alieni sono stati tutti cancellati, e quindi e' il giocatore ad aver vinto il livello di gioco.
In realta', una volta descritti gli oggetti, tutte queste operazioni sono piuttosto semplici. Quindi tutti gli agenti possono essere simulati da un unico programma in sequenza, in questo modo:

```
- Inizia qui:
    {
    - sposta gli alieni
    - controlla se e' stato premuto un tasto
    - se un tasto e' stato premuto:
        {
        - se e' una freccia sposta la navetta dell'utente
a destra o a sinistra
        - se e' il tasto di sparo metti un missile sullo
schermo
        }
    - se ci sono missili sullo schermo
      {
      - per ogni missile:
          {
          - sposta il missile in alto
          - controlla se il missile tocca un alieno, in
questo caso cancella l'alieno e il missile
          }
      }
    }
- ripeti dall' inizio finche' ci sono alieni e finche'
gli alieni sono abbastanza in alto
- Se gli alieni sono arrivati in fondo giocatore ha perso
- Se gli alieni sono finiti il giocatore ha vinto
```

Per chi non ha mai programmato un computer potrebbe essere difficile capire cosa e' scritto qui sopra, e quindi come dovrebbe funzionare il gioco. Per poter leggere correttamente quanto

sopra occorre considerare alcune cose:
Un computer e' in grado di eseguire un algoritmo, cioe' una sequenza di comandi. Ad ogni trattino "-" corrisponde quindi un comando, e i comandi saranno eseguiti uno alla volta, in sequenza. Il computer iniziera' quindi dal comando "inizia qui", e si ricordera' quale e' il punto di inizio.
Alcune parti del programma possono essere ripetute piu' volte, o eseguite solo sotto certe condizioni. Queste parti sono raggruppata tra parentesi graffe. Ad esempio le righe:

```
    - se un tasto e' stato premuto:
        {
        - se e' una freccia sposta la navetta dell'utente
a destra o a sinistra
        - se e' il tasto di sparo metti un missile sullo
schermo
        }
```

significano che se un tasto e' stato premuto verranno valutate le due ipotesi, che il tasto sia una freccia o lo sparo, mentre se il tasto non e' stato premuto nessuna delle righe tra le parentesi graffe verra' eseguita.
Quindi la riga "ripeti dall' inizio" ripetera', se le condizioni sono verificate, tutto quanto si trova tra la parentesi graffa che segue la dicitura "inizia qui" e la parentesi graffa che si trova sulla riga precedente al "ripeti dall' inizio".
E' da notare che e' stata utilizzata l' identazione per rendere piu' leggibile l' algoritmo: ogni volta che una parentesi graffa e' stata aperta le linee successive iniziano alcuni caratteri piu' avanti, finche' la parentesi graffa non viene nuovamente chiusa. Questo piccolo trucco nella scrittura del testo permette subito di capire, trovata una parentesi aperta, dove si trova la partentesi chiusa corrispondente, e quindi migliora la leggibilita'.

L' importante perche' il gioco funzioni correttamente e' che la sequenza descritta nell' algoritmo sia ripetuta abbastanza velocemente, idealmente alcune volte al secondo, in modo da non permettere al giocatore di notare un movimento a scatti.

Per realizzare un gioco "shoot-em all", si procede solitamente scrivendo delle librerie che descrivano gli oggetti che intervengono nel gioco, e poi descrivendo un "ciclo principale" del gioco, che viene ripetuto fino a termine partita, e che per ogni oggetto del gioco richiama delle funzioni che ne aggiornino lo stato.

I "Platforms"

I platforms sono giochi apparsi attorno ai primi anni ottanta. Lo scopo del gioco e' solitamente quello di percorrere uno "Schema", e su questo schema di raccogliere oggetti ed evitare ostacoli. Lo schema di un platform e' solitamente piuttosto grande, molto piu' grande dello schermo del computer. Mentre il personaggio del gioco si sposta lo schermo scorre sotto i suoi piedi, presentando sempre nuovi ostacoli ed oggetti.
Lo scopo del gioco e' solitamente percorrere l' intero schema fino ad una porta di uscita ed aumentare il proprio punteggio raccogliendo oggetti durante il percorso. Solitamente il personaggio puo' saltare da un piano all'altro dello schermo ed aiutarsi con piattaforme che funzionano da ascensori.
Lo schermo di un platform presenta delle particolarita' diverse da quelle di uno shoot-em all, che rendono questa tipologia di giochi piu' complessa da realizzare.
Per prima cosa lo schermo di un platform deve poter scorrere al movimento del personaggio. Questo significa che in memoria e'

necessario mantenere una mappa dell' intero schermo in modo da poterla utilizzare per disegnare le parti dello schermo mano a mano che queste diventano visibili. La mappa potrebbe essere semplicemente una immagine molto grande, visualizzata solo in parte, o una griglia divisa in un numero limitato di quadrati, ciascuno dei quali specifichera' cosa disegnare su un certo punto dello schermo.
Poi la schermata del platform presentera' un insieme di oggetti animati che si muoveranno indipendentemente l'uno dall' altro: piattaforme, ostacoli, mostri, che dobbiamo riuscire a gestire nel tempo limitato del ciclo di gioco, cercando di fare in modo che il ciclo stesso non diventi troppo lento mostrando movimenti a scatti. Questo richiede, oltre ad una libreria per gli sprites molto veloce, un buon progetto sia degli oggetti che del meccanismo con cui ciascuno di essi aggiorna la sua posizione, in modo da ridurre il tempo perso.
Inoltre lo scrolling dello schermo e' una operazione molto pesante, in quanto corisponde a ridisegnare l'intera schermata grafica del gioco. Anche utilizzando gli sprites questa operazione e' lenta, e quindi va fatta solo quando diventa indispensabile, cioe' quando il giocatore e' molto vicino al bordo dello schermo e sta ancora procedendo nella direzione che lo porterebbe ad uscirne. Su macchine veloci come il commodore 64 lo scrolling era un problema cosi' critico che il processore video VIC offriva un meccanismo hardware (cioe' realizzato con elettronica) per aiutare a realizzarlo spostando l'intero disegno della pagina.

