Università degli Studi di Urbino Carlo Bo
Corsi di Laurea in Informatica Applicata
Programmazione e Modellazione a Oggetti, a.a. 2022/2023

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

Elia Stefanini Matricola N° 298686
Gioele Mancinelli Matricola N° 301995

INDICE

1 Analisi

L'obiettivo del progetto è lo sviluppo di un'applicazione per giocare al "Gioco dell'Oca", un famosissimo e classico gioco da tavolo. Il Gioco dell'Oca è progettato per essere giocato su un unico dispositivo da un numero di giocatori compreso tra 2 e 5. Il gioco segue le regole tradizionali del Gioco dell'Oca, quindi ovviamente l'obiettivo sarà quello di guidare la propria pedina lungo un percorso numerato fino a raggiungere la casella finale e vincere la partita.

Descrizione del Gioco:

Il Gioco dell'Oca è un gioco da tavolo il cui scopo consiste nell'arrivare al termine di un percorso composto da una serie di caselle numerate da 1 a 63. I giocatori si spostano lungo il percorso in base al risultato ottenuto lanciando un dado. Ogni giocatore ha una propria pedina colorata che avanza del numero di caselle corrispondente al risultato del lancio del dado, per rendere il tutto più avvincente ed imprevedibile, alcune caselle "speciali" creeranno degli imprevisti che sposteranno gli equilibri della partita.

1.1 Requisiti

Requisiti dell'Applicazione "Gioco dell'Oca"
L'applicazione "Gioco dell'Oca" ha l'obiettivo di fornire una piattaforma per giocare al popolare gioco da tavolo "Gioco dell'Oca" su un computer. Il gioco segue le regole tradizionali del Gioco dell'Oca con alcune funzionalità aggiuntive per migliorare l'esperienza del giocatore.

Di seguito sono elencati i requisiti principali dell'applicazione:

- **Configurazione del Gioco:**
    - L'applicazione deve consentire ai partecipanti di selezionare il numero di giocatori (da 2 a 5) che parteciperanno al gioco.
    - Ogni giocatore deve avere un nome univoco (e non vuoto).
    - Ogni giocatore deve essere assegnato a un colore univoco.

- **Plancia di Gioco:**
    - L'applicazione deve mostrare una plancia di gioco virtuale, composta da una serie di caselle disposte in ordine.
    - La plancia deve contenere caselle numerate e alcune caselle speciali.
    - La casella finale è l'obiettivo del gioco (casella "Arrivo").

- **Lancio del Dado:**
    - L'applicazione deve consentire a ciascun giocatore di lanciare un dado virtuale.
    - Il risultato del lancio del dado deve determinare il numero di caselle che un giocatore deve avanzare sulla plancia.

