Minilaska

Generato da Doxygen 1.8.20

1 Descrizione generale del progetto

Autori

Riccardo Sale - Gabriele Nicoletti

1.0.0.1 Regolamento del gioco Implementazione di un gioco chiamato MiniLaska variante del gioco originale http://www.lasca.org/.
Rispetto al gioco originale miniLaska prevede le seguenti limitazioni:
• si può mangiare/conquistare una sola volta per mossa.
• le torri possono essere alte al massimo 3 pedine, superato questo limite,la pedina più in basso viene rimossa dalla scacchiera.

2 Indice dei tipi composti

2.1 Elenco dei tipi composti

Queste sono le classi, le struct, le union e le interfacce con una loro breve descrizione:

3 Indice dei file

3.1 Elenco dei file

Questo è un elenco dei file documentati con una loro breve descrizione:

4 Documentazione delle classi

4.1 Riferimenti per la struct pedina_t

Struct della pedina.

`#include <struct.h>`

Attributi pubblici

- char symbol
- int is_enhanced

4.1.1 Descrizione dettagliata

Struct della pedina.

Struct pedina costituita da due campi,symbol e is_enhanced.

4.1.2 Documentazione dei membri dato

4.1.2.1 is_enhanced `int pedina_t::is_enhanced`

Identifica se la pedina è un soldato o un ufficiale / valore 1->soldato / valore 2->ufficiale

4.1.2.2 symbol `char pedina_t::symbol`

Identifica il possessore della pedina o l'eventuale pedina vuota attraverso i caratteri '1' '2' '#'->pedina vuota

La documentazione per questa struct è stata generata a partire dal seguente file:

- struct.h

4.2 Riferimenti per la struct torre_t

Struct della torre.

`#include <struct.h>`

Attributi pubblici

- pedina_t pa [3]

4.2.1 Descrizione dettagliata

Struct della torre.

Struct torre costituita da un singolo campo, un array di tipo pedina_t con dimensione 3.

4.2.2 Documentazione dei membri dato

4.2.2.1 pa `pedina_t torre_t::pa[3]`

array di tipo pedina_t con dimensione massima 3.(altezza massima delle torri)

La documentazione per questa struct è stata generata a partire dal seguente file:

- struct.h

4.3 Riferimenti per la struct vector

Struct vector.

`#include <vector.h>`

Attributi pubblici

- size_t size
- size_t capacity
- vdata_t * data

4.3.1 Descrizione dettagliata

Struct vector.

Struct vector.

Nota

Quando si tenta di aggiungere un elemento ma la size è uguale a capacity allora
al vettore viene riservata un area di memoria maggiore e capacity viene incrementata.

4.3.2 Documentazione dei membri dato

4.3.2.1 capacity `size_t vector::capacity`

Indica la capacità "massima" di elementi dell'array

4.3.2.2 data `vdata_t* vector::data`

Puntatore agli elementi del vettore

4.3.2.3 size `size_t vector::size`

Indica il numero di elementi nel vettore.

La documentazione per questa struct è stata generata a partire dal seguente file:

- vector.h

5 Documentazione dei file

5.1 Riferimenti per il file board.h

Funzioni che eseguono operazioni sul board di gioco
.

```
#include "struct.h"
#include "find.h"
```

Funzioni

- void fill_board (torre_t board[7][7])
- void print_board (torre_t board[7][7])
- int * moves (torre_t board[7][7], char player, int *size, int *type_moves)

5.1.1 Descrizione dettagliata

Funzioni che eseguono operazioni sul board di gioco
.

5.1.2 Documentazione delle funzioni

5.1.2.1 fill_board() `void fill_board ( torre_t board[7][7] )`

Inizializza il board modificando quello dato in input.

Parametri

in,out	*board*	Campo di gioco

Restituisce

Restituisce il board inizializzato ai valori di base.(Scacchiera iniziale)

5.1.2.2 moves()

```
int* moves (
        torre_t board[7][7],
        char player,
        int * size,
        int * type_moves )
```

Genera l'array di mosse disponibili dati in input; il board e il giocatore attuale.
Assegna al parametro type_moves il tipo di mosse presenti nell'array.(spostamento singolo valore 1 - mangiata valore 2)

Nota

Se non è possibile effettuare nessuna mossa l'array non viene generato e a *typemoves viene assegnato -1.

Se *typemoves è stato assegnato a -1 ritorniamo quest'ultimo al posto del puntatore all'array.

La free del puntatore va effettuata quando a type_moves vengono assegnati i valori 1 o 2.

