RELAZIONE PROGETTO ALPHAONE

TOMMASO POPPI 1201270

Progetto del corso di programmazione ad oggetti a.a 2019/2020

- ## Scopo

  Lo scopo del progetto è di realizzare un gioco di scacchi utilizzando il linguaggio c++, che è in grado di riconoscere le possibili mosse di ogni pedina, di gestire i turni, lo scacco e lo scacco matto, le mosse speciali come arrocco ed en passant ,e di indicare chi è il vincitore.

- ## Ambiente di sviluppo

  - Sistema operativo: Ubuntu
  - versione Qt : 5.9.5
  - Compilatore: GNU g++ 5.4.0

- ## Ore effettivamente richieste

  - analisi preliminare del problema: 2 ore
  - progettazione modello e gui: 5 ore
  - apprendimento libreria qt: 5 ore
  - codifica modello: 20 ore
  - codifica gui: 12 ore
  - debugging e testing: 5 ore

- ## Suddivisione del lavoro

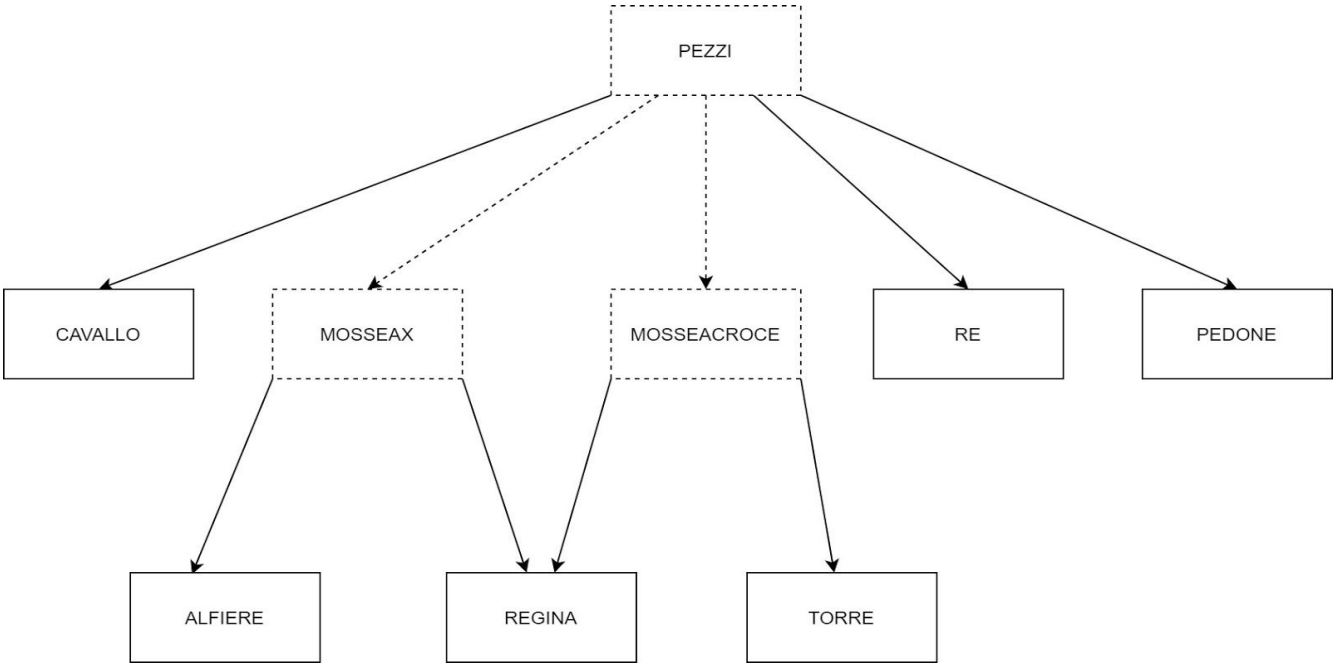
  Il progetto è stato per la maggior parte svolto insieme ad Andrea Mascari, in videoconferenza su zoom. Abbiamo utilizzato github per la condivisione del codice, solamente le seguenti classi sono state sviluppate individualmente:
  classi implementate da Andrea: Re, Cavallo, Torre, Mosseacroce, Chessbutton.
  classi implementate da me: Vector, Pedone, Mosseax, Alfiere, Regina.
  Il resto delle classi son state sviluppate in modo condiviso.

