UNIVERSITÀ DELLA CALABRIA

Dipartimento di ELETTRONICA, INFORMATICA E SISTEMISTICA

Anno accademico 2005/2006

Università della Calabria

Laboratorio di programmazione orientata agli oggetti:

Mastermind

Assimazza - GC 83308

Introduzione

Mastermind è un gioco di logica per due giocatori. Scopo del gioco è indovinare la combinazione scelta segretamente dall'altro giocatore entro dieci tentativi. Per ogni combinazione provata, il giocatore codificatore comunica al decifratore la somiglianza con la combinazione segreta usando un codice.

Ogni combinazione è formata da quattro pioli scelti tra sei colori, ognuno dei quali presente anche più di una volta, per un totale di 1296 permutazioni. A scelta è possibile lasciare uno o più fori piolo vuoti, passando così ad un totale di 2401 permutazioni.

Il giocatore codificatore è tenuto a valutare ogni tentata decodificazione usando dei pioli chiave di due colori diversi. Ogni piolo chiave corrisponde ad un piolo della combinazione, ma il decodificatore è allo scuro delle effettive associazioni.

Un piolo chiave bianco indica che un piolo combinazione è del giusto colore ma la sua collocazione è errata. Un piolo nero indica, invece, la correttezza sia del colore del piolo combinazione, che della sua posizione. Un piolo chiave assente indica che un piolo combinazione ha un colore sbagliato.

E’ possibile introdurre un limite di tempo per la riflessione in modo da accorciare i tempi di gioco, forzando il decifratore ad un ragionamento più attivo e ad un comportamento più azzardoso.

Come variante è possibile disputare due partite in contemporanea incrociando i ruoli dei giocatori, in modo da aggiungere un ulteriore livello di competizione.

Descrizione del caso di studio

L’applicazione sviluppa gran parte del regolamento del gioco, aggiungendo piccole modifiche in modo da rendere la partita più interessante.

Il progetto implementa solo la dinamica di gioco single-player, al termine della quale il giocatore potrà cimentarsi in una nuova partita nuovamente nel ruolo di decodificatore.

Al primo avvio, l'applicazione chiederà all'utente d'inserire il proprio nome; in mancanza del quale verrà utilizzato un nome di default ("Sconosciuto"). Il gioco terrà conto di questo campo nel caso si stabilisca un nuovo record.

Una finestra delle opzioni permetterà di scegliere i dettagli della partita. E' possibile attivare come disattivare la possibilità di formare combinazioni contenenti un foro vuoto. E' anche possibile abilitare un tempo massimo e settare i relativi minuti.

Dalla finestra delle opzioni è possibile modificare il nome del giocatore (purché esso sia meno di 25 caratteri), visualizzare le sue statistiche e resettarle.

Il gioco tiene memoria dei migliori punteggi usando una highscore, che è possibile azzerare a gioco fermo. Il calcolo del punteggio è basato sul numero di tentativi utilizzato e sulla percentuale di tempo occupato. Per ogni combinazione errata vengono sottratti 100 punti da un totale di 1000, successivamente, viene calcolata la percentuale di tempo rimasto e tale numero viene moltiplicato per 10. Nel caso il tempo massimo sia inferiore ai tre minuti, viene aggiunto un ulteriore bonus in proporzione inversa al tempo massimo.

All'uscita dal gioco, sia le opzioni che la highscore che i dettagli del giocatore, vengono salvati sotto un formato serializzato nel file dati.dat e caricati al successivo avvio.