Parametri

in	*board*	Campo di gioco
in	*player*	Giocatore per cui devo controllare le mosse disponibili
in,out	*size*	Puntatore int a cui assegnamo la dimensione dell'array di mosse
in,out	*type_moves*	Puntatore int a cui viene assegnato il tipo di mosse presenti nell'array

Restituisce

Puntatore all'array dinamico di tipo int che contiene le mosse trovate.

Puntatore di typemoves in caso non sia disponibile alcuna mossa.

5.1.2.3 print_board()

```
void print_board (
        torre_t board[7][7] )
```

Esegue la stampa del board dato in input.

Parametri

in	*board*	board di gioco

5.2 Riferimenti per il file find.h

Funzioni inerenti alla ricerca di particolari informazioni.
.

```
#include "struct.h"
```

Funzioni

- char find_player (int x, int y, torre_t board[7][7])
- int find_t_height (int x, int y, torre_t board[7][7])
- int find_t_strenght (int x, int y, torre_t board[7][7])
- void find_t_enemy_composition (int x, int y, int ∗amiche, int ∗nemiche, torre_t board[7][7])
- void find_mid (int ∗xmid, int ∗ymid, int x, int y, int x1, int y1)
- int find_enhanced (int x, int y, torre_t board[7][7])
- int find_score (torre_t board[7][7], int x, int y, int x1, int y1)

5.2.1 Descrizione dettagliata

Funzioni inerenti alla ricerca di particolari informazioni.
.

Funzioni inerenti alla ricerca di particolari informazioni, le informazioni sono legate a una singola torre.

5.2.2 Documentazione delle funzioni

5.2.2.1 find_enhanced()

```
int find_enhanced (
        int x,
        int y,
        torre_t board[7][7] )
```

Trova se la pedina è un soldato o un ufficiale.http://www.lasca.org/

Parametri

in	*x*	Coordinata x
in	*y*	Coordinata y
in	*board*	Campo di gioco

Restituisce

Restituisce se la pedina è potenziata o meno un valore di tipo int.

Valori di ritorno

0	Soldato
1	Ufficiale

5.2.2.2 find_mid() `void find_mid (`

```
int * xmid,
int * ymid,
int x,
int y,
int x1,
int y1 )
```

Date le coordinate di due pedine che corrispondono agli estremi della diagonale genera le coordinate della pedina centrale.

Parametri

`in,out`	*xmid*	Coord x della pedina centrale trovata
`in,out`	*ymid*	Coord y della pedina centrale trovata
`in`	*x*	Coord x della pedina 1 data in input
`in`	*y*	Coord y della pedina 1 data in input
`in`	*x1*	Coord x della pedina 2 data in input
`in`	*y1*	Coord y della pedina 2 data in input

5.2.2.3 find_player() `char find_player (`

```
int x,
int y,
torre_t board[7][7] )
```

Trova il giocatore che possiede la torre di pedine nella posizione del campo indicata dai parametri x e y.

Parametri

`in`	*x*	Coordinata x
`in`	*y*	Coordinata y
`in`	*board*	Campo di gioco

Restituisce

Giocatore trovato

Valori di ritorno

'1'	Giocatore 1
'2'	Giocatore 2
'cancelletto'	Casella vuota

5.2.2.4 find_score()

```
int find_score (
        torre_t board[7][7],
        int x,
        int y,
        int x1,
        int y1 )
```

Determina il punteggio di una mossa.

Nota

Il punteggio della mossa è calcolato in base alla strategia implementata.

Parametri

in	*board*	Campo di gioco
in	*x*	Coord x inziale
in	*y*	Coord y iniziale
in	*x1*	Coord x1 finale
in	*y1*	Coord y1 finale

Restituisce

Restituisce il punteggio calcolato.

Sono possibili 29 differenti punteggi.

5.2.2.5 find_t_enemy_composition()

```
void find_t_enemy_composition (
        int x,
        int y,
        int * amiche,
        int * nemiche,
        torre_t board[7][7] )
```

Trova da quante pedine la torre del nemico è formata.(dividendo in pedine nemiche ed amiche)

Nota

nemiche=pedine del possessore della torre.

amiche=pedine non appartenenti al possessore della torre.

Parametri

in	*x*	Coordinata x
in	*y*	Coordinata y
in,out	*amiche*	Numero di pedine amiche trovate
in,out	*nemiche*	Numero di pedine nemiche trovate
in	*board*	Campo di gioco / torre_t

5.2.2.6 find_t_height()

```
int find_t_height (
        int x,
        int y,
        torre_t board[7][7] )
```

Trova l'altezza della torre.

