1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

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

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

Membri:
Elia Lini, Federico De Piano, Udan Priyankarage

Indice

1 Analisi

1.1 analisi dei requisiti

L'obiettivo di questo progetto è di sviluppare un'applicazione in grado di simulare una partita di poker indiano a 4 giocatori, dove 1 è il giocatore e gli altri 3 sono simulati.
Il poker indiano è un gioco d'azzardo che si svolge in 10 round in cui ogni giocatore possiede 10 fiches e ad ogni round gli vengono servite 2 carte, una coperta ed una scoperta. Il mazzo è composto di 10 carte per seme che vanno dall'1 al 10 e i semi sono: quadri, cuori, fiori e picche. l'obiettivo del gioco è di avere più fiches possibili alla fine dei 10 round. Le fiches si ottengono vincendo un round, per vincere bisogna scommettere sulla propria combinazione, e per scommettere bisogna avere almeno 2 fiches: una per la tassa che bisogna pagare ad ogni round e l'altra per la puntata minima cioè, appunto, una fiches. La puntata massima per round è di quattro, sottolineando che la puntata non include la tassa, ciò vuol dire che in un turno si possono perdere al massimo 5 fiches. Se non si vuole perdere più del dovuto è possibile "foldare" comportando solo il pagamento della tassa senza necessità di puntare, tuttavia una volta "foldato" un round non è più possibile vincerlo. Le informazioni a disposizione sono la propria carta coperta che solo il giocatore che la possiede è in grado di vederla e la carta scoperta che invece sono in grado di vedere tutti. Ci sono tre possibili combinazioni: coppia, seme e maiale. Coppia come suggerisce il nome è quando in mano si ha una coppia di carte di egual valore(ad esempio coppia di otto), seme è quando si hanno in mano 2 carte dello stesso seme, e maiale quando non si ha nessuna delle 2 e quindi si punta sul valore della propria carta carta scoperta
Coppia è la combinazione che vale di più, quindi la coppia batte seme e maiale, vale più di maiale e maiale batte solo maiale. In caso di pareggio tra due coppie si stabilisce chi ha vinto guardando al valore delle carte che formano la coppia(coppia di 8 batte coppia di 6) nel caso entrambi abbiano la stessa coppia allora si controlla il seme della carta scoperta
perchè i semi hanno valore intrinseco, in particolare quadri > cuori > fiori > picche.
Nel caso di due semi allora se sono diversi si controlla chi ha il seme di valore maggiore altrimenti si controlla il valore della carta scoperta
Nel caso di due maiali si controlla prima il valore delle carte e poi se hanno lo stesso valore si controlla il seme. I round durano 15 secondi in cui avviene un solo giro di puntate.

1.2 Requisiti

Requisiti funzionali

- l’applicazione permette di puntare correttamente le fiches
- deve gestire e simulare una partita di poker indiano facendo rispettare le regole correttamente
- le partite sono composte da 10 round che sono tutti a tempo e allo scadere del tempo se un giocatore non ha puntato allora verrà considerato come se avesse foldato
- l'applicazione è in grado di simulare dei giocatori virtuali, in quanto le partite sono soltanto per un giocatore umano e 3 bot

