Università degli Studi di Urbino Carlo Bo
Corsi di Laurea in Informatica Applicata
Programmazione e Modellazione a Oggetti, a.a. 2023/2024

---

# Relazione per il Progetto del Corso di Programmazione e Modellazione a Oggetti

---

Elisa Geri mat. 299431

Alberto Iacobelli mat. 300792

Ottobre 2024

# Indice

# *Capitolo I*

# Analisi

## 1.1 Obiettivi

L'obiettivo del Team è lo sviluppo di un applicativo per giocare a Go (dama cinese).

Il titolo dell'applicativo prende il nome dal gioco originario "Go", antichissimo gioco da tavolo di strategia, nato in Cina e diffusosi successivamente in Giappone e Corea, dove ha avuto un'enorme popolarità.

Il Go si gioca su una griglia, posizionata su una scacchiera chiamata Goban, tipicamente composta da 19x19 linee che formano 361 intersezioni, sebbene siano comuni anche varianti su tavole di dimensioni inferiori come 9x9 o 13x13, utili soprattutto per principianti per partite più rapide.

I giocatori, "Bianco" e "Nero", si alternano nel piazzare una pietra (pedina) del proprio colore su un'intersezione libera della griglia.

Lo scopo principale del gioco è controllare il maggior numero di territori sulla griglia rispetto all'avversario, che avviene circondando con le proprie pietre determinate zone.

Le principali regole del Go sono:

- Piazzamento delle pietre: ogni giocatore, a turno, pone una pietra su una delle intersezioni libere. Una volta posizionata, la pietra non viene spostata a meno che non venga catturata.
- Libertà e cattura: Ogni pietra ha delle "libertà", cioè intersezioni libere adiacenti. Se una pietra o un gruppo di pietre viene completamente circondato, cioè perde tutte le proprie libertà, essa viene catturata e rimossa dal tavolo di gioco.
- Territorio: alla fine della partita, i giocatori contano le intersezioni che hanno circondato con le proprie pietre e si aggiungono il numero di pietre dell'avversario che sono riusciti a catturare.

Pertanto l'applicativo dovrà garantire il corretto svolgimento del gioco Go su una griglia 19x19, e si propone di offrire, nella sua forma classica, la possibilità di risolvere problemi di strategia e tattica, individuando le mosse migliori per raggiungere un obiettivo predefinito, migliorando così la propria comprensione strategica del gioco.

Grazie alla sua semplicità di regole e alla profondità strategica, Go è considerato uno dei giochi di strategia più complessi e affascinanti.

