



1506  
UNIVERSITÀ  
DEGLI STUDI  
DI URBINO  
CARLO BO

CORSO DI LAUREA IN  
**INFORMATICA APPLICATA**  
SCUOLA DI  
SCIENZE TECNOLOGIE E FILOSOFIA DELL'INFORMAZIONE

## ***Corso di Programmazione ad Oggetti e Ingegneria del Software***

Progetto per l'anno accademico 2018-2019

Studenti:

Alessandro Bernardi - Matricola: 284968

Denil Nicolosi - Matricola: 284720



# Indice

<b>1. Specifica del problema</b>	<b>5</b>
<b>2. Specifica dei requisiti</b>	<b>6</b>
2.1 Descrizione dei casi d'uso	7
<b>3. Analisi e progettazione</b>	<b>11</b>
3.1 Diagramma di robustezza	14
3.2 Diagramma delle classi	15
3.3 Modelli	16
3.3.1 Classe Pokemon	16
3.3.2 Classe Level1	16
3.3.3 Classe Level2	17
3.3.4 Classe Level3	17
3.3.5 Classe Skill	18
3.3.6 Classe Attack	18
3.3.7 Classe Defence	19
3.4 Controller	20
3.4.1 Interfaccia IController	20
3.4.2 Classe ControllerChoose	20
3.4.3 Classe ControllerGame	21
3.5 Viste	22
3.5.1 Classe FormChoose	22
3.5.2 Classe FormChange	23
3.5.3 Classe FormGame	24
3.6 Altre classi	25
3.6.1 Classe ChangeException	25
3.6.2 Classe SkillNotFoundException	25
3.6.3 Classe PokemonNotFoundException	26
3.6.4 Classe Program	26
<b>4. Implementazione</b>	<b>27</b>
4.1 Modelli	27
4.1.1 Classe Pokemon	27
4.1.2 Classe Level1	30
4.1.3 Classe Level2	31

4.1.4 Classe Level3	33
4.1.5 Classe Skill	34
4.1.6 Classe Attack	35
4.1.7 Classe Defence	36
4.2 Controller	37
4.2.1 Interfaccia IController	37
4.2.2 Classe ControllerChoose	37
4.2.3 Classe ControllerGame	43
4.3 Viste	50
4.3.1 Classe FormChoose	50
4.3.2 Classe FormChange	52
4.3.3 Classe FormGame	55
4.4 Altre classi	61
4.4.1 Classe ChangeException	61
4.4.2 Classe SkillNotFoundException	62
4.4.3 Classe PokemonNotFoundException	63
4.4.4 Classe Program	64
<b>5. Testing</b>	<b>65</b>
5.1 Test sull'efficacia del calcolo sul danno	65
5.2 Test sulla lettura dei pokémon da file csv	67
5.3 Test sulla scelta dei pokémon in numero diverso da tre	69
5.4 Test sulle evoluzioni	70
<b>6. Compilazione ed esecuzione</b>	<b>73</b>

## 1. Specifica del problema

Si richiede lo sviluppo di un videogioco per PC simile a Pokémon. Ci saranno due giocatori, che giocheranno a turno cercando di sconfiggersi a vicenda.

All'inizio di ogni partita un giocatore dovrà scegliere tre Pokémon fra quelli disponibili che possono avere attributi di erba, fuoco o acqua.

Un Pokémon può avere tre livelli:

- Nel primo livello avrà due mosse;
- Nel secondo livello avrà tre mosse;
- Nel terzo livello avrà quattro mosse.

Ogni mossa può essere di tipo attacco o difesa, mentre il livello aumenterà in base all'esperienza, che maturerà per ogni colpo a segno all'avversario.

Durante ogni turno, un giocatore potrà scegliere se cambiare pokémon o effettuare una mossa, considerando che a seguito del cambio pokémon il turno verrà terminato.

Quando un pokémon metterà a segno la mossa che gli permetterà di raggiungere il massimo valore di punti esperienza, esso, se possibile, evolverà automaticamente, recuperando tutti i punti salute persi, azzerando i punti esperienza e imparando una nuova mossa.

