1506
UNIVERSITÀ
DEGLI STUDI
DI URBINO
CARLO BO

UNIVERSITÀ DEGLI STUDI DI URBINO CARLO BO

Dipartimento di Scienze Pure e Applicate
Corso di Laurea in Informatica Applicata

Progetto Ingegneria del Software

GIOCO SCALE E SERPENTI

Autori:
Manoni Leonardo MAT[]
Incicco Matteo MAT[]

Anno Accademico 2015-2016

Indice

1 Specifica del Problema **1**
1.1 Progetto 1
1.2 Regole 1

2 Specifica dei requisiti **2**
2.1 Casi d'uso 2
2.2 Relazione tra casi d'uso 4

3 Analisi e Progettazione **7**
3.1 Ambiente e Linguaggio di programmazione 7
3.2 Scelte di Progetto 7
3.3 Interfaccia Grafica 9
3.4 Classi 11
3.5 Interfaccia 12
3.5.1 ImmettiNumeroGiocatori 12
3.5.2 ImmettiNomi 12
3.5.3 OrdineTiro 12
3.5.4 InizioGioco 12
3.6 Gioco 12
3.6.1 ControlloCaselleGioco 13
3.6.2 ControlloDestinazione 13
3.7 Giocatore 13
3.7.1 setNome 13
3.7.2 getNome 13
3.7.3 setNumeroUscito 13
3.7.4 getNumeroUscito 14
3.7.5 setPosizione 14
3.7.6 getPosizione 14
3.8 Mappa 14
3.8.1 Creazione 14
3.8.2 ControlloCaselle 14
3.8.3 ValidazioneDestinazione 14
3.9 Dado 15

3.9.1 Lancio 15
3.10 Casella 15
3.10.1 Vuota, Serpente, Scala 15
3.10.2 Partenza, Vittoria 15
3.11 Diagramma sequenziale 16

4 Implementazione **17**
4.1 Interfaccia 17
4.2 Gioco 22
4.3 Mappa 27
4.4 Dado 30
4.5 Giocatore 31
4.6 Casella 33
4.7 Scala 34
4.8 Serpente 35
4.9 Vuota 36
4.10 Partenza 37
4.11 Vittoria 38

5 Test **39**
5.1 White Box 39
5.1.1 Caso Numero 1 40
5.1.2 Caso Numero 2 41
5.1.3 Schema Logico 42

6 Compilazione ed esecuzione **44**
6.1 Istruzioni per la compilazione 44
6.2 Istruzioni per l'esecuzione 44

Capitolo 1

Specifica del Problema

1.1 Progetto

Nome Progetto: Gioco Scale e Serpenti, versione rivisitata del classico gioco delle Scale e dei Serpenti.

1.2 Regole

È un gioco da tavola nato in Inghilterra, dove due o più giocatori si sfidano cercando di raggiungere il prima possibile la casella più alta. I giocatori procedono del numero di caselle indicate dal lancio di un dado. All'interno delle caselle però si possono nascondere velenosi serpenti e utilissime scale. Se un giocatore capita nelle casella del serpente, scivolerà indietro, perdendo terreno, contrariamente, se un giocatore incappa in una casella contente una scala, riuscirà a balzare in avanti. Il giocatore vince quando arriva all'ultima casella decretando la fine del gioco.

Capitolo 2

Specifica dei requisiti

2.1 Casi d'uso

Di seguito riportiamo il diagramma dei casi d'uso, relativi al nostro progetto, per analizzare le funzionalità dell'applicazione. I casi d'uso costituiscono una tecnica utile a catturare i requisiti di un sistema e descrivere le interazioni tra l'utente ed il sistema stesso, fornendo un uno schema globale del comportamento del sistema.
Nel nostro caso è doveroso fare una distinzione tra GiocatoreNumero1 e Giocatore, questo perché ci sono delle azione che non tutti gli attori possono compiere. Per esempio solo il GiocatoreNumero1 può avviare il gioco e di conseguenza dovrà immettere il numero dei giocatori e i relativi nomi. Il Giocatore generico comprende tutti Giocatori diversi dal GiocatoreNumero1.

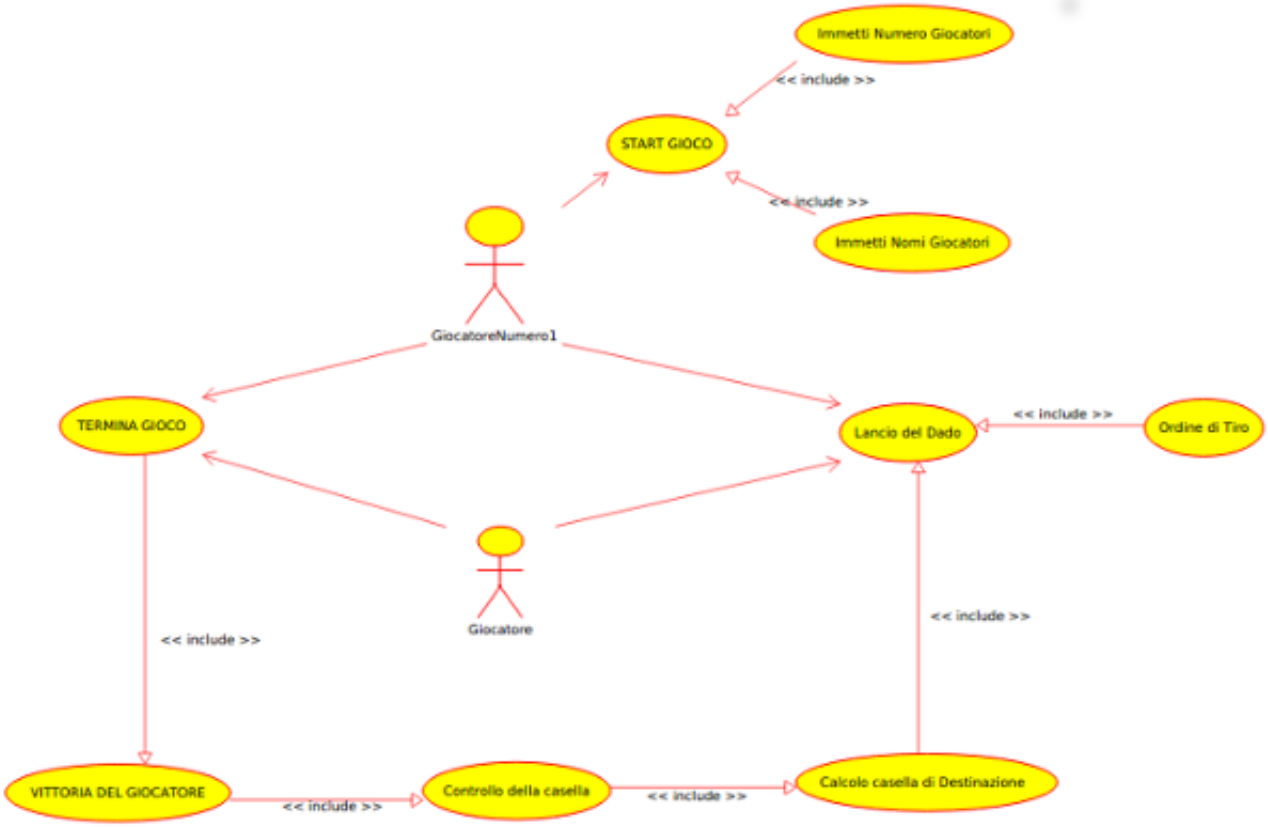

Figura 2.1: Casi d'uso

2.2 Relazione tra casi d'uso

Caso d'uso	Avvio applicazione (START GIOCO)
ID	#1
Attore	GiocatoreNumero1
Precondizioni	Il gioco deve essere eseguito
Corso d'azione base	Viene stampato a video il titolo e le istruzioni del gioco
Postcondizioni	Nessuna
Corso d'azione alternativo	Nessuna