Requisiti non funzionali

- l’applicazione deve essere facile da usare

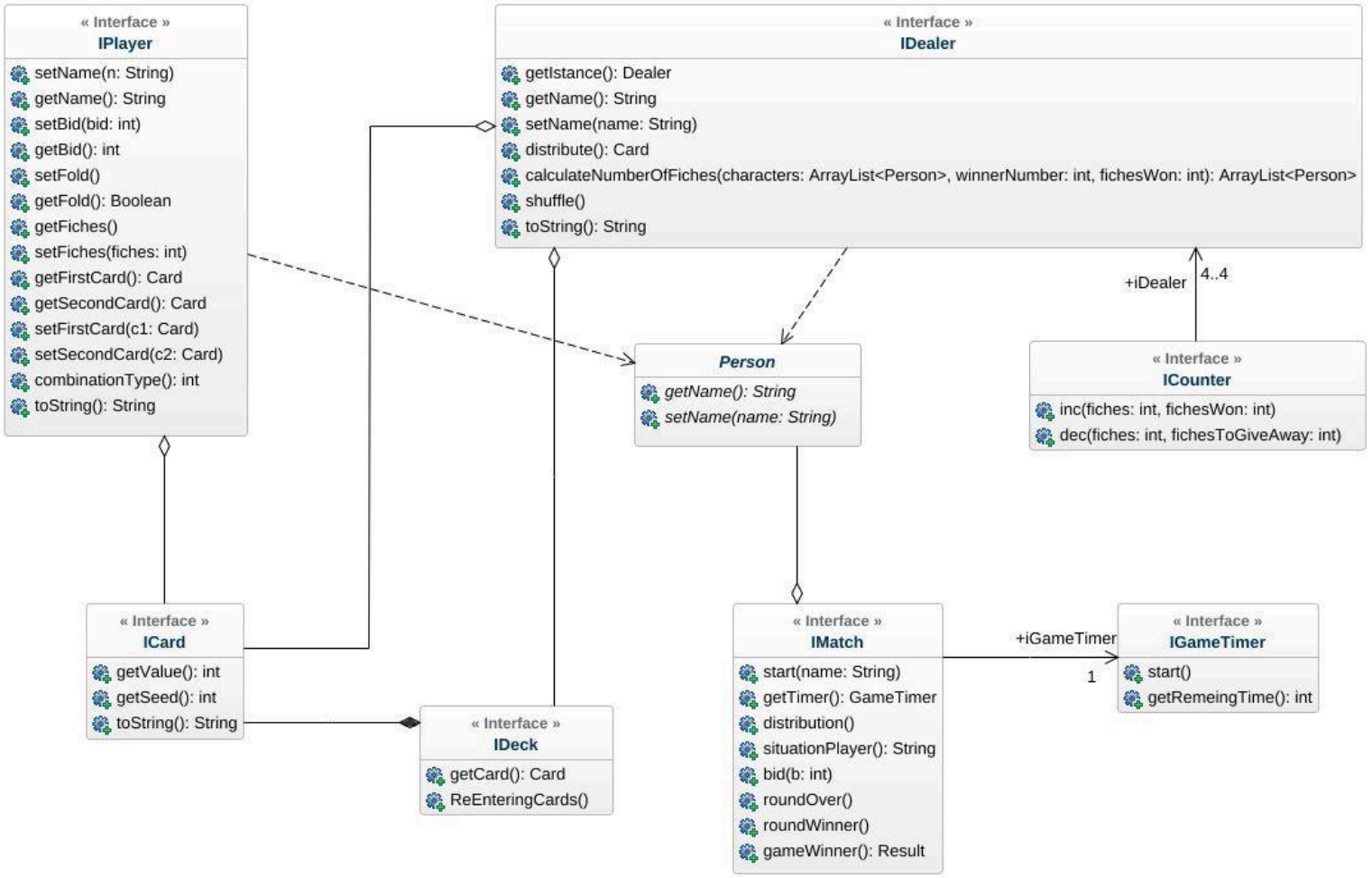
1.3 Modello del dominio

L’applicazione dovrà essere in grado di gestire partite di poker indiano con due carte per giocatore.

Una partita detta match identificata da IMatch, conterrà le principali informazioni sull’incontro:

a) I giocatori, identificati dall’entità IPlayer che sono in grado di puntare e foldare.
b) Il dealer, identificato dall’entità IDealer che distribuirà le carte, le vincite e che calcolerà il vincitore di ogni round.
c) Il Timer, utilizzato per determinare la durata dei 10 turni e identificato da IGameTimer.
d) Il player vincitore, alla conclusione del turno e alla conclusione del match.