Lo "schermo" e' un oggetto che negli shoot-em up non appariva: sotto gli sprite il disegno era fisso e non interagiva in alcun modo con il giocatore. In questo caso sono i movimenti del giocatore a cambiare il contenuto dello schermo, e gli oggetti

disegnati sullo schermo vincolano i movimenti del giocatore stesso. La mappa dello schermo dovra' contenere informazioni sulla presenza di piattaforme, di ostacoli in movimento o di tesori.
Lo schermo diventa quindi un oggetto, che e' composto da una mappa e da un meccanismo di scrolling.
La mappa potrebbe essere una matrice di caratteri, ad esempio qualcosa fatto in questo modo:

```
..............
.......X......
....WGGW....D.
....DDD.....TT
....WWWW..G.TT
............TT
...M.M...P..TT
...WWWWWWW..TT
```

che potrebbe descrivere la parte di schermo visibile del gioco in figura (The great giana sisters, un clone di Mario Bros, per commodore e amiga):

Quindi, W potrebbe stare per "wall" (muro), X per la posizione di partenza del giocatore, G per gift (battendo su questo mattoncino con un salto questo rilascia un premio), D per diamante, M per mostro e P per pianta.
Il giocatore puo' procedere nel gioco spostandosi a destra. La mappa dovrebbe quindi continuare sulla destra, in questo modo:

```
.........................
.......X.............D.....
....WGGW....D.......GWW....
....DDD.....TT..D..........
....WWWW..G.TTWWGW.........
............TT.............
...M.M...P..TT...M.......P
...WWWWWWW..TT...WWWW WWWW
```

(..e cosi' via..)

Lo schermo quindi dovra' mantenere informazioni sulla posizione del giocatore, e quando il gocatore e' troppo vicino al bordo della parte visibile dovra' ridisegnarsi, utilizzando una

parte diversa della mappa.
Una diversa griglia corrisponde ad una diversa difficolta' del gioco ed ad una differente presenza di ostacoli. Un platform e' facile da estendere per ottenere nuovi livelli semplicemente cambiando i contenuto della griglia.
Dovremmo descrivere il concetto di piattaforma, di premio, di mostro.
Una piattaforma sara' un oggetto su cui il personaggio puo' appoggiarsi. Ad ogni passo del ciclo di gioco lo sprite corrispondente al personaggio potrebbe toccare il suolo oppure no.
Se il suolo non viene toccato la posizione dello sprite dovra' essere abbassata (il personaggio sta cadendo o atterrando da un salto).
Se la posizione del personaggio diventa cosi' bassa da uscire dallo schermo questo e' caduto e non piu' recuperabile, e il giocatore ha perso. Devono essere descritte anche piattaforme mobili: sono piattaforme la cui posizione cambia continuamente in modo che il personaggio possa usarle come un ascensore o come un tappeto mobile. In questo caso se la piattaforma si alza e sta toccando il personaggio, anche il personaggio deve alzarsi della stessa misura.
Un premio sara' un oggetto immobile: quando un giocatore lo tocca il premio scompare ma al giocatore viene assegnato un punteggio, che si aggiunge ad un totale.
Un mostro sara' un oggetto animato. Un mostro puo' spostarsi a destra o a sinistra. Se il mostro si accorgera' che sotto di se non ha piu' suolo invertira' la sua direzione in modo da restare sempre sul terreno. Se c'e' un contatto tra il mostro ed il personaggio il giocatore ha perso.
Il ciclo principale, su cui torneremo nel capitolo riguardante la programmazione di giochi vera e propria, dovra' semplicemente

fare un passo di simulazione di tutti questi oggetti, facendo in modo che ciascuno di questi aggiorni il proprio stato.

```
- inizio:
- fissa la posizione dello schermo sulla mappa a zero.
- scorri la mappa e crea gli oggetti che sono descritti
nella griglia.
- visualizza gli oggetti
- Ripeti da qui:
  {
  - Aggiorna il giocatore leggendo la tastiera
  - Update the player reading the keyboard
  - Se il giocatore non tocca una piattaforma abbassa la
sua posizione.
  - Per ogni piattaforma mobile
    {
    - Aggiorna la posizione della piattaforma mobile.
Questo aggiornera' anche la posizione del giocatore se e'
sulla piattaforma.
    }
  - Per ogni mostro:
    {
    - aggiorna il mostro
    - se il giocatore tocca il mostro il giocatore e'
morto
    }
   - Se il giocatore tocca un premio aumentare il
punteggio e cancellare il premio
  }
- Finche' il giocatore non e' fuori schermo o morto, o
tocca la porta di uscita.
```

Questo e' un esempio di platform, ma questo schema di gioco basato su una griglia che descrive un "mondo" su cui il giocatore si puo' muove e' molto comune, e viene usato per molti altri giochi, fino ai moderni giochi in rete come ad esempio ultima on-line. Anche in questo caso ritorneremo piu' in dettaglio sul ciclo di gioco di un platform nei capitoli successivi per realizzare un esempio.

Giochi di tattica e meccanismi di intelligenza artificiale

Una tipologia di giochi particolare rispetto a quelle considerate in precedenza e' quella dei giochi che richiedono meccanismi di intelligenza artificiale. Ad esempio in una partita a dama contro il calcolatore e' necessario che il calcolatore sia in grado di stabilire la sua prossima mossa in maniera da migliorare la situazione complessiva delle sue pedine in gioco a scapito di quelle dell' avversario per vincere la partita.
Questa operazione e' piuttosto complessa e per giochi in cui le regole siano complesse e le possibilta' di movimento molte, come gli scacchi, puo' richiedere tempi di elaborazione molto lunghi e molte risorse. Un esempio di questo e' "deep blue", il supercalcolatore che divenne famoso come la macchina che fu in grado di battere nel 1996 il campione mondiale di scacchi Gary Kasparov. Anche se Kasparov, dopo un certo tempo, batte' nuovamente la macchina, il risultato era comunque spettacolare, ma per il calcolo di una singola mossa di deep-blue veniva utilizzata l' intera potenza di un calcolatore con trenta microprocessori RISC a centoventi megahertz, e con hardware specializzato per rendere piu' veloce la computazione, dedicato al solo gioco degli scacchi.
Anche dovendo realizzare giochi classici, con personaggi animati, puo' essere necessario muovere gli avversari in maniera apparentemente intelligente, e per farlo si possono utilizzare le stesse tecniche che vengono utilizzate per stabilire, passo per passo, la mossa successiva di una partita a scacchi o a dama.
La decisione di una mossa in una partita da parte del calcolatore e' basata su due concetti: quello di euristica e quello di regola. Una euristica e' una funzione matematica che il calcolatore usa per valutare la bonta' o meno della propria posizione di gioco. Il