Caso d'uso	Immetti numero di Giocatori
ID	#2
Attore	GiocatoreNumero1
Precondizioni	GiocatoreNumero1 deve aver eseguito il gioco
Corso d'azione base	Il gioco acquisisce il numero di giocatori
Postcondizioni	Il numero dei giocatori deve essere acquisito correttamente
Corso d'azione alternativo	c'è una validazione, se il numero dei giocatori acquisito non è compreso tra 2 e 4, si deve inserire il numero finché non è corretto

Caso d'uso	Immetti nome dei Giocatori
ID	#3
Attore	GiocatoreNumero1
Precondizioni	GiocatoreNumero1 deve aver inserito correttamente il numero dei Giocatori
Corso d'azione base	GiocatoreNumero1 inserisce i nomi di tutti i giocatori
Postcondizioni	I nomi dei giocatori devono essere acquisiti correttamente
Corso d'azione alternativo	Nessuna, non si può andare avanti finché non si inserisce i nomi. Se non si inserisce niente, e si preme invio, il programma associa il nome " " al giocatore

Caso d'uso	Lancio del Dado
ID	#4
Attore	Giocatore, GiocatoreNumero1
Precondizioni	GiocatoreNumero1 deve aver inserito tutte le informazioni richieste, quindi la fase di input deve essere terminata
Corso d'azione base	Viene lanciato un dado
Postcondizioni	Viene totalizzato un punteggio
Corso d'azione alternativo	Nessuna

Caso d'uso	Ordine di tiro
ID	#5
Attore	Applicazione (Gioco)
Precondizioni	Tutti i giocatori devono aver effettuato il lancio del dado
Corso d'azione base	l'applicazione ordina l'array dei Giocatori
Postcondizioni	I giocatori sono ordinati a seconda del punteggio totalizzato
Corso d'azione alternativo	Nessuna

Caso d'uso	Calcolo della Destinazione
ID	#6
Attore	Giocatore, GiocatoreNumero1
Precondizioni	Il giocatore deve aver lanciato il dado e deve avere totalizzato un numero
Corso d'azione base	Il giocatore si sposta nella casella di destinazione
Postcondizioni	Il giocatore sta in una casella (di possibili vari tipi)
Corso d'azione alternativo	Nessuna

Caso d'uso	Controllo della Casella
ID	#7
Attore	Giocatore, GiocatoreNumero1
Precondizioni	Il giocatore si trova in una nuova casella
Corso d'azione base	L'applicazione controlla il tipo di casella in cui si trova il giocatore
Postcondizioni	Avviene un'azione a seconda del tipo di casella in cui si trova il giocatore. Esempio: se la casella è di tipo Serpente si andrà al caso d'uso #4
Corso d'azione alternativo	Nessuna

Caso d'uso	Vittoria
ID	#8
Attore	Giocatore, GiocatoreNumero1
Precondizioni	Il giocatore deve aver controllato la casella in cui si trova
Corso d'azione base	Il giocatore ha vinto e stampa a video un messaggio relativo alla sua vincita
Postcondizioni	Fine del gioco (l'applicazione termina)
Corso d'azione alternativo	Nessuna

Caso d'uso	Termina Gioco
ID	#9
Attore	Giocatore, GiocatoreNumero1
Precondizioni	Un giocatore deve aver vinto, ovvero deve essere capitato sulla casella Vittoria (#8)
Corso d'azione base	Il programma termina
Postcondizioni	Nessuna
Corso d'azione alternativo	Se non ci sono vincitori significa che un giocatore ha terminato l'applicazione. Avendo un interfaccia a linea di comando, si può terminare il gioco con Control+C

Capitolo 3

Analisi e Progettazione

3.1 Ambiente e Linguaggio di programmazione

Per realizzare questo progetto è stato utilizzato C#, un linguaggio di programmazione orientato agli oggetti. C# nasce da Microsoft per la realizzazione del Framework .Net e successivamente viene approvato come standard ECMA.

L'ambiente di sviluppo adottato è Visual Studio Community 2013 che fa parte della piattaforma Microsoft Visual Studio. Visual Studio integra la tecnologia IntelliSense la quale permette di correggere eventuali errori sintattici (ed alcuni logici) senza compilare l'applicazione, possiede un debugger interno per il rilevamento e la correzione degli errori logici nel codice in runtime e fornisce diversi strumenti per l'analisi prestazionale.

Entrambe le scelte rispettano rigorosamente i vincoli della consegna del progetto.

3.2 Scelte di Progetto

Per realizzare il nostro progetto, abbiamo seguito per la maggior parte le regole tradizionali del gioco, modificandole a nostro piacimento ove abbiamo ritenuto necessario. Di seguito riportiamo le principali scelte di progetto adottate.

Tabellone
Il nostro Tabellone, chiamato Mappa in fase di sviluppo, è formato da 50 caselle.

Le 50 caselle saranno di tipo:
- Partenza (posizione 0)
-Vuota
-Serpente
-Scala

-Vittoria(psizione 49)

Ad ogni esecuzione del programma, avremo mappe completamente diverse, che seguiranno un vero e proprio senso logico ma non sempre avranno stessi numeri di caselle Serpente e di casella Scale. Un vincolo che abbiamo adottato sono: tre caselle Vuote dopo una casella Serpente o Scale.

Dado
Il dado utilizzato è il classico dado da gioco, cubico con le facce marcate da numeri naturali che vanno da uno a sei. Abbiamo deciso di utilizzare solo un dado alla volta.

Giocatori
Il numero di giocatori massimo ammesso per ogni partita è di 4. Anche se il numero dei giocatori non influisce nel funzionamento dell'applicazione, è stata presa questa decisione per rendere più veloce e meno complicata l'esecuzione del gioco.

Funzionamento Gioco
Fase iniziale: Il Giocatore, all'avvio del programma, inserisce il numero dei Giocatori (da due a quattro) e i relativi nomi.

Fase 2: Tutti i partecipanti tirano il dado per stabilire l'ordine di Tiro durante il gioco. Il giocatore con punteggio pisubù alto sarà il primo a giocare. In caso di parità, l'ordine sarà determinato dalla precedenza di tiro, esempio:

giocatori: Leonardo, Matteo, Ugo
Leonardo lancia e totalizza 4
Matteo lancia e totalizza 6
Ugo lancia e totalizza 4

Ordine di lancio:
Matteo
Leonardo
Ugo

In questo caso, Matteo, ha totalizzato 6, sarà al primo a giocare. Leonardo e Ugo hanno entrambi totalizzato 4, ma Leonardo avendo tirato prima di Ugo, giocherà per secondo.

Fase 3: Stabilito l'ordine di tiro, inizia il gioco vero e proprio. Tutti i giocatori partono dalla casella Partenza. Matteo tira il dado, e si sposterà nella

casella di destinazione. Sulla casella di Destinazione avviene un controllo.

L'esito del controllo può portare a diverse possibilità: se Matteo finisce su una casella Vuota, passa il turno a Leonardo, altrimenti se capita in una casella Serpente o Scala, Matteo sarà costretto a ritirare il dado e a retrocedere o avanzare di un numero di caselle tante quanto il punteggio totalizzato dal lancio.