Il calcolo del danno è calcolato in base a quattro fattori:

- Bonus dato dagli attributi dei due pokémon;
- Danno della mossa;
- Attacco del pokémon attaccante;
- Difesa del pokémon attaccato;

ATTACCATO → ATTACCANTE ↴	Fuoco	Acqua	Erba
Fuoco	1	0,5	2
Acqua	2	1	0,5
Grass	0,5	2	1

$$[danno] = ([danno\ mossa] + [attacco\ attaccante]\% [danno\ mossa]) * [bonus]$$

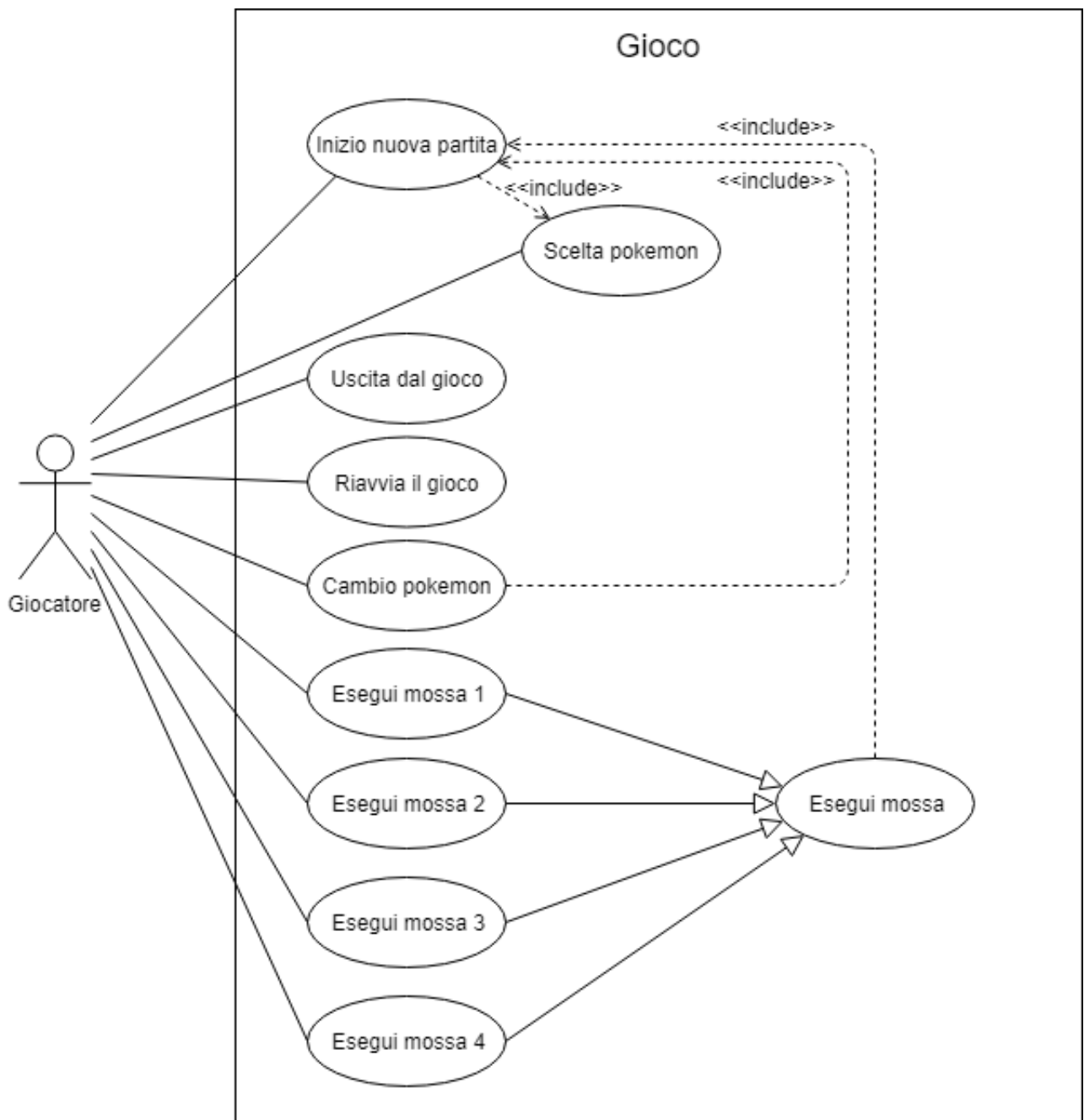
$$[danno\ percepito] = [danno] - [difesa\ attaccato]\% [danno]$$

## 2. Specifica dei requisiti

Per descrivere i requisiti funzionali del sistema si utilizzeranno i casi d'uso.

I casi d'uso funzionano descrivendo le interazioni tipiche tra gli utenti di un sistema e il sistema stesso, fornendo una descrizione di come viene utilizzato un sistema.

