

Progetto Ingegneria del Software

Anno Accademico 2016-2017

Sessione estiva

SNAKE

Realizzazione di un software for playing

Applicazione Windows Form e Console in C#

Realizzato da:

Davide Marrancone - 273567

Jonathan Zaccaria - 271960

Sommario

1.	Specifica del problema	4
2.	Specifica dei Requisiti.....	5
2.1	Relazioni e descrizione dei casi d'uso.....	5
2.2	Caso 1 - Visualizzare il menu.....	6
2.3	Caso 2 - Visualizzare il form contenente le istruzioni.....	7
2.4	Caso 3 - Uscire dal programma dal menu iniziale	8
2.5	Caso 4 - Inizio Partita	8
2.6	Caso 5 - Input di movimento.....	9
2.7	Caso 6 - Mangia.....	10
2.8	Caso 7 - Collisione con il bordo	11
2.9	Caso 8 - Collisione Snake.....	11
2.10	Caso 9 - Morte.....	12
2.11	Caso 10 - Rigioca.....	13
2.12	Caso 11 - Esci	14
3.	Analisi e progettazione.....	15
3.1	Linguaggio di programmazione.....	15
3.2	Ambiente di sviluppo.....	15
3.3	GUI.....	16
3.4	Design Pattern.....	20
3.5	Organizzazione progetto.....	21
3.6	Classi.....	22
3.6.1	Cibo.....	22
3.6.2	Impostazioni.....	24
3.6.3	Pezzo.....	25
3.6.4	Controller.....	26
3.6.5	Snake.....	27
3.6.6	Partita.....	29
3.6.7	Program.....	30
3.6.8	Inizio	31

3.6.9 Istruzioni	31
3.6.10 Campo Gioco.....	32
4. Implementazione.....	33
4.1 Pezzo.....	34
4.2 Impostazioni.....	34
4.3 Controller.....	35
4.4 Cibo.....	36
4.5 CiboGiallo.....	37
4.6 CiboRosso.....	37
4.7 Snake.....	38
4.8 Partita.....	40
4.9 Inizio.....	43
4.10 Istruzione.....	43
4.11 CampoGioco.....	44
4.12 Program.....	45
5. Test	46
6. Compilazione ed esecuzione	54

1. Specifica del problema

Nel progetto corrente andremo a realizzare una versione del gioco “Snake” in modalità giocatore singolo.

Lo scopo del gioco è quello di muovere l’oggetto serpente (snake) controllato dall’utente attraverso i quattro tasti direzionali, verso gli oggetti cibo rappresentati da un quadrato che potrà essere giallo o rosso.

Quando il serpente si scontra con il cibo le sue dimensioni aumentano di 1 quadrato e il punteggio sale in base alla tipologia di cibo mangiato.

Nel gioco sono stati introdotti due diverse tipologie di cibo:

- Il cibo **giallo** che incrementa il punteggio di 100.
- Il cibo **rosso** che incrementa il punteggio di 200.

Il gioco può terminare al verificarsi di una collisione da parte del giocatore con il bordo del campo di gioco o con la collisione della testa del serpente su uno dei segmenti del suo corpo.

In questo gioco non esiste il concetto di partita vinta o persa ma solo quello di gioco terminato in quanto l’unico obiettivo è fare un punteggio alto e far aumentare la lunghezza del serpente.

All’avvio del gioco apparirà una finestra che permetterà all’utente di scegliere tra 3 opzioni:

- Istruzioni: Se l’utente andrà a cliccare su questo bottone il programma mostrerà un altro form contenente un’immagine con una presentazione del gioco e i comandi da usare per giocare.
- Gioca: Se l’utente andrà a cliccare su questo bottone il programma farà partire il form di gioco e l’utente potrà iniziare a giocare.

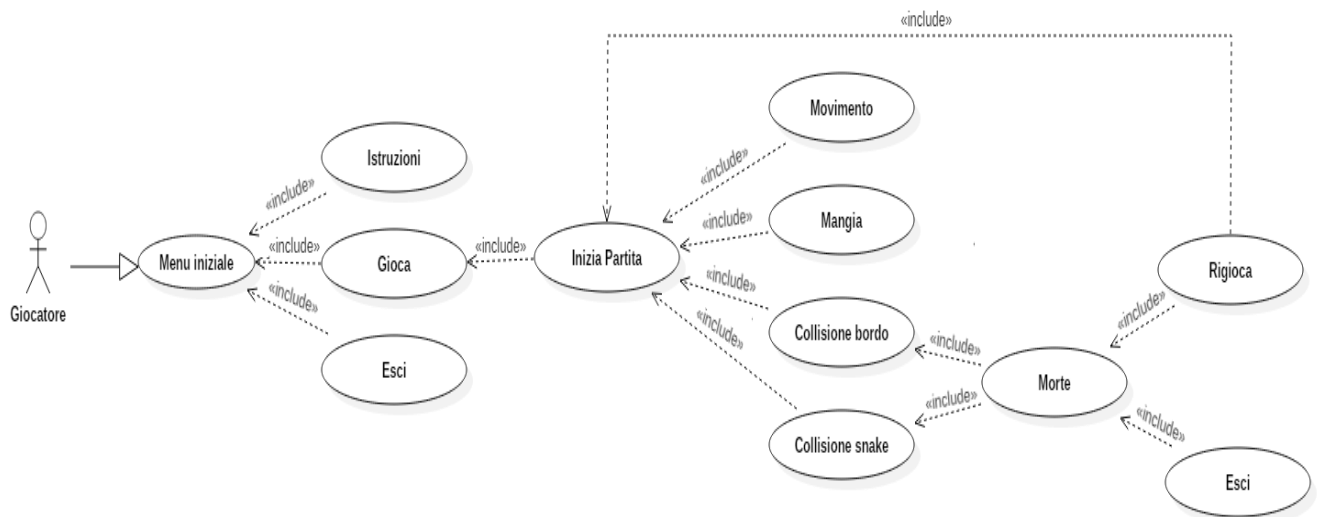
- Esci: Se l'utente andrà a cliccare su questo bottone il programma si chiuderà.

Una volta che l'utente inizierà a giocare, al termine della partita verranno comunicati il punteggio e la lunghezza ottenuta e verranno mostrati 2 bottoni: rigioca e esci.

2. Specifica dei Requisiti

In questa sezione andremo a illustrare i casi d'uso con le loro relazioni e descrizioni.

Il seguente diagramma dei casi d'uso ci mostrerà cosa sarà possibile fare nel programma.



Il diagramma ci mostra il percorso che l'attore (in questo caso il giocatore) potrà intraprendere, esso all'avvio dell'applicativo si troverà d'avanti al menu iniziale e potrà scegliere tra: istruzioni, gioca ed esci.

L'unico caso da illustrare è il caso Gioca in quanto gli altri due non includono o estendono altri casi d'uso, per fare ciò ricorreremo alle tabelle dei casi d'uso.

2.1 Relazioni e descrizione dei casi d'uso

In questa sezione andremo ad illustrare i vari casi d'uso individuati. Il giocatore una volta avviato il programma si troverà d'avanti al form rappresentante il menù iniziale del gioco come illustrato nel caso d'uso 1.

2.2 - Caso 1

Caso d'uso	Visualizzare il menù
ID	1
Attore	Giocatore
Precondizioni	Avviare il programma
Corso d'azione base	Avvio del gioco
Post condizioni	Scelta dal menù tra: Istruzioni, Gioca e Esci.
Percorso Alternativo	/

Da questo form il giocatore potrà intraprendere 3 azioni scegliendo tra: istruzioni, gioca ed esci.

Se il giocatore clicca su Istruzioni, verrà aperto un nuovo form chiamato form "Istruzioni" contenente le istruzioni del gioco.

Dentro questo form l'unica azione che il giocatore può intraprendere è quella di uscire e tornare al menù iniziale in quanto è un form puramente grafico.

2.3 - Caso 2

Caso d'uso	Visualizzare il form contenente le istruzioni
ID	2
Attore	Giocatore
Precondizioni	Il giocatore è nel form del menù iniziale
Corso d'azione base	Premere sul button "Istruzioni"
Post condizioni	Una volta finito di visualizzare le istruzioni il giocatore torna nel menu iniziale tramite il button "chiudi"
Percorso Alternativo	/

- Istruzioni

Se il giocatore clicca sul button "esci" il programma si arresta.

2.4 - Caso 3

- Esci

Caso d'uso	Uscire dal programma dal menù iniziale
ID	3
Attore	Giocatore
Precondizioni	Il giocatore è nel form del menù iniziale
Corso d'azione base	Premere sul button "Esci"
Post condizioni	Il programma termina
Percorso Alternativo	Chiusura del programma tramite gestione delle attività o tramite gestione risorse.

Se il giocatore invece clicca sul button "Gioca" la partita potrà avere inizio e si aprirà il form di gioco.

2.5 - Caso 4

- Gioca

Caso d'uso	Gioca - Inizia Partita
ID	4
Attore	Giocatore
Precondizioni	Il giocatore è nel form del menù iniziale

Corso d'azione base	Premere sul button "Gioca"
Post condizioni	Il giocatore si trova nel form CampoGioco e il gioco avrà inizio
Percorso Alternativo	Al termine della partita se si clicca su rigioca il giocatore potrà riiniziare la partita

Dal caso d'uso 4 il giocatore potrà iniziare a giocare e svolgere tutte le azioni che il gioco prevede.

Ora le andremo ad illustrare:

2.6 - Caso 5

- Movimento: è l'input direzionale dato dall'utente.