Abbiamo scelto di utilizzare questa tecnica per rendere più imprevedibili ed eterogenei i percorsi dei vari giocatori così che lo stesso serpente possa essere più o meno dannoso ad un utente piuttosto che ad un altro.

Fase 4: alla fine di ogni serie di tiro (quando tutti i giocatori hanno completato lo stesso turno) viene stampato a video la situazione di tutti i giocatori.

Fase 5: il gioco termina nel momento in cui un giocatore capita nella casella Vittoria. La casella Vittoria deve essere raggiunta con precisione altrimenti si retrocede per il numero rimanente di passi, esempio:

La Casella Vittoria è statica e si trova nella casella 49.
posizione attuale di Ugo $\bar{4}7 \rightarrow$ Casella Vuota

Ugo tira il dado:

situazione A:
totalizza 2
casella di destinazione 49 $\rightarrow$ Casella Vittoria
UGO VINCE, GIOCO TERMINA

situazione B
totalizza 5
casella di destinazione 46 $\rightarrow$ Casella Vuota
47$\rightarrow$48$\rightarrow$49$\rightarrow$48$\rightarrow$47$\rightarrow$46 (ogni freccia rappresenta un movimento)
Ugo passa il turno a Matteo.

3.3 Interfaccia Grafica

Abbiamo scelto un'interfaccia a linea di comando. Il progetto è stato organizzato usando un architettura multi tier, cioè un architettura in cui le varie funzionalità del software sono logicamente separate ovvero suddivise su più strati o livelli software differenti in comunicazione tra loro. Nel nostro caso le operazioni di input e output sono confinate nella classe Interfaccia, separate dalla logica funzionale del programma che si trova nella classe Gioco.

Sono stati rispettati tutti principi della programmazione ad oggetti come l'incapsulamento e occultamento della informazione (information hiding), l'ereditarietà e il polimorfismo.

3.4 Classi

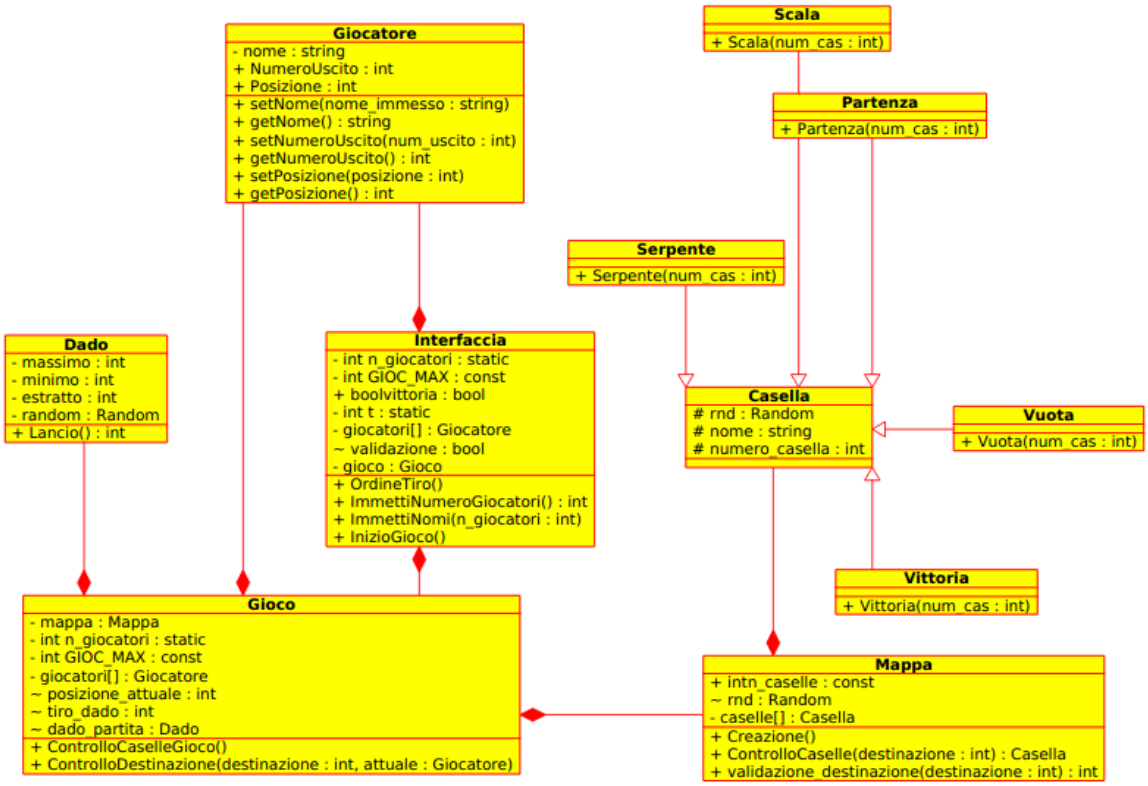

Figura 3.1: Classi

3.5 Interfaccia

3.5.1 ImmettiNumeroGiocatori

Questo metodo permette di acquisire il numero di giocatori. C'è una validazione, il metodo non accetta caratteri, stringhe e punteggiatura. Finchè non viene inserito un numero intero da 2 a 4, il programma continuerà a richiedere tale inserimento. La scelta di poter giocare solo con un massimo di quattro giocatori deriva dal fatto che abbiamo cercato di rendere una partita leggera e non troppo lunga. Inizialmente viene creato un array con quattro posti (numero massimo), solo dopo l'inserimento del numero riusciamo a ridimensionare tale struttura dati se necessario.

3.5.2 ImmettiNomi

Questo metodo crea tutti i giocatori inserendoli nell'apposito array per poi acquisire i relativi nomi richiamando la funzione setNome (nella classe Giocatore). Acquisisce una stringa di caratteri.

3.5.3 OrdineTiro

Questo metodo riprende l'array dei giocatori. Richiama la funzione Lancio (nella classe Dado) e setta tale risultato del lancio ad ogni giocatore richiamando la funzione setNumeroUscito nella classe Giocatore. Una volta che tutti i giocatori hanno tirato il dado, riordina l'array dei giocatori in ordine decrescente valutando il parametro appena salvato. Per riordinare l'array è stato utilizzato l'algoritmo di Insertsort. Nonostante la sua Complessità Computazionale quadratica è stato ugualmente scelto questo algoritmo per la sua facile implementazione non curandoci minimamente della sua complessità in quanto gli array da riordinare non supereranno mai i quattro elementi. Terminata l'operazione di riordino, il metodo stampa a video il nuovo ordine di gioco dei giocatori.

3.5.4 InizioGioco

Metodo per un certo senso banale e inutile perché non fa altro che creare l'istanza Gioco. Si è deciso di inserire ugualmente questo metodo per separare la parte di input con la parte di gestione del gioco vero e proprio, in modo da avere una visuale del codice più precisa e organizzata.

3.6 Gioco

E' la classe principale di tutto il gioco. Contiene la logica funzionale del progetto, gestisce i turni tra i giocatori, controlla le caselle e decreta il vincitore.

Come vediamo dallo schema sovrastante la classe Gioco crea una Composizione con le classi Giocatore, Mappa e Dado, quindi proprio qui vengono create tutte le istanze per far entrare in gioco tutte le classi del programma.

3.6.1 ControlloCaselleGioco

Metodo che gestisce il turno vero e proprio dei giocatori. E' la parte centrale del progetto. Un ciclo (for, che termina solo in caso di vittoria di un giocatore) fa lanciare il dado, a turno, a ogni giocatore; viene calcolata la destinazione per poi passare alla parte di controllo richiamando il metodo ControlloDestinazione. Finito il turno di gioco (cioè tutti i giocatori hanno finito lo stesso turno), il metodo stampa a video la posizione di tutti i giocatori.