Ogni classe concreta implementata la relativa interfaccia (il cui nome sarà: il nome della classe preceduto da ‘I’).
Person è una classe astratta estesa da Player e Dealer.
IDealer possiederà IDeck e lo userà per estrarre le carte e poi userà le carte, che i IPlayer contiene, per compiere il suo ruolo da mazziere distribuendo le carte a tutti i giocatori. IDeck conterrà le carte (identificate dall’entità ICard), ognuna delle quali avrà un seme e un numero.IDealer conterrà il counter (identificati dall’entità ICounter) per eseguire il calcolo dei punti.


« Interface »
IPlayer
setName(n: String)
getName(): String
setBid(bid: int)
getBid(): int
setFold()
getFold(): Boolean
getFiches()
setFiches(fiches: int)
getFirstCard(): Card
getSecondCard(): Card
setFirstCard(c1: Card)
setSecondCard(c2: Card)
combinationType(): int
toString(): String
« Interface »
IDealer
getIstance(): Dealer
getName(): String
setName(name: String)
distribute(): Card
calculateNumberOfFiches(characters: ArrayList<Person>, winnerNumber: int, fichesWon: int): ArrayList<Person>
shuffle()
toString(): String
+iDealer
4..4
Person
getName(): String
setName(name: String)
« Interface »
ICounter
inc(fiches: int, fichesWon: int)
dec(fiches: int, fichesToGiveAway: int)
« Interface »
ICard
getValue(): int
getSeed(): int
toString(): String
« Interface »
IDeck
getCard(): Card
ReEnteringCards()
« Interface »
IMatch
start(name: String)
getTimer(): GameTimer
distribution()
situationPlayer(): String
bid(b: int)
roundOver()
roundWinner()
gameWinner(): Result
+iGameTimer
1
« Interface »
IGameTimer
start()
getRemeingTime(): int

2 Design

2.1 Architettura

L'architettura del nostro software è basata sul pattern MVC (Model View Controller).

Il modello MVC separa gli aspetti dell'applicazione in tre componenti.

1) Model, contenente i dati e la logica dell'applicazione.
2) View, che visualizza i dati del modello e invia gli input dell'utente al controller.
3) Controller, che in base all'input dell'utente aggiorna View o Model.

Usando questo pattern si elimina ogni interazione diretta tra Model e View.

Il Controller fa uso dei metodi pubblici delle entità del modello per cambiare i dati in base agli eventi.

Abbiamo inoltre usato il pattern Observer per controllare e gestire il verificarsi di un evento. In base a questo modello di pattern i generatori di eventi (sono nella view) e gli osservatori di essi (sono nel modello) vengono gestiti dal controller (che si comporta da intermediario).

2.2 Design dettagliato

In questa sezione vogliamo approfondire alcuni elementi del design con maggior dettaglio.

Oltre ai pattern MVC e Observer è stato utilizzato il design pattern creazionale Singleton per evitare di avere più istanze di Dealer.

