LEZIONE 2018/11/14 - SCOPA

Gabor Galazzo 20024195 A.A. 2018/2019

Il programma è uno scheletro per il gioco di carte "scopa". Rappresenta le carte, il mazzo, i giocatori e il tavolo. Lo scheletro è programmato per funzionare nel seguente modo:

- Situazione iniziale:
  - Giocatori: 4 stack inizializzati e vuoti
  - Carte sul tavolo: lista inizializzata e vuota
  - Mazzo di carte: coda di 40 carte “ordinate”
- Prima mano:
  - Mescolare il mazzo
  - Dare le carte ai giocatori (3 a testa)
  - Mettere 4 carte sul tavolo
- Fase di gioco:
  - Per ogni mano, ogni giocatore gioca a turno una carta fino a che le carte in mano sono esaurite
  - A questo punto, si danno altre 3 carte a testa ai giocatori
- Condizione di fine partita:
  - Le carte del mazzo sono terminate, e i giocatori hanno esaurito le carte dell’ultima mano

Il programma da la possibilità di segure il gioco “passo a passo":

- Per ogni carta giocata da un giocatore, visualizzare la carta giocata e la situazione del gioco:
  - che carte sono in mano ad ogni giocatore
  - che carte sono sul tavolo
  - che carte sono ancora nel mazzo

Scelte progettuali

Ogni entità del progetto è rapppresentata e condivisibile tramite un *header file* associato che fornisce funzioni e strutture.
Il gioco della *scopa* viene emulato da un'entità **engine** che ha come dipendenze tutti i domini del gioco e li gestisce secondo le regole presabilite.
L'interazione con l'utente è gestita nel **main()**.
Dato che non è stato esplicitamente rischiesto si è optato per ignorare i memory leak generati dalle *malloc()*

SRC TREE:

```
PRJ:.
├───20181114_CARTE
│   │   makefile
│   │   README.md
│   ├───out //Generated output dir
│   │   ├───src
│   │   │       main
```

```
|   |   └──test
|   |           test
|   ├──src
|   |   |   main.c //Main
|   |   |
|   |   ├──engine // Contiene gli engin dei giochi di carte
|   |   |       scopa.c
|   |   |       scopa.h
|   |   |
|   |   └──model // Contiene il dominio del gioco
|   |           card.c
|   |           card.h
|   |           deck.c
|   |           deck.h
|   |           domain.h
|   |           player.c
|   |           player.h
|   |           table.c
|   |           table.h
|   └──test
|           test.c //File di test
└──shared // Filoe condivisi (librerie)
        liste_code.c
        liste_code.h
```

Descrizione delle strutture dati utilizzate

Le strutture utilizzte sono estensioni dei tipi **coda** e **pila** o della struttura **lista**, precedentemente implementate nella libreria liste_code.h

Carta - src/model/card.h

```
typedef enum s { // Rappresentazione dei SEMI
   DIAMOND = 0,
   CLUBS = 1,
   HARTH = 2,
   SPADES = 3
} Suit;
typedef struct {
   int value;
   Suit suit;
} Card;

// Inizializza la carta
Card* card__init(int value, Suit suit);
// Stampa una carta su fp
void card__print(Card* card, FILE* fp);
//Stampa generico per foreach di List
void card__print_for_list(void* card);
```

Mazzo - src/model/deck.h

```
//il mazzo è una coda
typedef Queue Deck;

//Inizializza il mazzo
Deck* deck__init();
//mischia il mazzo tramite "ruffle shuffle"
void deck__shuffle(Deck* deck);
//mischia un numero rando di volte tra 3 e 6 il mazzo
void deck__good_shuffle(Deck* deck);
//togli una carta dala cima del mazzo e te la restituisce
Card* deck__draw(Deck* deck);

//Utililties

//Stampa il deck
void deck__print(Deck* deck);
//Restituisce se il mmazzo è vuoto
int deck__is_empty(Deck* deck);
// Restituiisce il numero di carte dentro il deck
int deck__count(Deck* deck);
```

Tavolo - src/model/table.h

```
//Il Tavolo è una lista
typedef List Table;

// Inizializza il tavolo
Table table__init();
//Prende una carta card e la mette sul tavolo
void table__play(Table *table, Card *card);

//Utilities

//Stampa tutte le carte che ci sono sul tavolo
void table__print(Table table);
```