Parametri

in	*x*	Coordinata x
in	*y*	Coordinata y
in	*board*	Campo di gioco

Restituisce

Altezza torre

Valori di ritorno

'0'	Altezza 1
'1'	Altezza 2
'2'	Altezza 3

5.2.2.7 find_t_strenght()

```
int find_t_strenght (
        int x,
        int y,
        torre_t board[7][7] )
```

Trova se la torre è debole o forte.(concetto di gioco di laska -> http://www.lasca.org/)

Parametri

in	*x*	Coordinata x
in	*y*	Coordinata y
in	*board*	Campo di gioco

Restituisce

Valore integer che identifica se la torre è forte o debole.

Valori di ritorno

'0'	Torre debole
'1'	Torre forte \reterr '-1' Eventuale errore (non possibile)

5.3 Riferimenti per il file game_engine.h

Funzioni relative al ciclo di gioco.
.

```
#include "board.h"
#include "find.h"
```

Funzioni

- void game_setup (char ∗p1, char ∗p2)
- int read_coord (char ∗in, int ∗xy, const int ∗arr, const int ∗dim)
- void print_move (int dim, int ∗arr, char player, char ∗p1, char ∗p2)
- void game_engine_cpu (torre_t board[7][7], char ∗p1)
- void game_engine_pvp (torre_t board[7][7], char ∗p1, char ∗p2)

5.3.1 Descrizione dettagliata

Funzioni relative al ciclo di gioco.
.

Giocatore vs Giocatore / Giocatore vs CPU .

5.3.2 Documentazione delle funzioni

5.3.2.1 game_engine_cpu() `void game_engine_cpu (`

```
torre_t board[7][7],
char * p1 )
```

Si occupa del ciclo di gioco di una partita player contro CPU.

Parametri

`in`	*board*	Campo di gioco
`in,out`	*p1*	Nickname giocatore 1

5.3.2.2 game_engine_pvp() `void game_engine_pvp (`

```
torre_t board[7][7],
char * p1,
char * p2 )
```

Si occupa del ciclo di gioco di una partita player contro player.

Parametri

in	*board*	Campo di gioco
in,out	*p1*	Nickname giocatore 1
in,out	*p2*	Nickname giocatore 2

5.3.2.3 game_setup() `void game_setup (`

```
char * p1,
char * p2 )
```

Determina la modalità di gioco per la partita corrente.

Parametri

in,out	*p1*	Nome del player_1
in,out	*p2*	Nome del player_2
in	*board[7][7]*	Campo di gioco

5.3.2.4 print_move() `void print_move (`

```
int dim,
int * arr,
char player,
char * p1,
char * p2 )
```

Stampa "user friendly" dell' array arr contenente le mosse disponibili per un certo giocatore.

Parametri

in	*dim*	Dimensione array arr
in,out	*arr*	Array con mosse disponibili
in	*player*	Giocatore che deve effettuare la mossa
in,out	*p1*	Nickname giocatore 1
in,out	*p2*	Nickname giocatore 2

5.3.2.5 read_coord() `int read_coord (`

```
char * in,
int * xy,
const int * arr,
const int * dim )
```

Lettura da tastiera delle coordinate relative alla mossa che si vuole effettuare.

Nota

in= stringa di caratteri su cui effettuo la lettura da tastiera.

xy= array di interi in cui inserisco le coordinate convertite prese da in.

arr= array con le mosse ditponibili.

dim= dimensione dell'array arr.

Parametri

in,out	*in*	Coordinate lette da tastiera
in,out	*xy*	Coordinate convertite
in,out	*arr*	Array con le mosse disponibili
in,out	*dim*	Dimensione arr

Restituisce

Ritorna un codice relativo alla correttezza dei dati inseriti da tastiera.

Valori di ritorno

-1	Errore nell'inserimento dati
0	Dati inseriti correttamente

5.4 Riferimenti per il file recursive.h

Funzione ricorsiva che si occupa della valutazione del punteggio dei singoli rami.

```
#include "struct.h"
#include "vector.h"
```

Funzioni

- void f_ricorsiva (torre_t board[7][7], char player, int value, int depth, int x, int y, int x1, int y1, vector_t *vector, int *counter, int tp)

5.4.1 Descrizione dettagliata

Funzione ricorsiva che si occupa della valutazione del punteggio dei singoli rami.