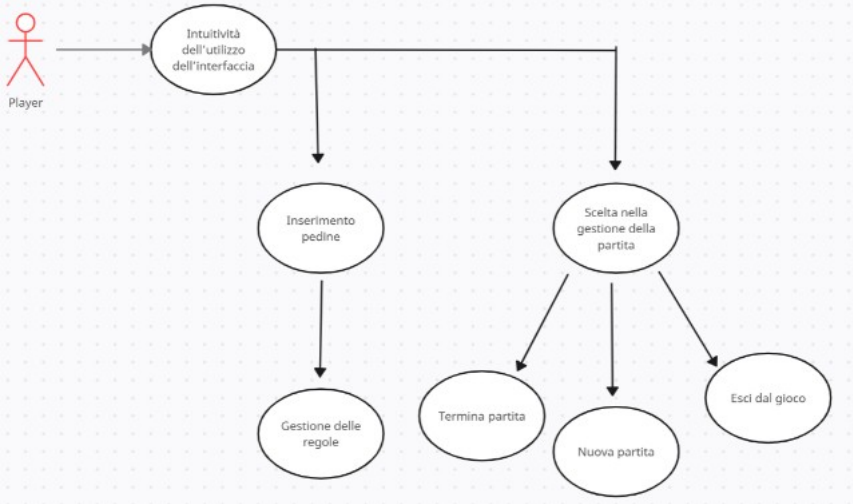



Diagramma dei casi d'uso

## 1.2 Requisiti

### 1.2.1 Requisiti funzionali

- L'applicazione permette di posizionare le pietre sulla griglia
- L'applicazione sarà composta da una griglia 19x19
- L'applicazione dovrà essere in grado di gestire una classica partita di Go, applicando quindi le regole base di quest'ultima
- Il giocatore deve avere la possibilità di terminare la partita corrente, mostrando il risultato raggiunto fino a quel momento
- Il giocatore deve avere la possibilità di iniziare una nuova partita
- Il giocatore deve avere la possibilità di uscire dall'applicazione
- Sarà possibile visualizzare il punteggio raggiunto dai giocatori, e il rispettivo vincitore
- L'applicazione emetterà un messaggio di errore nel caso in cui l'utente effettui una mossa non valida

### 1.2.2 Requisiti non funzionali

- Interfaccia utente semplice ed intuitiva, consente anche ai nuovi giocatori di apprendere velocemente il gioco
- Risposte rapide alle azioni dell'utente, come il piazzamento di una pietra