Per il progetto sviluppato, le principali interazioni sono le seguenti:



## 2.1 Descrizione dei casi d'uso

<b>Caso d'uso</b> : Inizio nuova partita.
<b>ID</b> : 1
<b>Attori</b> : Giocatore
<b>Pre-condizioni</b> : Entrambi i giocatori devono aver scelto 3 pokémon.
<b>Eventi base</b> : L'utente avvia la partita dal pulsante "next" in seguito alla scelta dei pokémon.
<b>Post-condizioni</b> : Viene avviata la battaglia tra i due giocatori.
<b>Percorsi alternativi</b> : N/A

<b>Caso d'uso</b> : Scelta del pokémon.
<b>ID</b> : 2
<b>Attori</b> : Giocatore
<b>Pre-condizioni</b> : Avviare l'applicazione.
<b>Eventi base</b> : L'utente avvia l'applicazione.
<b>Post-condizioni</b> : Inizio di una nuova partita.
<b>Percorsi alternativi</b> : N/A

<b>Caso d'uso</b> : Uscita dal gioco.
<b>ID</b> : 3
<b>Attori</b> : Giocatore
<b>Pre-condizioni</b> : Applicazione in esecuzione
<b>Eventi base</b> : L'utente chiude la finestra del gioco.
<b>Post-condizioni</b> : L'applicazione viene chiusa.
<b>Percorsi alternativi</b> : Pop-up a fine partita.

<b>Caso d'uso :</b> Riavvia il gioco.
<b>ID :</b> 4
<b>Attori :</b> Giocatore
<b>Pre-condizioni :</b> E' stata terminata una partita.
<b>Eventi base :</b> l'utente riavvia il gioco dal pop-up a fine partita.
<b>Post-condizioni :</b> L'applicazione viene riavviata.
<b>Percorsi alternativi :</b> N/A

<b>Caso d'uso :</b> Cambio pokémon
<b>ID :</b> 5
<b>Attori :</b> Giocatore
<b>Pre-condizioni :</b> <ul style="list-style-type: none"> <li>• E' il turno del giocatore che vuole effettuare il cambio pokémon.</li> <li>• Il giocatore ha più di un pokémon in vita.</li> </ul>
<b>Eventi base :</b> l'utente clicca sul pulsante "change pokémon" durante la partita.
<b>Post-condizioni :</b> <ul style="list-style-type: none"> <li>• Il pokémon del giocatore viene sostituito</li> <li>• Il giocatore termina il turno.</li> </ul>
<b>Percorsi alternativi :</b> Alla morte di un pokémon viene richiesto il cambio.

<b>Caso d'uso :</b> Esegui mossa
<b>ID :</b> 6
<b>Attori :</b> Giocatore
<b>Pre-condizioni :</b> E' il turno del giocatore che vuole effettuare la mossa.
<b>Eventi base :</b> l'utente clicca sul nome di una mossa.
<b>Post-condizioni :</b> <ul style="list-style-type: none"> <li>• Viene inflitto l'eventuale danno della mossa.</li> <li>• Il giocatore termina il turno.</li> </ul>
<b>Percorsi alternativi :</b> N/A



<b>Caso d'uso</b> : Esegui mossa 1
<b>ID</b> : 7
<b>Attori</b> : Giocatore
<b>Pre-condizioni</b> : E' il turno del giocatore che vuole effettuare la mossa.
<b>Eventi base</b> : l'utente clicca sul nome della prima mossa.
<b>Post-condizioni</b> : Viene eseguita la prima mossa.
<b>Percorsi alternativi</b> : N/A

<b>Caso d'uso</b> : Esegui mossa 2
<b>ID</b> : 8
<b>Attori</b> : Giocatore
<b>Pre-condizioni</b> : E' il turno del giocatore che vuole effettuare la mossa.
<b>Eventi base</b> : l'utente clicca sul nome della seconda mossa.
<b>Post-condizioni</b> : Viene eseguita la seconda mossa.
<b>Percorsi alternativi</b> : N/A

<b>Caso d'uso</b> : Esegui mossa 3
<b>ID</b> : 9
<b>Attori</b> : Giocatore
<b>Pre-condizioni</b> : <ul style="list-style-type: none"> <li>• E' il turno del giocatore che vuole effettuare la mossa.</li> <li>• Il pokémon attaccante è almeno di livello 2</li> </ul>
<b>Eventi base</b> : l'utente clicca sul nome della terza mossa.
<b>Post-condizioni</b> : Viene eseguita la terza mossa.
<b>Percorsi alternativi</b> : N/A