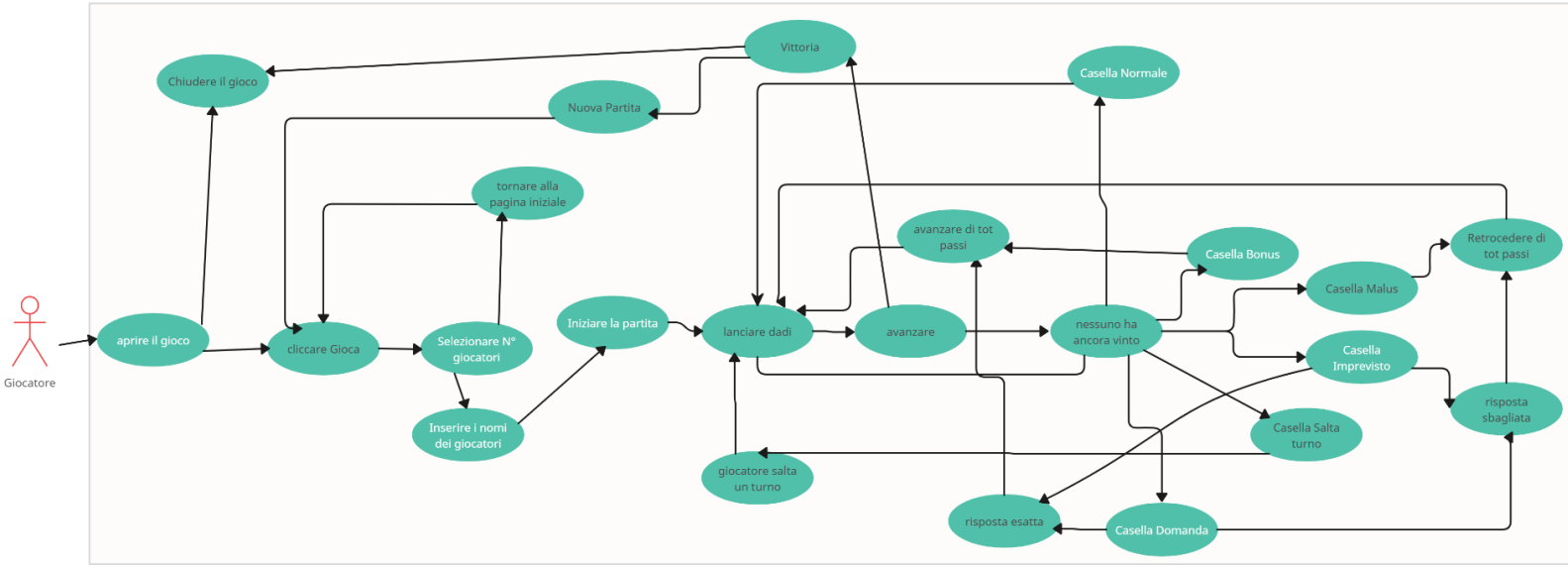
- **Turno dei Giocatori:**
    - Il gioco si svolge in turni, con ogni giocatore che completa il suo turno prima di passare al successivo.
    - Durante il proprio turno, un giocatore deve lanciare il dado, avanzare del numero di caselle risultante dal lancio del dado, e in caso rispettare gli effetti delle caselle speciali.

- **Effetti delle Caselle Speciali:**
    - Alcune caselle speciali hanno effetti che influenzano il percorso di un giocatore, ad esempio, alcune caselle possono far avanzare o retrocedere il giocatore di un numero specifico di caselle.
    - Le caselle speciali possono anche implicare azioni come saltare il turno o rispondere a delle domande.

- **Vittoria e Restart:**
    - Il gioco termina quando un giocatore raggiunge o supera la casella "Arrivo."
    - L'applicazione deve annunciare il giocatore vincitore.
    - Dopo una partita, i giocatori devono avere l'opzione di rigiocare ricominciando nuovamente il gioco.

- **Interfaccia Utente:**
    - L'applicazione deve fornire un'interfaccia utente intuitiva che visualizza la plancia di gioco, lo stato dei giocatori e i controlli per gestire il gioco.
    - Deve esserci un pulsante per lanciare il dado e un display per mostrare i risultati del dado.

Diagramma dei Casi d'Uso



1.2 Modello del dominio

Nel modello del dominio applicativo del Gioco dell'Oca, identifichiamo le principali entità e relazioni che compongono il problema da risolvere.

Entità Principali:

- **Giocatore:**
  - **Attributi:** nome, colore, posizione, deveSaltareTurno
  - **Descrizione:** Rappresenta un giocatore partecipante al gioco. Ogni giocatore ha un nome, un colore associato, una posizione corrente sulla plancia di gioco e un attributo che indica se deve saltare il turno.

- **Plancia:**
  - **Attributi:** caselle
  - **Descrizione:** Rappresenta la plancia di gioco, che è composta da una serie di caselle ordinate. La plancia contiene tutte le informazioni relative alle caselle della plancia.

- **Casella:**
  - **Attributi:** Numero, Simbolo
  - **Descrizione:** Rappresenta una casella sulla plancia di gioco. Ciascuna casella ha un numero e può avere un simbolo associato per distinguerla da caselle di altri tipi.

- **Casella Speciale:**
  - **Attributi:** Effetto
  - **Descrizione:** Rappresenta una casella speciale sulla plancia con un effetto specifico. Le caselle speciali possono influenzare il percorso di un giocatore, ad esempio, facendolo avanzare, tornare indietro, ecc.