risultato della funzione euristica sara' piu' alto migliore sia la posizione di gioco del giocatore. Ad esempio, un euristica classica utilizzata nel gioco degli scacchi consiste nel dare un punteggio positivo a tutti i propri pezzi, dipendente dal tipo di pezzo, e un punteggio negativo agli stessi pezzi dell'avversario. Qualcosa del tipo "Se la mia regina e' in gioco guadagno dieci punti, mentre ne perdo dieci se e' in gioco quella dell' avversario". In questo caso il calcolatore cerchera' le mosse che gli consentono di aumentare il piu' possibile l'euristica, quindi cerchera' di mantenere in gioco la propria regina e di mangiare quella avversaria.
Cambiando i punteggi del singolo pezzo sulla funzione euristica il calcolatore scegliera' mosse differenti. Se ad esempio dessi ad un alfiere un valore minore rispetto ad una torre il calcolatore preferirebbe, nella stessa situazione, mangiare la torre avversaria piuttosto che l' alfiere.
Mentre per giochi semplici la funzione euristica puo' essere uno strumento sufficiente per un pezzo per decidere la propria prossima mossa, per giochi complessi come gli scacchi questo non puo' essere sufficiente: se un giocatore di scacchi decidesse la propria prossima mossa solamente guardando l' attuale posizione della scacchiera e senza pensare ad una tattica che lo porti a sferrare un attacco non potrebbe mai vincere la partita.
E' necessario poter non solo decidere una mossa, ma pianificare la sequenza di mosse che permetta il maggiore aumento dell'euristica. Se il giocatore avversario non fa, durante questa sequenza, quanto ci aspettiamo, e' necessario ripetere il calcolo e ricavare una nuova tattica.
Le regole del gioco, oltre ad essere un metodo per fare in modo che il calcolatore muova i propri pezzi in maniera congrua e che possa controllare che anche l'utente lo faccia, sono anche un metodo per poter definire una tattica durante la partita.

Per fare un esempio proviamo a descrivere al calcolatore le regole della dama, semplicemente perche' queste sono piu' semplici di quelle degli scacchi. Il computer descrivera' la posizione di un pezzo secondo due coordinate, X e Y:

- Per ogni pezzo del mio colore:

(i pezzi della dama si muovono diagonalmente, sulle caselle non occupate da altri pezzi)

- I pezzi possono avere una X compresa tra 1 e 8 e una Y compresa tra 1 e 8.

- Se la casella davanti a sinistra (X-1, Y-1) e' libera, e' regola valida che il pezzo proceda di un passo in avanti a sinistra

- Se la casella davanti a destra e' libera, e' regola valida che il pezzo proceda di un passo in avanti a destra

- Se il pezzo e' una dama e la casella dietro a sinistra e' libera, e' regola valida che il pezzo proceda di un passo indietro a sinistra

- Se il pezzo e' una dama e la casella dietro a destra e' libera, e' regola valida che il pezzo proceda di un passo indietro a destra

(Nella dama italiana classicamente, una dama non puo' essere mangiata da una pedina.. questo complica un poco le regole)

- Se la casella avanti a sinistra contiene una pedina avversaria e' regola valida cancellare la pedina avversaria e spostarsi di due passi avanti a sinistra

- Se la casella avanti a destra contiene una pedina avversaria e' regola valida cancellare la pedina avversaria e spostarsi di due passi avanti a destra

- Se sono una dama e la casella indietro a sinistra contiene una dama o una pedina avversaria e' regola valida cancellarla e spostarsi di due passi indietro a sinistra.

- Se sono una dama e la casella indietro a destra contiene una dama o una pedina avversaria e' regola valida cancellarla e spostarsi di due passi indietro a destra.

- Se un pezzo ha una Y di 1 (E' arrivato all'altro capo della scacchiera) e il pezzo e' una pedina, questa diventa una dama.

- Se non esistono mosse valide la partita e' persa.

Le regole descritte in questo modo, e ovviamente tradotte in un linguaggio di programmazione, descrivono tutti i possibili movimenti dei pezzi della dama. E' necessario descrivere al calcolatore lo scopo del gioco. Questo viene fatto con la funzione euristica.
Per fare una euristica per la dama potremmo ragionare in questo modo:
- La funzione potrebbe assegnare un punteggio positivo alla presenza di ogni mia pedina sulla scacchiera, ad esempio una pedina potrebbe valere cinque punti, e un punteggio negativo alla presenza di ogni pedina avversaria, che potrebbe togliere cinque punti.
- La funzione potrebbe ancora assegnare un punteggio positivo alla presenza di ogni mia dama, ad esempio di trenta punti, e un punteggio negativo dello stesso valore per la presenza di ogni dama avversaria.
- Ogni pedina potrebbe avere un punteggio aggiuntivo legato alla sua distanza dal bordo della scacchiera su cui puo' diventare dama, ad esempio zero se la distanza e' di sette passi, uno se la distanza e' di sei passi, e cosi' via. Questo per spiegare al calcolatore che e' meglio che le pedine tentino di diventare

dame. Le pedine avversarie potrebbero introdurre lo stesso punteggio negativo.
La scacchiera puo' essere definita come una matrice di otto per otto numeri, e su ciascuna posizione potrebbe esserci ad esempio +1 se c'e' una mia pedina, -1 se c'e' una pedina avversaria, +2 se c'e' una mia dama e -2 se c'e' una dama avversaria, 0 se la casella e' libera. L' euristica e' allora una funzione a cui puo' essere passata come parametro una scacchiera e ne viene ritornata una valutazione.
L' euristica assieme alle regole sono tutto quanto serve per spiegare al calcolatore un gioco di strategia. A questo punto e' possibile definire il ciclo principale del gioco. Se i giochi sono abbastanza semplici e' possibile per il calcolatore stabilire la propria prossima mossa per "forza bruta", cioe' per enumerazione tra tutte le possibili mosse:

```
Ciclo principale di gioco:
- inizio
- pongo la scacchiera nella posizione iniziale
- ripeti da qui:
  {
  - attendo la mossa del giocatore
  - valuto e faccio la mia mossa
  - valuto se uno dei due ha vinto la partita
  }
- se nessuno dei due ha vinto ripeti.
```

E per valutare la mia mossa posso procedere a questo modo:

```
- voglio memorizzare la migliore euristica e la mossa
corrispondente.
- per ogni mio pezzo sulla scacchiera:
  {
  - per ogni mossa valida per il pezzo:
    {
    - genero una nuova scacchiera modificata secondo la
```

```
mossa
    - per ogni pezzo dell' avversario sulla scacchiera
modificata:
      {
      - per ogni mossa valida per il pezzo:
        {
          - genero una nuova scacchiera modificate
secondo la mossa
          - calcolo l' euristica
          - se l' euristica e la migliore incontrata fino
ad ora
            {
             - memorizzo l' euristica e le due mosse
corrispondenti
            }
        }
      }
    }
  }
- eseguo la mossa memorizzata
```