1. Introduzione

Il progetto utilizzerà un'implementazione model-view-controller quindi interfaccia grafica e modello son implementate separatamente ed entrambe comunicano attraverso l'uso di un controller.
Le classi che fanno parte del modello sono tutte le classi che fanno parte della gerarchia dei pezzi e la classe scacchiera, mentre le classi appartenenti alla vista sono sono "scacchi" ,"chessbutton" e "prombutton", che verranno discusse in seguito.

2. Modello

Gerarchia realizzata.



- **Pezzi**

  La classe astratta Pezzi è la classe base che sta in testa alla gerarchia. I campi dati presenti all'interno di questa classe sono:
  parent che è un puntatore alla classe scacchiera, che fa riferimento alla scacchiera dove è stato costruito il pezzo;
  colore che indica il colore della pedina;
  pos che indica la posizione attuale della pedina all'interno della scacchiera (quindi può essere $0 \leq pos < 64$);
  I metodi implementati ritornano il colore e la posizione della pedina visto che i campi dati sono protetti, mentre i metodi virtuali permettono di modificare la posizione grazie a setPosizione, visualizzare le mosse possibili del pezzo

grazie al metodo move() che ritorna un vettore di interi con le possibili posizioni, e infine clone che ritorna un puntatore a una copia del pezzo.
Il metodo move dunque è il metodo polimorfo principale di questa gerarchia.

- **Mosseax e Mosseacroce**
  Mosseax e mosseacroce son altre due classi astratte derivate da pezzi, hanno una derivazione virtuale visto che è presente una gerarchia a diamante, dunque questo permette di ridurre lo spreco di memoria e impedisce di produrre ambiguità e così permette di accedere alla classe pezzi. Queste due classi servono per descrivere il comportamento delle mosse dell'alfiere (mosseax) e della torre (mosseacroce). Infatti le classi alfiere e torre sono classi derivate da queste due.

- **Regina**
  regina è la classe che chiude il diamante quindi ha ereditarietà multipla, è derivata da mosseax e mosseacroce. L'ereditarietà multipla dunque permette di ereditare il metodo move() di mosseax che sarebbero le mosse possibili dell'alfiere e permette di ereditare il metodo move() di mosseacroce cioè le mosse possibili della torre. In regina viene ridefinito il metodo move() che esegue l'append dei due vettori ritornati dalle due chiamate delle move() delle classi che la derivano. In questo modo abbiamo sfruttato l'ereditarietà multipla per creare le mosse possibili della regina.
- **Re e Torre**
  Re è una classe derivata direttamente da pezzi, è una classe particolare perchè oltre alle mosse possibili può eseguire le due mosse speciali dell'arrocco (arrocco lungo e arrocco corto) che vengono ritornate in un vettore grazie al metodo MoveArrocco(), dunque nel metodo move() oltre alle mosse possibili del re viene effettuata un'append con le mosse dell'arrocco e viene restituito il tutto in un vettore.
  Per verificare se è possibile fare l'arrocco abbiamo bisogno di un campo dati in più rispetto alle altre pedine classiche, è il campo moved che serve per verificare se il re è stato mosso almeno una volta (moved=true) oppure no (moved=false), e dunque visto che nell'arrocco viene coinvolta anche la torre, anche la classe torre ha bisogno del campo dati moved insieme al metodo hasmoved() che ritorna moved.
- **Pedone**
  Anche la classe pedone è una classe particolare rispetto alle altre, appunto perchè è in grado di effettuare la mossa speciale dell' en passant, e inoltre deve controllare se puo effettuare il salto di due caselle se non si è ancora mossa. Quindi anche in questa classe viene creato un nuovo metodo enpassant() che ritorna un vettore con le mosse speciali e dunque in move viene effettuata l'append tra il vettore delle mosse possibili classiche e il