- **Dado:**
  - **Attributi:** Numero di Facce
  - **Descrizione:** Rappresenta un dado virtuale utilizzato per determinare quanti passi un giocatore deve muovere la propria pedina avanti nella plancia.

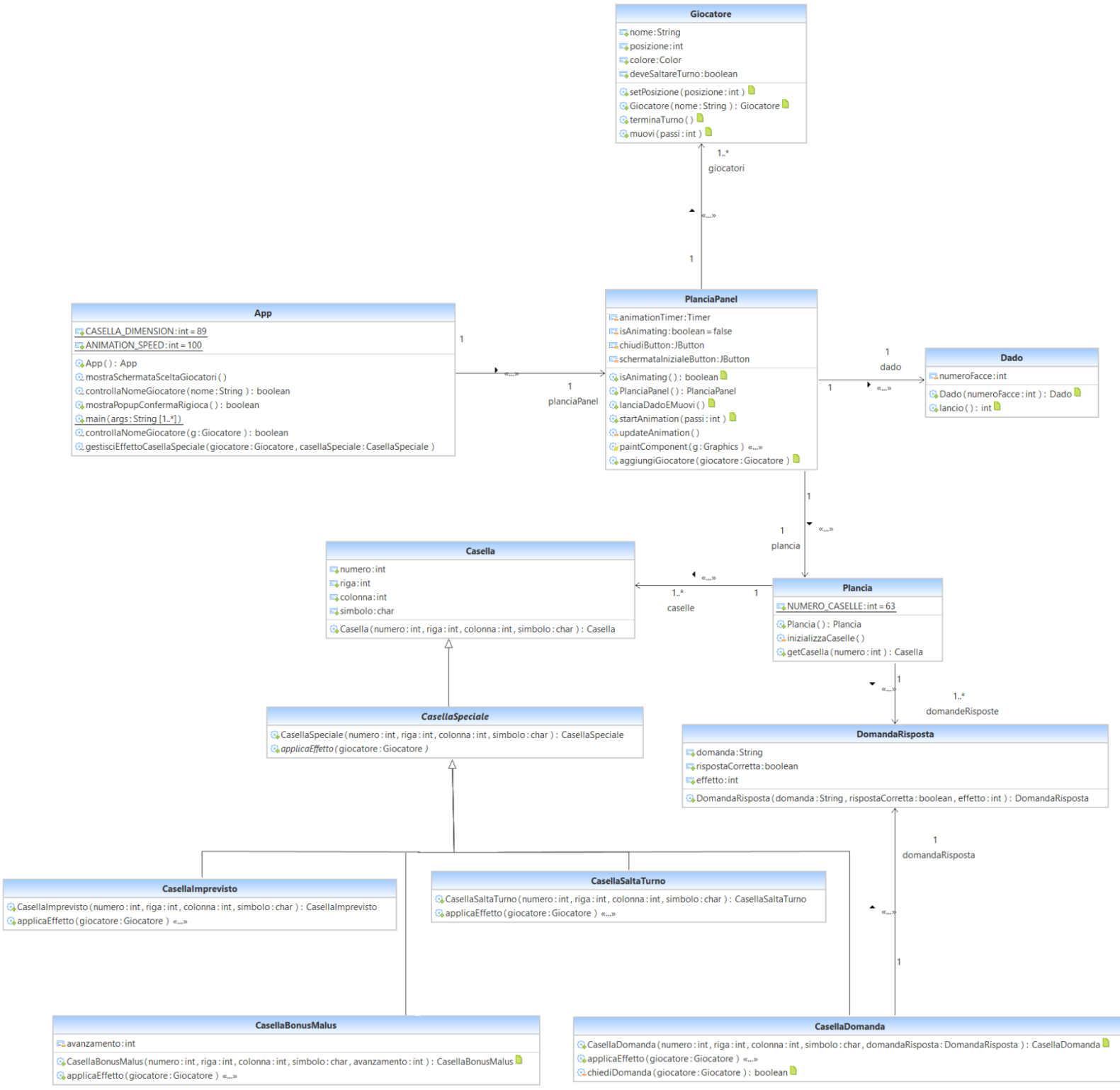
Relazioni Principali:

- **Giocatori sulla Plancia:**
    - Un'associazione tra la Plancia e i Giocatori tiene traccia delle posizioni dei giocatori sulla plancia.
    - Ciascun giocatore ha una posizione corrente sulla plancia.