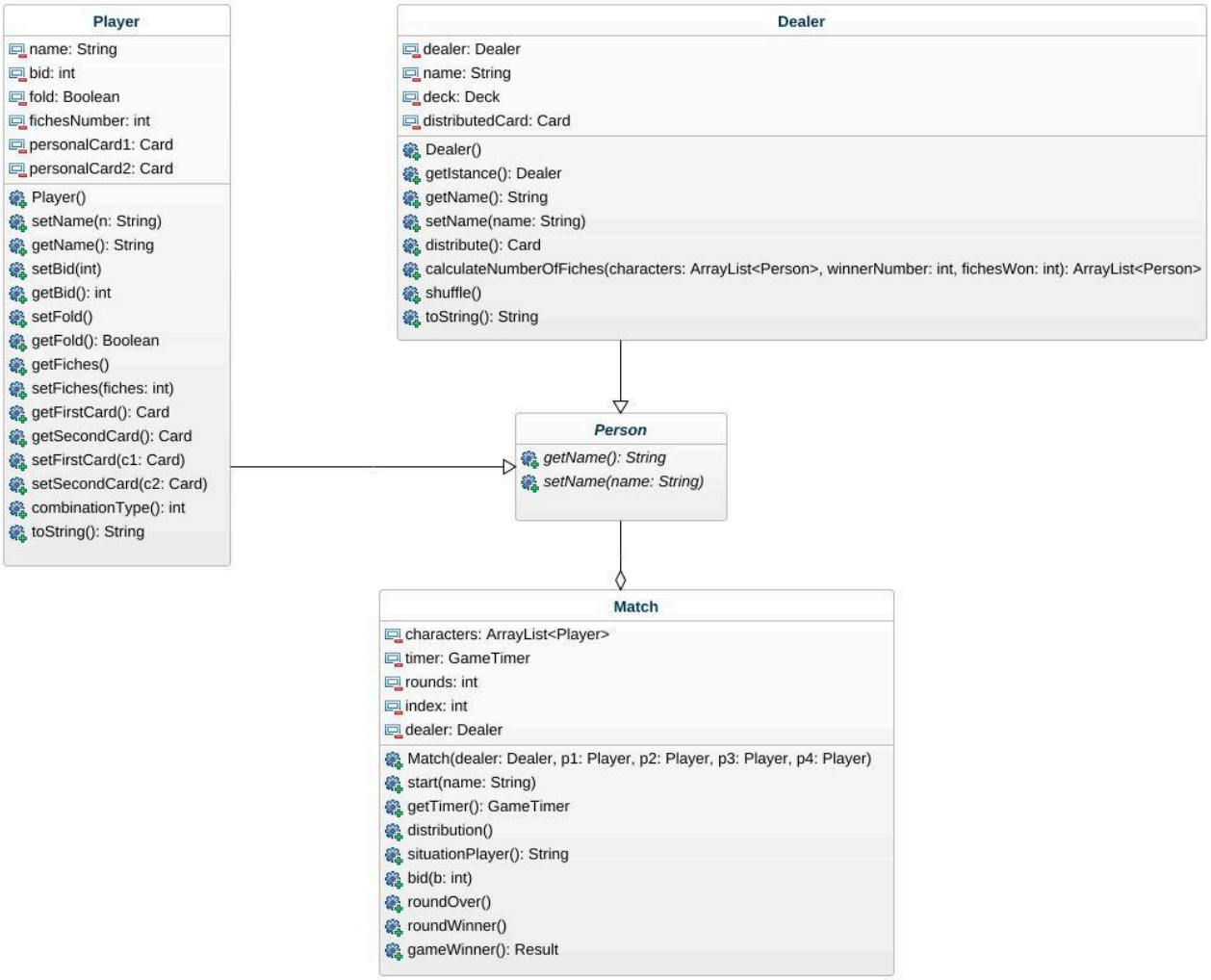
Per realizzarlo abbiamo messo privato il costruttore e aggiunto un metodo che restituisce un'istanza di Dealer nel caso non sia mai stato istanziato in precedenza.

Le entità che usufruiscono del singleton sono il Controller e la classe Match.

Abbiamo inoltre deciso di usare il design pattern Template Method nei metodi della classe Person, che consente di creare uno scheletro per le classi Player e Dealer.

2.3 Design del Modello

Nota: la seguente immagine è per i primi tre problemi affrontati.



2.3.1 Creazione del Player e del Dealer

Problema

Il dealer è un'entità che deve essere istanziata una sola volta.

Una parte fondamentale del gioco è distinguere tra i player il giocatore umano.

Soluzione

Per evitare la possibilità che venga istanziato un secondo dealer abbiamo usato il pattern creazionale Singleton.

Abbiamo deciso di inserire tutti i player in un arraylist per rendere più semplice l'identificazione dei player, in quanto il giocatore umano è e sarà sempre indicizzato a 0.

Questa soluzione consente uno scorrimento agevole dei player.

2.3.2 Gestione fold e bid

Problema

Nel poker si possono fare due cose o puntare o foldare, sarebbe sbagliato se qualcuno dei giocatori facesse sempre la stessa puntata e non foldasse mai.

Soluzione

Abbiamo stabilito una puntata massima di 4 ed una minima di 1 (senza considerare la tassa) per far si che la partita non possa finire in un solo round, i bot infatti fanno puntate sempre diverse all'interno di questo intervallo attraverso il metodo random della classe math, la possibilità dei bot di foldare è stata permessa ai bot attraverso la capacità di puntare 0 che il programma identifica come un fold .