vettore ritornato da enpassant(). Per verificare se un mossa enpassant può essere fatta, ce bisogno di aggiungere un campo dati chiamato pass, questo viene messo a false di default quando viene creato un pedone, viene messo a true quando il pedone effettua la mossa di due caselle, e infine viene riportato a false quando passa un turno (in questo modo permette di effettuare le en passant solamente nel primo turno in cui è possibile effettuare le en passant, quindi se non viene eseguito alla prima mossa non è più possibile farlo).
Inoltre nella classe pedone viene aggiunto un campo dati firstmove (che di default è =false) che permette di verificare se è possibile effettuare la mossa di due caselle.

- **Scacchiera**

  Scacchiera è la classe principale del modello, il suo campo dati principale è board che rappresenta la scacchiera, infatti è un vettore di puntatori di pezzi e viene costruito a partire dal basso, quindi ogni posizione della scacchiera corrisponde all'elemento n-esimo del vettore.

56	57	58	59	60	61	62	63
48	49	50	51	52	53	54	55
40	41	42	43	44	45	46	47
32	33	34	35	36	37	38	39
24	25	26	27	28	29	30	31
16	17	18	19	20	21	22	23
8	9	10	11	12	13	14	15
0	1	2	3	4	5	6	7

  Il campo dati statico caselle serve per indicare la lunghezza del vettore, e il campo dati turn serve per gestire i turni.
  Il costruttore di scacchiera costruisce una scacchiera con tutti i pezzi a partire dalla posizione zero, quindi dal basso, iniziando a posizionare i pedoni bianchi e infine posiziona i neri a partire dalla posizione 48.
  Il distruttore si occupa di distruggere tutti gli elementi puntati dalla board così da non produrre spazzatura, quindi viene richiamato il distruttore di pezzi, per ogni pezzo presente nella scacchiera.
  Il costruttore di copia di scacchiera effettua delle chiamate polimorfe richiamando il metodo clone(), così da poter clonare tutti i pezzi della scacchiera che si vuole copiare.
  Il metodo doMove(pos1,pos2) si occupa di effettuare una mossa cioè di spostare una pedina dalla posizione pos1 a pos2 lanciando varie eccezioni:

-arrocco_exc() se la mossa effettuata è un arrocco.
-enpassant_exc() se la mossa effettuata è l'enpassant.
-mossa_illegale() se la mossa è illegale (cioè quando il re è sotto scacco e faccio una mossa che non elimina lo scacco)
-mossa_impossibile() se la mossa effettuata non rientra nelle mosse possibili di quel pedone.
In doMove andrò prima a provare la mossa su una scacchiera di prova e andrò poi a verificare se in questa scacchiera dopo aver effettuato la mossa c'è uno scacco quindi chiamo il metodo W(p) in prova, che fa esattamente questo.
Sempre nel metodo doMove viene chiamato enpassant e arrocco che si occupano di gestire le mosse speciali, e dopo aver effettuato la mossa domove chiama il metodo cambiaturno che si occupa di cambiare il turno, mettere i il campo dati pass dei pedoni a false, verifica se c'è una promozione e in caso positivo lancia l'eccezione promozione(), e infine controlla se c'è un vincitore chiamando il metodo winner.
Il metodo promozione si occupa di modificare il pedone con una delle pedine scelte da giocatore quindi questo metodo viene chiamato dal controller.
E infine il metodo reset permette di riportare la scacchiera nello stato iniziale iniziale.
Nei metodi doMove, Winner, Mosse vengono effettuate delle chiamate polimorfe al metodo move() di Pezzi.

