Novytskyy Maksym **Progetto miniLaska**

## Indice



## Introduzione

Il progetto miniLaska è una versione ridotta del gioco originale http://www.laska.org con le seguenti limitazioni:

- E’ possibile conquistare solo una pedina per turno
- La massima altezza delle pedine è tre, se si supera il limite la pedina viene tolta dal gioco
- Se la partita arriva a 100 mosse viene considerata come parità

Oltre a partite normali dove il giocatore può scegliere contro chi giocare è stata aggiunta la modalità per testare le cpu tra di loro.

## Vincoli tecnici

Per far funzionare il gioco bisogna assicurarsi di aver installato sul pc il compilatore gcc.
Successivamente si può usufruire del Makefile presente nella cartella che utilizzando il semplice comando “make” dentro al terminale. Che eseguirà il seguente comando:

```
gcc -ansi -pedantic -Wno-unused-result -O2 src/main.c src/board.c src/move.c
src/participants.c -o miniLaska
```

## Organizzazione e tempi

Il progetto è stato completamente svolto dal sottoscritto. Per realizzare il progetto esso è stato suddiviso in parti:
- Realizzare la struttura scacchiera, le funzioni per gestirla.
- Struttura per tenere conto dei movimenti disponibili.
- Sviluppare la cpu per poter giocare contro il giocatore.

N.B. Per insoddisfazione della prima versione cpu è stata realizzata una la seconda versione più efficace.

## Difficoltà progetto

Durante lo svolgimento si sono riscontrati alcuni problemi durante la realizzazione delle funzioni per gestire il movimento è la ricerca di movimenti disponibili, pero sono stati risolti con il debugging e utilizzando la funzione readGame in tal modo è stato possibile inserire casi specifici è individuare dove si trovava il problema. Successivamente si sono presentate delle difficoltà sul come realizzazione la prima cpu. Alla fine è stato deciso di utilizzare il metodo brute force cioè far girare la cpu per ogni possibile combinazione fino ad un certo punto. Dopo aver completato la prima cpu realizzare la seconda cpu è stata un esperienza più agevole perché la struttura rimane molto simile alla prima.

## Struttura

Il progetto è suddiviso in quattro parti. Il main contiene il menu principale e i vari sotto menu.
Il participants che contiene funzioni per gestire il giocatore e varie cpu. Board per gestire la scacchiera, movimenti e informazioni come turno e vincitore. Move che registra e gestisce i movimenti disponibili.

Main

Il main contiene il menu principale dal quale possiamo scegliere di giocare, fare test, cambiare profondità delle cpu e uscire.
Selezionando giocare passeremo alla funzione *gameManager* che chiedere delle configurazioni partita ad esempio contro di chi vogliamo giocare. Dopo aver scelto la partita comincia chiamando la funzione *gameControl* che gestisce i turni tra i vari partecipanti.
La seconda scelta chiama la funzione *testCpu* al interno della quale chiede quanti test eseguire, tipo di test (se una cpu contro un altra oppure tutte le cpu tra di loro) e se si sceglie la prima specificare le cpu.
Dopo aver selezionato viene chiamata la funzione *testManager* che esegue il numero di partite selezionate e ne tiene conto del punteggio(anche in questo caso viene usata la funzione *gameControl*).

Partecipants

Al interno del file sono presenti funzioni che gestiscono il giocatore e le cpu.
Per far giocare il giocatore si utilizza *playerMove* che trova e fa vedere le mosse disponibili e successivamente attende l'input la mossa scelta. Se il giocatore sbaglia quattro volte l'input il programma termina.
Abbiamo la semplice funzione *randomMoveCpu* che sceglie a caso tra le mosse disponibili.
Infine abbiamo le due cpu avanzate la *cpuScan* e *cpuEat*:
-La *cpuScan* utilizza la funzione ricorsiva *cpuScanRec* per passare per tutte le possibilità fino ad un certa profondità, quando arriva alla fine chiama la funzione *valueOfBoard* che assegna un punteggio alla scacchiera in base alle pedine presenti dentro, questi valori vengono ritornati dalla funzione e sommati. Quando la funzione ricorsiva finisce di funzionare il valore che ritorna viene successivamente diviso dal numero di combinazioni possibili che viene incrementato al interno della funzione ricorsiva. Terminato questo la funzione sceglie la mossa con il punteggio maggiore.
-La *cpuEat* utilizza la funzione ricorsiva *cpuEatRec* pero questa volta quando passa per le possibili mosse tiene conto delle catture dei giocatori aumentando o diminuendo il valore, alla fine questo valore viene riportato e sommato tra tutte le mosse possibili e infine diviso dal numero di combinazioni possibili. In più quando trova una vittoria o sconfitta viene sommato al valore un numero +-100 e diviso per il livello di profondità in tal modo la sconfitta o vittoria più vicina a più priorità