- **Caselle sulla Plancia:**
    - La Plancia è costituita da una sequenza di Caselle ordinate.
    - Ogni casella ha un numero univoco.
    - Alcune caselle possono essere Caselle Speciali con effetti specifici.

Questo modello del dominio rappresenta le entità chiave e le relazioni che definiscono il problema del Gioco dell'Oca. Le interfacce e le classi del modello dell'applicazione saranno direttamente ispirate da questo modello del dominio durante la fase di progettazione.

Schema UML


Giocatore
nome:String
posizione:int
colore:Color
deveSaltareTurno:boolean
setPosizione(posizione:int)
Giocatore(nome:String):Giocatore
terminaTurno()
muovi(passi:int)
1..*
giocatori
«...»
1
App
CASELLA_DIMENSION:int=89
ANIMATION_SPEED:int=100
App():App
mostraSchermataSceltaGiocatori()
controllaNomeGiocatore(nome:String):boolean
mostraPopupConfermaRigioca():boolean
main(args:String[1..*])
controllaNomeGiocatore(g:Giocatore):boolean
gestisciEffettoCasellaSpeciale(giocatore:Giocatore, casellaSpeciale:CasellaSpeciale)
1
«...»
1
planciaPanel
PlanciaPanel
animationTimer:Timer
isAnimating:boolean=false
chiudiButton:JButton
schermataInizialeButton:JButton
isAnimating():boolean
PlanciaPanel():PlanciaPanel
lanciaDadoEMuovi()
startAnimation(passi:int)
updateAnimation()
paintComponent(g:Graphics) «...»
aggiungiGiocatore(giocatore:Giocatore)
1
dado
1
«...»
Dado
numeroFacce:int
Dado(numeroFacce:int):Dado
lancio():int
1
«...»
1
plancia
Casella
numero:int
riga:int
colonna:int
simbolo:char
Casella(numero:int, riga:int, colonna:int, simbolo:char):Casella
«...»
1..*
caselle
1
Plancia
NUMERO_CASELLE:int=63
Plancia():Plancia
inizializzaCaselle()
getCasella(numero:int):Casella
1
«...»
1..*
domandeRisposte
CasellaSpeciale
CasellaSpeciale(numero:int, riga:int, colonna:int, simbolo:char):CasellaSpeciale
applicaEffetto(giocatore:Giocatore)
DomandaRisposta
domanda:String
rispostaCorretta:boolean
effetto:int
DomandaRisposta(domanda:String, rispostaCorretta:boolean, effetto:int):DomandaRisposta
1
domandaRisposta
«...»
1
CasellaImprevisto
CasellaImprevisto(numero:int, riga:int, colonna:int, simbolo:char):CasellaImprevisto
applicaEffetto(giocatore:Giocatore) «...»
CasellaSaltaTurno
CasellaSaltaTurno(numero:int, riga:int, colonna:int, simbolo:char):CasellaSaltaTurno
applicaEffetto(giocatore:Giocatore) «...»
CasellaBonusMalus
avanzamento:int
CasellaBonusMalus(numero:int, riga:int, colonna:int, simbolo:char, avanzamento:int):CasellaBonusMalus
applicaEffetto(giocatore:Giocatore) «...»
CasellaDomanda
CasellaDomanda(numero:int, riga:int, colonna:int, simbolo:char, domandaRisposta:DomandaRisposta):CasellaDomanda
applicaEffetto(giocatore:Giocatore) «...»
chiediDomanda(giocatore:Giocatore):boolean

2 Design

2.1 Architettura

Per quanto riguarda l'architettura del Gioco dell'Oca, è doveroso e molto importante sottolineare che, sebbene non sia stato implementato esplicitamente il pattern Model-View-Controller (MVC), abbiamo adottato un approccio modulare che separa le diverse responsabilità all'interno dell'applicazione.