3.6.2 ControlloDestinazione

Questo metodo permette di controllare il tipo di casella in cui il giocatore è capitato. Se il giocatore capita nelle caselle Serpente o Scala, il metodo fa tirare di nuovo il dado, calcola la destinazione e reinvoca lo stesso metodo ricorsivamente. In più, se il giocatore arriva alla casella numero 49, cioè la casella Vittoria, stampa a video la notizia.

3.7 Giocatore

Questa classe contiene tutti gli attributi relativi al giocatore, come il nome, la posizione, il numero per l'ordine di tiro. Fondamentalmente è un contenitore di questi attributi e si è deciso di creare una classe apposita per favorire la modularità ed il riuso del codice. Le istanze di questa classe vengono create tante quante sono il numero di giocatori, in più attraverso l'uso di setter e getter risulta più facile inserire informazioni e attributi alle varie istanze Giocatore.

3.7.1 setNome

Metodo per inserire l'attributo nome ad ogni istanza Giocatore.

3.7.2 getNome

Metodo che restituisce il nome del Giocatore.

3.7.3 setNumeroUscito

Metodo per inserire l'attributo numeroUscito ad ogni istanza Giocatore, che permette il riordino dell'array.

3.7.4 getNumeroUscito

Metodo che restituisce il numero uscito dal lancio del dado in fase di ordine di tiro.

3.7.5 setPosizione

Metodo per inserire l'attributo posizione ad ogni istanza Giocatore. La posizione permette di sapere in quale casella si trova momentaneamente il giocatore.

3.7.6 getPosizione

Metodo che restituisce la posizione attuale del Giocatore.

3.8 Mappa

Questa classe rappresenta il tabellone. Attraverso dei parametri specifici ogni volta che va in esecuzione l'applicazione questa classe crea una mappa di caselle di vario tipo, quindi è l'unica che interagisce con la classe Casella.

3.8.1 Creazione

Questo metodo crea la mappa di caselle. Assegna la posizione 0 alla casella Partenza e la posizione 49 alla casella Vittoria. Poi passa ad assegnare caselle Vuota, caselle Serpente e caselle Scala,a random, alle posizioni da 1 a 48. Unico vincolo è che dopo una casella Serpente o Scala, devono esserci per forza 3 caselle Vuota. Così facendo avremo un rapporto di 1:4 tra caselle Serpente+Scala e caselle Vuota. Questo rapporto non può essere dimostrato matematicamente, ma da vari e scrupoli test effettuati siamo riusciti a fare questa considerazione.

3.8.2 ControlloCaselle

Metodo che restituisce il tipo di casella. Viene invocato dalla classe Gioco in fase di controllo della casella di destinazione.

3.8.3 ValidazioneDestinazione

Come detto in precedenza, i giocatori per vincere devono per forza capitare sulla casella Vittoria. Non possono oltrepassarla. Stesso motivo non possono stazionare su caselle minori di quella di Partenza. Questo metodo gestisce proprio questo problema. Valida la casella di Destinazione modificandola quando necessario.

3.9 Dado

Classe in cui viene simulato il dado. Modificando l'attributo statico (numero massimo), possiamo aumentare il numero dei dadi di gioco. Nel nostro caso il numero massimo che si può totalizzare è 6.

3.9.1 Lancio

Questo metodo simula il lancio del dado. Casualmente restituisce un numero compreso tra i range settati nella classe Dado.

3.10 Casella

Classe che rappresenta le posizioni nel tabellone. Si è deciso di affrontare la creazione di questa classe con le sue classi derivate per svariati motivi. In primo luogo servivano tipi di caselle diverse, si poteva scegliere altre strade per affrontare il problema, ma abbiamo preferito seguire la logica della programmazione ad oggetti. In un secondo luogo, nel momento in cui si voglia riutilizzare ilcodice per ampliare interfacce grafiche, un impostazione del genere rende molto più comprensibile e meno difficoltoso il lavoro da fare.

3.10.1 Vuota, Serpente, Scala

Sono Classi figlie di Casella. Non hanno metodi. L'unico attributo che hanno è il nome.

3.10.2 Partenza, Vittoria

Sono Classi figlie di Casella. In fase di sviluppo si era deciso che dovevano esistere solo 3 tipi principali di caselle (Vuota, Serpente, Scale) e che la casella Partenza e Vittoria dovevano essere classi figlie di Vuota, perché fondamentalmente tali caselle devono essere vuote. Ma con il proseguire del progetto abbiamo adottato questa scelta in quanto tutte le classi figlie di Casella sono vuote (comprese Serpente e Scala) se non per il nome datogli, perché le operazioni di modifica della posizione avvengono in Gioco, quindi ci riallacciamo alle motivazioni date di sopra per giustificare la nostra scelta. Questa decisione ci sembra più pulita e concettualmente più giusta.

3.11 Diagramma sequenziale

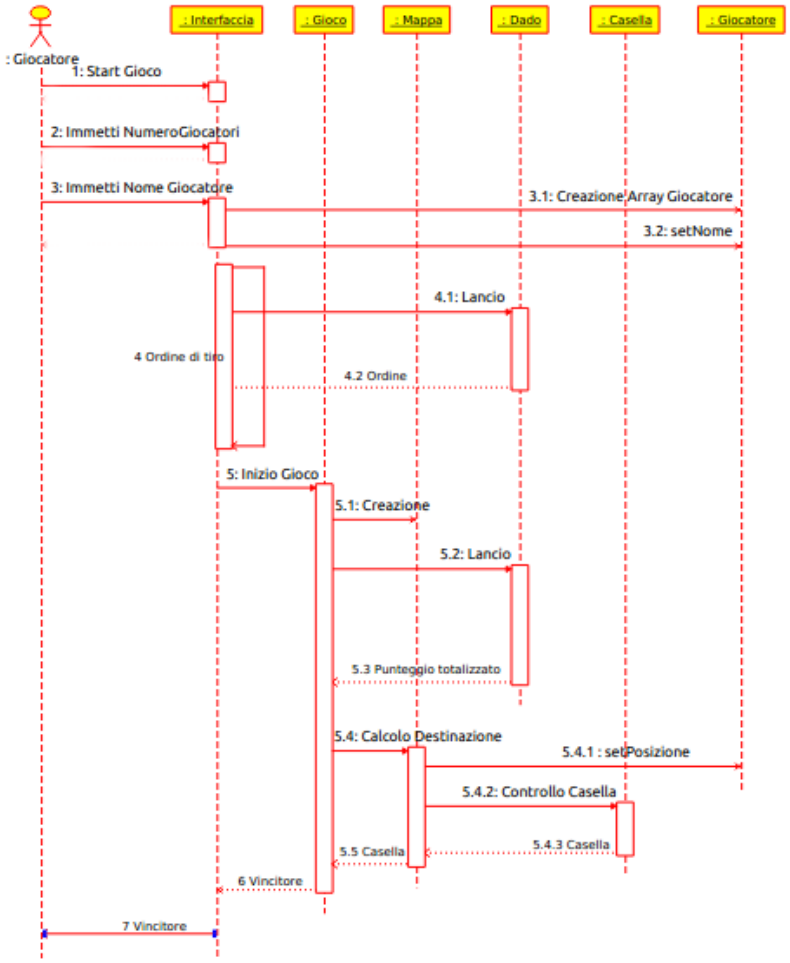

Figura 3.2: diagramma sequenziale

Capitolo 4

Implementazione

Di seguito viene trascritto il codice sorgente di tutte le classi.