Giocatore - src/model/player.h

```
//Il giocatore è una pila (le carte che ha in male)
typedef Stack Player;

//Inizializza il giocatore
Player player__init();
// Dato un deck il giocatore ne pesta num_cards carte
void player__draw(Deck* deck, Player* player, int num_cards);
// Verifica se il giocatore ha la mano vuota
```

```
int player__empty_hand(Player player);+
// Fa giocare una carta (pop sulla lista)
Card* player__play(Player* player);

// Utilities

// Stampa tutte le carte in mano ad un giocatore
void player__print(Player player);
```

Engine di Scopa - src/engine/scopa.h

```
#define MAX_PLAYERS 4

typedef struct {
   Table table;
   Deck* deck;
   Player players[MAX_PLAYERS];
   int num_players;
   int turn;
} Scopa_e;

//Inizializza l'engin con n giocatori con la mano
// vuota, un mazzo non mescolato e il tavolo
// vuoto
Scopa_e* scopa_engine__init(int num_players);

// Passa al sucessivo "frame" di gioco
void scopa_engine__step(Scopa_e* scopa_engine);

// Mischia il mazzo, fa pescare a tutti i giocatori
// tre carte e ne mette quattro sul tavolo
void scopa_engine__reset(Scopa_e* scopa_engine);

// Stampa la condizione di gioco
void scopa_engine__print_status(Scopa_e* scopa_engine);

// Verifica se siamo arrivati ad
// una condizione di gioco finito
int scopa_engine__is_game_end(Scopa_e* scopa_engine);
```

Algoritmo di "mescolamento"

```
algoritmo deck__shuffle(Deck deck)
    // Condizione di ritorno:
    // se il mazzo ha 1 carta è mescolato
    IF size(deck) < 2 THEN return
    Deck support  COSTO S: 1
    // Divido il mazzo in due
    FOR i = 0 TO size(deck)/2  COSTO T: n/2
```

```
        aggiungo a support la prima carta di deck
    // Mescolo i due sottomazzi che ho creato
    deck__shuffle(support)  COSTO T: T(n/2); COSTO S: S(n/2)
    deck__shuffle(deck)  COSTO T: T(n/2); COSTO S: S(n/2)
    Deck result  COSTO S: 1
    // Effettuo il "vero" ruffle shuffle
    WHILE(!is_empty(deck) && !is_empty(support))  COSTO T: n
        // Fintanto che ho carte nei sottomazzi
        // lancio una moneta (0 o 1) per capire da
        // quale mazzo pescare una carta per comporre
        // il mazzo mescolato "result".
        // Se in un mazzo non ci sono più carte metto in result
        // tutte le carte rimanenti
        IF (coin_flip() || is_empty(support) && !is_empty(deck)) THEN
            aggiungo a result la prima carta di deck
        ELSE
            aggiungo a result la prima carta di support
    cambio il riferimento a deck con result
```

Dal precedente algoritmo si può dedurre la seguente relazione di ricorrenza:

$$\begin{cases} T(0) = T(1) = 0 \\ T(n) = 2T(\frac{n}{2}) + \frac{3}{2}n \end{cases}$$

Dato che la relazione di ricorrenza precedente è della forma

$$T(n) = \begin{cases} aT(\frac{n}{b}) \Leftarrow n > k \\ c \Leftarrow n \leq k \end{cases}$$

è applicabile il "*Teorema Master*"

$$\begin{cases} a = 2 \\ b = 2 \\ f(n) = \frac{3}{2}n \\ g(n) = n^{log_b a} = n^{log_2 2} = n^1 = n \end{cases}$$

Dala relazione possiamo stabilire che:

$$f(n) = \Theta(g(n)) \Rightarrow T(n) = \Theta(n \cdot log(n))$$

Con un osservazione un po' più accurata si può anche stabilire che qusto algoritmo non è il massimo nei termini di "fattore spazio", infatti, se applichiamo il teorema master contando lo spazio e non il numero di operazioni viene fuori che:

$$S(n) = \Theta(n)$$