La mossa che verra' eseguita e' quella che dopo una mossa mia e una dell' avversario risulta dare il risultato migliore sulla scacchiera. Per ogni mossa che posso fare valuto anche la contromossa dell' avversario, e alla fine scelgo la mossa che, dopo la contromossa, lascia la scacchiera nella condizione migliore.
Lo stesso meccanismo puo' essere ripetuto piu' volte, e il risultato e' che verra' definita una tattica in questo modo: io decido la mia mossa in previsione di cosa potro' fare io e di cosa fara' l'avversario nelle successive N mosse.
Questo meccanismo si chiama di "forza bruta" perche' il calcolatore, per vincere la partita sfrutta il vantaggio che ha nell' essere molto piu' veloce nello svolgere calcoli e memorizzare informazioni rispetto ad un giocatore umano. Per decidere la mossa successiva, anziche' basarsi sull' intuito, infatti, valuta tutte le possibili mosse, e tutte le possibili contromosse alle mosse fatte, e cosi' via per N livelli. E' una cosa che una persona non potrebbe mai fare perche' non sarebbe in grado di

memorizzare, e non avrebbe la pazienza di seguire, tutte le possibili variazioni.
Se sulla scacchiera ho quattro pedine che posso muovere, valutero' due o tre mosse per ogni pedina. Per ciascuna di queste due o tre mosse valutero' un numero di contromosse che corrisponde al numero di pedine che l' avversario puo' muovere per il numero di mosse che puo' fare ogni pedina e cosi' via.
Questo tipo di algoritmo genera una "esplosione combinatoria". Questo significa che il tempo necessario per scendere di N livelli e quindi definire una tattica aumenta esponenzialmente con il numero di livelli che voglio percorrere.
Il limite di questo tipo di algoritmi e' dato proprio dalla impossibilita' di scendere di un numero di livelli molto elevato, a meno di non fare semplificazioni. Ad esempio potremmo scartare a priori tutte le mosse che gia' dopo una prima sequenza mossa/risposta dell' avversario peggiorano sensibilmente l' euristica. Queste scelte possono accelerare molto la ricerca, ma vanno fatte pensando attentamente, perche' ad esempio in questo caso potremmo stare eliminando tutte le mosse che implicano il sacrificio di una mia pedina per ottenere un vantaggio maggiore, e questo potrebbe essere errato.
La scelta e' solitamente di tagliare qualcosa quando la situazione e' palesemente negativa, e di limitare il numero di livelli di simulazione a quanto necessario per ottenere una risposta in tempi ragionevoli e un gioco abbastanza accurato.
Anche in questo caso, nei prossimi capitoli, potremo provare a descrivere un gioco basato sulla intelligenza artificiale e sui sistemi a regole.
Occorre pensare che il sistema di intelligenza artificiale a regole non e' utilizzato solamente per giochi di stategia ma anche per gli avversari ad esempio in giochi come i platform. In questo caso pero', solitamente, la base di regole e' molto semplice, e

non e' necessario scendere di molti livelli per elaborare una buona strategia. In molti casi e' sufficiente un euristica, altrimenti si corre il rischio che l' avversario sia troppo intelligente, e considerando il fatto che i suoi tempi di reazione sono enormemente piu' veloci di quelli del giocatore umano, che diventi imbattibile.

Giochi in rete - meccanismi client-server

Gia' alla fine degli anni ottanta, in molte sale giochi era possibile giocare a giochi con piu' postazioni, collegate in modo che giocatori su diverse postazioni potessero giocare alla stessa partita dello stesso gioco. All' epoca era impossibile che un home computer potesse fare le stesse cose, e nel caso migliore esistevano giochi in cui due giocatori potessero giocare assieme utilizzando entrambi i joysticks connessi allo stesso computer. Oggi, e gia' da alcuni anni, tutti i computer che vengono prodotti hanno una scheda di rete. L' avvento di internet ha cambiato drasticamente il modo di vedere un sistema operativo. Oggi un computer e' diventato una macchina il cui scopo principale e' di comunicare. I programmi utilizzati piu' frequentemente sono quelli per scambiarsi la posta, per chattare online e per leggere pagine web.
Anche i giochi hanno quindi subito la stessa evoluzione. Oggi molti giochi sono fatti in modo che piu' giocatori, ciascuno dalla sua macchina, possano condividere la stessa "mappa", e quindi muoversi su uno stesso schema di gioco.
Questo per molte e diverse tipologie di giochi: dai giochi di corsa di automobili a giochi con mappe estremamente

complesse, come ad esempio "ultima online", che sono costruiti, piu' che come un gioco, come una completa simulazione di un mondo virtuale. Lo scopo del gioco non e' tanto quello di eliminare i nemici o gli animali che si trovano nel mondo virtuale, e sono comunque pilotati dal computer, ma e' piu' che altro quello di aggregarsi con altri giocatori per costruire un gruppo "solido", che permetta di affrontare assieme gli avversari.
La presenza di un sistema per chattare all'interno del gioco permette di trasformare il terreno di gioco in un punto di aggregazione: una persona puo' entrare nel gioco semplicemente per incontrare gli avatar degli amici e per fare due chiacchiere, e magari per risolvere qualche enigma assieme.
Una partita ha una durata molto elevata, che puo' essere anche di mesi. Quando il personaggio non sta giocando lo si puo' lasciare ad esempio ad allenarsi o a svolgere un lavoro ripetitivo, per poi ripartire quando si ha tempo per giocare ancora. I personaggi di un gioco di questo tipo, o meglio i loro "avatar" (la loro rappresentazione nel mondo virtuale) formano comunita', si sposano, costruiscono le proprie case, evolvendo lo stato del mondo.
Tecnicamente realizzare un gioco del genere richiede la realizzazione di due programmi distinti: un server e un client. Un server e' un programma che gira su un computer, solitamente un mainframe con buona velocita' di calcolo e una capacita' di memoria sufficiente a mantenere l'intera mappa.
Il server si occupa di comunicare con i client, inviare a questi la mappa e modificarla in risposta a messaggi ricevuti. Il client si occupa di visualizzare la mappa inviata dal server, di accettare i tasti premuti dal giocatore e di inviare al server le modifiche da effettuare sulla mappa stessa. Un gioco puo' avere piu' programmi client, grafici o testuali. L' importante e' che questi