2.3.3 Gestione dei Risultati

Problema

Al termine di ogni round bisogna identificare chi ha vinto e distribuire la vincita di conseguenza.

Soluzione

Il vincitore prende tutte le fiches puntate, suddetto vincitore viene stabilito dal dealer il quale ha un metodo che implementa un algoritmo, per determinare chi ha vinto seguendo le regole stabilite nell'analisi, quindi prima confrontando il valore della carta scoperta e se le carte scoperte hanno lo stesso valore allora controlla i semi delle carte, ovviamente solo di chi ha pareggiato.

2.4 Design del Controller

Problema

Il Controller ha il compito di fare l'intermediario tra View e Modello. Nel nostro caso esso deve aggiornare la View in caso di scelta dell'utente e alla fine di ogni turno, inoltre deve aggiornare i dati del modello come bid, fold e il nome dell'utente stesso.

Soluzione

Abbiamo scelto di usare una sola classe come Controller.

Questa classe Controller si compone delle classi: View, Modello, Result e GameTimer.

Il Controller memorizza le istanze di view e modello e contatta i metodi di queste ultime a fronte di un evento. La classe Controller sta in ascolto degli input dell'utente e gestisce l'ordine con cui chiamare le funzioni delle due istanze in modo da garantire una corretta esecuzione della partita. Il Controller è anche sempre al corrente dello stato della partita (YOU-WON, YOU-DREW, YOU-LOST, CONTINUE).

2.5 Design della View

La View ha il compito di mostrare all'utente lo stato della partita e raccogliere gli input dell'utente, inoltrandoli seguentemente al controller.

La View è composta da una singola entità: IndianPokerViewImpl.

IndianPokerViewImpl si occupa di stampare a schermo la situazione della partita e il timer, inoltre permette l'inserimento della scelta dell'utente.

Il giocatore può inserire la sua bid nell'apposita area di testo e confermare premendo il pulsante bid, può inoltre passare premendo fold e quittare premendo quit.

3 Sviluppo

3.1 Testing automatizzato

Per il sistema di entità sono state testate le caratteristiche principali di ciascuna delle macro-implementazioni ovvero:

• Per il package Characters sono state testate le classi Dealer e Player nella loro interezza.

• Per il package Elements sono stati testati tutti i metodi principali delle classi presenti.

• Per il package Enums non sono presenti metodi e quindi non si è rivelato necessario scrivere tests relativi a questo package .

• Per il package Match è stata testata la classe nella sua interezza.

3.1.1 Characters

• Costruttore, abbiamo testato la corretta costruzione della classe sfruttando gli attributi della classe presa in considerazione.

• Metodi per gestire le fiches del giocatore (Player).

• Metodi per gestire la bid e il fold del giocatore (Player).

• Metodi (del Dealer) per la determinazione del vincitore e delle fiches di tutti i giocatori.

• Design Pattern Singleton (del Dealer), abbiamo testato il funzionamento del pattern esaminato.

3.1.2 Elements

• Costruttore, abbiamo testato la corretta costruzione della classe sfruttando gli attributi della classe presa in considerazione.

• Metodi di inizializzazione come lo start() nel GameTimer e deckInitializetion() nel Deck.

• Metodi getter e setter di tutte le classi.

• Metodi (del Deck) per l'estrazione casuale.

3.1.3 Match

• Costruttore, abbiamo testato la corretta costruzione della classe sfruttando i suoi attributi.

• Metodi di inizializzazione come lo start() e distribution() sfruttando gli lo stato degli oggetti che gestiscono

• Metodi getter e setter.

• Metodi per la determinazione del vincitore.

3.2 Metodologia di lavoro

Strumenti Utilizzati:

• GitHub: repository principale.

• Eclipse: ambiente di sviluppo utilizzato.

• GenMyModel: utilizzato nel processo di creazione e modellazione dell'UML.