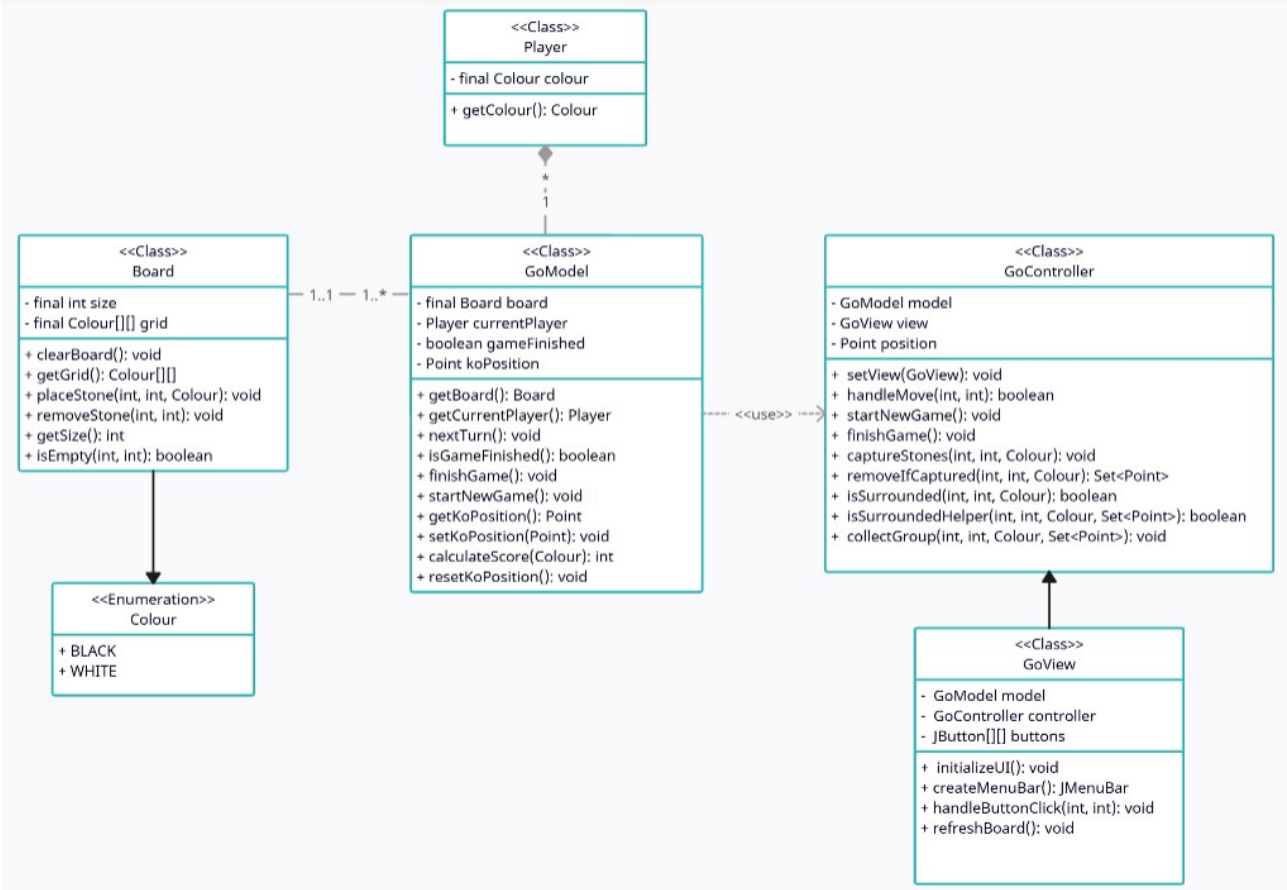
## 1.3 Modello del dominio

L'applicazione dovrà essere in grado di gestire una classica partita di Go.

Il tipo di partita è definito nel GoModel, che contiene informazioni sull'inizio e la fine del gioco, sul cambio turno, le regole base del Go e la determinazione del vincitore della partita.

Le principali componenti dell'applicazione sono:

- Player, che identifica il giocatore
- Board, che contiene le condizioni della scacchiera, che consente di effettuare delle mosse come: aggiungere delle pietre, rimuoverle e cambiare turno
- L'enumerato Colour, che identifica colore
- Il GoController, che  si occupa della gestione delle interazioni in maniera consequenziale
- La GoView, che si occupa della parte grafica



Schema UML dell'analisi del problema, con rappresentate le principali entità e i rapporti fra  loro

# *Capitolo II*

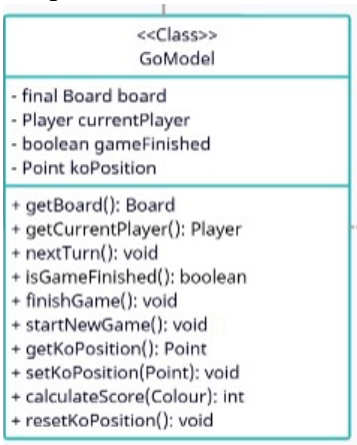
# Design

## 2.1 Architettura

Per lo sviluppo dell'architettura del gioco Go si è deciso di utilizzare il design pattern MVC(Model-View-Controller).
Questa architettura separa la logica di gioco dall'interfaccia grafica(GUI), migliorando la modularità e semplificandone lo sviluppo.

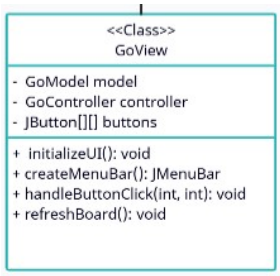
### 2.1.1 Model

Il modello rappresenta il gioco stesso, quindi contiene informazioni sull'inizio e la fine della partita, sulle interazioni e l'applicazione delle regole base.
Il suo scopo è quello di dare al Controller un modo per interagire con il gioco, facilitando l'implementazione delle mosse che l'utente può compiere e gestendo il cambio turno.



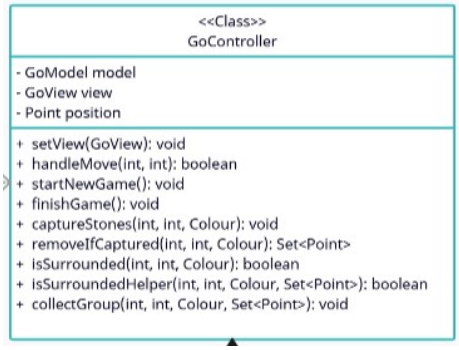

### 2.1.2 View

La View è la schermata dell'applicazione che gestisce la parte grafica, la User Experience e l'interazione con l'utente.
È compito della View registrare e informare il Controller delle interazioni dell'utente con l'applicazione, aspettando la sua risposta per eventuali cambiamenti nei dati.
La View è stata realizzata utilizzando Java Swing.
È appositamente separata dal Controller in modo tale che la modifica della libreria grafica richieda solo la riscrittura della parte relativa alla View, lasciando invariata quella relativa al Controller.