rispettino un protocollo, cioe' delle regole di comunicazione precise con il server.
In ogni caso in cui due macchine debbano comunicare e' necessario che le regole di questa comunicazione siano precise altrimenti potrebbero accadere inconvenienti. Un comando non capito potrebbe causare problemi non solamente per il client che lo ha inviato ma anche per tutti gli altri giocatori. In questo tipo di gioco "interrompere" il mondo virtuale per un errore del server o per problemi di rete ha conseguenze gravi per i giocatori: tutti potrebbero essere costretti a reiniziare la partita da zero.
Bisogna inoltre considerare che un client potrebbe "cadere", cioe' sparire dalla rete improvvisamente, o uno dei pacchetti di comunicazione tra il client e il server potrebbe andare perso, e il tutto va progettato in modo da tenerne conto e evitare conseguenze catastrofiche (per il mondo virtuale).
Questo tipo di gioco richiede la possibilta' di essere espanso da parte dei giocatori, altrimenti diventerebbe noioso. Un giocatore deve poter allora inventare un nuovo oggetto, un tipo diverso di avversario, degli enigmi e cosi' via.
Ogni gioco di questo tipo offre un linguaggio di scripting. I giocatori piu' esperti hanno solitamente il diritto di "scriptare" sul gioco, cioe' di aggiungere al gioco nuovi oggetti descrivendoli con un linguaggio che non debba essere compilato ma che il "motore" di gioco (il ciclo di gioco di un gioco in rete) sia in grado di accettare direttamente. Le funzioni scritte in questo linguaggio di scripting verranno eseguite invece di quelle integrate nel gioco ogni volta che il nuovo oggetto viene creato e usato.
Realizzare un gioco di questo tipo richiede allora almeno la capacita' di utilizzare due linguaggi: uno compilato (viene tradotto in linguaggio macchina prima dell' esecuzione) e

veloce, con cui realizzare gli sprites e tutta l' interfaccia, e uno interpretato, anche se piu' lento, con cui realizzare gli oggetti, in modo da poterli aggiungere mentre il gioco sta girando, senza dover fermare e ricompilare il programma.
La comunicazione tra il server e il client del gioco avviene solitamente utilizzando il protocollo tcp/ip, che e' il protocollo utilizzato dalle macchine di internet per comunicare tra loro. Ad ogni macchina e' assegnato un numero univoco nella rete mondiale.
Due macchine, per comunicare, devono semplicemente conoscere ciascuna l'indirizzo dell' altra. Alcune macchine della rete si occupano di fare routing, cioe' di stabilire, dato il mittente e il destinatario del pacchetto, quale direzione dare al pacchetto per raggiungere le prossima macchina che fa routing o il destinatario.
Quindi, per noi, avere due macchine che siano nella stessa stanza e collegate con un cavetto o avere due macchine che si trovino in due diversi punti di internet e' la stessa cosa. Unica differenza tra i due casi e' la velocita' con cui le macchine possono comunicare tra loro.
Una volta la velocita' della rete era un parametro estremamente critico nello sviluppo di un gioco con piu' giocatori.
Oggi questo parametro diventa meno problematico perche' le velocita' delle connessioni che utilizziamo sono molto maggiori e sufficienti a permettere una partita con un gioco che abbia anche un alto livello di interattivita'.
L' unico problema di velocita' e' dato dal server: il server e' infatti costretto a comunicare contemporaneamente con tutti i clients, e quindi se ogni client utilizza un kilobyte di banda ogni secondo e al nostro server sono connessi venti clienti, il server avra' necessita' di venti kilobytes al secondo di banda.
Per avere buone prestazioni su giochi con decine o centinaia di

utenti si sceglie di solito di affidare il server ad una "server farm", cioe' di posizionarlo in una sala connessa direttamente a linee ad alta velocita', piuttosto che tenerlo su una linea internet domestica. Le nuove adsl che vengono oggi offerte, con una velocita' di "upload" (dati dal vostro computer verso la rete internet) attorno al megabit al secondo, e i contratti "flat" sempre piu' comuni, garantiscono oggi una velocita' sufficiente per poter avere un server per un gioco on-line a casa propria.
Il client puo' comunicare con il server in due modi: a pacchetti o tramite una socket.
A pacchetti significa che, quando il client deve comunicare qualcosa al server, impacchetta il messaggio in qualcosa di simile ad una lettera, e lo invia tutto in una volta sperando che il server lo riceva. Il server, allo stesso modo, puo' comunicare le informazioni sulla mappa ad un client, sperando che questo le riceva.
La comunicazione a pacchetti e' pero' inaffidabile: c'e' il rischio che un pacchetto inviato da una macchina non arrivi mai a destinazione, e che quindi parte della comunicazione sia persa. Nel caso di reti ethernet, realizzate con doppini di cavi telefonici la possibilita' di perdere pacchetti e' ragionevolmente bassa, e l'eventualita' del pacchetto perso non si presenta molto spesso. Ma se il gioco deve girare su una rete meno affidabile, come ad esempio su una rete wireless, che in caso di rumorosita' puo' arrivare a livelli di perdita di pacchetti attorno al trenta percento o superiore,e' necessario considerare seriamente il problema. Se il client inviasse una richiesta al server e si fermasse ad esempio ad attendere una risposta, avremmo il trenta per cento di possibilita' che la richiesta sia persa, quindi che la risposta non arrivi, e il client rimanga fermo in attesa all'infinito.
Inoltre, come per una lettera, il pacchetto potrebbe arrivare a destinazione con un ritardo casuale, e quindi due pacchetti

inviati in un certo ordine da un client ad un server potrebbero arrivare a destinazione in ordine inverso.
Il meccanismo dei pacchetti, pero', e' molto utile perche' si tratta di un meccanismo assolutamente asincrono: quando il server si vede arrivare un pacchetto processa la richiesta in esso contenuta, e a tempi fissi invia a tutti i clienti la nuova mappa. Quando non ci sono pacchetti in arrivo il server e' sempre libero di svolgere altri compiti.
La socket e' un meccanismo piu' complesso: tra il client e il server viene attivato un canale di comunicazione stabile, e ogni volta che il client deve inviare un messaggio al server scrive qualcosa sulla socket. Il server si vedra' arrivare il messaggio e potra' rispondere sulla stessa socket.
La cosa e' un po' piu' complessa, per prima cosa perche' le socket sono realizzate utilizzando i pacchetti, e per aprire una socket e mantenerla aperta vengono inviati molti pacchetti, e poi perche' il server, anziche' attendere semplicemente l' arrivo di messaggi, dovra' poter gestire contemporaneamente molte connessioni, sempre aperte, con tutti i suoi clienti.
D' altro canto il meccanismo delle socket garantisce piena affidabilita'. Qualunque messaggio inviato verso il server arrivera' sicuramente a destinazione, come siamo sicuri che la risposta verso il client sara' restituita, e il messaggio rispettera' l'ordine temporale di invio. Il meccanismo e' simile a quello di una telefonata.
L' uso di un meccanismo o dell' altro dipendera' dal tipo di gioco che stiamo realizzando. Se realizziamo un gioco per cui i clienti devono essere sempre connessi, e non ci sono molti messaggi da inviare, come per esempio un poker on-line, il sistema migliore potrebbero essere le socket: non corriamo alcun rischio che un giocatore perda il proprio turno o faccia la scelta sbagliata perche' un pacchetto viene perso.