4.1 Interfaccia

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GiocoScaleSerpenti
9  {
10     class Interfaccia
11     {
12         //inizializzazione variabili globali
13         static int n_giocatori;
14         const int GIOC_MAX = 4;
15         public bool boolvittoria = false;
16
17         //variabile di controllo per fase test
18         static int t = 0;
19
20         //creo l'array giocatori dove verranno inserite tutte le
21         // informazioni riguardanti i giocatori
22         static Giocatore[] giocatori = new Giocatore[GIOC_MAX];
23  //main
24         static void Main(string[] args)
25         {
26             //stampa a video istruzioni iniziali
27             Console.WriteLine("Progetto Ingegneria del Software");
28             Console.WriteLine("Manoni Leonardo & Incicco Matteo");
29             Console.WriteLine("Gioco delle Scale e dei Serpenti");
```

```
30              Console.ReadLine();
31
32
33              //Scelta numero giocatori, l'utente inserira' a video
34              //il numero dei giocatori
35              //con validazione dati di input.
36                  //la variabile booleana serve per validare l'input
37              bool validazione = true;
38
39              while (validazione)
40              {
41                  n_giocatori = ImmettiNumeroGiocatori();
42                  if (n_giocatori >= 2 && n_giocatori <= 4)
43                  {
44                      validazione = false;
45                  }
46              }
47
48              //l'utente inserira' tutti i nomi dei giocatori
49              //non c'e' validazione, utilizziamo stringhe, e'
50              //possibile inserire qualunque carattere
51              t = ImmettiNomi(n_giocatori);
52
53                          //istruzioni per Test
54              if (t == 1)
55              {
56                  WhiteBox wb = new WhiteBox();
57                  wb.test_destinazione();
58                          }
59              else
60              {
61                  //ordine di tiro
62                  //tutti i giocatori tirano il dado,
63                  //a seconda dei punteggi si stabilisce l'ordine
64                  //di gioco
65                  OrdineTiro();
66
67                  //inizio del gioco e dei turni di gioco
68                  InizioGioco();
69              }
70
71              }
72          //funzione che acquisisce da tastiera il numero di giocatori
73          static int ImmettiNumeroGiocatori()
74          {
75              System.Console.WriteLine("Immetti numero giocatori
76                                                  (Da 2 a 4):");
77
78
```

```
79
80          //acquisizione
81              string stringa_appoggio = Console.ReadLine();
82          //c# acquisisce solo stringhe, convertiamo la stringa
83          // in int
84              Int32.TryParse(stringa_appoggio, out n_giocatori);
85
86              return n_giocatori;
87          }
88
89
90          //acquisizione dei nomi dei giocatori
91          //questa funzione permette all'utente di inserire i nomi
92          // dei giocatori
93          static int ImmettiNomi(int n_giocatori)
94          {
95              string nome_immesso;
96              int test = 0;
97
98
99              for (int i = 0; i < n_giocatori; i++)
100             {
101                 Console.WriteLine("Immetti nome giocatore numero:
102                                                     " + (i + 1));
103                 nome_immesso = Console.ReadLine();
104                                 //istruzione per test
105                 if (nome_immesso == "test")
106                 {
107                     i = 5;
108                     test = 1;
109                 }
110                 else
111                 {
112                    //creazioni istanze giocatori
113                    giocatori[i] = new Giocatore();
114                    //settiamo il nome ad ogni giocatore
115                    //richiamando setNome
116                    giocatori[i].setNome(nome_immesso);
117                 }
118
119             }
120
121             return test;
122
123         }
124
125         //funzione che ordina i giocatori in base al punteggio
126         //del dado
127         //tutti i giocatori tirano il dado, e in base al loro punteggio
```

```
128         //avranno la posizione di tiro
129
130         static void OrdineTiro()
131         {
132             Console.WriteLine();
133             Console.WriteLine("TUTTI I GIOCATORI TIRANO IL DADO");
134             Console.WriteLine("L'ORDINE DI GIOCO VERRA' STABILITO
135                               \r\n IN ORDINE DECRESCENTE A SECONDA
136                               \r\n DEL PUNTEGGIO TOTALIZZATO ");
137             Console.ReadLine();
138
139             //istituisco l'ordine di gioco facendo tirare ad ogni
140             //giocatore il dado
141             for (int i = 0; i < n_giocatori; i++)
142             {
143                 //istanza dado per richiamare il metodo lancio
144                 Dado dado_ordine = new Dado();
145                 int nOrdineUscito;
146
147                 //tiro dado
148                 nOrdineUscito = dado_ordine.Lancio();
149
150                 //setto il numero uscito con l'array giocatori
151                 giocatori[i].setNumeroUscito(nOrdineUscito);
152                 Console.WriteLine(giocatori[i].getNome() +
153                                   " ha totalizzato " + nOrdineUscito);
154                 Console.ReadLine();
155
156
157             }
158
159
160             //insertsort
161             //ordino l'array giocatori in ordine decrescente
162             Giocatore appoggio = new Giocatore();
163             for (int i = 1; i < n_giocatori; ++i)
164             {
165                 appoggio = giocatori[i];
166                 int appoggio_numero = appoggio.getNumeroUscito();
167                 int j = (i - 1);
168
169                 while (j >= 0 &&
170                 appoggio_numero > giocatori[j].getNumeroUscito())
171                 {
172
173                     giocatori[j + 1] = giocatori[j];
174                     j--;
175                 }
176
```

