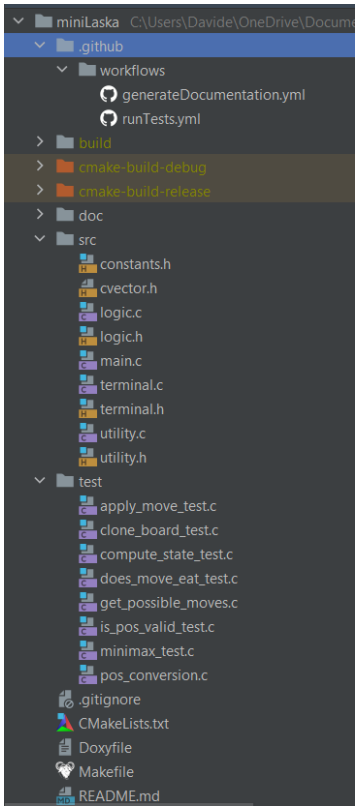
Relazione MiniLaska

Christian Bottone, Davide Cazzin, Giulia Papa – 19 gennaio 2020 – Università Ca' Foscari

Questa relazione intende descrivere la struttura del progetto miniLaska, spiegare come compilare il progetto, l'organizzazione del lavoro tra i diversi componenti del gruppo, le funzionalità aggiuntive che abbiamo ritenuto importante includere e le principali difficoltà incontrate.

1 Struttura del progetto



1.1 Suddivisione in file

Vista la dimensione del progetto abbiamo deciso di organizzare il progetto in diversi file, utilizzando i file header *.h per dichiarare le funzioni e i file source *.c per l'implementazione delle funzioni. Abbiamo suddiviso le diverse funzioni per categoria. Quelle che si occupano della logica del gioco laska nel file logic, mentre quelle che gestiscono la grafica del terminale nel file terminal, delle funzioni di aiuto nel file utility e le constanti nel file constants.h.

Grazie a questa suddivisione, se in un qualsiasi momento avessimo cambiato idea e deciso, ad esempio, di utilizzare una grafica 3D al posto di una 2D realizzata attraverso il terminale, sarebbe stato necessario soltanto aggiungere un nuovo file implementando la grafica 3D, senza dover riscrivere le funzioni relative alla logica del programma, in quanto le funzioni della grafica sono indipendenti da quelle della logica.

1.2 Git e GitHub

Dato che lo sviluppo del progetto è stato realizzato da più persone, abbiamo ritenuto molto importante utilizzare un Version Control Systems (VCS) per la gestione dei file e delle varie versioni, nel nostro caso abbiamo scelto di utilizzare git come VCS e GitHub per l'hosting. In questo modo siamo riusciti a lavorare tutti e tre sullo stesso codice ed a commentare i cambiamenti che i diversi membri del gruppo hanno apportato.

1.3 Tests

Abbiamo ritenuto importante creare delle classi di test, in modo tale che dopo aver scritto una funzione la si potesse provare indipendentemente dal resto del codice. In questo modo, oltre ad essere sicuri della correttezza di una funzione, nel caso in cui venga successivamente modificata in un refactoring del codice, non sarà necessario controllare manualmente tutti i casi possibili per assicurarsi se il comportamento del gioco non sia stato compromesso.

1.4 Documentazione

La documentazione generata con doxygen si trova all'interno della cartella denominata "doc". È presente sia la documentazione sotto forma di pagina web sia quella sotto forma di file PDF denominata con il nome di refman.pdf.

2 COME COMPILARE

Per compilare il progetto si può decidere se farsi aiutare da un IDE come CLion oppure compilare tramite terminale eseguendo il comando *make* nella cartella principale del progetto, in questo modo verrà generato l'eseguibile miniLaska all'interno della cartella "build".

Eseguendo *make debug* si può compilare il progetto in modalità debug, compilando quindi anche i test che possono essere eseguiti individualmente o lanciando *ctest* all'interno della cartella build.

3 ORGANIZZAZIONE DEL LAVORO

La parte di progettazione è stata svolta da Davide con l'aiuto di Giulia, insieme hanno definito le funzioni, le struct e gli enum necessari e li hanno suddivisi per categoria in file diversi come spiegato precedentemente. Successivamente le funzioni definite sono state suddivise tra i tre i membri del gruppo che le hanno implementate insieme a dei test utili per verificarne la loro accuratezza.

In linea di massima, Giulia ha realizzato le funzioni relative alla gestione della grafica del terminale e ed ha aiutato nel perfezionamento dell'algoritmo minimax; Christian ha implementato parte dei metodi che gestiscono la logica del gioco insieme ai relativi test; Davide ha scritto le basi del gioco, la documentazione delle funzioni, le restanti funzioni relative alla logica e si è assicurato che il programma funzionasse correttamente nella sua interezza.