Se stiamo realizzando un gioco interattivo, in cui il server invia continuamente ai clients lo stato della mappa mentre i cients riportano al server informazioni sulle scelte del giocatore, la consequenzialita' temporale tra i messaggi diventa meno importante, come il rischio che uno di questi sia perso: se non ricevo la mappa dal server ora, manterro' quella vecchia, per poi ricevere quella nuova tra un secondo. Diventa inoltre piu' pesante il traffico di rete ed e' necessario considerare il tempo perso per aprire e mantenere aperta una connessione affidabile, che potrebbe essere eccessivo.
Solitamente in questi casi si preferisce comunicare usando i pacchetti.
In figura e' mostrata una schermata di una versione di "ultima on-line". Si puo' chiaramente riconoscere la "griglia" data dalla mappa che il server passa al client per desrivere il gioco, evidenziata dal mattoncino del pavimento.

A grandi linee possiamo ora provare a descrivere un algoritmo che ci possa permettere di creare un server di gioco on line, molto semplificato ma sull' esempio di "ultima on-line".
Per prima cosa sara' necessario definire una mappa: la nostra mappa sara' semplicemente una matrice, come quella utilizzata sui platform, ma di dimensioni molto maggiori per poter descrivere un mondo che richieda tempo per essere completamente attraversato, e che possa contenere molti personaggi. Il motore scorrera' la mappa e aggiornera' lo stato e le interazioni tra tutti gli oggetti. Un passo di simulazione e' una operazione molto lunga, quindi potremmo fare delle semplificazioni, come ad esempio scegliere di simulare solo gio oggetti "attivi" del gioco, cioe' quelli che effettivamente sono visibili dagli utenti, e di lasciare inalterati quelli che si trovano in una posizione dove nessun utente li vede. Questo puo' accelerare di molto il motore del gioco, anche se ne complica un po' la realizzazione.
Ogni elemento su una mappa avra' pero' un proprio stato, e non potra' essere descritto solamente con un carattere, come abbiamo fatto in precedenza. Dobbiamo quindi per prima cosa focalizzarci sul concetto di "oggetto", e descrivere come questo oggetto e' realizzato e come puo' interagire con altri oggetti. Gli oggetti dovranno poter avere dimensioni diverse e quindi occupare piu' riquardi della mappa.

Ad esempio, potremmo iniziare descrivendo i seguenti oggetti di un paesaggio di montagna:

Un albero e' un oggetto che occupa uno spazio di un riquadro per un riquadro. Ha una eta'. Ogni diecimila passi di gioco la sua eta' aumenta. La sua immagine cambia in funzione dell'eta'. Ogni trentamila passi di gioco l' albero genera un altro albero,

che posiziona casualmente in un riquadro, se vuoto, entro una distanza di uno scaccho dalla sua posizione. Se il riquadro scelto e' pieno il nuovo albero non viene generato. Dopo centomila passi di gioco l' albero sparisce (l' ultima immagine dovrebbe mostrare un albero secco).
Un sasso e' un oggetto che occupa un riquadro. Non svolge nessuna azione. Occupando dei riquadri i personaggi non potranno camminare su questi riquadri e quindi e' possibile formare delle barriere.
Una pecora e' un oggetto che occupa un riquadro. Ogni cento passi di simulazione cambia la sua posizione spostandosi casualmente di una casella. Se la nuova casella scelta come destinazione contiene un altro oggetto la pecora non si sposta.
E cosi' via. Alcuni degli oggetti non faranno nulla mentre altri oggetti dovranno interagire. Per prima cosa e' allora necessario classificare gli oggetti, come ad esempio inanimati, piante, animali, etc.. e costruire una lista di categorie, magari in grado di crescere.

Quindi potremmo descrivere un oggetto in questo modo:
nome (una sequenza di caratteri, ad esempio "sasso" o "pecora")
tipo (ad esempio "terreno", "inanimato" o "animale innocuo", o "animale aggressivo")
collegamento ad una immagine
script di definizione del comportamento
script di interazione con il giocatore
stato dell'oggetto (congelato/in gioco)

Per descrivere gli oggetti useremo un linguaggio di scripting: questo significa che ogni oggetto conterra' al suo interno un programma, e che questo programma ne descrivera' il comportamento. Ogni volta che sara' necessario simulare

l'oggetto il suo script verra' chiamato. Lo script deve avere la possibilita' di leggere il contenuto della mappa vicino all'oggetto, di modificare la mappa, di modificare l'immagine dell' oggetto.
Allo stesso modo possiamo definire un altro script da lanciare quando il giocatore viene in contatto con l'oggetto. Casi particolari, come ad esempio combattimenti con animali o mostri, possono essere gestiti direttamente dal motore di gioco.
Passiamo all'oggetto tutte le informazioni sul giocatore e diamogli la possibilta' di cambiarle.
Il nostro giocatore sara' descritto principalmente dalla sua posizione, da oggetti che potrebbe avere con se, da un numero che ne indica lo stato di salute per giochi che implichino battaglie, e cosi' via.

Il nostro server dovra' quindi fare questo ciclo di gioco:

```
- per ogni giocatore collegato
{
  - data la posizione del giocatore, segnala come oggetti
non congelati tutti quelli in un intorno di dieci caselle
dalla posizione del giocatore
}
- per tutti gli oggetti sulla mappa
{ - se l'oggetto non e' congelato
  {
    - esegui lo script dell'oggetto passandogli la mappa
    - congela l' oggetto (se il giocatore e' in zona si
scongela al prossimo giro)
  }
}
- se ho ricevuto pacchetti dai giocatori
  {
    - per ogni pacchetto ricevuto
      {
        - aggiorna la posizione del giocatore e lo stato
secondo le info ricevute dal client
        - update the position of the player and his
state, using the info received from the client
        - se il giocatore e' a contatto con un oggetto
chiama lo script di interazione giocatore-oggetto
```

```
        - se il giocatore e' a contatto con un avversario
gestisci il combattimento
      }
  }
}
```

