Relazione per il progetto software "Dots and Boxes" di Mattia Incoronato e Donoval Candolfi

Presentazione ed introduzione alla struttura del software

Prima di inziare lo sviluppo vero e proprio del programma, ci siamo ritrovati a dover analizzare con cura le specifiche del progetto e tutte le regole del gioco in questione. Per iniziare, abbiamo quindi iniziato a schematizzare su carta tutti gli elementi che sarebbero stati utili ai fini dello sviluppo, in modo tale da avere una visione più chiara e completa dei passi da fare.

Ragionando sui vari elementi di cui è composto Dots and Boxes, abbiamo pensato di andare a suddividere ognuno di questi, in modo tale da andare a lavorare su specifiche componenti che avrebbero poi collaborato insieme.
Difatti, dopo diverse idee proposte, abbiamo deciso come sarebbe dovuto essere strutturato il software.

Partendo dall'idea che il gioco è composto da una griglia di Box, il passo successivo è stato quello di pensare a come siano formate. Ogni Box è composta da 4 linee, ed ogni linea collega due punti distinti della griglia.
Ecco perchè abbiamo deciso di partire dall'elemento "Dot", ovvero il punto, grazie al quale è possibile creare la griglia di gioco, visto che quest'ultima non è altro che un insieme definito di punti.

Dopo il Dot, abbiamo sviluppato la classe Line e la classe Box. Dopodichè abbiamo iniziato a lavorare sulla struttura della griglia.

Oltre a questi elementi di gioco, abbiamo lavorato anche al Controller, ovvero l'arbitro della partita e alla struttura del Player, ovvero il giocatore che può essere reale o un bot controllato dal pc.

Tutti questi elementi vanno a collaborare fra di loro per creare poi la struttura completa del software.
Durante lo sviluppo non ci siamo soffermati in particolare sulle funzionalità o sulle opzioni di gioco, ma piuttosto sull'architettura. L'obiettivo era quello di rendere il software modulabile, suddiviso e pronto per eventuali estensioni future.

Architettura del software e suddivisione in pacchetti

1. Pacchetto "it.unicam.cs.pa.dotsandboxes.structure"

Questo pacchetto comprende le classi che formano la struttura degli elementi del gioco. La loro responsabilità è quella di sancire la struttura delle componenti usate durante le partite.

Classi:

- *Dot (public)*

La classe Dot rappresenta la struttura di un punto, che è formato da due interi che funzionano da coordinate (X e Y). Nel costruttore è stato inserito anche un controllo per assicurarsi che il punto venga creato fra coordinate maggiori o uguali a 0.
Inoltre è stato inserito un metodo chiamato *distance* che ha l'obiettivo di controllare che la distanza fra due punti non sia maggiore di 1.
Il Dot è l'elemento che andrà poi a formare la griglia.

-Line (public)

La linea è un altro elemento fondamentale del gioco, visto che ci serve per poter rappresentare sulla griglia l'unione di due punti.
Ogni *Line* è infatti costituita da 2 *Dot* e può essere costruita solamente fra due punti adiacenti e non sovrapposti.
Nella classe sono presenti i metodi Getters per ottenere il punto 1 o il punto 2 della linea, metodi per verificare se la linea è orizzontale o verticale ed un metodo per ottenere una "normal form".
Abbiamo indicato una forma normale delle linee in modo tale da poter effettuare confronti ragionando sempre sullo stesso tipo di struttura della linea.

-Box (public)

La classe Box sancisce la struttura di un "quadrato", ovvero quella struttura da costruire con le linee per poter ottenere punti.
È infatti costituito da 4 *Line* che formano i quattro lati della *Box.*
Nel costruttore è stato implementato un controllo per assicurarsi che le linee che formano il quadrato siano adiacenti.

-Grid (public, implementa IGrid)

La classe *Grid*, che implementa l'interfaccia *IGrid* ha come responsabilità quella di creare la matrice di gioco ed effettuare alcuni controlli.
La matrice è composta da un array bidimensionale di *Dot.*
All'interno della classe viene sfruttato il Listener per notificare se eventuali Box vengono create sulla griglia.
È presente un metodo per l'inserimento delle linee (a cui viene passato come paramentro una Linea definita dal giocatore) ed uno Stream utilizzato per controllare se ci sono le condizioni per creare un nuovo Box. In caso affermativo un nuovo Box viene creato ed aggiunto ad una lista.

Interfacce :

-BoxCreatedListener (public)
È un interfaccia che è stata poi utilizzata come Listener per notificare quando vengono create nuove *Box* sulla griglia.

-IGrid (public)
È un interfaccia che si occupa di definire i metodi che deve implementare la griglia.