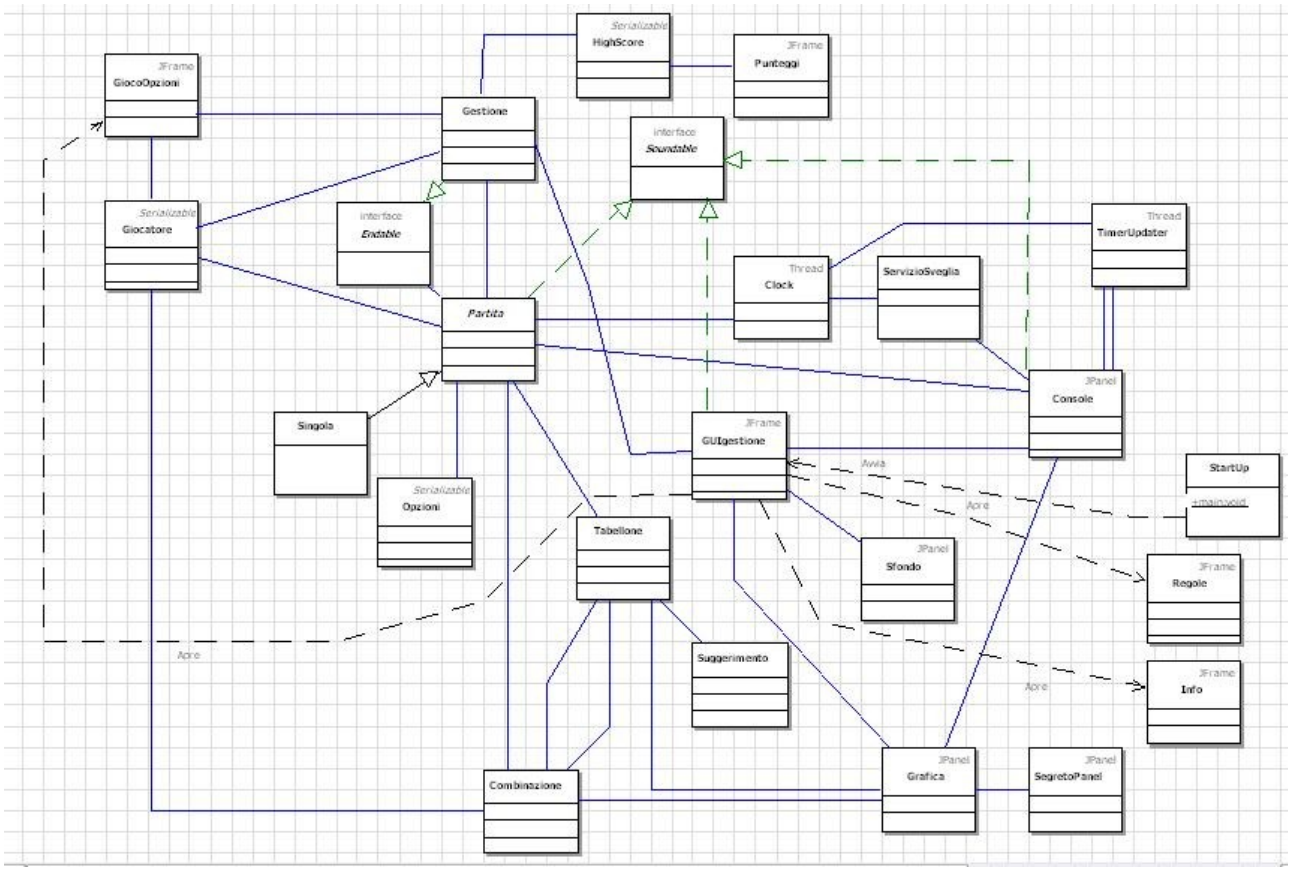
Ad ogni partita il giocatore umano è sempre il decodificatore, mentre il computer seleziona casualmente una combinazione segreta. Il giocatore può scegliere, usando il mouse, quali colori posizionare e dove, confermando premendo il pulsante OK. Ad ogni tentativo andato male, il computer mantiene la vecchia combinazione provata, cosicché il giocatore debba solo modificarla. Se dovesse scadere il tempo limite, il computer convaliderà la combinazione tentata così come si trova allo stato attuale.

Nel caso si decida di abbandonare o terminare la partita, essa verrà conclusa con una sconfitta, previo avviso.