Caso d'uso	Input di movimento
ID	5
Attore	Giocatore
Precondizioni	Il giocatore inizia la partita
Corso d'azione base	Premere uno dei 4 tasti direzionali di input
Post condizioni	Il serpente (snake) si muove in base all'input dato
Percorso Alternativo	Non premere nessun tasto direzionale. In questo caso snake continuerà a muoversi verso il basso

Nel caso il giocatore non dovesse comunicare nessun input direzionale snake continuerà a muoversi verso il basso.

2.7 - Caso 6

- **Mangia:** si verifica se snake entra in collisione con un oggetto cibo.

Caso d'uso	Mangia.
ID	6
Attore	Snake.
Precondizioni	Il giocatore sta giocando.
Corso d'azione base	Tramite input direzionale snake si dirige verso l'oggetto cibo e ci entra in collisione.
Post condizioni	Una volta entrato in collisione l'oggetto cibo scompare, ne viene generato uno nuovo, il punteggio viene incrementato in base al cibo mangiato e snake si allunga di 1 unità.
Percorso Alternativo	Anche se non viene premuto nessun tasto direzionale e l'oggetto cibo viene creato nelle coordinate (20,y >1) dalla percorso alternativo #4 l'oggetto verrà comunque mangiato.

Come detto precedentemente durante il gioco si potranno incontrare due oggetti cibo diversi ma il caso d'uso resta del tutto speculare per entrambi, infatti ci sarà un aumento di lunghezza di snake e un aumento del punteggio, l'unica differenza tra i due oggetti è che l'oggetto cibo rosso incrementerà il punteggio di 200 mentre quello giallo di 100.

2.8 - Caso 7

- Collisione con il bordo: si verifica quando snake entra in collisione con uno dei 4 bordi di gioco.

Caso d'uso	Collisione con il bordo.
ID	7
Attore	Giocatore.
Precondizioni	Il giocatore da input direzionali.
Corso d'azione base	Snake entra in collisione con uno dei 4 bordi del gioco.
Post condizioni	Caso morte.
Percorso Alternativo	Se il giocatore non da input direzionali snake entrerà in collisione con il bordo inferiore nelle coordinate (20,maxYpb).

2.9 - Caso 8

- Collisione con snake: si verifica quando la testa di snake si scontra con un componente del suo corpo

Caso d'uso	Collisione Snake
ID	8

Attore	Giocatore
Precondizioni	Il giocatore comunica input direzionali e snake deve avere almeno una lunghezza di 5 unità, ovvero deve aver mangiato almeno 4 oggetti cibo.
Corso d'azione base	La testa di snake entra in collisione con una delle parti del suo corpo.
Post condizioni	Caso morte.
Percorso Alternativo	/

Il caso d'uso 8 e 9 portano entrambi alla stessa post condizione, ovvero il caso morte.

2.10 - Caso 9

- Morte: post condizione del caso 8 e 9, si verifica al verificarsi di uno dei due casi d'uso e porta alla fine della partita.

Caso d'uso	Morte.
ID	9
Attore	Giocatore.
Precondizioni	Collisione con il bordo o con snake illustrate nei casi d'uso 8 e 9.
Corso d'azione base	Il gioco termina e il giocatore non può più comunicare input direzionali.
Post condizioni	Il gioco termina, vengono mostrati punteggio e lunghezza snake e appariranno i button: Rigioca e

	Esci.
Percorso Alternativo	/

Verificatosi il caso morte il giocatore non potrà più comunicare gli input direzionali e verrà visualizzata una label contenente i dati della partita.

2.11 - Caso 10

Appariranno anche 2 button: Rigioca ed Esci.

Caso d'uso	Rigioca
ID	10
Attore	Giocatore
Precondizioni	Morte
Corso d'azione base	Il giocatore clicca sul button Rigioca.
Post condizioni	Il giocatore riinizia la partita, caso d'uso 4
Percorso Alternativo	/

2.12 - Caso 11

Caso d'uso	Esci
ID	11
Attore	Giocatore
Precondizioni	Morte
Corso d'azione base	Il giocatore clicca sul button Esci.
Post condizioni	Il giocatore torna nel menù iniziale.
Percorso Alternativo	/

3. Analisi e progettazione

In questa sezione andremo a descrivere le scelte progettuali effettuate per la realizzazione del programma.

3.1 Linguaggio di programmazione

Per la realizzazione del programma è stato utilizzato il linguaggio di programmazione c#, esso è un linguaggio orientato ad oggetti, funzionale e ad eventi sviluppato da Microsoft strettamente legato allo sviluppo di .NET framework, infatti molte delle sue astrazioni, come classi e metodi sono particolarmente adatte a gestire il .NET framework.

Andiamo ad elencare alcune delle caratteristiche base della sintassi:

- Nomi di variabili, funzioni, classi e altri elementi sono sempre sensibili alle minuscole, ovvero "case-sensitive".
- Ogni specifica dev'essere chiusa dal carattere punto e virgola (;).
- Le parentesi graffe ({}) sono usate per raggruppare specifiche.
- Secondo le consuetudini dei linguaggi orientati agli oggetti, le specifiche sono di regola raggruppate in *metodi* (ovvero funzioni), i *metodi* sono raggruppati in *classi*, e le *classi* sono raggruppate nei namespace.

3.2 Ambiente di sviluppo.

Per lo sviluppo del progetto è stato utilizzato l'ambiente di sviluppo Visual Studio Community 2017 che utilizza .NET framework versione 4.5.2 .

3.3 GUI

Per migliorare l'aspetto visivo del gioco abbiamo optato per una applicazione Windows form di tipo grafica.

Le classi grafiche che sono state usate nel progetto sono:

- PictureBox che permette di visualizzare una immagine.
- Button, un oggetto che risponde ad un evento click.
- Label che fornisce testo descrittivo per un controllo.

Il progetto è diviso in 3 form:

- Inizio
- Istruzioni
- CampoGioco

Il form "inizio" può essere considerato il menù iniziale del gioco in quanto è formato da 3 button che permettono di aprire gli altri 2 form del progetto oppure di uscire.



L'immagine del gioco non è visualizzata tramite picturebox ma è stata collocata nella proprietà backgroundImage del form.

Il form "istruzioni" raggiungibile tramite l'evento click sul button "istruzioni" contiene solamente 1 button ("chiudi") che consente di tornare al menù iniziale una volta visualizzate le istruzioni del gioco, e 1 picturebox nel quale è stata allocata l'immagine contenente le istruzioni del gioco.



Questo form è puramente visivo in quanto non è collegato in nessun modo al gioco, ma è stato creato per fornire le istruzioni e i comandi di gioco al giocatore.

Il form “CampoGioco” raggiungibile tramite il button “gioca” del menù iniziale, è il form nel quale si svolge il gioco.

Esso graficamente è formato da:

1 picturebox(pbCampoGioco) nel quale si svolgerà il gioco.

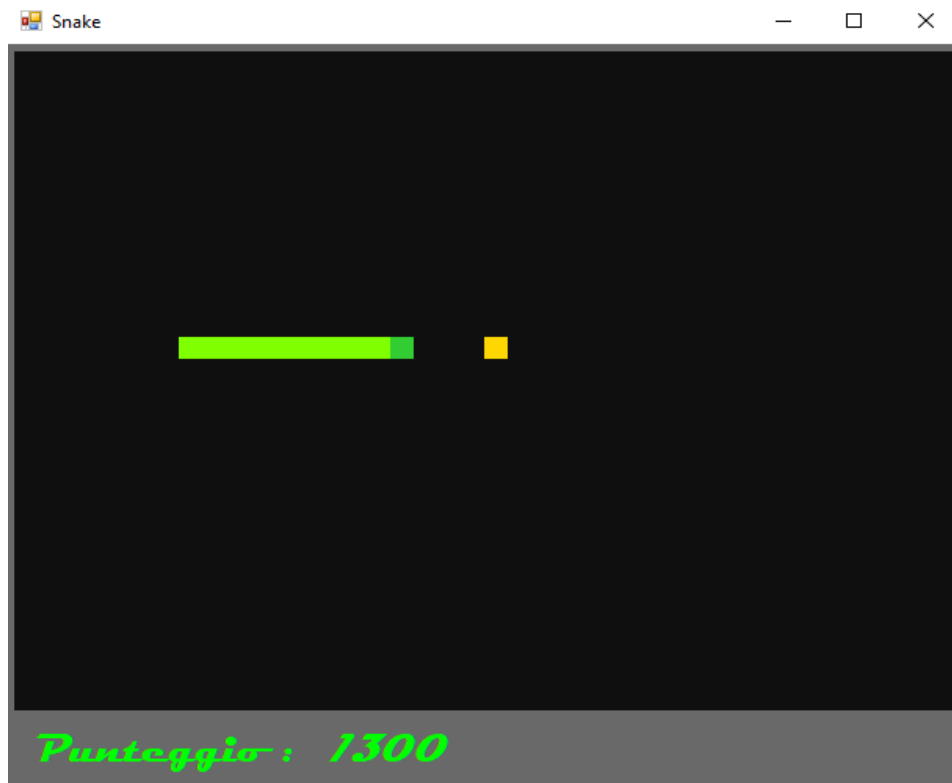
3 label divise in:

- lblFineGioco: è una label che appare al termine della partita, essa contiene il punteggio e la lunghezza raggiunte da snake al termine della partita.
- lblPunteggio: è la label che ci consente di vedere il punteggio in tempo reale durante la partita.
- lblGrafica: è una label puramente grafica al quale è stata modificata solo la proprietà text per far visualizzare(“Punteggio :”)

Ingegneria del Software:

Progetto sessione estiva - 2017

Oltre a questi oggetti nel form sono presenti anche 2 button che verranno visualizzati solo al termine della partita, essi sono btnRigioca e btnEsci che ci consentiranno di ricominciare la partita o di tornare al menù iniziale.