```
177                 giocatori[j + 1] = appoggio;
178
179             }//fine algoritmo insertsort
180
181             Console.WriteLine("ORDINE DI GIOCO");
182
183                         //stampa a video dell'ordine di gioco
184             for (int i = 0; i < n_giocatori; i++)
185             {
186                 Console.WriteLine(giocatori[i].getNome() + "
    " +
187                                   giocatori[i].getNumeroUscito());
188             }
189
190             Console.ReadLine();
191         }
192
193
194
195         //inizio Gioco
196         static void InizioGioco()
197         {
198             //creo istanza gioco passando per paramertro
199             //il numero giocatori e  l'array contemente i giocatori
200             new Gioco(n_giocatori, giocatori);
201
202         }
203
204
205
206     }
207
208
209 }
```

4.2 Gioco

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6  using System.IO;
7
8  namespace GiocoScaleSerpenti
9  {
10     class Gioco
11     {
12         //creazione mappa di gioco Random,
13         //ad ogni esecuzione del programma verra' creata una nuova
14         // mappa
15         static Mappa mappa = new Mappa();
16
17         //variabili globali
18         static int n_giocatori;
19         const int GIOC_MAX = 4;
20
21         //creo l'array giocatori dove verranno inserite tutte le
22         //informazioni riguardanti i giocatori
23         static Giocatore[] giocatori = new Giocatore[GIOC_MAX];
24
25         //costruttore
26         public Gioco(int n_g, Giocatore[] gio)
27         {
28             //inserisco i parametri passati nelle variabili create
29             n_giocatori = n_g;
30             for (int i = 0; i < n_giocatori; i++)
31             {
32                 giocatori[i]=gio[i];
33
34             }
35
36                         //chiamo metodo per iniziare turno di gioco
37               ControlloCaselleGioco();
38
39             }
40
41         //metodo iniziare con i turni di gioco
42         static void ControlloCaselleGioco()
43         {
44             int destinazione;
45             int posizione_attuale;
46             int tiro_dado;
47             Dado dado_partita = new Dado();
```

```
48
49
50
51          //ogni giocatore tira il dado, e avanza di posizione a
52          //seconda del punteggio totalizzato
53              for (int i = 0; i < n_giocatori; i++)
54              {
55
56                  //prendo la posizione attuale
57                  //inizialmente sara' settata a 0
58                  posizione_attuale = giocatori[i].getPosizione();
59
60                  tiro_dado = 0;
61
62                  //giocatore i lancia il dado
63                  tiro_dado = dado_partita.Lancio();
64                  Console.WriteLine(giocatori[i].getNome() +
65                                    "ha lanciato:  " + tiro_dado);
66
67                  //destinazione raggiunta
68                  destinazione = posizione_attuale + tiro_dado;
69
70                  //creo istanza Giocatore per usarla come appoggio
71                  Giocatore attuale = new Giocatore();
72                  attuale = giocatori[i];
73
74
75                  //caselle <0 e >49 non esistono
76                  if (destinazione < 0)
77                  {
78                      destinazione = -destinazione;
79                  }
80
81                  if (destinazione >= 50)
82                  {
83                      destinazione = 49 - (destinazione - 49);
84                  }
85
86                  //una volta raggiunta la destinazione devo controllare
87                  //dove il giocatore e' capitato
88                  ControlloDestinazione(destinazione, attuale);
89
90                  if (destinazione == 49)
91                  {
92                      //se c'e' un vincitore, usciamo dal for
93                      i = n_giocatori;
94                  }
95
96
```

```
97                //facciamo ripartire il turno se nessuno ha vinto
98                //quindi se non c'e' vincitore, settiamo i a -1
99                if (i == (n_giocatori - 1))
100               {
101                   i = -1;
102                   for (int j = 0; j < n_giocatori; j++)
103                   {
104                       if (j == 0)
105                       {
106                           Console.WriteLine("\n\r");
107                           Console.WriteLine("_____________________
108                                   ________________________________");
109                       }
110
111                   Console.WriteLine(giocatori[j].getNome() +
112                                     " e' in posizione:  " +
113                                    giocatori[j].getPosizione());
114
115                       if (j == n_giocatori - 1)
116                           Console.WriteLine("_________________________
117                                     ________________________________");
118                   }
119               }
120
121               Console.ReadLine();
122
123           }
124       }
125
126
127           //metodo per controllare la casella destinazione
128       static void ControlloDestinazione(int destinazione,
129                                         Giocatore attuale)
130       {
131           //metto la cassella su una variabile di controllo
132           Dado dado_partita = new Dado();
133           Casella controllo;
134
135                   //richiamo il metodo ControlloCaselle per scoprire
136                   // su quale casella
137           //il giocatore e' capitato
138           controllo = mappa.ControlloCaselle(destinazione);
139
140
141           //inizio fase dei controlli
142           if (controllo.GetType().FullName ==
143              ("GiocoScaleSerpenti.Vuota"))
144           {
145               Console.WriteLine("Destinazione--> " + destinazione +
```

```
146                                 ": --> Casella Vuota");
147
148             //se la casella e' vuota, setto la posizione destinazione
149             //come attuale ed esco dal ciclo
150
151                 attuale.setPosizione(destinazione);
152
153
154             }
155             else if (controllo.GetType().FullName =
156                     ("GiocoScaleSerpenti.Serpente"))
157             {
158                 Console.WriteLine("Destinazione--> " + destinazione +
159                                   ": --> Casella Serpente\n\r "
160                                   + attuale.getNome() +
161                                   " deve rilanciare il dado");
162
163                 //se la casella e' un serpente, rilancio il dado e'
164                 //ritorno indietro
165                 int ripiego;
166                 ripiego = dado_partita.Lancio();
167                 Console.WriteLine(attuale.getNome()
168                                 + " ha lanciato:  " +
169                                 ripiego);
170
171                 //destinazione dopo il lancio
172                 destinazione = destinazione - ripiego;
173
174                 //validazione destinazione
175                 destinazione = mappa.validazione_destinazione(destinazione);
176
177                 //setto la destinazione modificata come posizione attuale
178                 attuale.setPosizione(destinazione);
179                 //ricontrollo la posizione
180                 //ricorsione di funzione
181                 ControlloDestinazione(destinazione, attuale);
182
183             }
184                 else if (controllo.GetType().FullName ==
185                         ("GiocoScaleSerpenti.Scala"))
186             {
187                 Console.WriteLine("Destinazione--> " + destinazione +
188                                   ": --> Casella Scala\n\r "
189                                    + attuale.getNome() +
190                                    " deve lanciare il dado:");
191                 //se e' una Scala, ritiro il dado
192                 int aumento;
193                 aumento = dado_partita.Lancio();
194                 Console.WriteLine(attuale.getNome() + " ha lanciato: "
```

```
195                                     + aumento);
196
197                  //la destinazione viene aumentata del risultato
198                  //del lancio
199                 destinazione = destinazione + aumento;
200
201             //validazione destinazione
202             destinazione = mappa.validazione_destinazione(destinazione);
203
204                 //ricontrollo la posizione
205                 //ricorsione di funzione
206                         ControlloDestinazione(destinazione, attuale);
207             }
208             else if (controllo.GetType().FullName ==
209                     ("GiocoScaleSerpenti.Vittoria"))
210             {
211                 Console.WriteLine(attuale.getNome() + " ha VINTO!!!!");
212                 Console.WriteLine("FINE GIOCO");
213
214             }
215             else if (controllo.GetType().FullName ==
216                     ("GiocoScaleSerpenti.Partenza"))
217             {
218                 Console.WriteLine("Destinazione--> " + destinazione +
219                                     ": --> Casella Partenza");
220
221                 attuale.setPosizione(destinazione);
222             }
223         }
224
225     }
226 }
```

4.3 Mappa

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GiocoScaleSerpenti
9  {
10     class Mappa
11     {
12         //variabile statica, il numero del tabellone e' sempre 50
13         const int n_caselle = 50;
14
15                 Random rnd = new Random();
16
17         //inizializzo array di caselle
18                 Casella[] caselle = new Casella[50];
19
20         //metodo costruttore Mappa
21         public Mappa()
22         {
23             Creazione();
24         }
25
26         //Metodo Creazione Mappa
27         //Vincolo: dopo una casella Serpente o Scala deve essercene
28         //3 vuote
29         public void Creazione()
30         {
31             //come prima cosa, creiamo l'istanza Partenza nella
32             // posizione 0,
33             //e l'istanza Vittoria nella posizione 49
34             for (int i = 0; i < n_caselle; i++)
35             {
36                 if (i == 0)
37                 {
38                     caselle[i] = new Partenza(i);
39
40                 }
41                 else
42                     if (i == 49)
43                     {
44                         caselle[i] = new Vittoria(i);
45                     }
46                     else
47                     {
```

```
48                          //a Random inseriamo o la casella Vuota, o
49                          //la Serpente o la Scala
50                          int appoggio;
51                          appoggio = rnd.Next(0, 3);
52                          if (appoggio == 0)
53                          {
54                              caselle[i] = new Vuota(i);
55                          }
56                          else if (appoggio == 1)
57                          {
58                              caselle[i] = new Serpente(i);
59
60                              // creo 3 caselle vuote
61                              if (i < 46)
62                              {
63                                  caselle[i + 1] = new Vuota(i + 1);
64                                  caselle[i + 2] = new Vuota(i + 2);
65                                  caselle[i + 3] = new Vuota(i + 3);
66                                  i = i + 3;
67                              }
68                          }
69                          else
70                          {
71                              caselle[i] = new Scala(i);
72
73                              // creo 3 caselle vuote
74                              if (i < 46)
75                              {
76                                  caselle[i + 1] = new Vuota(i + 1);
77                                  caselle[i + 2] = new Vuota(i + 2);
78                                  caselle[i + 3] = new Vuota(i + 3);
79
80                                  i = i + 3;
81                              }
82
83                          }
84
85                      }
86                  }
87          }
88
89          //metodo che restituisce il tipo di casella.
90          //
91          public Casella ControlloCaselle(int destinazione)
92          {
93              //inizializziamo un oggetto casella che ci servira' come
94              //appoggio
95              Casella controllo;
96
```