Abbiamo scelto di non utilizzare un framework architetturale specifico, in quanto la natura del gioco è relativamente semplice e mira a offrire un'esperienza autonoma su un singolo computer. Non è richiesta la gestione di complessi flussi di comunicazione tra componenti o il supporto a molteplici visualizzazioni o interfacce utente. Invece, abbiamo sviluppato un'architettura più leggera e diretta che è sufficiente per soddisfare i requisiti del nostro progetto.

Interazione tra Componenti

- **Interfaccia Utente (UI):**
  L'interfaccia utente è responsabile della visualizzazione del gioco e dell'interazione con l'utente. Utilizza Swing di Java per creare l'interfaccia grafica, inclusi pulsanti, pannelli e etichette. La UI è in grado di avviare nuove partite, gestire i lanci dei dadi e controllare i turni dei giocatori.

- **Logica del Gioco:**
  La logica del gioco è responsabile delle regole del gioco e delle meccaniche. Questo componente include classi come Plancia, Giocatore, Casella, Dado, ecc. Questa parte della logica del gioco si occupa di definire le regole del gioco e di gestire la logica per spostare i giocatori lungo il tabellone.

- **Controllore del Gioco:**
  La classe App funge da controllore principale. Essa coordina l'interazione tra l'interfaccia utente e la logica del gioco. La classe App è responsabile di inizializzare il gioco, controllare i turni dei giocatori, applicare gli effetti delle caselle speciali e determinare il vincitore, questa classe utilizza anche un timer per gestire l'animazione delle mosse dei giocatori.

2.2 Design dettagliato

In questa sezione, esamineremo alcuni aspetti del design del Gioco dell'Oca con un maggiore livello di dettaglio, fornendo informazioni più specifiche su come sono state affrontate alcune parti rilevanti dell'applicazione.

Gestione dei Giocatori

- **Problema:**
  È necessario tenere traccia dei giocatori, consentendo loro di muoversi sulla plancia, mantenendo le loro posizioni e gestendo le loro interazioni.
- **Soluzione**:
  Per gestire i giocatori, è stata creata una classe Giocatore. Questa classe contiene informazioni come il nome del giocatore, il colore associato e la posizione attuale sulla plancia. È stata scelta questa struttura dati per rappresentare in modo chiaro e intuitivo i giocatori e facilitare il controllo del gioco. Poiché il numero di giocatori è limitato a un massimo di cinque, non è stato necessario implementare soluzioni più complesse.
- **Schema UML:**
  Nella struttura dell'applicazione, la classe Giocatore è una semplice classe con i campi dati per nome, colore e posizione.

Gestione delle Caselle

- **Problema:**
  Il gioco richiede una plancia con caselle, alcune delle quali hanno caratteristiche speciali o eventi imprevisti. È necessario rappresentare in modo efficiente la plancia e gestire il comportamento delle caselle speciali.
- **Soluzione:**
  Per affrontare questo problema, è stata creata una struttura gerarchica di classi per la gestione delle caselle. La classe base di questa gerarchia è denominata **Casella**, che rappresenta una casella generica sulla plancia. Le caselle speciali sono gestite come sottoclassi di **Casella**. La classe **CasellaSpeciale** è una classe astratta che estende **Casella**, fornendo un'interfaccia comune per le caselle speciali. Le sottoclassi come **CasellaBonusMalus**, **CasellaSaltaTurno**, **CasellaImprevisto**, e **CasellaDomanda** implementano comportamenti specifici associati a ciascun tipo di casella speciale. Questa struttura gerarchica consente di differenziare facilmente il comportamento delle caselle e di applicare gli effetti desiderati quando un giocatore atterra su di esse.
  Ad esempio, CasellaBonusMalus fornisce un bonus o un malus di tot passi al giocatore, mentre CasellaSaltaTurno richiede al giocatore di saltare un turno…
- **Schema UML:**

Casella è la classe base, e le caselle speciali come **CasellaSpeciale**, **CasellaBonusMalus**, **CasellaSaltaTurno**, **CasellaImprevisto**, e **CasellaDomanda** ereditano da essa. Questa gerarchia semplifica il design e consente di applicare le varie logiche specifiche alle caselle speciali, consentendo una gestione efficiente ed estendibile delle caselle nel gioco.