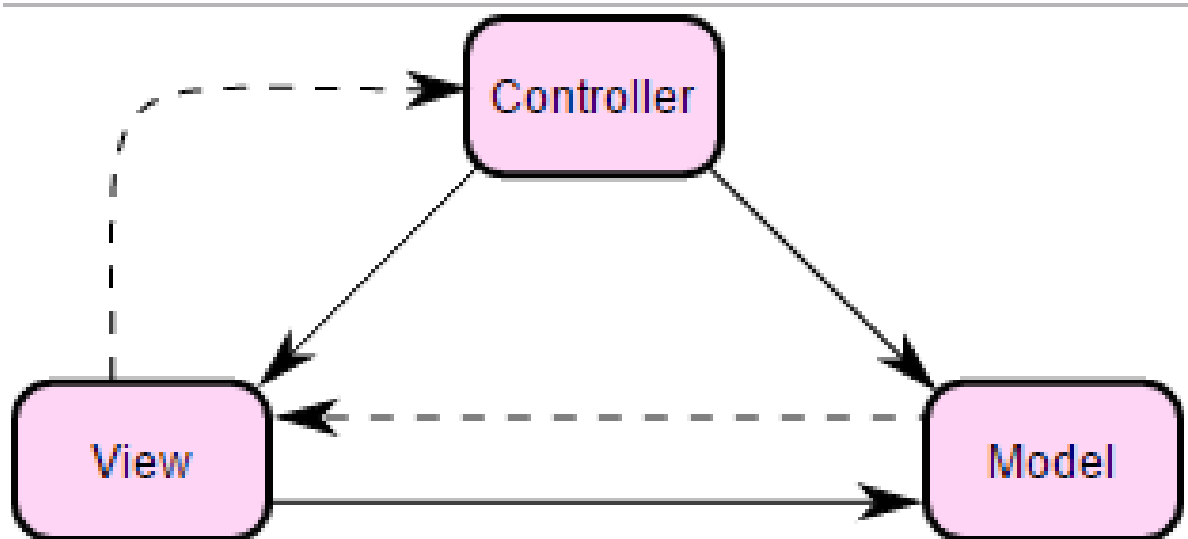
3.4 Design Pattern

Per lo sviluppo del progetto abbiamo deciso di optare per l'utilizzo del design pattern MVC (Model View Control).

Esso è particolarmente utilizzato per le applicazioni che hanno a loro interno un'interfaccia grafica in quanto permette di distaccare la parte logica da quella grafica.

Questo design pattern è formato da 3 componenti principali:

- Model : gestisce direttamente i dati, la logica e le regole dell'applicazione.
- View : ci comunica graficamente le informazioni in output.
- Controller : accetta l'input dato dall'utente e lo converte in comandi per la vista e il modello.



3.5 Organizzazione progetto

Il progetto è stato sviluppato sulla base del design pattern MVC tuttavia non è stato possibile avere un completo distacco della parte logica da quella grafica in quanto esse sono strettamente collegate fra loro e presentavano riferimenti ciclici.

Tuttavia è stato possibile applicare le regole principali del design pattern al progetto “Snake_mvc”.

Snake_mvc è formato da 12 classi che sono state divise per rappresentare i 3 componenti del MVC come segue:

MODELLO	Cibo.cs
	CiboGiallo.cs
	CiboRosso.cs
	Impostazioni.cs
	Partita.cs
	Pezzo.cs
	Snake.cs

VISTA	Inizio.cs
	Istruzioni.cs
	CampoGioco.cs

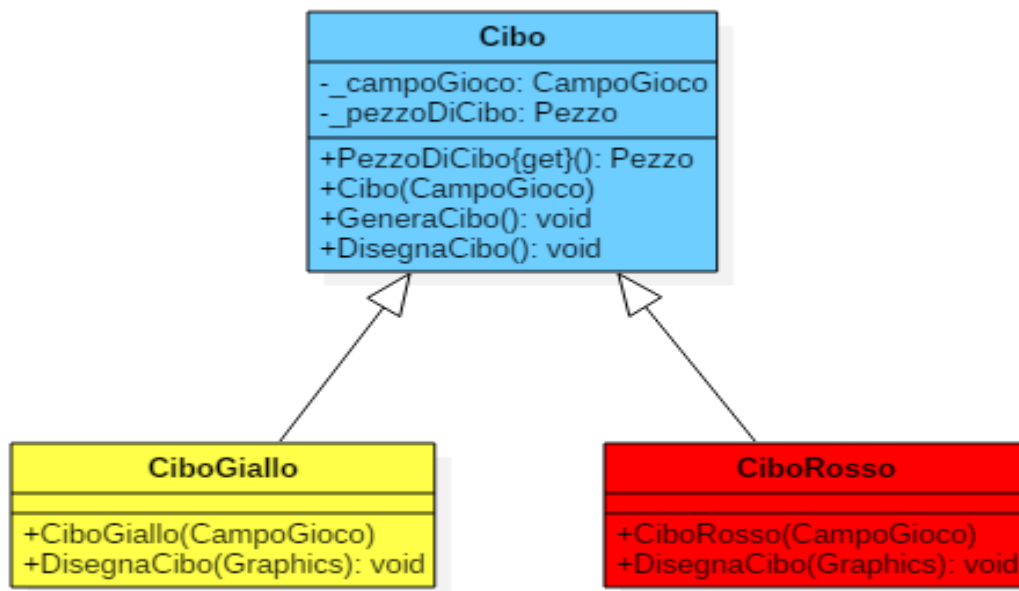
CONTROLLER	Controller.cs
-------------------	---------------

Queste classi saranno spiegate dettagliatamente nel successivo punto della relazione.

3.6 Classi

Andremo ora a descrivere tutte le classi presenti in “Snake_mvc” con relativo schema UML.

3.6.1 Cibo.cs



Nella classe cibo è stato possibile inserire i due concetti di ereditarietà e di polimorfismo, la classe padre è rappresentata dalla classe Cibo che sarà abstract ed eredita le 2 classi figlie CiboGiallo e CiboRosso che rappresenteranno i 2 oggetti di tipo cibo presenti nel gioco.

L'ereditarietà permette alle 2 classi figlio di ereditare il metodo virtual `DisegnaCibo` permettendoci di applicare le modifiche volute nelle sotto classi.

Le librerie utilizzate sono:

```
using System;
using System.Drawing;
```

Il polimorfismo permette alle 2 classi figlio “CiboGiallo” e “CiboRosso” di sovrascrivere il metodo virtual void “DisegnaCibo” modificandone così le proprietà da oggetto a oggetto in base a cosa l’ingegnere del software desidera.

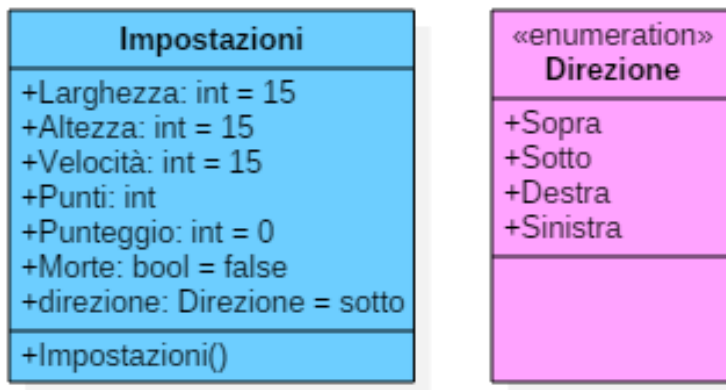
In questa classe è presente anche il concetto di incapsulamento, lo possiamo notare nel metodo pubblico PezzoDiCibo che ci permette attraverso il “get” di leggere le informazioni contenute nell’attributo privato _pezzoDiCibo.

L’incapsulamento, nella metodologia di programmazione orientata agli oggetti impedisce l’accesso ai dettagli di implementazione permettendo così di accedere indirettamente alle strutture dati interne.

Andiamo ora a esaminare gli attributi e i metodi di questa classe.

- _campoGioco: istanza privata del form CampoGioco.
- _pezzoDiCibo: istanza privata del costruttore Pezzo, serve a creare un nuovo oggetto pezzo che sarà associato al cibo.
- PezzoDiCibo: metodo pubblico che ci permette di accedere alle proprietà di sola lettura di _pezzoDiCibo.
- Cibo: costruttore della classe Cibo.cs contenente riferimenti a CampoGioco.
- GeneraCibo: metodo pubblico void usato per generare gli oggetti di tipo cibo e posizzionarli in maniera random nel form, al suo interno sono presenti 2 variabili locali: maxXpb e maxYpb che ci consentiranno di non far posizionare gli oggetti cibo fuori dalla picturebox di gioco.
- DisegnaCibo: metodo pubblico virtual void che verrà ereditato dalle 2 sottoclassi, esso rappresenta il polimorfismo in quanto passa alle classi figlio tutti i parametri che dovranno essere modificati in base alla rappresentazione che si vorrà dare ai 2 oggetti.

3.6.2 Impostazioni.cs



La classe impostazione contiene tutti quegli attributi che ci saranno utili per il corretto funzionamento del gioco.

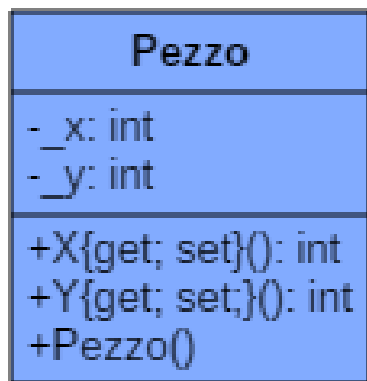
Fa parte della classe Impostazioni.cs anche l'enumerazione "Direzione" che ci consentirà di avere delle costanti associate alla direzione che andremo a richiamare nelle varie classi.