-GridFactory (interfaccia funzionale)
È un interfaccia funzionale che serve per ritornare un istanza di una classe che implementa *IGrid*

2. Pacchetto "it.unicam.cs.pa.dotsandboxes.player"

In questo pacchetto sono presenti le classi che hanno la responsabilità di definire il giocatore reale e il bot.

Interfacce:

-Player (public)
Player è un interfaccia che definisce il metodo che tutti i giocatori devono implementare per giocare, ovvero *drawLine*.

Classi :

-AbstractPlayer (abstract , implementa player)
La classe AbstractPlayer implementa *Player* ed ha il compito di definre il costruttore base per qualunque tipo di giocatore (reale o bot). Il costruttore prende come parametro la griglia, in modo tale che ogni giocatore abbia come riferimento la matrice di gioco.

-RealPlayer (public, estende AbstractPlayer)
La classe RealPlayer definisce il giocatore reale. Estende *AbstractPlayer* e comprende di un metodo per l'inserimento da tastiera delle coordinate di una nuova linea.

-Bot (public, estende AbstractPlayer)
La classe *Bot* estende A*bstractPlayer* e a differenza del RealPlayer, il costruttore richiede non solo la griglia, ma anche una *Strategy,* ovvero una tattica con la quale giocare e decidere quale linea disegnare.

-PlayerWithPoints (public)
L'oggetto PlayerWithPoints ha lo scopo di tenere traccia dei punti dei giocatori. Il costruttore richiede infatti che venga passato un giocatore come paramentro.
Sono presenti i metodi per aggiungere punti ed il Getter per poter ritornare il valore dei punti accumulati.

3. Pacchetto "it.unicam.cs.pa.dotsandboxes.manager"

In questo pacchetto sono presenti le classi che hanno la responsabilità di gestire le meccaniche di gioco, come ad esempio la gestione dei turni.

Interfacce:

-Strategy (public)
L'interfaccia *Strategy* definisce il metodo che ogni strategia "reale" deve implementare. Definendo il nome del metodo che devono implementare le strategie, non c'è bisogno di effettuare modifiche anche al bot che utilizza la strategia, visto che continuerà a fare riferimento ad un metodo con lo stesso nome.

Classi :

-DumbStrategy (public , implementa Strategy)
La classe DumbStrategy definisce una strategia "poco furba", capace di giocare scegliendo casualmente dove posizionare la linea.
La *DumbStrategy* può essere implementata dal *Bot* per poter giocare in autonomia.

-Utils (public, final)
Implementa i metodi di Factory. Quando c'è bisogno di stampare una nuova griglia aggiornata, richiamo il metodo di Utils, in modo tale che si viene a creare una nuova griglia usata su un istanza di GridFactory.

-Controller (public)
Il controller è fondamentalmente l'arbitro della partita. Le sue responsabilità sono quelle di gestire i turni, controllare che ci siano ancora mosse disponibili e dichiarare il vincitore.
Il suo costruttore viene definito con due giocatori e la griglia passati come parametri.

-ControllerManager (public)
Il *ControllerManager* ha l'obiettivo di gestire il *Controller*. Mentre il Controller ci aiuta a capire cosa dobbiamo gestire, il ControllerManager ci aiuta a capire il come. Inoltre richiama *Utils* per incaricarlo allo svolgimento del suo ruolo.
Ha la responsabilità di definire le modalità di gioco per quanto riguarda i giocatori, quindi gli attori in campo.

4. Pacchetto "it.unicam.cs.pa.dotsandboxes

Nel pacchetto è presente solo la classe *Main*, che ha la responsabilità di avviare il gioco.
È presente una piccola interfaccia testuale per permettere all'utente di selezionare la modalità di gioco (giocatori reali o bot) e la grandezza della griglia.
Il gioco termina quando le mosse disponbili sono finite.

Estendibilità del codice

Il codice permette l'estendibilità grazie alla possibilità di poter inserire nuovi tipi di *RealPlayer,Bot* e nuovi tipi di *Strategy.* Le interfacce preposte sono in grado di garantire la possibilità di aggiungere nuovi elementi al gioco.
Inoltre il *ControllerManager* è in grado di gestire diversi giocatori con diverse strategie.

Cenni conclusivi

Durante lo sviluppo del progetto, abbiamo riscontrato sicuramente diverse difficoltà per quanto riguarda le scelte di design del software. Il lavoro svolto ci ha sicuramente permesso di comprendere le difficoltà del lavoro di gruppo e di quanto non sia sempre semplice far collidere visioni diverse.

Per lavorare in coppia abbiamo sfruttato le possibilità di collaborazione offerte da GitHub. Abbiamo infatti sfruttato, oltre al sito web, il plugin di GitHub per Eclipse. In questo modo è stato semplice aggiornare il progetto per entrambi senza dover effettuare esportazioni e condivsioni in altri modi.

Mattia Incoronato
Donoval Candolfi