• Word: utilizzato per la scrittura della relazione.

Abbiamo suddiviso il lavoro tra noi (Elia Lini, Federico De Piano, Udan Priyankarage) in modo equo tenendo in considerazione l'opinione dei singoli membri.

Divisione lavoro di implementazione del codice delle classi (MODELLO)

Si specifica innanzitutto che i membri del gruppo responsabili per la creazione di una classe sono anche responsabili per la creazione del relativo test.

Elia Lini: responsabile della classi Card, Deck, GameTimer, inoltre è responsabile della creazione delle interfacce e la classe astratta Person.

Udan Priyankarage: responsabile delle classi Counter e Dealer (esclusa l'algoritmo di distribuzione della vincita).

Federico De Piano: responsabile della creazione della classe Player e degli algoritmi di distribuzione della vincita (calculateNumberOfFiches(ArrayList<Player> characters, **int** winnerNumber, **int** fichesWon)) in Player.

Nota: la classe Match, essendo una classe che va ad usare tutte le altre è stata sviluppata insieme, test compreso.

Divisione lavoro di implementazione del CONTROLLER

Il controller è stato sviluppato in cooperativa di ogni membro del gruppo.

Elia Lini: responsabile del metodo play().

Udan Priyankarage: responsabile del costruttore e del metodo privato game().

Federico De Piano: responsabile per la creazione di tutti gli altri metodi.

Divisione lavoro di implementazione della VIEW

La View è stata sviluppata in cooperativa di ogni membro del gruppo.

Elia Lini: responsabile del costruttore di IndianPokerViewImpl.

Federico De Piano: responsabile della creazione dei restanti metodi della View (IndianPokerViewImpl).

Udan Priyankarage: responsabile della creazione della creazione dell'observer (IndianPokerViewObserver) e dell'interfaccia della View (IndianPokerView)

Divisione lavoro creazione degli UML

Sono stati creati in presenza di tutti i membri del gruppo sulla base dell'opinione e delle necessità di tutti.

Divisione lavoro creazione testing

Ognuno è responsabile di testare il codice da lui/ loro creato.

Divisione lavoro creazione della relazione

Udan Priyankarage: Testing Automatizzato, Metodologia di lavoro

Elia Lini: Modello del dominio, Architettura

Federico De Piano: Analisi, Requisiti

Durante la scrittura del Design dettagliato ciascun membro è stato responsabile della propria sezione.

Udan Priyankarage: Creazione del Player e Dealer

Federico De Piano: Gestione fold e bid, Gestione dei Risultati

Elia Lini: Design del Controller, Design della View

Le Note di sviluppo sono un insieme di commenti scritti singolarmente da ogni singolo membro del gruppo.

3.3 Note di sviluppo

Tramite questo progetto siamo andati oltre a concetti puramente tecnici imparando l'uso dell'editor, l'importanza del lavoro di gruppo e del testing del codice.

- Elia Lini ha consolidato gli aspetti tecnici imparati durante il corso e l'uso di Git.

  Inoltre ha imparato l'importanza della creazione di un buon design a partire dalle idee iniziali per una corretta realizzazione del progetto.

  La difficoltà principale è stata quella di permettere allo user di giocare una partita che sembri più realistica possibile nonostante esso sia l'unico giocatore umano.

- Udan Hallawa Priyankarage: lo sviluppo di questo progetto ha permesso di migliorare le conoscenze tecniche del linguaggio Java, l'uso di Git e la creazione di Uml. Inoltre ha imparato l'importanza di una buona comunicazione tra i membri del team e il confronto con gli altri.

- Federico De Piano ha consolidato le conoscenze necessarie ad un'analisi dei requisiti completa, gestire la funzionalità di algoritmi complessi, le tecniche di java e della programmazione OO in generale, l’uso di git come strumento per un team di sviluppo e l’importanza dei meeting tra i membri anche in sessioni di programmazione di gruppo in contemporanea.

  La challenge principale è stata fare un algoritmo di distribuzione della vincita funzionante e soddisfacente e un’analisi che fosse esaustiva.