La classe Impostazioni è formata dai seguenti attributi:

- **Larghezza:** intero pubblico e statico che viene settato a 15, esso nel progetto serve a impostare alcune grandezze riferite all'asse x
- **Altezza:** intero pubblico e statico che viene settato a 15, esso nel progetto serve a impostare alcune grandezze riferite all'asse y
- **Velocità:** intero pubblico e statico che viene settato a 15, esso verrà collegato con il timer per impostare la velocità dello spostamento di snake.
- **Punti:** intero pubblico e statico che serve ad aumentare il punteggio.
- **Punteggio:** intero pubblico e statico settato a 0, esso rappresenta il punteggio raggiunto dal giocatore durante la partita.

- **Morte:** attributo booleano pubblico e statico che servirà a verificare se il giocatore perde, esso è settato a false di default e verrà settato a true appena si verificherà una delle condizioni che faranno morire snake.
- **Direzione:** attributo di tipo enum pubblico e statico che servirà a spostare snake, esso è settato nella direzione “sotto” quindi appena iniziato il gioco , snake si sposterà verso il basso.

3.6.3 Pezzo.cs



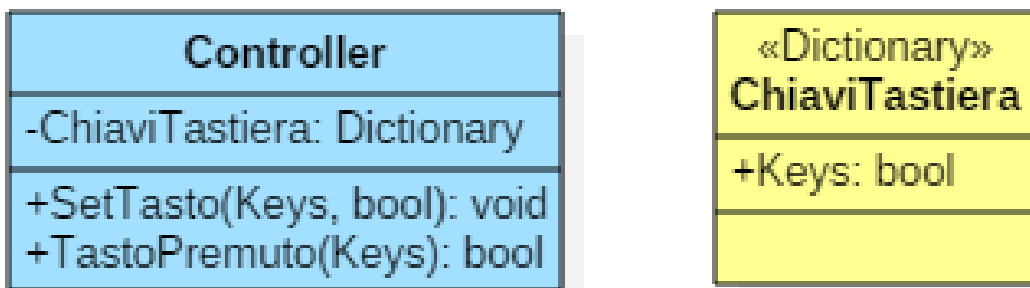
La classe pezzo serve a fornire delle coordinate x e y che sono state usate durante la progettazione del codice.

Gli attributi e i metodi di questa classe sono:

- `_x`: attributo privato intero che rappresenterà l'asse x
- `_y`: attributo privato intero che rappresenterà l'asse y
- `X`: metodo pubblico che ci permette di accedere alle proprietà di scrittura e lettura di `_x`.
- `Y`: metodo pubblico che ci permette di accedere alle proprietà di scrittura e lettura di `_y`.

- Pezzo: metodo costruttore nel quale verranno settati `_x = 0` e `_y = 0`.

3.6.4 Controller.cs



Classe che gestisce i 4 input direzionali dati dall'utente tramite Keys, in controller è presente anche una Dictionary ovvero una sottoclasse che ottiene una raccolta contenente le chiavi della classe Dictionary.

Le librerie utilizzate sono:

```
using System.Collections.Generic;
using System.Windows.Forms;
```

La classe Controller è formata da:

- ChiaviTastiera: campo privato e statico di tipo Dictionary che conterrà le Keys e un valore booleano.
- SetTasto: metodo pubblico statico e void, esso serve a collegare le ChiaviTastiera(Keys) con il valore booleano (TastoPremuto).
- TastoPremuto: metodo pubblico statico e void che serve a controllare se un determinato tasto è premuto o no, il risultato di ciò viene associato ad una variabile locale chiamata "StatoTasto".

3.6.5 Snake.cs

Snake
- _pezziDelSerpente: list<Pezzo> - _campoGioco: CampoGioco - _partita: Partita
+ _PezziDelSerpente{get;}: list<Pezzo> + Snake(CampoGioco, Partita) + CreaSnake(): void + Muori(): void + Mangia(): void + DisegnaPezzoSnake(Graphics, int, Brush): void + MuoviGiocatore(): void

Snake.cs è la classe rappresentante l'oggetto snake ovvero il serpente che controlleremo durante il gioco.

Le librerie utilizzate sono:

```
using System.Collections.Generic;  
using System.Drawing;
```

Al suo interno troveremo dei metodi strettamente legati all'oggetto snake ma anche dei metodi che si interfacciano con le varie azioni di gioco come ad esempio "Mangia" che vede un coinvolgimento dell'oggetto cibo.

La classe Snake.cs è formata come segue:

- `_pezziDelSerpente`: lista privata della classe `Pezzo` che oltre alla creazione dell'oggetto snake sarà utilizzata per rappresentarne graficamente il movimento.
- `_campoGioco`: istanza privata del form `CampoGioco`.
- `_partita`: istanza privata della classe `Partita`.

- `_PezziDelSerpente`: metodo pubblico che ci permette di accedere alle proprietà di sola lettura di `_pezziDelSerpente`.
- `Snake`: costruttore della classe `Snake.cs`, contiene riferimenti a `CampoGioco` e `Partita`.
- `CreaSnake`: metodo pubblico void che serve alla creazione di snake attraverso l'add alla lista `_pezziDelSerpente` di un `Pezzo` chiamato "testa".
- `Muori`: metodo pubblico void che servirà a indicare la fine della partita, esso viene collegato all'attributo `Morte` presente in `Impostazioni.cs`.
- `Mangia`: metodo pubblico e void che viene richiamato quando l'oggetto snake entra in collisione con un oggetto cibo applicandone le conseguenze come l'aumento della grandezza dell'oggetto snake, l'incremento del punteggio e la creazione di un nuovo oggetto `Cibo`.
- `DisegnaPezzoSnake`: metodo pubblico e void che serve alla rappresentazione grafica di snake attraverso la classe `System.Drawing.Graphics`.
- `MuoviGiocatore`: metodo pubblico e void che serve a rappresentare il movimento di snake dato dall'input utente e gestisce le collisioni con il bordo o con lo stesso oggetto snake.

3.6.6 Partita.cs

Partita
- _campoGioco: CampoGioco - _snake: Snake - _cibo: Cibo
+Partita(CampoGioco) +GeneraNuovoCibo(): void + _Cibo({get;}): Cibo +IniziaPartita(): void +AggiornaSchermo(object, EventArgs): void +DisegnaForm(object, PaintEventArgs): void

La classe Partita.cs è un vero e proprio gestore, in quanto gestisce passo per passo lo sviluppo della partita.

Le librerie utilizzate sono:

```
using System;  
using System.Drawing;  
using System.Windows.Forms;
```

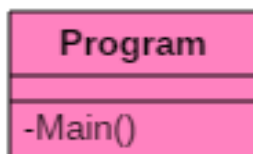
Questa classe contiene passo per passo i metodi che il programma deve richiamare per il corretto funzionamento del gioco.

Partita.cs è formata da:

- _campoGioco: istanza privata del form CampoGioco.
- _snake: istanza privata della classe snake.
- _cibo: istanza privata della classe Cibo.
- Partita: costruttore della classe Partita.cs, contiene riferimenti a CampoGioco.
- GeneraNuovoCibo: metodo pubblico void utilizzato per creare un nuovo oggetto cibo, contiene una variabile random che consente di scegliere tra i 2 oggetti cibo disponibili per poi generarlo.

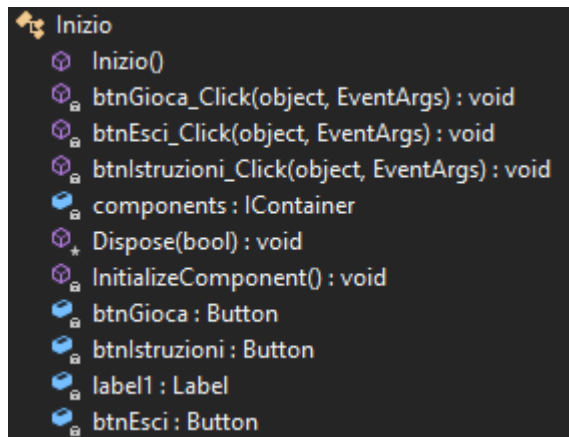
- `_Cibo`: metodo pubblico che ci permette di accedere alle proprietà di sola lettura di `_cibo`.
- `IniziaPartita`: metodo pubblico void che imposta su `visible = false` gli oggetti label e button che diventeranno visibili al giocatore una volta finita la partita, oltre a questo re inizializza le impostazioni di gioco e crea un nuovo oggetto snake.
- `AggiornaSchermo`: metodo pubblico e void che contiene riferimenti agli oggetti e agli eventi, esso non permette al giocatore di muovere snake sopra sé stesso e blocca la picturebox in caso di morte.
- `DisegnaForm`: metodo pubblico e void che contiene riferimenti agli oggetti e agli eventi, permette di disegnare l'oggetto snake e gli oggetti cibo e rende visibile i vari oggetti label e button in caso di morte.

3.6.7 Program.cs



La classe `Program.cs` serve all'avvio dell'applicazione, al suo interno è presente un'istanza di `Inizio.cs` per segnalare al programma che al suo avvio dovrà aprire il form `Inizio.cs`.