Tutti i componenti del gruppo hanno controllato il codice scritto dagli altri. Durante lo sviluppo sono state aggiunte eventuali funzioni di aiuto quando ritenute necessarie.

4 FUNZIONALITÀ AGGIUNTIVE

4.1 DIFFICOLTÀ DEL COMPUTER

All'inizio della partita viene chiesto al giocatore la modalità di gioco, Player VS Player oppure Player VS Computer. In seguito, verrà chiesto con che colore vuole controllare il giocatore, bianco o nero. Nel caso in cui venga scelto Player VS Computer verrà anche chiesto il livello del computer che desidera affrontare tra facile, medio o difficile.

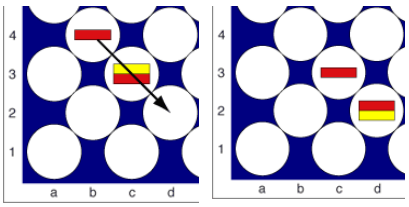
```
Select the game mode:
 1 - Player VS Player
 2 - Player VS Computer
2
Do you want to play with white or black pieces?
 1 - White
 2 - Black
1
Select the level of the computer
  1 - Easy
  2 - Medium
  3 - Hard
3
Allow hints [Y/N]: y
Clear console after every move [Y/N]: n
```

4.2 AIUTI

Ad inizio partita viene chiesto se si vogliono abilitare gli aiuti. Successivamente nel turno in cui si vuole richiedere un aiuto, basterà selezionare l'opzione 0 come descritto dall'interfaccia. Quando viene chiesto un aiuto il gioco suggerirà la mossa migliore che il giocatore può effettuare in quel turno, calcolata con l'algoritmo minimax e la stessa profondità utilizzata dal computer a difficoltà difficile.

```
Turn 1 - White to move
Possible moves:
0 - Hint
1 - a3-b4
2 - c3-b4
3 - c3-d4
4 - e3-d4
5 - e3-f4
6 - g3-f4
Type the number of the selected move: 0
Best move: c3-b4
```

4.3 Liberazione di una pedina



Questa regola, che non è ben descritta nel sito lasca.org e non è prevista in alcune versioni del gioco, prevede che se si cattura una pedina appartenente all'avversario nella quale c'è almeno una pedina del proprio colore la propria pedina verrà liberata e si cattura la pedina avversaria.

5 Principali difficoltà incontrate

5.1 ANSI C

Attenersi allo standard ANSI C non è stato semplice, infatti molte funzionalità comunemente utilizzate in molti altri linguaggi di programmazione al giorno d'oggi non erano ancora state implementate in C89. In particolare, abbiamo sentito la mancanza delle funzioni inline, i tipi bool e i commenti in una linea in stile C++. Abbiamo sopperito a queste mancanze utilizzando il define al posto delle funzioni inline, definendo il nostro tipo bool e utilizzando i commenti multi-riga soltanto su una singola riga.

5.2 Strategia del Computer

Per quanto riguarda la scrittura del codice, la parte più difficile è stata migliorare la strategia del computer. Infatti, inizialmente avevamo problemi negli stati finali della partita, la CPU non riusciva a vincere dato che per calcolare una strategia vincente bisognava considerare più di 9-10 mosse in avanti. Per questo, dopo aver implementato l'alpha-beta pruning, abbiamo migliorato la funzione utilizzata per calcolare l'euristica di una scacchiera.

Per calcolare l'euristica abbiamo deciso di tenere anche conto della differenza tra il numero di pedine della CPU e quelle dell'avversario e di considerare la distanza tra le pedine nel caso si fosse in vantaggio numerico di almeno tre pezzi. In particolare, più le pedine in vantaggio sono vicine a quelle avversarie meglio sarà considerata la formazione.

5.3 Memory Leaks

Una problematica incontrata è stata quella dei memory leaks. Infatti, durante lo sviluppo abbiamo notato che la memoria RAM utilizzata dal gioco aumentava ad ogni turno fino ad arrivare anche a mezzo gigabyte nel caso in cui la difficoltà del computer fosse stata impostata a difficile, quando le funzioni vengono eseguite ricorsivamente il maggior numero di volte. Era chiaro che era presente un memory leak da qualche parte. Dopo aver controllato più volte senza successo il codice alla ricerca di una variabile che veniva allocata ma non deallocata, abbiamo deciso di utilizzare Valgrind, un tool molto vasto che può anche essere utilizzato per controllare gli accessi alla memoria e vedere se si verificano memory leaks, lo abbiamo lanciato da una shell Bash con la seguente configurazione:

valgrind --tool=memcheck --leak-check=yes ./miniLaska

Grazie a Valgrind abbiamo trovato un memory leak che si verificava in un caso particolare e che ci era sfuggito a "occhio nudo". Dopo aver sistemato questo memory leak la memoria utilizzata dal gioco è nell'ordine dei kilobytes, come controllato con */usr/bin/time -v* e valgrind.