### 2.1.3 Controller

Il Controller è la componente che si occupa di gestire le interazioni dell'utente nella View in maniera consequenziale, comunicando così le modifiche al Model. Una volta che il Model ha completato l'elaborazione della richiesta di modifica, il Controller notificherà alla View, in modo che quest'ultima possa aggiornarsi in maniera coerente secondo le regole specificate nel Model.




## 2.2 Design dettagliato

Un problema che si è dovuto affrontare fin dall'inizio è stato quello di gestire le azioni a disposizione del giocatore, infatti essendo il Go un gioco basato sulla conquista dei territori occorreva andare a specificare quali fossero le mosse consentite e quali no.
Per questa ragione ci si è concentrati sullo sviluppo delle regole, così da poter ricreare una versione verosimile del Go.
Le regole principali del gioco sono:

- Regola del Ko, che previene situazioni di catture infinite reciproche e di ripetizione ciclica delle mosse, sostanzialmente impedisce ad un giocatore di fare una mossa che ripristinerebbe la posizione esatta che c'era sul tavolo prima della mossa dell'avversario.
  Ad esempio si suppone che il giocatore Nero cattura una pietra Bianca in una posizione, a questo punto il Bianco potrebbe voler catturare la pietra Nera appena piazzata, ma questo porterebbe alla stessa situazione iniziale. La regola del Ko vieta questa ripetizione immediata, e il Bianco dovrà fare una mossa diversa altrove.

- Regola della cattura delle pietre, permette ai giocatori di eliminare le pietre dell'avversario dal tavolo.
  La cattura avviene isolando le pietre nemiche in modo che non abbiano più libertà, o "libertà di movimento"

### 2.2.1 Elisa Geri

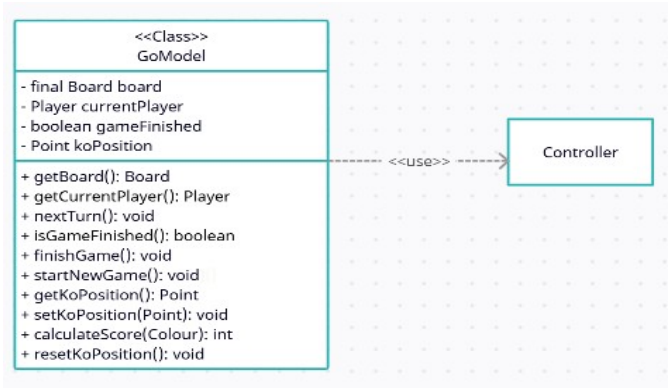
#### 2.2.1.1 GoModel

GoModel è una classe che fornisce una serie di metodi e attributi utili per la gestione delle azioni del giocatore, all'interno della quale è stato possibile risolvere il problema della regola del Ko.
Gli attributi di questa classe consentono di determinare in quale posizione il giocatore ha inserito le pietre, e di conseguenza verificare se la rispettiva cella sia vuota o già riempita.
I metodi al suo interno rappresentano una delle principali regole del Go, ossia la regola del Ko.

Quest'ultima viene gestita attraverso la creazione dei metodi:

- getKoPosition(), che ritorna la posizione del Ko
- setKoPosition() , che imposta la posizione del Ko
- resetKoPosition(), che resetta la posizione del Ko

Se il giocatore dovesse provare a posizionare una pietra nelle celle non consentite apparirà a schermo un messaggio d'errore.
Un altro punto saliente riguarda la gestione del cambio turno, la possibilità di iniziare una nuova partita e di controllarne lo stato.