3.6.8 Inizio.cs

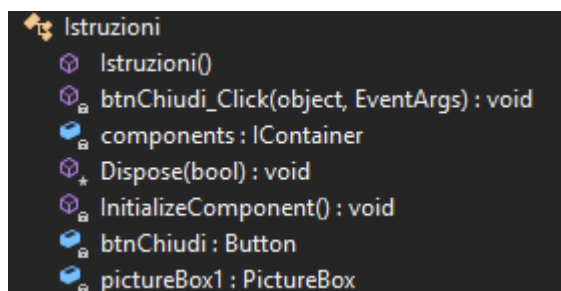


```
Inizio
Inizio()
btnGioca_Click(object, EventArgs) : void
btnEsci_Click(object, EventArgs) : void
btnIstruzioni_Click(object, EventArgs) : void
components : IContainer
Dispose(bool) : void
InitializeComponent() : void
btnGioca : Button
btnIstruzioni : Button
label1 : Label
btnEsci : Button
```

Inizio è il form del menù iniziale, al suo interno sono presenti 1 label puramente grafica e 3 button con i relativi eventi click, essi sono:

- btnGioca_Click: contiene un'istanza di CampoGioco e rende visibile il form CampoGioco all'utente.
- btnIstruzioni_Click: contiene un'istanza di Istruzioni e rende visibile il form Istruzioni all'utente.
- btnEsci_Click: serve al close dell'applicazione.

3.6.9 Istruzioni.cs

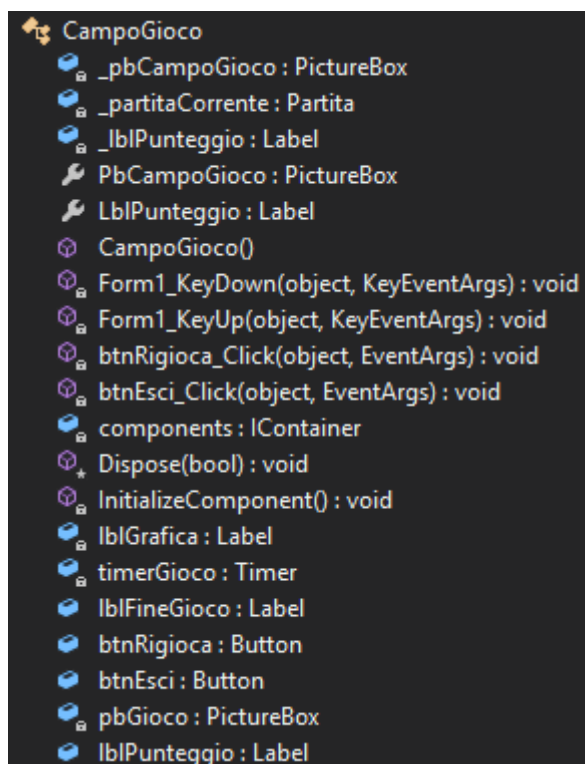


```
Istruzioni
Istruzioni()
btnChiudi_Click(object, EventArgs) : void
components : IContainer
Dispose(bool) : void
InitializeComponent() : void
btnChiudi : Button
pictureBox1 : PictureBox
```

Istruzioni è un form grafico raggiungibile dal menù iniziale, esso contiene una pictureBox con un imageBackGround contenente le istruzioni di gioco e 1 button chiudi.

L'unico evento click presente è btnChiudi_Click che viene utilizzato per il close del form istruzioni.

3.6.10 CampoGioco.cs



CampoGioco è il form di gioco raggiungibile dall'evento click Gioca del menù iniziale, questo form sarà quello in cui il giocatore potrà cimentarsi nel gioco.

Oltre ai componenti grafici illustrati a pag. 13 possiamo notare la presenza di un timer chiamato timerGioco che sarà la base del funzionamento del gioco grazie alla proprietà Interval che sarà collegata alla velocità e la proprietà Tick che servirà ad aggiornare lo schermo, queste 2 proprietà permetteranno alla parte grafica di avere una continuità nel gioco.

In Campogioco.cs sono presenti anche 2 eventi form:

- KeyUp: si verifica quando si rilascia un tasto, all'interno del codice KeyUp verrà associata alla proprietà bool = false di SetTasto.
- KeyDown: si verifica quando si preme un tasto, questo evento verrà associato alla proprietà bool = true di SetTasto comunicando così l'input dato dall'utente.

4 Implementazione

In questa sezione andremo a scrivere e commentare il codice del progetto snake_mvc.

4.1 Pezzo.cs

```
namespace Snake_mvc
{
    public class Pezzo
    {
        private int _x;
        private int _y;
        // proprietà di incapsulamento per accedere alle proprietà di lettura e
        // scrittura di _x
        public int X
        {
            get { return _x; }
            set { _x = value; }
        }
        // proprietà di incapsulamento per accedere alle proprietà di lettura e
        // scrittura di _y
        public int Y
        {
            get { return _y; }
            set { _y = value; }
        }

        // costruttore
        public Pezzo()
        {
            // inizializzazione attributi
            _x = 0;
            _y = 0;
        }
    }
}
```

4.2 Impostazioni.cs

```
namespace Snake_mvc
{
    // enumerazione della Direzione
    public enum Direzione
    {
        Sopra,
        Sotto,
        Destra,
        Sinistra
    }
}
```

Ingegneria del Software:

Progetto sessione estiva - 2017

```
public class Impostazioni
{
    // attributi
    public static int Larghezza;    // grandezza x
    public static int Altezza;      // grandezza y
    public static int Velocità;     // velocità di spostamento
    public static int Punti;        // intero che sarà sommato al punteggio
    public static int Punteggio;    // punteggio del giocatore
    public static bool Morte;       // variabile booleana per controllare l'evento
    public static Direzione direzione; // enumerazione

    // costruttore
    public Impostazioni()
    {
        // inizializzazione attributi
        Larghezza = 15;
        Altezza = 15;
        Velocità = 15;
        Punteggio = 0;
        Morte = false;
        direzione = Direzione.Sotto;    // snake inizierà a muoversi verso il basso
    }
}
```

4.3 Controller.cs

```
using System.Collections.Generic;
using System.Windows.Forms;

namespace Snake_mvc
{
    // classe controller per gli input dati dall'utente
    class Controller
    {
        private static Dictionary<Keys, bool> ChiaviTastiera = new Dictionary<Keys,
        bool>();

        public static void SetTasto(Keys key, bool TastoPremuto)
        {
            ChiaviTastiera[key] = TastoPremuto;
        }

        public static bool TastoPremuto(Keys tasto)
        {
            bool StatoTasto;
            if (ChiaviTastiera.TryGetValue(tasto, out StatoTasto))
            {
                return StatoTasto;
            }
            return false;
        }
    }
}
```

4.4 Cibo.cs

```
using System;
using System.Drawing;

namespace Snake_mvc
{
    // classe astratta dell'oggetto cibo
    abstract class Cibo
    {
        private CampoGioco _campoGioco;
        private Pezzo _pezzoDiCibo = new Pezzo();

        // incapsulamento per accedere alle proprietà di lettura
        public Pezzo PezzoDiCibo
        {
            get { return _pezzoDiCibo; }
        }

        // costruttore
        public Cibo(CampoGioco view)
        {
            _campoGioco = view;
        }

        // metodo per generare il cibo
        public void GeneraCibo()
        {
            // variabili che rappresentano la grandezza massima della picturebox
            int maxXpb = _campoGioco.PbCampoGioco.Size.Width / Impostazioni.Larghezza;
            int maxYpb = _campoGioco.PbCampoGioco.Size.Height / Impostazioni.Altezza;

            // funzione random per posiziobare il cibo nel campo da gioco
            Random random = new Random();
            _pezzoDiCibo = new Pezzo();
            _pezzoDiCibo.X = random.Next(0, maxXpb);
            _pezzoDiCibo.Y = random.Next(0, maxYpb);
        }

        // metodo virtuale per disegnare il cibo
        public virtual void DisegnaCibo(Graphics gioco)
        {
        }
    }
}
```

4.5 CiboGiallo.cs

```
using System.Drawing;

namespace Snake_mvc
{
    // classe erede di Cibo
    class CiboGiallo : Cibo
    {
        // costruttore con riferimento alla base
        public CiboGiallo(CampoGioco view) : base(view)
        {
        }
        // metodo sovrascritto presente in Cibo
        public override void DisegnaCibo(Graphics gioco)
        {
            // rettangolo rappresentante il cibo con le varie proprietà
            gioco.FillRectangle(Brushes.Gold,
                               new Rectangle(PezzoDiCibo.X * Impostazioni.Larghezza,
                                              PezzoDiCibo.Y * Impostazioni.Altezza,
                                              Impostazioni.Larghezza, Impostazioni.Altezza));
            // l'oggetto CiboGiallo è associato a 100 punti
            Impostazioni.Punti = 100;
        }
    }
}
```

4.6 CiboRosso.cs

```
using System.Drawing;

namespace Snake_mvc
{
    // classe erede di Cibo
    class CiboRosso : Cibo
    {
        // costruttore di CiboRosso con riferimento alla base
        public CiboRosso(CampoGioco view) : base(view)
        {
        }
        // metodo sovrascritto presente in Cibo
        public override void DisegnaCibo(Graphics gioco)
        {
            // rettangolo rappresentante il cibo con le varie proprietà
            gioco.FillRectangle(Brushes.Red,
                               new Rectangle(PezzoDiCibo.X * Impostazioni.Larghezza,
                                              PezzoDiCibo.Y * Impostazioni.Altezza,
                                              Impostazioni.Larghezza, Impostazioni.Altezza));
            // l'oggetto CiboRosso è associato a 200 punti
            Impostazioni.Punti = 200;
        }
    }
}
```