Gestione del Dado

- **Problema:**
  È necessario il lancio del dado e visualizzare il risultato.
- **Soluzione:**
  Per gestire l'animazione del dado, è stata creata una classe Dado con un metodo lancio() che simula il lancio del dado restituendo un valore casuale tra 1 e 6. Questo design semplifica la gestione dell'animazione e consente di mostrare il risultato del dado in modo visuale in un'etichetta vicino al pulsante per lanciare il dado.
- **Schema UML:**
  Nella struttura UML, Dado è una classe indipendente utilizzata per generare il risultato del lancio del dado. Non sono stati applicati design pattern specifici per questa parte del progetto.

Gestione della Plancia di Gioco

- **Problema:**
  È necessario rappresentare la plancia di gioco e gestire il movimento dei giocatori su di essa.
- **Soluzione:**
  Per affrontare questo problema, è stata creata una classe PlanciaPanel. Questa classe rappresenta la plancia di gioco e gestisce l'interfaccia grafica per il tabellone. È responsabile della rappresentazione grafica delle caselle, dei giocatori sul tabellone e dell'animazione del movimento dei giocatori.
- **Schema UML:**
  Nello schema UML, PlanciaPanel è responsabile della rappresentazione visuale della plancia di gioco e dei giocatori. Questa classe integra le caselle e i giocatori sul tabellone. Non sono stati utilizzati design pattern specifici per questa parte del progetto.

In sintesi, il Gioco dell'Oca è stato sviluppato senza l'uso esplicito di design pattern formali in quanto l'approccio adottato è mirato a garantire una chiara comprensione del codice e una facile manutenzione, evitando complessità non necessarie.

3 Sviluppo

3.1 Testing automatizzato

Per effettuare il testing automatizzato la scelta è ricaduta sul framework JUnit 5. Il testing si è focalizzato in modo particolare sulle varie classi che costituiscono il modello alla base del programma,

Sono stati testati i metodi principali delle varie classi del modello con apposite classi di test, generando dati di prova per verificare il funzionamento dei vari metodi. In caso di fallimento dei test, sospendevamo lo sviluppo delle funzionalità fino all'individuazione di una correzione per risolvere il problema che comportava il fallimento.

Le classi di test sono:

- AppTest;
- CasellaTest;
- CasellaSpecialeTest;
- DadoTest;
- GiocatoreTest;
- PlanciaTest,

3.2 Metodologia di lavoro

Essendo questo progetto una prima esperienza sotto vari punti di vista, tra cui la dimensione particolarmente estesa rispetto a progetti passati e la realizzazione di una GUI, la fase di analisi e la fase di design sono state precedute e affiancate da molto tempo speso in ricerche, in particolar modo relative ai principi di funzionamento di Swing e dei suoi componenti

La fase di design dettagliato è partita dalla realizzazione, affiancata dal testing automatizzato, del modello alla base del gioco, delineando per ognuna delle classi in modo preliminare quelle che erano le proprietà che ci si aspettava che sarebbero state necessarie per garantire il corretto funzionamento del meccanismo alla base del gioco. Alcune classi del modello hanno poi subito delle modifiche nel corso delle fasi successive dello sviluppo.

Alcuni componenti delle viste, prima di essere inseriti nel programma, sono stati collaudati in un piccolo programma di test completamente separato dal progetto, in

quanto come anticipato sopra, essendo noi alle prime armi con progetti con interfaccia grafica, abbiamo potuto verificarne la corretta rappresentazione grafica e il corretto comportamento a seguito delle interazioni da parte dell'utente.

Per annotare idee relative all'implementazione e al miglioramento del progetto abbiamo utilizzato un semplice documento di testo condiviso su google drive, mentre per quanto riguarda la condivisione del progetto, ci abbiamo lavorato insieme ma sviluppando solo uno alla volta, anche per evitare conflitti e sincronizzazioni da gestire.

3.2.1 Divisione del lavoro

Nel corso dello sviluppo del nostro progetto di Gioco dell'Oca in Java, abbiamo avuto l'opportunità di lavorare come una squadra di due persone. La nostra collaborazione è stata fondamentale per garantire il successo complessivo del progetto e il raggiungimento degli obiettivi stabiliti.