```
97              //in input la funzione acquisce il numero destinazione,
98              //avviene il controllo e restituisce il tipo di casella
99              controllo=caselle[destinazione];
100             return controllo;
101         }
102
103
104         //metodo che ci permette di validare la destinazione nel momento
105         // in cui si supera la casella 49 o la 0
106         public int validazione_destinazione(int destinazione)
107         {
108             //se superiamo la casella 49, dobbiamo calcolare l'avanzo e
109             //sottrarlo dalla casella 49
110             if (destinazione >= 50)
111             {
112                 destinazione = 49 - (destinazione - 49);
113             }
114             //se invece siamo sotto zero, basta cambiare di segno
115                     else if (destinazione < 0)
116             {
117                 destinazione = -(destinazione);
118             }
119
120             return destinazione;
121         }
122
123
124
125     }
126 }
```

4.4 Dado

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GiocoScaleSerpenti
8  {
9      class Dado
10     {
11         //variabili globali
12         private int massimo;
13         private int minimo;
14         private int estratto;
15         private Random random = new Random();
16
17         // costruttore di dado
18         public Dado()
19         {
20             massimo = 6;
21             minimo = 1;
22         }
23
24         //metodo che permette di prelevare un numero random
25         // dentro certi
26         //valori (minimo e massimo)
27         public int Lancio()
28         {
29             estratto = random.Next(minimo, massimo);
30             return estratto;
31         }
32
33     }
34 }
```

4.5 Giocatore

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GiocoScaleSerpenti
8  {
9      class Giocatore
10     {
11         //variabili globali
12         private string nome;
13         public int NumeroUscito;
14         public int Posizione;
15
16         //costruttore
17         public Giocatore()
18         {
19             Posizione = 0;
20         }
21
22         //assegna il nome
23         public void setNome(string nome_immesso)
24         {
25           nome = nome_immesso ;
26         }
27
28         //restituisce il nome
29         public string getNome()
30         {
31             return nome;
32         }
33
34         //assegna il numero uscito dal dado in fase
35         //di ordine dei giocatori
36         public void setNumeroUscito(int num_uscito)
37         {
38             NumeroUscito = num_uscito;
39         }
40
41         //restituisce il numero uscito dal dado in fase di ordine
42         // dei giocatori
43         public int getNumeroUscito()
44         {
45             return NumeroUscito;
46         }
47
```

```
48          //assegna la posizione attuale
49          public void setPosizione(int posizione)
50          {
51            Posizione=posizione;
52          }
53
54          //restituisce la posizione attuale
55          public int getPosizione()
56          {
57              return Posizione;
58          }
59
60
61      }
62  }
```

4.6 Casella

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GiocoScaleSerpenti
8  {
9      //classe madre delle caselle:
10     //Vuota, Serpente, Partenza, Vittoria e Scala
11     abstract class Casella
12     {
13         protected Random rnd;
14         protected string nome;
15         protected int numero_casella;
16
17      }
18 }
```

4.7 Scala

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GiocoScaleSerpenti
9  {
10     class Scala : Casella
11     {
12         //classe figlia di Casella
13         public Scala(int num_cas)
14         {
15             nome = "scala";
16             numero_casella = num_cas;
17         }
18     }
19 }
```

4.8 Serpente

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GiocoScaleSerpenti
9  {
10     //classe figlia di Casella
11     class Serpente : Casella
12     {
13                 public Serpente(int num_cas)
14         {
15             nome = "serpente";
16             numero_casella = num_cas;
17         }
18     }
19 }
```

4.9 Vuota

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GiocoScaleSerpenti
8  {
9      //classe figlia di casella
10     class Vuota : Casella
11     {
12                 public Vuota(int num_cas)
13         {
14             nome = "Vuota";
15             numero_casella = num_cas;
16         }
17     }
18 }
```

4.10 Partenza

```
1
2  using System;
3  using System.Collections.Generic;
4  using System.Linq;
5  using System.Text;
6  using System.Threading.Tasks;
7
8  namespace GiocoScaleSerpenti
9  {
10     class Partenza : Casella
11     {
12         //classe figlia di Casella
13         public Partenza(int num_cas)
14         {
15             nome = "Partenza";
16             numero_casella = num_cas;
17         }
18     }
19 }
```

4.11 Vittoria

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace GiocoScaleSerpenti
8  {
9      //classe figlia di Casella
10     class Vittoria : Casella
11     {
12         public Vittoria(int num_cas)
13         {
14             nome = "Vittoria";
15             numero_casella = num_cas;
16         }
17     }
18 }
```

Capitolo 5

Test

5.1 White Box

Il "white box" è un tipo di test (anche chiamato test strutturale), che viene effettuato per rilevare degli errori in uno o più parti di codice di un sistema software.

Alla base di questa metodologia c'è appunto l'analisi del codice. Si devono, quindi, identificare i cammini da percorrere ed effettuare anche in modo manuale, con l'uso di diagrammi di flusso che mostrano i diversi cammini del programma e delle tabelle di traccia, che simulano l'esecuzione del codice.

Entrando nel dettaglio (nel nostro programma) si evince una porzione di codice, in particolare, la quale necessita di test con questa metodologia. Questa porzione di codice riguarda un particolare comportamento del gioco agli estremi della mappa. Nella prime caselle c'è la possibilità che si prenda un serpente. Di conseguenza si ha anche la possibilità di effettuare più passi indietro di quelli che sono stati fatti in avanti.

5.1.1 Caso Numero 1

Il giocatore A arriva nella casella numero 3. Supponiamo che in quest'ultima vi sia un serpente il giocatore A si ritroverà a lanciare il dado per sapere di quante caselle tornare indietro. Se il dado restituirà un numero più alto della casella in cui si trova il giocatore ci si troverà di fronte ad un particolare comportamento del gioco. Ammettiamo che il numero estratto sia 6. Il giocatore dovrà quindi effettuare 6 passi indietro. Di conseguenza il normale comportamento sarà il seguente:

	A	←	←	←	←	←	A	
-4	-3	-2	-1	0	1	2	3	4

Il giocatore finirà nella casella numero -3 ma, ovviamente, la casella numero -3 non esiste nella mappa. Quindi dovrà tornare indietro (in avanti) dei passi rimanenti.

←	←	←	A	
0	1	2	3	4
	→	→	**A**	

5.1.2 Caso Numero 2

La stessa cosa vale per la parte finale della mappa.

	A	→	→	→	→	→	A	
45	46	47	48	49	50	51	52	53

La casella massima (ovvero quella della vittoria è la numero 49).

Il giocatore arriva nella casella numero 46. Se quest'ultima contiene una scala il giocatore si troverà a dover rilanciare il dado, questa volta avanzando nuovamente invece che indietreggiando. Se il giocatore estrarrà il numero 6 dovrà fare 6 passi in avanti. Come sopra, un normale comportamento è il seguente: Ma, l'ultima casella disponibile nella nostra mappa è la numero 49. Quindi si troverà a dover tornare indietro dei passi rimanenti.