4.7 Snake.cs

```
using System.Collections.Generic;
using System.Drawing;

namespace Snake_mvc
{
    class Snake
    {
        private List<Pezzo> _pezziDelSerpente = new List<Pezzo>();
        private CampoGioco _campoGioco;
        private Partita _partita;

        // proprietà di incapsulamento per accedere alle proprietà di lettura della
        lista _pezziDelSerpente
        public List<Pezzo> _PezziDelSerpente
        {
            get { return _pezziDelSerpente; }
        }

        // costruttore
        public Snake(CampoGioco campoGioco, Partita partita)
        {
            _campoGioco = campoGioco;
            _partita = partita;
        }

        // metodo per creare la testa di snake
        public void CreaSnake()
        {
            Pezzo testa = new Pezzo();
            testa.X = 20; // cordinata x della creazione della testa
            testa.Y = 1; // cordinata y della creazione della testa
            _pezziDelSerpente.Add(testa);
        }

        // metodo muori
        public void Muori()
        {
            // imposta a true la variabile booleana morte
            Impostazioni.Morte = true;
        }

        // metodo mangia
        public void Mangia()
        {
            // aggiungi un rettangolo al corpo
            Pezzo rettangolo = new Pezzo
            {
                X = _pezziDelSerpente[_pezziDelSerpente.Count - 1].X,
                Y = _pezziDelSerpente[_pezziDelSerpente.Count - 1].Y
            };
            _pezziDelSerpente.Add(rettangolo);

            // aggiorna punteggio sommandogli i punti
            Impostazioni.Punteggio += Impostazioni.Punti;
            _campoGioco.LblPunteggio.Text = Impostazioni.Punteggio.ToString();
        }
    }
}
```

Ingegneria del Software:

Progetto sessione estiva - 2017

```
// genera un nuovo oggetto cibo dopo che è stato mangiato
_partita.GeneraNuovoCibo();
}

// Metodo per disegnare snake
public void DisegnaPezzoSnake(Graphics gioco, int i, Brush coloreSnake)
{
    gioco.FillRectangle(coloreSnake,
        new Rectangle(_pezziDelSerpente[i].X * Impostazioni.Larghezza,
            _pezziDelSerpente[i].Y * Impostazioni.Altezza,
            Impostazioni.Larghezza, Impostazioni.Altezza));
}

// Metodo movimento (comprende le collisioni con il bordo e con il cibo)
public void MuoviGiocatore()
{
    for (int i = _pezziDelSerpente.Count - 1; i >= 0; i--)
    {
        // muovi la testa
        if (i == 0)
        {
            switch (Impostazioni.direzione)
            {
                case Direzione.Destra:
                    _pezziDelSerpente[i].X++;
                    break;
                case Direzione.Sinistra:
                    _pezziDelSerpente[i].X--;
                    break;
                case Direzione.Sopra:
                    _pezziDelSerpente[i].Y--;
                    break;
                case Direzione.Sotto:
                    _pezziDelSerpente[i].Y++;
                    break;
            }

            // ottieni x e y massimi della picturebox
            int maxXpb = _campoGioco.PbCampoGioco.Size.Width /
Impostazioni.Larghezza;
            int maxYpb = _campoGioco.PbCampoGioco.Size.Height /
Impostazioni.Altezza;

            // rileva collisioni con i bordi del gioco(PictureBox)
            if (_pezziDelSerpente[i].X < 0 || _pezziDelSerpente[i].Y < 0
                || _pezziDelSerpente[i].X >= maxXpb || _pezziDelSerpente[i].Y >=
maxYpb)
            {
                Muori();
            }

            // rileva collisioni con il corpo
            for (int j = 1; j < _pezziDelSerpente.Count; j++)
            {
                if (_pezziDelSerpente[i].X == _pezziDelSerpente[j].X &&
                    _pezziDelSerpente[i].Y == _pezziDelSerpente[j].Y)
                {
                    Muori();
                }
            }

            // rileva collisioni con il cibo
            if (_pezziDelSerpente[0].X == _partita._Cibo.PezzoDiCibo.X &&
```

```
        _pezziDelSerpente[0].Y == _partita._Cibo.PezzoDiCibo.Y)
    {
        Mangia();
    }
}
else
{
    // muovi il corpo
    _pezziDelSerpente[i].X = _pezziDelSerpente[i - 1].X;
    _pezziDelSerpente[i].Y = _pezziDelSerpente[i - 1].Y;
}
}
}
}
}
```

4.8 Partita.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace Snake_mvc
{
    class Partita
    {
        private CampoGioco _campoGioco;
        private Snake _snake;
        private Cibo _cibo;

        public Partita(CampoGioco campoGioco)
        {
            _campoGioco = campoGioco;
        }

        // metodo per generare casualmente gli oggetti cibo
        public void GeneraNuovoCibo()
        {
            Random sceltaCibo = new Random();
            if (sceltaCibo.NextDouble() >= 0.3)
            {
                _cibo = new CiboGiallo(_campoGioco); // se la la funzione random
                restituisce un double da 0.31 a 1.0 genera un oggetto CiboGiallo
            }
            else
            {
                _cibo = new CiboRosso(_campoGioco); // se la la funzione random
                restituisce un double da 0.01 a 0.3 genera un oggetto CiboRosso
            }

            _cibo.GeneraCibo(); // richiamo al metodo GeneraCibo
        }
        // incapsulamento
        public Cibo _Cibo
        {
            get { return _cibo; }
        }

        public void IniziaPartita()
```


Ingegneria del Software:

Progetto sessione estiva - 2017

```
{
    // nascondi la label e i 2 button
    _campoGioco.lblFineGioco.Visible = false;
    _campoGioco.btnRigioca.Visible = false;
    _campoGioco.btnRigioca.Enabled = false;
    _campoGioco.btnEsci.Visible = false;
    _campoGioco.btnEsci.Enabled = false;

    // imposta le impostazioni di default
    new Impostazioni();

    // crea un nuovo oggetto giocatore
    if(_snake != null)
        _snake._PezziDelSerpente.Clear();

    _snake = new Snake(_campoGioco, this);
    _snake.CreaSnake();

    // crea un nuovo oggetto cibo
    GeneraNuovoCibo();
    // associa alla label punteggio l'attributo punteggio
    _campoGioco.lblPunteggio.Text = Impostazioni.Punteggio.ToString();
}

// richiamato ad ogni tick del timer
public void AggiornaSchermo(object sender, EventArgs e)
{
    // controllo morte
    if (Impostazioni.Morte)
    {
    }
    else
    {
        // Controllo per non far muovere snake sopra se stesso
        if (Controller.TastoPremuto(Keys.Right) && Impostazioni.direzione !=
Direzione.Sinistra)
            Impostazioni.direzione = Direzione.Destra;

        else if (Controller.TastoPremuto(Keys.Left) && Impostazioni.direzione !=
Direzione.Destra)
            Impostazioni.direzione = Direzione.Sinistra;
        else if (Controller.TastoPremuto(Keys.Up) && Impostazioni.direzione !=
Direzione.Sotto)
            Impostazioni.direzione = Direzione.Sopra;
        else if (Controller.TastoPremuto(Keys.Down) && Impostazioni.direzione !=
Direzione.Sopra)
            Impostazioni.direzione = Direzione.Sotto;

        _snake.MuoviGiocatore();
    }
    // Blocca la picturebox di gioco
    _campoGioco.PbCampoGioco.Invalidate();
}

// metodo per disegnare
public void DisegnaForm(object sender, PaintEventArgs e)
{
    Graphics gioco = e.Graphics;

    if (!Impostazioni.Morte)
```

Ingegneria del Software:

Progetto sessione estiva - 2017

```
{
    // imposta il colore di snake
    Brush coloreSnake;

    // disegna snake
    for (int i = 0; i < _snake._PezziDelSerpente.Count; i++)
    {
        if (i == 0)
            coloreSnake = Brushes.LimeGreen; // colora la testa di snake
        else
            coloreSnake = Brushes.Chartreuse; // colora il corpo di snake

        // richiamo al metodo disegna snake (classe Snake)
        _snake.DisegnaPezzoSnake(gioco, i, coloreSnake);

        // richiamo al metodo disegna cibo (classe cibo)
        _cibo.DisegnaCibo(gioco);
    }
}

else
{
    // associamo alla stringa morte il punteggio e la lunghezza di snake
    string morte = ("Gioco Finito! \nPunteggio Finale: " +
Impostazioni.Punteggio + "\nSnake di lunghezza: " + _snake._PezziDelSerpente.Count);
    // rendiamo visibili la label e i 2 button
    _campoGioco.lblFineGioco.Text = morte;
    _campoGioco.lblFineGioco.Visible = true;
    _campoGioco.btnRigioca.Visible = true;
    _campoGioco.btnRigioca.Enabled = true;
    _campoGioco.btnEsci.Visible = true;
    _campoGioco.btnEsci.Enabled = true;
}
}
}
```

4.9 Inizio.cs

```
using System;
using System.Windows.Forms;

namespace Snake_mvc
{
    public partial class Inizio : Form
    {
        public Inizio()
        {
            InitializeComponent();

            // visualizza il form CampoGioco dove è possibile iniziare a giocare
            private void btnGioca_Click(object sender, EventArgs e)
            {
                Form campoGioco = new CampoGioco();
                campoGioco.Show();
            }

            // esce dal programma
            private void btnEsci_Click(object sender, EventArgs e)
            {
                this.Close();
            }

            // visualizza il form contenente le istruzioni del gioco
            private void btnIstruzioni_Click(object sender, EventArgs e)
            {
                Form Istruzioni = new Istruzioni();
                Istruzioni.Show();
            }
        }
    }
}
```

4.10 Istruzioni.cs

```
using System;
using System.Windows.Forms;

namespace Snake_mvc
{
    public partial class Istruzioni : Form
    {
        public Istruzioni()
        {
            InitializeComponent();

            private void btnChiudi_Click(object sender, EventArgs e)
            {
                this.Close();
            }
        }
    }
}
```