Dato che il campo di gioco e' abbastanza grande e il linguaggio di scripting non e' molto veloce c'e' il rischio che l' utente riesca a vedere che l'aggiornamento degli oggetti sullo schermo inizia da un angolo e gradualmente scorre fino all'angolo opposto. Lo stesso problema si pone ad esempio quando dobbiamo visualizzare uno sprite molto grande o quando dobbiamo far scorrere lo sfondo.
Una tecnica che viene utilizzata per rendere meno visibile l' aggiornamento, e che puo' essere utilizzata in maniera generale anche per gli sprites o altri problemi simili, e' di interlacciare l'aggiornamento del campo di gioco.
Se una volta marcati come attivi o congelati gli oggetti l' aggiornamento viene fatto in maniera interlacciata l' effetto "onda" rigurdante l' aggiornamento degli oggetti dall'alto al basso sara' molto meno visibile.
Il sistema piu' semplice e' di scorrere prima tutte le righe pari della griglia, e quindi di aggiornare gli oggetti su queste, e poi di ripetere la cosa sulle righe dispari. La cosa puo' essere fatta in maniera piu' complessa, semplicemente cambiando l'ordine con cui gli oggetti vengono aggiornati, per dare l'impressione di maggiore fluidita' possibile nell'aggiornamento dello stato.
La stessa tecnica viene utilizzata ad esempio per trasmettere le immagini televisive. Anziche trasmettere un intero schermo venticinque volte al secondo si preferisce trasmettere cinquanta volte al secondo dei mezzi schermi, contenenti alternativamente le linee pari e le linee dispari da visualizzare. Il risultato e' che non e' possibile vedere sfarfallii sull' immagine televisiva.
Una cosa che possiamo notare del ciclo di gioco e' che questo sta

svolgendo due operazioni distinte: sta aggiornando il terreno di gioco, modificando lo stato degli oggetti e spostando quelli animati, e dall'altra parte sta servendo gli utenti e cambiando le informazioni del loro avatar in risposta alle richieste del client. Le due operazioni potrebbero essere distinte: un algoritmo potrebbe occuparsi degli aggiornamenti del gioco virtuale e un altro delle interazioni con il giocatore e tra i giocatori e il mondo virtuale. I due algoritmi potrebbero girare contemporaneamente e agire sulla stessa mappa.
La cosa va valutata attentamente in quanto fornisce vantaggi e svantaggi. E' un vantaggio ad esempio il fatto che anziche' scrivere un algoritmo molto complesso e' possibile risolvere il problema scrivendo piu' algoritmi semplici ed indipendenti. E' uno svantaggio perche' c'e' la necessita' di avere la mappa accessibile da tutti i programmi che girano in parallelo, e quindi di fare in modo che l' accesso alla mappa non causi problemi di concorrenza.
Riguardo alla velocita', se noi sviluppiamo per una macchina con un singolo processore l'utizzare diversi processi ci costringe a doverli coordinare. Il codice necessario a coordinare i processi e il tempo perso ad attendere che sia disponibile la mappa, solitamente, rende questi programmi piu' lenti di un unico programma che svolga lo stesso compito. Se la macchina ha piu' microprocessori, pero', il risultato potrebbe essere molto vantaggioso perche' ogni processo potrebbe girare su un diverso microprocessore con un deciso aumento della velocita' rispetto ad un processo singolo.
Il fatto di usare il linguaggio di scripting ci permette di andare a caricare la definizione di un nuovo oggetto mentre il sistema sta girando, senza la necessita' di riavviarlo. Mentre il gioco e' in esecuzione e' possibile allora inserire un nuovo tipo di avversario, espandere la mappa e cosi' via.

Giochi tridimensionali

Da alcuni anni a questa parte la potenza di calcolo di un computer domestico e' diventata sufficiente a poter eseguire in tempo reale un algoritmo di render. Il "render" e' un algoritmo che, data la descrizione di un mondo tridimensionale, ne riesce a ricavare una vista bidimensionale da una posizione scelta, e permette di riportarla sul monitor.
In questo tipo di giochi la mappa vista come una griglia di oggetti che interagiscono con l'utente potrebbe essere sostituita con una descrizione "vettoriale" di un mondo in tre dimensioni, o semplicemente, mantenendo a due dimensioni la mappa del gioco, i personaggi e gli oggetti potrebbero essere sostituiti da figure tridimensionali, e il loro render ricalcolato momento per momento per dare l'effetto di essere realmente in un mondo tridimensionale.
Alcuni esempi di giochi tridimensionali per personal computer

sono ad esempio "descent" o "Grim fandango". Sono due

esempi di due giochi molto diversi.
Su descent una navetta spaziale si muove all'interno di un labirinto tridimensionale, e deve seguire una nave guida per raggiungere una posizione del labirinto, distruggendo nel frattempo le navette avversarie. Il gioco e' basato su una mappa tridimensionale, e per dare l'impressione di muoversi in un mondo sempre diverso e' utilizzato il meccanismo delle "textures".
Grim fandango e' una avventura. Il protagonista deve muoversi e

risolvere degli enigmi, come altri precedenti giochi della Lucas Arts, e interagire con altri personaggi pilotati dal computer, con cui deve dialogare per avere informazioni. In questo caso sono i personaggi, il protagonista e lo sfondo del gioco ad essere realizzati in tridimensionale, e l' animazione riguarda il movimento dei personaggi stessi.
Per descrivere un oggetto nelle tre dimensioni viene utilizzato

un programma di CAD. Essendo infatti lo schemo del computer bidimensionale, per poter descrivere un oggetto tridimensionale e' necessario avere l'aiuto di un programma, che permetta di descrivere l'oggetto in termini di linee e triangoli.
Mentre uno sprite in un gioco a due dimensioni e' solitamente descritto con una metrice di punti di diversi colori, un disegno in tre dimensioni deve essere descritto in maniera diversa.
Solitamente si utilizza una descrizione dell' oggetto vettoriale. In pratica la figura viene descritta in termini di superfici a contatto l'una con l' altra.
Le superfici sono descritte utilizzando le coordinate dei vertici, ed eventualmente tipizzando le superfici in termini di colore, trasparenza, riflessione della luce.
Per fare un esempio proviamo a descrivere una piramide a base triangolare in termini di coordinate X,Y,Z.
La base sara' composta da punti tutti sul piano X,Y, quindi con coordinata Z pari a 0. In particolare sara' ad esempio il triangolo di coordinate (0,0,0) - (8,0,0) - (4,6,0), che dovrebbe essere un triangolo equilatero.
Le tre facce superiori saranno composte dagli stessi punti piu' un vertice nella posizione (4,3,6), e quindi avranno le seguenti coordinate:
Triangolo (0,0,0) - (8,0,0) - (4,3,6)
Triangolo (8,0,0) - (4,6,0) - (4,3,6)
Triangolo (4,6,0) - (0,0,0) - (4,3,6)

L' insieme dei quattro triangoli descrive completamente la figura.
Il fatto di utilizzare una rappresentazione vettoriale ha dei vantaggi. Per esempio e' possibile moltiplicare tutte le coordinate per due e ottenere una piramide di dimensioni doppie da quella descritta.