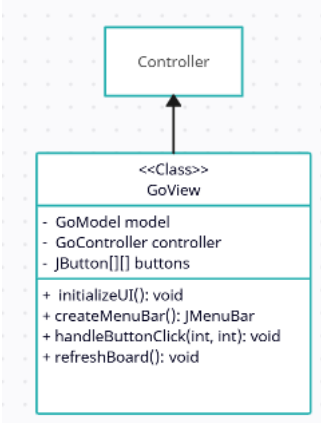

### 2.2.1.2 GoView

La GoView rappresenta l'interfaccia utente del gioco, la caratteristica fondamentale che deve presentare è la facilità di utilizzo, infatti l'utente deve essere in grado di capire con facilità il funzionamento del gioco.
Nel caso del gioco Go si è ottenuto ciò creando dei pulsanti per ciascuna posizione nel tabellone, in questo modo il giocatore può vedere chiaramente dove sta posizionando la pedina.
Un'altra parte importante riguarda il Menù, infatti per rendere migliore l'esperienza di gioco si è scelto l'inserimento di un menù contenete tre scelte che l'utente può compiere:

- Terminare la partita, in questo caso il gioco terminerà e verranno visualizzati a schermo il vincitore e il numero di pedine con le quali ha vinto

- Iniziare una nuova partita, prima di poter iniziare una nuova partita occorrerà terminare quella in corso, dopodiché si potrà selezionare questa opzione che consentirà di resettare il tabellone ed iniziare una nuova partita

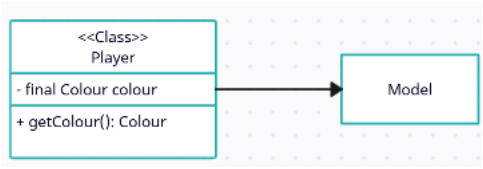
- Uscire dal gioco, in questo caso si chiuderà l'applicazione



### 2.2.2 Alberto Iacobelli

#### 2.2.2.1 Player

La classe Player rappresenta il giocatore nel gioco Go, specificando il colore delle sue pietre.
Il metodo principale è quello che va a restituire il colore del giocatore.

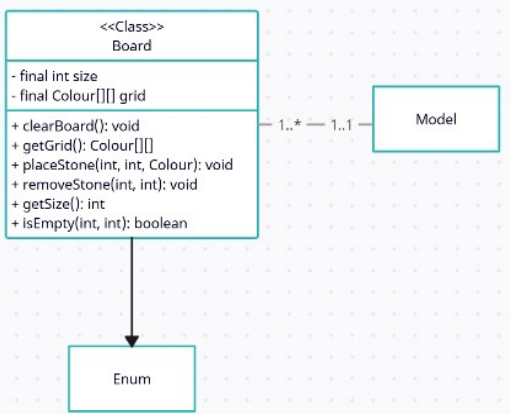



#### 2.2.2.2 Board

La Board è fondamentale in quanto rappresenta il tabellone di gioco, attraverso di essa vengono gestite la dimensione del Goban e la posizione di ciascuna pedina attraverso una griglia bidimensionale, dove ciascuna cella potrà essere di colore:

- Nero
- Bianco
- Vuoto(inteso come valore null)

Oltre a questo i metodi contenuti nella Board consentono di gestire la regola della cattura delle pietre. Alla Board si collega l'enumerazione Colour che definisce i due colori principali per le pedine.




#### 2.2.2.3 GoController

Rappresenta una parte fondamentale, in quanto funge da intermediario tra il modello(GoModel) e l'interfaccia(GoView).
Per gestire questa funzione si va a verificare, in base alla posizione scelta dall'utente:

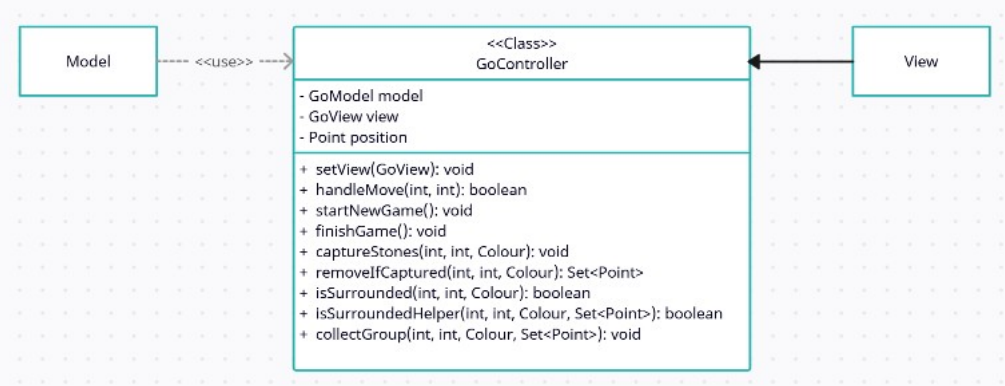
- Se la cella è vuota(non perché catturata), si colorerà del colore della pietra del giocatore(Bianco o Nero).

- Occupata da una pietra nera, si controlleranno i territori adiacenti, se è presente un gruppo di pietre nere, la pietra bianca(o pietre bianche) interna verrà catturata attraverso il metodo captureStones(). La cella assumerà il colore iniziale, poiché la pietra verrà rimossa dal metodo removeIfCaptured(). Inoltre si provvederà a controllare la regola del Ko utilizzando i metodi della GoModel.

- Occupata da una pietra bianca, si controlleranno i territori adiacenti, se è presente un gruppo di pietre bianche, la pietra nera(o pietre nere) interna verrà catturata attraverso il metodo captureStones(). La cella assumerà il colore iniziale, poiché la pietra verrà rimossa dal metodo removeIfCaptured(). Inoltre si provvederà a controllare la regola del Ko utilizzando i metodi della GoModel.

- Se la cella è vuota perché è stata precedentemente catturata una pietra, sarà possibile reinserire al suo interno un'altra pietra solo se il colore corrisponde a quello delle pietre che hanno conquistato il territorio.
  Questa tattica viene utilizzata come autodifesa dal giocatore.

Per verificare se una pietra è circondata in tutte le direzioni si utilizzano i metodi isSurrounded() e isSurroundedHelper().



# *Capitolo III*

# Sviluppo

## 3.1 Testing automatizzato

I test automatizzati sono stati utilizzati per rendere il  lavoro il più efficiente possibile.

Per implementare i test automatizzati è stato utilizzato Junit.

I test effettuati sono contenuti nel package Test, e si basano sul corretto funzionamento dei metodi, e sono:

- Test testBoardInizialization(), verifica che il tabellone venga creato con la dimensione corretta, che le caselle siano inizialmente vuote
- Test testPlaceStone(), verifica che le pietre possano essere posizionate correttamente e che la posizione sia aggiornata
- Test testCaptureStone(), verifica che le pietre vengano rimosse
- Test testKoRule(), simula una situazione di Ko e verifica che lo stesso stato non possa ripetersi immediatamente

## 3.2 Metodologia di lavoro

Nella fase iniziale del progetto si è discusso su quale specifica sarebbe stata ideale da implementare, dopo numerose idee si è optato per la creazione del gioco Go, ideando il diagramma dei casi d'uso.

Dopodiché si è ragionato sul grafico UML, che sarebbe stato utile poi per l'implementazione del gioco. È seguita la fase di implementazione vera e propria in cui è avvenuta la suddivisione dei lavori  all'interno del team.

Infine si è svolto lo studio della interfaccia grafica, in particolare su come dovesse essere più chiara e semplice possibile da utilizzare, poi è stata implementata ed è stato aggiunto un menù che consentisse all'utente di gestire la partita.

Il team è composto da  due membri e, vista la lontananza, gli incontri si sono svolti attraverso l'utilizzo della piattaforma Google Meet.

### 3.2.1 Porzione di Elisa Geri

Implementazione delle funzionalità del GoModel

Implementazione dell'interfaccia grafica GoView

### 3.2.1 Porzione di Alberto Iacobelli

Implementazione delle funzionalità del GoController

Implementazione delle funzionalità del Player

Implementazione delle funzionalità della Board e creazione dell'enumerato per distinguere le pietre