Profilo implementativo

Nel progetto è stato fatto largo uso delle librerie swing per la grafica, mentre per la parte logica, a parte le librerie di default, sono state usate le util ed occasionalmente altre librerie come le io.

E’ possibile distinguere tre blocchi principali: grafica, gestione e gioco. Le classi del gruppo grafica si curano di implementare la GUI utente e la grafica del gioco. Le classi del gruppo di gestione, si curano di arbitrare l’applicazione e gestire i dati, mentre le classi del gruppo di gioco, sono il vero e proprio motore di ogni partita. Sono state inoltre utilizzate due interfacce, allo scopo di poter segnalare alcuni oggetti di un evento accaduto, e di marcare quegli oggetti come avvisabili.



Classi grafiche:

- GUIgestione — Finestra principale del gioco
- SfondoPanel — Pannello interno per lo sfondo di GUIgestione
- Info — Finestra dei riconoscimenti
- Regole — Finestra delle regole del gioco
- Punteggi — Finestra dei punteggi migliori
- GiocoOpzioni — Finestra delle opzioni
- Grafica — Pannello grafico del tabellone
- SegretoPanel — Pannello interno della combinazione segreta
- Console — Pannello di controllo del gioco
- TimerUpdater — Propaga alla grafica lo scandire dei secondi

Classi di gestione:

- Gestione — Gestione logica del gioco
- Giocatore — Astrazione logica del giocatore e suoi dettagli
- ServizioSveglia — Notifica un evento alle classi iscritte
- Soundable — Marca una classe come avvisabile dalla precedente
- Opzioni — Indica le opzioni selezionate prima di una partita
- Endable — Interfaccia per notificare il termine di una partita
- StartUp — Avvia l’applicazione
- HighScore — Gestione logica dei punteggi migliori

Classi di gioco:

- Partita — Classe astratta che definisce una generica partita
- Singola — Partita giocatore singolo contro il computer
- Clock — Orologio del gioco
- Tabellone — Assicura uno svolgimento del gioco coerente
- Combinazione — Definisce logicamente una generica combinazione
- Suggerimento — Definisce logicamente un quartetto di pioli chiave.

STRATEGIE E SCELTE PROGETTUALI.

E’ stata posta molta attenzione a mantenere rigidamente separate grafica e logica dell’applicazione. Inoltre si è scelto di lasciare la possibilità di specificare nuove varianti o modalità di gioco a chiunque fosse interessato. La classe astratta partita, infatti, implementa solo i metodi indispensabili al buon funzionamento della parte gestionale, mentre le routine più legate al gioco, invece, sono state implementate dalla classe Singola. Così facendo, si dà la possibilità di estendere la classe Partita ulteriormente, creando nuove varianti di gioco con la minima modifica del codice.

Poiché le parti logiche e grafiche sono state rigidamente separate, nasce il problema della notifica interna degli eventi. Infatti, nessuna classe logica ha un riferimento ad una classe grafica, per cui, a prima vista, sembra impossibile poter notificare la grafica o le classi parent dell’occorrenza di un evento. Tale problematica è particolarmente evidente nel caso di una terminazione senza errore di una partita, o nel caso dello scadere del tempo limite. Tali casi sono stati gestiti con due diversi approcci.

Nel primo caso, è stato sufficiente far implementare alla classe Gestione l’interfaccia Endable, contenente un metodo solo, il cui compito è gestire una corretta terminazione della partita. All’atto di creazione della classe Console (interfaccia controller del giocatore), viene passata la classe Gestione come parametro istanza di Endable. In tal modo Console non potrà invocare i metodi di Gestione, impedendo un uso scorretto o malizioso del codice. Quando necessario, Console invocherà il metodo dell’interfaccia Endable notificando la partita.