Un disegno vettoriale puo' essere scalato alle dimensioni volute per disegnarlo sul monitor. Questo e' molto utile perche' ci permette di definire oggetti a cui possiamo avvicinarci o da cui possiamo allontanarci ottenendone comunque una vista precisa senza "sgranare". Questo viene utilizzato per esempio per avere un effetto distanza sugli oggetti. Un oggetto lontano e' piu' piccolo, e un oggetto vicino e' piu' grande, senza la necessita' di utilizzare differenti sprites.
La descrizione che abbiamo fatto di una piramide per triangoli non e' casuale. In realta' superfici piu' complesse, come ad esempio dei quadrati o dei rettangoli, sono descrivibili come insiemi di triangoli complanari. E' possibile, con un certo grado di approssimazione, descrivere in termini di triangoli ogni superficie.
Una volta descritta la figura in termini di triangoli, per poterla disegnare sullo schermo, e' necessario definire delle luci e una "camera", cioe' il punto da cui l'oggetto verra' visualizzato.
Le luci possono essere puntiformi o direttive. Le superfici della figura possono riflettere la luce e quindi presentare un diverso colore e una diversa luminosita a seconda della distanza dalla sorgente di luce, e magari dalle caratteristiche della sorgente stessa. Ad esempio per avere l'effetto di un tramonto sara' possibile utilizzare delle luci di colore rosso, mentre potremmo utilizzare della luce bianca per avere l'effetto di una giornata di sole.
La "camera" descrive il punto da cui l' oggetto viene visto e la direzione dello sguardo dell' osservatore. La camera viene posizionata come sarebbe posizionata una macchina fotografica. Spostando la camera cambia il punto da cui l'oggetto viene visualizzato. L' animazione dell'oggetto puo' essere ottenuta spostando la camera, spostando le luci o cambiando gradualmente la posizione dei vertici di alcuni dei triangoli della

figura.
La visualizzazione dell' oggetto tridimensionale, cioe' il "render", e' un algoritmo complesso, anche tenendo presente il fatto che, oltre alla semplice visualizzazione delle superfici degli oggetti, deve essere possibile valutare il colore degli oggetti secondo le fonti di luce, la posizione dell' osservatore, il tipo di superficie, le "textures", e effetti avanzati come ad esempio l' effetto "foschia", che viene utilizzato in giochi come quelli della serie "quake".
Solo per avere un idea di come potrebbe funzionare, un semplice algoritmo di render che veniva utilizzato dai programmi di CAD negli anni '90, era quello di ordinare tutti i triangoli di cui la figura era composta secondo la loro distanza dall' osservatore, di ruotarli secondo il punto di vista dell' osservatore, e di disegnarli partendo da quelli piu' lontani a quelli piu' vicini. Il risultato era che i triangoli piu' lontani potevano essere coperti dai piu' vicini, nello stesso modo in cui alcune superfici degli oggetti che guardiamo possono essere coperte da altre superfici dell' ogetto stesso o di altri oggetti.
In realta' questo metodo di render oggi risulta troppo primitivo, perche' non gestisce correttamente superfici che si compenetrino l'una con l'altra e perche' non consente di avere colori diversi per diversi punti della stessa superfice. Ad esempio non e' possibile ottenere superfici a specchio, che riflettano oggetti vicini o che riflettano le fonti di luce.
Oggi l' immagine viene costruita valutando, per ogni punto dello schermo, quale sia il colore di questo punto, secondo le riflessioni della luce generata dalle sorgenti nelle superfici.
La cosa e' estremamente complessa in termini di elaborazione e solitamente viene, almeno in parte, demandata a schede video con espansioni tridimensionali, in grado di gestire l' ordinamento di triangoli e di aiutare sui calcoli.

Per poter ottenere effetti piu' realistici, come gia' detto prima, alcuni giochi sovrappongono all'immagine tridimensionale delle "textures", cioe' srovrappongono alle superfici dell'immagine tridimensionale dei disegni bidimensionali, scalati in modo di adattarsi all'oggetto.
Come gia' detto, la descrizione di oggetti tridimensionali puo' essere fatta in termini di triangoli, ma descrivere un personaggio di un gioco, soprattutto se animato, puo' essere una operazione difficile da fare, e quindi e' il caso di affidarsi ad un programma di disegno tridimensionale che ci aiuti a farlo.
Un buon programma per il disegno di oggetti tridimensionali e' ad esempio "blender". Blender e' un programma open-source, il che significa che e' possibile scaricarne gratuitamente i sorgenti da internet e compilarlo per il proprio computer senza pagare licenze. In realta' blender e' supportato da una comunita' di utenti molto vasta e quindi e' possible ottenere gratuitamente da internet una copia di blender gia' compilata, solo da installare, per linux o per windows.
Blender e' un programma che non e' molto facile da utilizzare, e probabilmente per il primo utilizzo e' necessario una lettura del manuale per ottenere un disegno, ma 'e comunque un buon aiuto nel descrivere oggetti tridimensionali. Oltre a questo blender permette di esportare sia "viste" degli oggetti disegnati, cioe' delle "fotografie" dell' oggetto prese dalla camera, sia l' intero disegno dell'oggetto in tridimensionale in un formato standard che puo' essere caricato dalle librerie grafiche com openGl o DirectX, che sono principalmente utilizzate, appunto, per scrivere giochi.
Esistono anche programmi commerciali equivalenti a blender, come ad esempio "maya", ma sono solitamente supportati da una comunita piu' ristretta e hanno la stessa difficolta' nell' essere utilizzati.

Questi programmi permettono solitamente di plasmare gli oggetti tridimensionali "a mano libera", spostandone i vertici con il mouse, a differenza di altri programmi per il disegno tridimensionale, come ad esempio "AutoCAD", che essendo progettati piu' che altro per il disegno tecnico offrono strumenti che permettono di disegnare figure estremamente precise, anche con un interfaccia che permetta di definirne i vertici utilizzando dei comandi dati da tastiera, ma non sono solitamente adatti per il disegno di oggetti per i giochi, dove piu' che una rigorosita' geometrica del disegno e' necessario poter plasmare l' oggetto in modo semplice per ottenere una figura gradevole alla vista. Inoltre blender ci permette di animare gli oggetti ottenuti definendo un insieme di "punti" dell'immagine che possono essere spostati, e di costruire delle animazioni.