5.4.2 Documentazione delle funzioni

5.4.2.1 f_ricorsiva() `void f_ricorsiva (`

```
torre_t board[7][7],
char player,
int value,
int depth,
int x,
int y,
int x1,
int y1,
vector_t * vector,
int * counter,
int tp )
```

Trova il punteggio di ogni ramo generato dalla mossa passata alla funzione, salvandolo in un vettore.

Parametri

in	*board*	Campo di gioco
in	*player*	Giocatore
in	*value*	Valore iniziale della prima mossa passata -> diventa il punteggio del ramo.
in	*depth*	Profondità della mini-max custom
in	*x*	coord x iniziale (mossa)
in	*y*	coord y iniziale (mossa)
in	*x1*	coord x1 finale (mossa)
in	*y1*	coord y1 finale (mossa)
in,out	*vector*	Puntatore del vettore in cui andiamo ad inserire il punteggio di ogni ramo trovato
in,out	*counter*	Numero di rami trovati
in,out	*tp*	Tipo mossa da effettuare

5.5 Riferimenti per il file shift.h

Funzioni che operano uno scambio.

`#include "struct.h"`

Funzioni

- void shift (torre_t campo[7][7], int x, int y, int x1, int y1, int tp)
- void shift_player (char *player)

5.5.1 Descrizione dettagliata

Funzioni che operano uno scambio.

5.5.2 Documentazione delle funzioni

5.5.2.1 shift()

```
void shift (
        torre_t campo[7][7],
        int x,
        int y,
        int x1,
        int y1,
        int tp )
```

Modifica il campo in base alle mosse date in input.
Se mossa tipo 1 -> spostamento della pedina in una diagonale adiacente.
Se mossa tipo 2 -> Mangiata.

Parametri

in,out	*board*	Campo di gioco
in	*x*	coord x iniziale
in	*y*	coord y iniziale
in	*x1*	coord x1 finale
in	*y1*	coord y1 finaòe
in	*tp*	valore 1 o 2 in base al tipo di mossa

5.5.2.2 shift_player()

```
void shift_player (
        char * player )
```

Scambia il giocatore attuale con quello "inattivo".

Parametri

in,out	*player*	Giocatore attuale / char pointer

5.6 Riferimenti per il file struct.h

Struct inerenti alle componenti del gioco.

Composti

- struct pedina_t

Struct della pedina.

- struct torre_t

  Struct della torre.

5.6.1 Descrizione dettagliata

Struct inerenti alle componenti del gioco.

5.7 Riferimenti per il file vector.h

Libreria che implementa un vettore nel linguaggio C.

```
#include "stdio.h"
```

Composti

- struct vector

  Struct vector.

Ridefinizioni di tipo (typedef)

- typedef int **vdata_t**
- typedef struct vector **vector_t**

Funzioni

- vector_t * v_create ()
- void v_free (vector_t *v)
- void _v_check_index (const vector_t *v, size_t index)
- vdata_t v_get (const vector_t *v, size_t index)
- void _v_check_extend (vector_t *v)
- void v_push_back (vector_t *v, vdata_t value)

5.7.1 Descrizione dettagliata

Libreria che implementa un vettore nel linguaggio C.

5.7.2 Documentazione delle funzioni

5.7.2.1 _v_check_extend() `void _v_check_extend ( vector_t * v )`

Controlla se la size è uguale alla capacity, in quel caso effettua una realloc andando ad aumentare la memoria disponibile,
inoltre aggiorna il valore di capacity

Parametri

in	*v*	Il vettore.

5.7.2.2 _v_check_index()

```
void _v_check_index (
        const vector_t * v,
        size_t index )
```

Verifica che l'indice index sia minore di v-> size.

Parametri

in	*v*	Il vettore.
in	*index*	L'indice desiderato.

5.7.2.3 v_create()

```
vector_t* v_create ( )
```

Crea un nuovo vettore.

Nota

Se la malloc da errore, andiamo in exit failure.

5.7.2.4 v_free()

```
void v_free (
        vector_t * v )
```

Libera la memoria che era stata riservata al vettore.

5.7.2.5 v_get()

```
vdata_t v_get (
        const vector_t * v,
        size_t index )
```

Ritorna l'elemento nella posizione index.
Termina se l'indice è più grande di size.

Parametri

in	*v*	Il vettore.
in		

5.7.2.6 **v_push_back()** `void v_push_back (`

```
vector_t * v,
vdata_t value )
```

Inserisce value in coda al vettore.

Parametri

in	*v*	Il vettore.
in	*value*	Il valore da inserire.

Indice analitico