Il secondo caso, verificatosi ad ogni scadenza del tempo, è stato risolto con l’ausilio di una classe ServizioSveglia. Essa ha il compito di notificare a comando, tutti quegli oggetti che ne abbiano precedentemente fatto richiesta. La classe Clock ha come istanza un ServizioSveglia, creato alla creazione del Clock ed inizialmente vuoto. Successivamente, gli oggetti interessati, che implementano Soundable, si iscrivono al ServizioSveglia, che li memorizzerà in una linked list di Soundable. Quando Clock lo deciderà, invocherà avvisa() notificando tutti gli iscritti. Poiché ServizioSveglia maneggia solo dei Soundable, non ha alcun accesso ad ulteriori dettagli degli iscritti. Un problema simile si è ripresentato quando si è reso necessario comunicare alla interfaccia grafica dell’applicazione la corretta terminazione di una partita. In tal caso si è utilizzata la stessa strategia, istanziando un nuovo ServizioSveglia nella classe Console.

Un ulteriore artifizio usato per garantire maggiore flessibilità ed efficienza dell’applicazione, è la classe Opzioni. Essa si limita semplicemente a memorizzare le opzioni particolari, scelte dall’utente. Una istanza della classe, viene fornita come parametro al costruttore di Partita, impedendo, così, un accesso completo a tutti i metodi della classe Gestione ed un loro conseguente uso sconsiderato. Inoltre, con

questo accorgimento, non è necessario salvare l'intera classe Gestione su disco, ma solo il suo contenuto informativo.

Infine, per consentire una rapida modifica del codice, le costanti comuni sono state dichiarate public e static, in modo da poter costruire qualsiasi oggetto riferendosi direttamente alla costante necessaria, evitando l'uso di costanti multiple, difficili da mantenere coerenti in caso di modifica, o, peggio, di "numeri magici" che avrebbero reso impossibili future modifiche al codice e difficile la sua interpretazione da parte di altre persone.

STRUTTURE DATI

La politica di progettazione adottata ha prediletto una maggiore complessità delle relazioni inter-classe a fronte di una maggiore semplicità delle strutture dati. Di seguito si riportano le strutture dati più interessanti.

La classe tabellone si fonda su due vettori, uno per i pioli chiave ed uno per le combinazione tentate, entrambi array di oggetti. A sua volta, una combinazione è un vettore di interi costanti, che codificano i singoli colori, mentre un quartetto di pioli chiave viene codificato da un double, la parte intera per i pioli neri, quella decimale per i bianchi. Incapsulando, queste due strutture in due oggetti distinti, si riesce ad ottimizzare l'efficienza e la correttezza dei metodi, in contrapposizione ad una implementazione alternativa, che vedeva sia le combinazioni che i suggerimenti, gestiti da un'unica matrice in un'unica classe.

Un'altra struttura interessante, è la classe HighScore. Le sue fondamenta sono una matrice di dieci righe e due colonne, il cui scopo è mantenere in memoria un massimo di dieci punteggi associati a dieci nomi. Per assicurare una corretta visualizzazione, inoltre, la classe riempie automaticamente con spazi vuoti ogni record che abbia una stringa troppo corta. Ad ogni partita terminata con una vittoria, si tenta d'inserire il nome del giocatore ed il suo punteggio nella high score. Nel caso il giocatore abbia stabilito un nuovo record, i vecchi valori vengono scalati di un posto, e l'ultimo record cancellato. L'esito dell'inserimento viene notificato restituendo un valore booleano.

Le altre strutture dati utilizzate sono piuttosto semplici, poiché delegano i compiti più complessi agli oggetti stessi, distribuendo il uniformemente il codice e concretizzando la filosofia della programmazione orientata agli oggetti.

Conclusioni

In conclusione, l’applicazione trattata, permette qualche svago nelle ore di pausa, accogliendo l’utente con un’interfaccia agevole e immediata, lasciando le porte aperte a futuri sviluppi e perfezionamenti impiegando alcuni semplici, ma molto astuti, artifizi progettativi.

Assimazza GC