Laboratorio di informatica – classe 4ª – data 29/11/2016

Gioco snake

Si vuole realizzare il gioco snake. Lo scopo è quello di muovere un "serpente" in una griglia allo scopo di "mangiare" dei bersagli che compaiono a coordinate casuali. Si vuole separare la gestione della UI da quella del movimento dello snake, in modo che il gioco possa essere realizzato con UI diverse. A questo scopo si ipotizza la realizzazione di un oggetto che incapsula:

1. Il movimento dello snake nella griglia.
2. La cattura del bersaglio, con conseguente "allungamento" dello snake.
3. La posizione di ogni elemento dello snake (alternativamente o in aggiunta: fornire la posizione degli elementi per i quali è stata modificata).

Note sull'implementazione della UI

Per visualizzare la griglia, compresi snake e bersaglio, si può immaginare di creare una matrice di controlli (`Panel` , ad esempio) che rappresentano le celle della griglia. Questi verranno colorati per identificare il bersaglio e lo snake. (Le coordinate degli elementi dello snake corrisponderanno alle coordinate della matrice contenente il controlli.)

Problema: durante la creazione, trasformare le coordinate di riga e colonna della griglia nelle coordinate in pixel del controlli, da applicare alle proprietà `Left` e `Top`.

Implementazione alternativa

Invece di creare un controllo per ogni cella della griglia, si creeranno (e distruggeranno) i soli controlli necessari a visualizzare il bersaglio e lo snake.

Problema1: durante la creazione di un controllo, trasformare la sua posizione nella griglia (riga e colonna) nelle coordinate in pixel da applicare alle proprietà `Left` e `Top`.

Problema2: distruggere i controlli.

Implementazione dello snake

Nel realizzare lo snake vi sono due questioni. La prima è che l'implementazione deve essere completamente indipendente dalla UI. La seconda riguarda il procedimento da utilizzare per muovere lo snake, e cioè per stabilire le coordinate dei suoi elementi.

Progetto ipotetico dello snake

Segue lo scheletro di una ipotetica classe che implementa lo snake:

```
public class Snake
{
    public Snake(Dimensione dimGriglia, Posizione posIniziale) {...}

    public bool Muovi(Direzione direzione, Posizione posBersaglio) {...}
```

```
    public Posizione[] ElementiSnake() {...}
}
```

Tipi di supporto per gestire coordinate e direzione

```
public enum Direzione
{
    Sinistra,
    Destra,
    Basso,
    Alto
}
```

```
public struct Posizione
{
    public int X;
    public int Y;
}
```

```
public struct Dimensione
{
    public int Larghezza;
    public int Altezza;
}
```

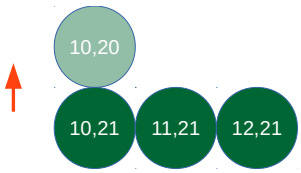
Alcune note:

- I tipi `Direzione`, `Posizione` e `Dimensione` semplificano la gestione delle coordinate e del movimento dello snake. (Nota bene: la loro implementazione come *record* è ipotetica; un'implementazione alternativa può prevedere dei costruttori e la verifica che le coordinate utilizzate siano valide.)
- Il metodo `Muovi()` riceve la direzione e la posizione dell'attuale bersaglio. Lo snake dovrà spostarsi nella direzione indicata e stabilire se il bersaglio è stato catturato. Il metodo restituisce *true* se il bersaglio è stato catturato, *false* in caso contrario.
- `ElementiSnake()` restituisce un vettore delle posizioni degli elementi dello snake, partendo dalla testa. (Questo sarà utilizzato per aggiornare la UI.)

Rappresentazione interna e movimento dello snake

Lo snake si può rappresentare mediante una lista di posizioni (X, Y); il primo elemento della lista è la *testa*. Implementare il movimento è semplice: prima di muovere la *testa* in accordo alla direzione specificata, si assegna a ogni elemento la posizione dell'elemento che lo precede.

Prima



Dopo

Se la testa oltrepassa i confini della griglia, basta reimpostare le sue coordinate in modo che venga posizionata sul confine opposto.

Cattura del bersaglio e allungamento dello snake

Se, dopo essersi mosso, lo snake cattura il bersaglio, è sufficiente aggiungere un nuovo elemento la cui posizione sia uguale a quella che aveva l'ultimo elemento prima che lo snake si spostasse.