3. vista

- ## scacchi

La classe scacchi implementa l'interfaccia grafica della GUI di alphaone utilizzando QVBoxLayout per il layout principale dove poi ci aggiungo il menù attraverso il metodo addMenu() e poi subito sotto al menu aggiungo un QGridLayout formato da una griglia di bottoni, i bottoni e la griglia vengono creati dal metodo addButtons() che aggiunge ad ogni posizione della griglia un'icona.
Il metodo finestrapromozione serve per far apparire una finestra quando avviene una promozione e da questa finestra si può scegliere la pedina, dunque nella finestra ogni pedina è rappresentata da un bottone e grazie a questo metodo, il click di ogni bottone viene collegato a una funzione del controller che permette di eseguire la promozione.
Il metodo mostravincitore e finestraeccezioni fa apparire una nuova finestra nel quale indica chi è il vincitore oppure le varie eccezioni.
I metodi cancellamosse, colora , aggiornaicone, Promozione, eliminapedina servono per modificare la griglia mentre si gioca, quindi vengono chiamate dal controller in base alle scelte dell' utente.

Il metodo reset serve per resettare la griglia e riportarla allo stato iniziale.
nota: spesso in questa classe vengono usate delle formule complesse come ad esempio “56-(k/8)*8+k%8”, questo è dovuto al fatto che l’implementazione del modello ha una griglia di riferimento differente rispetto alla griglia di riferimento della vista, infatti come spiegato sopra la board del modello parte dal basso a sinistra mentre nella vista la griglia è implementata in questo modo da qt:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

- ### ChessButton e PromButton

  Sono due classi che sono derivate dalla classe QPushButton.
  ChessButton è il tipo di bottone che è presente in ogni posizione della scacchiera che ha due campi dati importanti: pos che ricorda la posizione del bottone all’interno della griglia cioè del gridlayout, e red che serve per vedere se un bottone è colorato di rosso e quindi se posso spostare una pedina in quel bottone.
  Il segnale buttonclicked viene emesso quando eseguo lo slot clicked() cioè quando clicco il bottone perché nel costruttore faccio la connect di clicked() con il segnale clicked() implementato da QPushButton().
  PromButton è il tipo di bottone che è presente nella finestra che appare quando avviene una promozione. Ha bisogno di due campi dati: pos che memorizza la posizione in cui bisogna effettuare la promozione e tipo che indica se il bottone si riferisce a una promozione di torre di regina di cavallo o alfiere. Il segnale buttonclicked viene emesso quando eseguo lo slot clicked(), e lo slot si attiva quando viene cliccato il bottone perchè nel costruttore faccio la connect con clicked() che è un metodo definito da QPushButton.

4. Controller

Il controller ha bisogno di interagire sia con il modello e sia con la vista quindi abbiamo bisogno di due campi dati che sono dei puntatori a scacchiera (che rappresenta il modello), ed a scacchi (che rappresenta la vista).
Il controller permette di impostare qualsiasi vista grazie al metodo setView e quindi essere liberi di cambiare interfaccia grafica.

lo slot coloramuovi può:
-permettere di visualizzare le mosse possibili di una pedina, quindi esegue il metodo vedimosse (che esegue una chiamata polimorfa chiamando move())
-effettuare una mossa, quindi chiama il metodo EseguiMossa (che gestisce le eccezioni delle varie mosse)
questo slot viene chiamato quando un qualsiasi bottone di tipo chessbutton viene cliccato nella griglia (nella vista).
Lo slot eseguiPromozione viene chiamato quando viene cliccato un bottone di tipo prombutton, quindi quando appare la finestra della promozione e viene cliccato uno dei 4 bottoni.
Infine lo slot reset che chiama entrambi i metodi reset di scacchiera e vista.

❖ Signal e Slots

La classe ChessButton connette il segnale “clicked” derivato da QPushButton con lo slot clicked di ChessButton che a sua volta questo slot emette un segnale “buttonclicked” che è collegato al controller allo slot coloramuovi.
La classe PromButton utilizza la stessa implemetazione di ChessButton solamente che il segnale buttonclicked è collegato allo slot EseguiPromozione di controller, e inoltre questo segnale è anche collegato allo slot close() della finestra della promozione.
Nella vista (scacchi.h) è presente il menù con due funzionalità: exit e reset; exit è collegato alla chiusura del programma, mentre reset è collegato allo slot reset di controller.

- **Nota:**
  nei file consegnati è presente anche il file .pro (che abbiamo modificato aggiungendo QT +=widgets) dunque sarà possibile compilare il progetto utilizzando solamente i comandi qmake⇒make.