```
}  
}
```

4.11 CampoGioco.cs

```
using System;  
using System.Windows.Forms;  
  
namespace Snake_mvc  
{  
    public partial class CampoGioco : Form  
    {  
        // istanze  
        private PictureBox _pbCampoGioco;  
        private Partita _partitaCorrente;  
        private Label _lblPunteggio;  
  
        // proprietà di incapsulamento per accedere alle proprietà di lettura di  
        _pbCampoGioco  
        public PictureBox PbCampoGioco  
        {  
            get { return _pbCampoGioco; }  
        }  
        // proprietà di incapsulamento per accedere alle proprietà di lettura di  
        _lblPunteggio  
        public Label LblPunteggio  
        {  
            get { return _lblPunteggio; }  
        }  
        // costruttore  
        public CampoGioco()  
        {  
            InitializeComponent();  
            _lblPunteggio = lblPunteggio;  
            _pbCampoGioco = pbGioco;  
  
            // eventi key del form  
            KeyDown += Form1_KeyDown;  
            KeyUp += Form1_KeyUp;  
  
            // imposta le impostazioni di default  
            new Impostazioni();  
  
            _partitaCorrente = new Partita(this);  
            _partitaCorrente.IniziaPartita();  
            pbGioco.Paint += _partitaCorrente.DisegnaForm;  
  
            // imposta la velocità e il timer(start)  
            timerGioco.Interval = 1000 / Impostazioni.Velocità;  
            timerGioco.Tick += _partitaCorrente.AggiornaSchermo;  
            timerGioco.Start();  
        }  
        // se un tasto è premuto allora SetTasto = true  
        private void Form1_KeyDown(object sender, KeyEventArgs e)  
        {  
            Controller.SetTasto(e.KeyCode, true);  
        }  
        // se un tasto viene rilasciato allora SetTasto = false  
        private void Form1_KeyUp(object sender, KeyEventArgs e)  
        {  
            Controller.SetTasto(e.KeyCode, false);  
        }  
    }  
}
```

Ingegneria del Software:

Progetto sessione estiva - 2017

```
    }
    // evento click di btnRigioca
    private void btnRigioca_Click(object sender, EventArgs e)
    {
        _partitaCorrente.IniziaPartita(); // inizia nuova partita
    }
    // evento click di btnEsci
    private void btnEsci_Click(object sender, EventArgs e)
    {
        this.Close(); // chiudi
    }
}
}
```

4.12 Program.cs

```
using System;
using System.Windows.Forms;