La suddivisione del lavoro è stata accuratamente pianificata per sfruttare le nostre competenze e interessi individuali. Questa sezione esplorerà come abbiamo organizzato il lavoro tra i due membri del team:

- Elia Stefanini,
- Gioele Mancinelli.

Progettazione del Gioco

- **[Elia Stefanini]** si è concentrato sulla progettazione generale del gioco, sviluppando il concetto di base, definendo le regole, e creando uno schema iniziale dell'architettura del software.

- **[Gioele Mancinelli]** ha contribuito alla progettazione concentrandosi sull'interfaccia utente e sulla visualizzazione grafica del gioco, progettando la schermata iniziale e l'aspetto complessivo del tabellone.

Sviluppo del Backend

- **[Gioele Mancinelli]** ha contribuito allo sviluppo del backend, ha assunto la responsabilità principale per lo sviluppo del backend del gioco, implementando classi come Giocatore, Dado, e gestendo la logica del gioco, inclusi i movimenti dei giocatori e il controllo del turno.

- **[Elia Stefanini]** ha assunto la responsabilità principale per lo sviluppo del backend del gioco, concentrandosi sulla creazione di tutte le caselle e le caselle speciali, e sulla gestione degli effetti associati a queste caselle.

Sviluppo della Grafica

- **[Gioele Mancinelli]** ha svolto un ruolo chiave nello sviluppo del frontend, creando l'interfaccia utente grafica del gioco. Questo ha incluso la schermata iniziale, il tabellone di gioco, il lancio dei dadi e le finestre di dialogo interattive.
- **[Elia Stefanini]** ha collaborato nell'implementazione di elementi grafici, fornendo un feedback importante sul design e contribuendo all'integrazione dei componenti di gioco con l'interfaccia utente.

Test e Debugging

- Entrambi i membri del team hanno contribuito al processo di testing del gioco, individuando e risolvendo bug, e valutando il funzionamento generale del software. Questa fase è stata cruciale per garantire un'esperienza di gioco senza errori.

Coordinamento

- Entrambi i membri del team si sono impegnati a coordinare il lavoro, partecipando a videochiamate regolari per condividere idee e sviluppare insieme il progetto, discutere i progressi e risolvere eventuali sfide o ostacoli che si sono presentati.

Grazie a questa cooperazione, siamo riusciti a sviluppare un Gioco dell'Oca in Java che speriamo sarà divertente e coinvolgente per gli utenti. La nostra esperienza in questo progetto ci ha insegnato l'importanza della collaborazione e della divisione efficace del lavoro.

3.3 Note di sviluppo

Gradle: il setup del progetto è stato gestito con Gradle, del quale ho tentato di apprendere le basi del funzionamento, in modo da poter gestire in modo semplice l’eventuale necessità di importare librerie esterne, che tuttavia non sono state utilizzate.

4 Considerazioni finali

Siamo in generale abbastanza soddisfatti per l'esito del progetto e per il risultato, soprattutto in relazione alla nostra inesperienza in lavori di questa dimensione.

Questo progetto è stato anche la nostra prima esperienza, tra le altre cose, nella realizzazione di un'interfaccia grafica, nell'uso di Gradle e nel team work per progetti comunque di questa "portata", quindi, essendo comunque consapevoli della presenza di elementi migliorabili nel progetto, legati ad esempio ad una conoscenza poco approfondita dei pattern di progettazione, siamo comunque soddisfatti di aver ottenuto un programma funzionante e un gioco anche se semplice ma comunque divertente e un buon passatempo soprattutto se giocato tra amici.

Questo progetto è stato sicuramente uno tra i più difficili da noi affrontati però anche il più stimolante, perché ci ha costretto comunque a fare ricerche e a cercare di comprendere le soluzioni adatte alle nostre idee.

In futuro intendiamo approfondire gli studi sul linguaggio Java, focalizzandoci in particolare sugli aspetti con i quali abbiamo scarsa confidenza e che non sono stati usati in questo progetto, per fare in modo in futuro di migliorare la qualità del nostro codice.