A	→	→	→	
45	46	47	48	49
A	←	←		

5.1.3 Schema Logico

Per ovviare al problema precedentemente descritto è stata implementata una funzione nella classe Mappa chiamata `validazione_destinazione`. Questo metodo sarà dichiarato pubblico in quanto verrà richiamato nella classe gioco e, appunto, dalla classe WhiteBox. Questo metodo sarà dichiarato pubblico in quanto verrà richiamato nella classe Gioco e, appunto, dalla classe WhiteBox. La funzione ha come parametro la destinazione finale del giocatore e ritornerà come valore la destinazione stessa (opportunamente corretta se rientrerà nei due casi sopra descritti).
Schema logico della funzione:

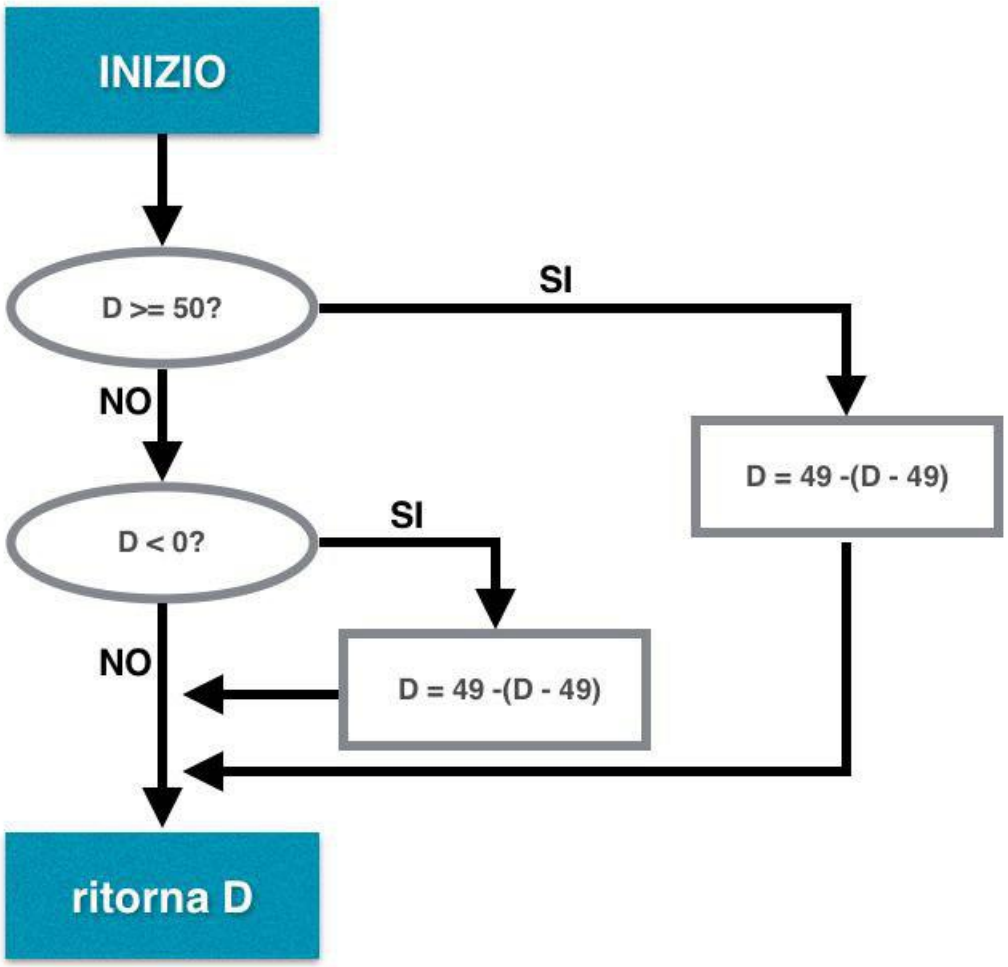

Figura 5.1: Schema Logico della Funzione

Per attivare la modalità test sarà necessario inserire semplicemente la parola “test” come nome dei giocatori.
Di seguito viene riportato uno screen dell’esecuzione dei test.

```
public int validazione_destinazione(int destinazione)
{
    if (destinazione >= 50)
    {
        destinazione = 49 - (destinazione - 49);
    }
    else if (destinazione < 0)
    {
        destinazione = -(destinazione);
    }

    return destinazione;
}
```

Figura 5.2: Funzione validazione destinazione

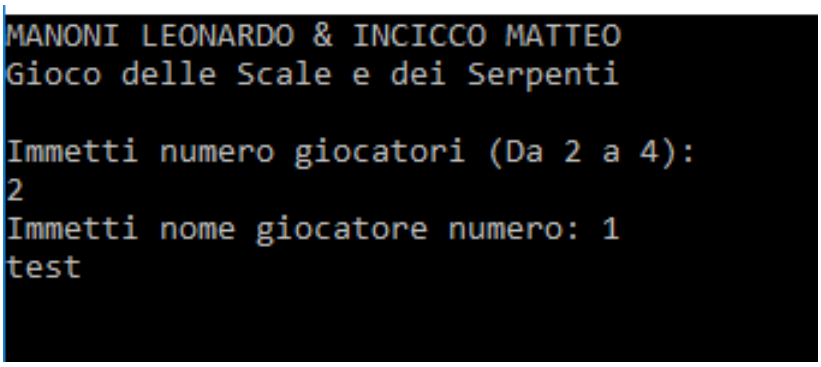

Figura 5.3: Screen accesso modalità test

```
test
MODALITA' TEST:
In questa modalità verrà testata la funzione che
valida la destinazione di un giocatore

Nel caso in cui ci si trovi in una delle
caselle iniziali, e si prenda una casella serpente
c'è la possibilità di tornare indietro di più caselle
di quante ne siano state percorse fin'ora.
ESEMPIO: ammettiamo il caso in cui ci troviamo nella
casella 4, ed in questa vi sia un serpente. Se dal dado
si estraesse il numero 5 i passi del giocatore dovrebbero
essere i seguenti : 4 -> 3 -> 2 -> 1 -> 0 -> 1.
E non: 4 -> 3 -> 2 -> 1 -> 0 -> -1.


Validazione Serpente iniziale:
Input destinazione: -2
Usciti dalla funzione la destinazione
corretta dovrebbe essere 2.


ESITO: ok


Validazione Scala finale:
Input destinazione: 52
Usciti dalla funzione la destinazione
corretta dovrebbe essere 46.


ESITO: ok



FINE TEST
```

Figura 5.4: screen esecuzione test

Capitolo 6

Compilazione ed esecuzione

6.1 Istruzioni per la compilazione

Istruzioni per la compilazione:
1. Estrarre l'archivio GiocoScaleSerpenti.tar.gz
2. Navigare nel percorso GiocoScaleSerpenti/
3. Avviare il file di Visual Studio GiocoScaleSerpenti.sln
4. Attende il caricamento dei componenti
5. Selezionare le voci di menu Compila –> Compila soluzione
6. L'applicazione è stata correttamente compilata

6.2 Istruzioni per l'esecuzione

1. Navigare nel percorso GiocoScaleSerpenti/GiocoScaleSerpenti/bin/Debug
2. Avviare il file GiocoScaleSerpenti.exe
3. Viene aperto il terminale e l'applicazione è pronta per essere utilizzata

Sono stati effettuati test su macchine con diversi Sistemi Operativi e con diversi hardware. L'applicazione è stabile. Di seguito riportiamo l'elenco dei test.

Sistema Operativo	CPU	Ram	Crash
Windows 8.1	4core	8gb	NO
Windows 8	2core	4gb	NO
Windows 7	2core	4gb	NO

Sono stati utilizzati:
LATEX
TEXMAKER
GIMP
UMBRELLO