namespace Snake_mvc
{
    static class Program
    {
        /// <summary>
        /// Punto di ingresso principale dell'applicazione.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Inizio()); // avvio del form del menù iniziale
        }
    }
}
```

5. Test

Black-Box Test.

Il black-box test è un metodo di test del software che esamina la funzionalità di un'applicazione senza esaminare le sue strutture interne o le sue funzioni.

I casi di prova sono costruiti attorno alle specifiche e ai requisiti, vale a dire che cosa dovrebbe fare l'applicazione.

Il progettista di test seleziona gli ingressi validi e non validi e determina l'output corretto.

Iniziamo ora la fase di test.

Come prima cosa bisogna avviare l'applicativo ovvero recarsi nella cartella Snake_mvc/obj/Debug ed avviare l'applicazione snake_mvc.exe.

Fatto ciò vediamo aprirsi il menù iniziale come visto nel caso d'uso n.1



Una volta aperto il menù iniziale andiamo a testare il programma iniziando dalla visualizzazione delle istruzioni come nel caso d'uso n.2

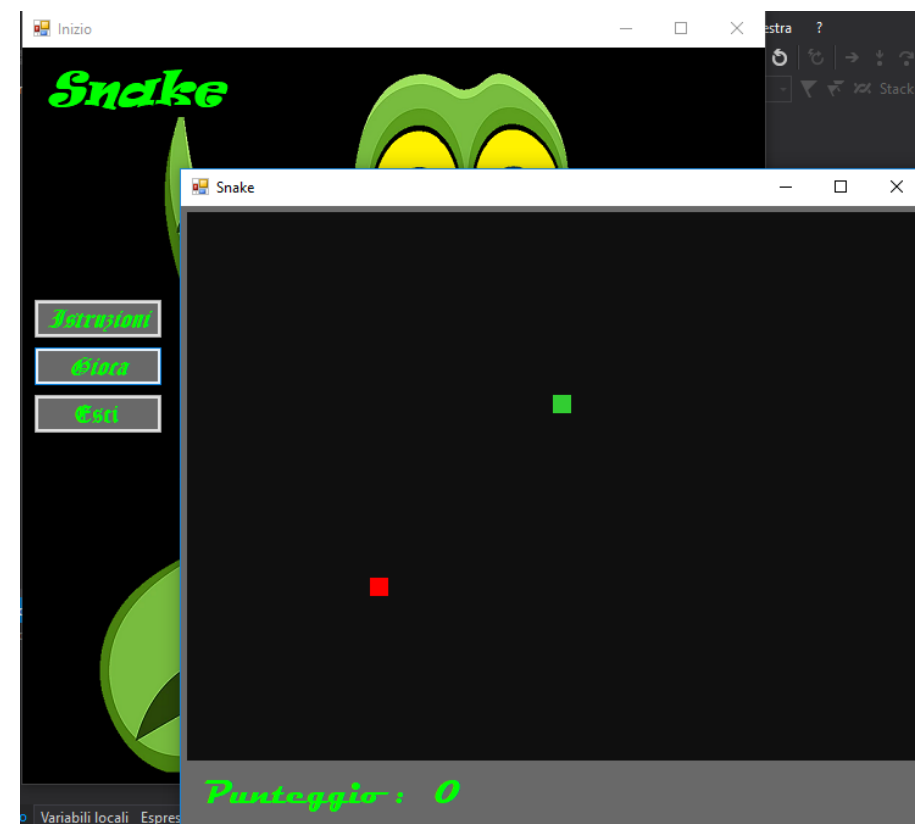
Ingegneria del Software:

Progetto sessione estiva - 2017



Una volta visualizzate le istruzioni torniamo nel menù iniziale chiudendo la finestra contenente le istruzioni facendo click sull'apposito comando chiudi verificando così la post condizione del caso d'uso n.2

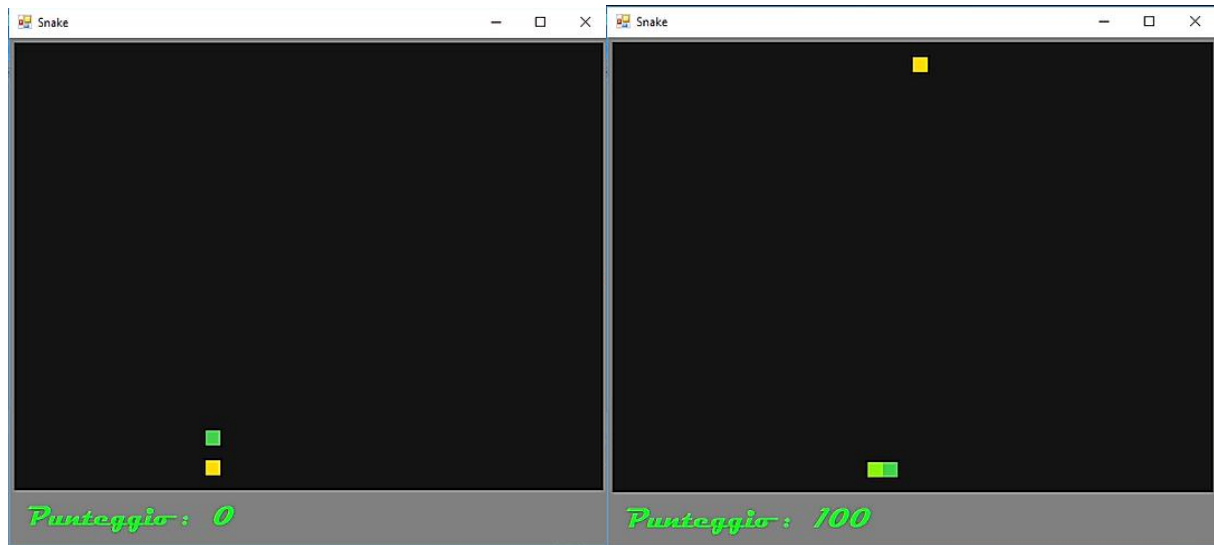
Tornati nel menù iniziale avviamo il gioco cliccando su gioca, apparirà così la finestra di gioco come nel caso d'uso n.4



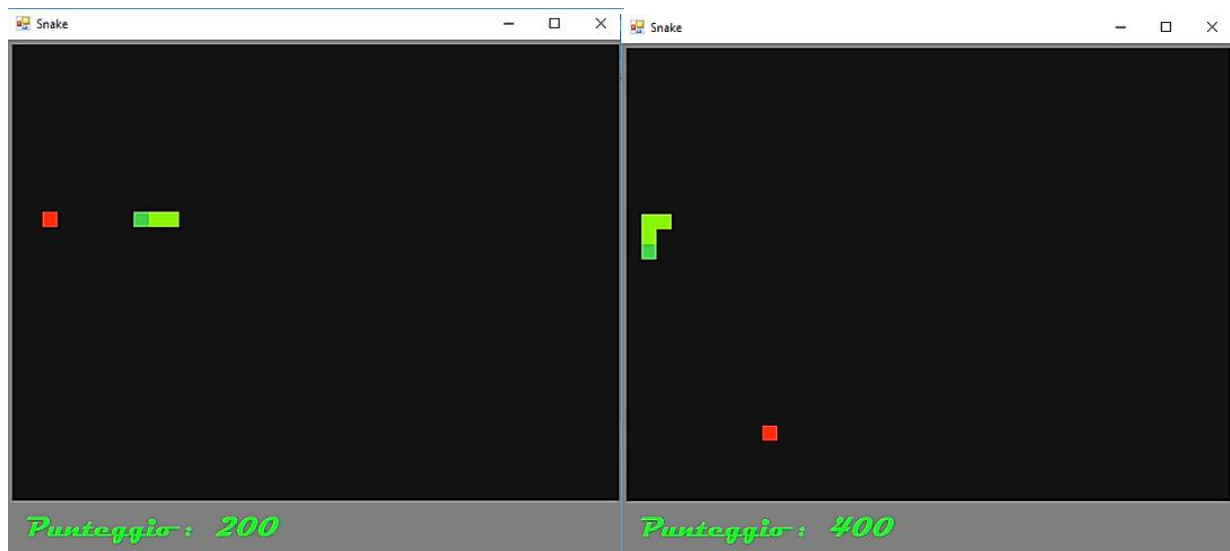
Ingegneria del Software:

Progetto sessione estiva - 2017

Andiamo ora a testare tutti i casi d'uso legati al gioco partendo dall'azione di mangiare del caso d'uso n.6 testando così anche le sue post-condizioni ovvero l'aumento di punteggio e l'allungamento di snake.



Nel caso l'oggetto mangiato sia un cibo rosso l'aumento di punteggio sarà di 200.



Ingegneria del Software:

Progetto sessione estiva - 2017

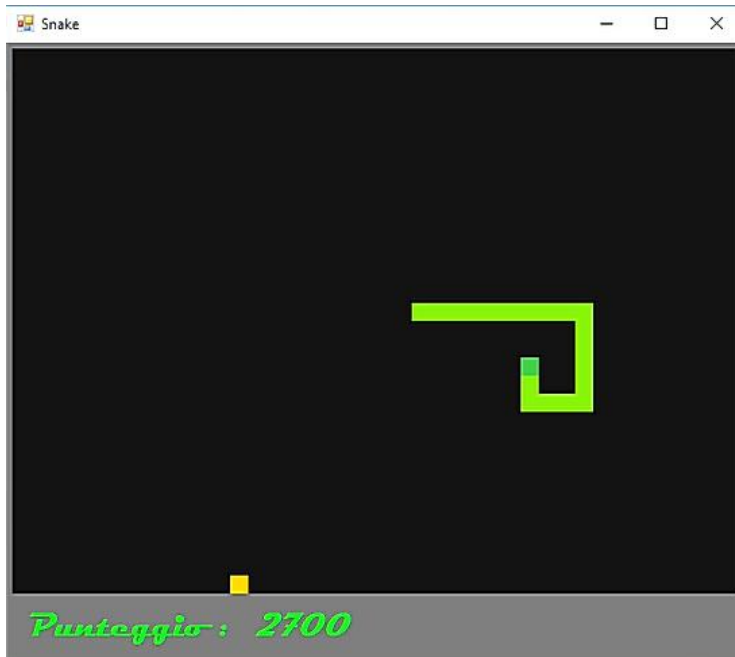
Durante il gioco si potranno verificare degli eventi chiamati collisioni, andiamo a testare l'evento "collisione bordo" nel quale snake va a scontrarsi con uno dei bordi del campo di gioco come illustrato nel caso d'uso n.7.



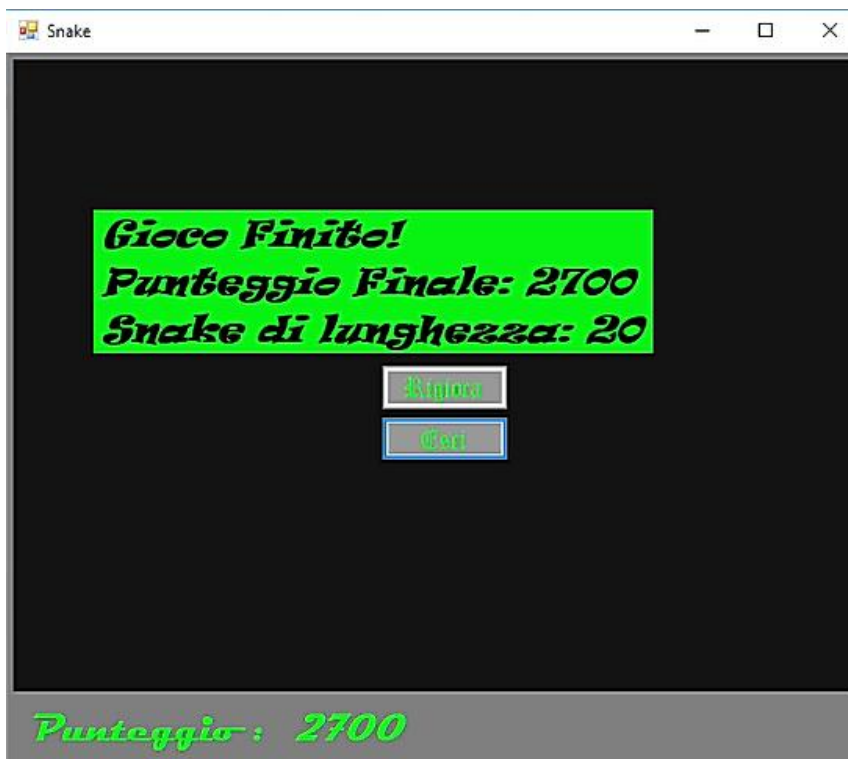
Al verificarsi della collisione si verificherà la sua post-condizione ovvero il caso morte con l'apparizione del punteggio ottenuto, la lunghezza e la possibilità di rigiocare o uscire



Un altro evento collisione che si può verificare durante il gioco è la collisione di snake con se stesso, caso d'uso n.8. nel quale la testa di snake entra in collisione con una parte del suo corpo.



Una volta verificatosi questo evento si avrà una post-condizione uguale a quella appena vista nella collisione con il bordo.

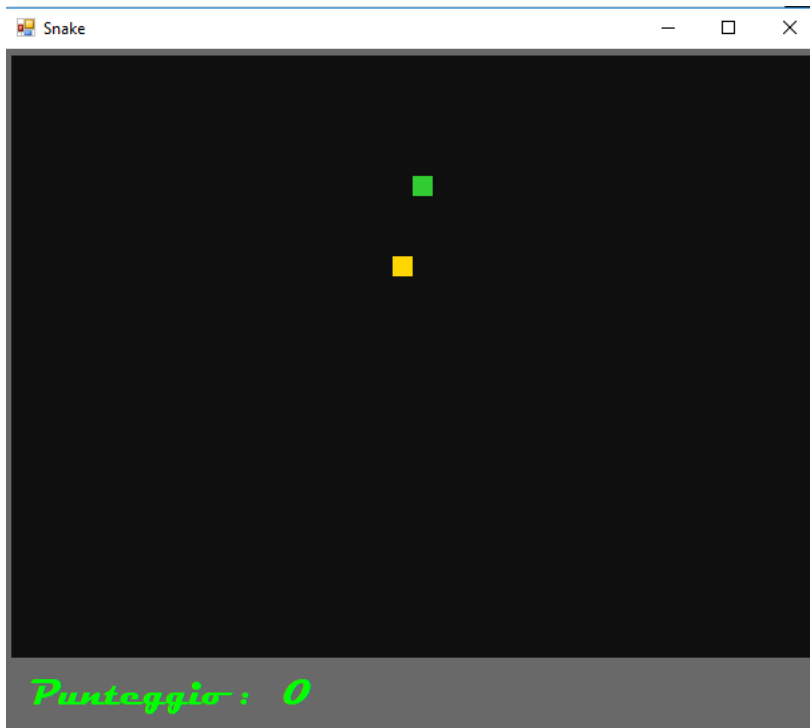


Ingegneria del Software:

Progetto sessione estiva - 2017

Infine procediamo a testare i 2 comandi rigioca ed esci.

Cliccando su rigioca il gioco riinizierà da capo, caso d'uso n.10



Mentre se il giocatore farà click su esci, il form CampoGioco si chiuderà tornando così al menù iniziale.



Per chiudere l'applicazione basterà fare click su esci come visto nel caso d'uso n.3, per testare ciò abbiamo avviato l'applicazione tramite visual studio per controllarne la terminazione effettiva.

Il programma '[81140] Snake_mvc.exe' è terminato con il codice 0 (0x0).

6. Compilazione ed esecuzione

L'ambiente di sviluppo utilizzato è visual studio community 2017 con .net Framework versione 4.5.2.

Per l'avvio dell'applicazione è consigliato dunque avere una versione di Windows con .net Framework versione 4.5.2 o superiore.

Per compilare il progetto si dovrà:

- Estrarre la cartella snake_mvc dal file in formato .rar
- Avviare il file snake_mvc.sln (soluzione)
- Una volta avviato il compilatore cliccare su "compila"

Al termine della compilazione saranno disponibili due cartelle nelle quali sarà possibile eseguire l'applicativo raggiungendo il file .exe attraverso il seguente percorso:

snake_mvc / obj / debug / snake_mvc.exe

Il programma è stato eseguito e quindi testato su diversi computer

Sistema Operativo	RAM	CORE	Architettura
Windows 10	4 Gb	Intel(r) Core(TM) i5- 6200U CPU @ 2.30GHz 2.40GHz	64 bit
Windows 10	8 Gb	Intel(r) Core(TM) i7- 6500U CPU @ 2.50GHz 2.59GHz	64 bit
Windows 10	8 Gb	Intel(r) Core(TM) i5- 4210U CPU @ 1.70GHz 2.40GHz	64 bit