Move

Al suo interno è presente una struttura che tiene conto delle mosse disponibili, essa è composta da due contatori che tengono conto del numero di mosse normali e quelle di cattura, e due array dinamici(2d array pointer to pointer) che al loro interno sono contenute le coordinate delle pedine e coordinate di destinazione. Tra le funzioni per gestire i movimenti ci sono *createMove* usato per azzerare i contatori e mettere i puntatori a null, per registrare i movimenti vengono usate *regNormalMove* e *regEatMove* al loro interno vengono incrementati i contatori, assegnata memoria dinamica(malloc e realloc) e inseriti dentro le coordinate. Per pulire la memoria viene utilizzata la *destroyMove*, successivamente ci sono le funzioni per visualizzare le mosse *printNormaMoves* e *printEatMoves* in più al loro interno si possono visualizzare il valore di una mossa usato dalle cpu.
Infine sono presenti le funzioni di utilità per estrarre informazioni dalla struttura come *getCountEat*, *getCountNormal*, *getMoveEat* e *getMoveNormal*(per le ultime due funzioni bisogna specificare quale mossa si vuole estrarre e quale delle coordinate).

Struttura del array *eatMove* e *normalMove*

normal moveMove { {coord x start, coord y start, coord x end, coord y end}, … }
eatMove { {coord x start, coord y start, coord x eat, coord y eat, coord x end, coord y end}, ...}

Board

Contiene la scacchiera (3d array, pointer to pointer) allocata dinamicamente, alcune informazioni come turno e vincitore. Dentro si possono trovare anche funzioni per gestirla come *createBoard* e *destroyBoard* per allocare e

liberare la memoria, *initBoard* e *initPlayers* per inizializzare(riempire la scacchiera di spazzi vuoti e inserire le pedine dei giocatori). Funzioni per gestire i movimenti come *normalMovement* e *eatMovement*, e funzioni usate per individuare movimenti disponibili con *checkWhite* e *checkBlack* che al suo interno dopo aver individuato i movimenti chiamano funzioni di move.h per registrarli. Altre funzioni presenti sono *showBoard* per visualizzare la scacchiera *copyBoard* per copiare(se una delle due scacchiere e nulla il programma esce con errore) e altre semplici funzioni per utilità: *changeTurn*, *getTurn*, *getWinner*, *selectWinner*, *drawGame*.

## Risultati

Il programma funziona correttamente senza errori, è stata anche controllata la presenza di leak memoria tramite **valgrind** che non ha rilevato problemi.
Per testare la funzionalità della cpu sono stati realizzati alcuni test tra le varie cpu, ecco i risultati:

Random  white side: 2728 wins.  Random black side: 4145 wins.  Draws: 3127
Random  white side: 472 wins.  CpuScan black side: 9114 wins.  Draws: 414
Random  white side: 1 wins.  CpuEat black side: 9967 wins.  Draws: 32
CpuScan white side: 8552 wins.  Random black side: 737 wins.  Draws: 711
CpuScan white side: 6132 wins.  CpuScan black side: 21 wins.  Draws: 3847
CpuScan white side: 0 wins.  CpuEat black side: 8487 wins.  Draws: 1513
CpuEat white side: 9887 wins.  Random black side: 0 wins.  Draws: 113
CpuEat white side: 9990 wins.  CpuScan black side: 0 wins.  Draws: 10
CpuEat white side: 0 wins.  CpuEat black side: 28 wins.  Draws: 9972

Dalla qualle possiamo osservare la cpuScan per la maggior parte dei casi riesce a battere la random pero perde completamente controla cpuEat. In più vediamo che la cpuEat resce a distruggere tutti  nel giocare contro se stessa va nella maggior parte dei casi in parità.