<b>Caso d'uso :</b> Esegui mossa 4
<b>ID :</b> 10
<b>Attori :</b> Giocatore
<b>Pre-condizioni :</b> <ul style="list-style-type: none"> <li>• E' il turno del giocatore che vuole effettuare la mossa.</li> <li>• Il pokémon attaccante è di livello 3</li> </ul>
<b>Eventi base :</b> l'utente clicca sul nome della quarta mossa.
<b>Post-condizioni :</b> Viene eseguita la quarta mossa.
<b>Percorsi alternativi :</b> N/A

### 3. Analisi e progettazione

Per la realizzazione del progetto è stato sfruttato il pattern di sviluppo MVC (Model-View-Controller). Il pattern consente di dividere i compiti tra i componenti software:

- **Model:** fornisce le classi con i dati utili al programma ed i metodi per accedervi.
- **View:** visualizza i dati dei vari *Model*, e si occupa dell'interazione tra l'utente e la struttura sottostante.
- **Controller:** riceve i comandi dall'utente tramite il *View*, esegue operazioni che possono interessare il *Model* e solitamente cambia lo stato del *View*.

Questo schema implica la separazione fra la logica applicativa, a carico del *Controller* e del *Model*, e l'interfaccia utente a carico del *View*.

Per l'interfaccia utente sono disponibili delle Form, messe a disposizione dal framework di Microsoft .NET.

#### Input:

Gli input della applicazione sono le varie azioni che il giocatore esegue nei controlli utente (bottoni, combo box, ecc ...).

#### Output:

L'output è formato dalle varie Form e dalle MessageBox mostrate a video dal programma.

#### Documentazione

Il progetto è stato ampiamente documentato grazie allo strumento *Doxygen*, generando una pagina web navigabile al seguente indirizzo:

<https://denilnicolosi.github.io/Pokemon-game/> .

In particolare, il codice sorgente è stato commentato con tag xml, utili sia per la generazione della documentazione, che per l'utilizzo di *IntelliSense* nell'IDE Visual Studio per visualizzare informazioni rapide sui tipi e membri.

#### Versioning

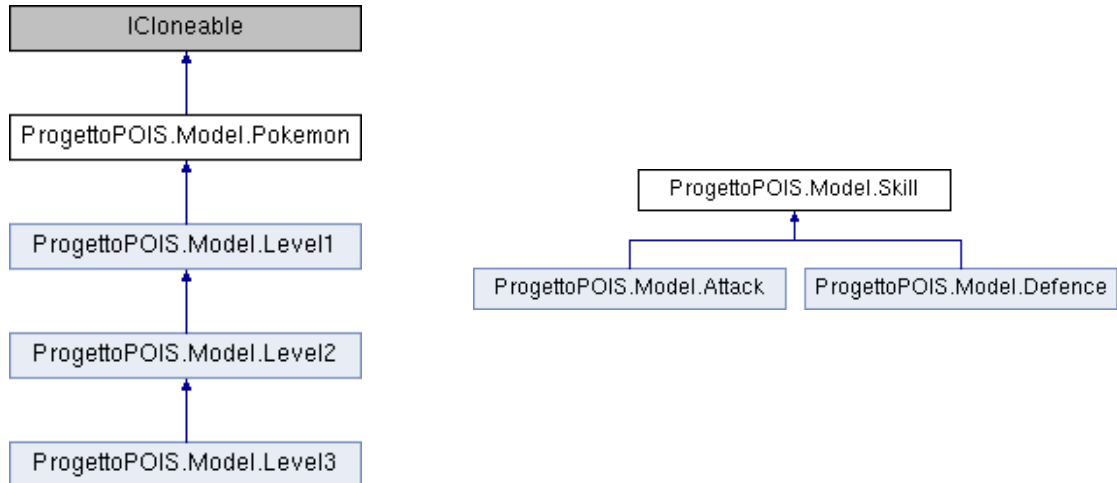
Come strumento di controllo di versione è stato utilizzato *GIT*. Inoltre il progetto è stato sviluppato sulla piattaforma GitHub.

Utilizzando come linguaggio di programmazione C# ed essendo orientato ad oggetti, le principali peculiarità utilizzate sono :

- **Incapsulamento**

- **Ereditarietà**

L'ereditarietà è stata sfruttata principalmente nei modelli.



Le immagini mostrano la struttura delle classe ereditate

- **Polimorfismo**

Viene utilizzato il polimorfismo per poter trattare tutte le classi derivate del *Pokemon* e della *Skill* come classe base, in modo da rendere più generale il codice indipendentemente dal livello.

In particolare è stato utilizzato l'**upcasting** per eseguire il cast delle classi derivate nella classe padre, mentre è stato utilizzato il **downcasting** per risalire alla classe derivata dalla classe padre.

E' stata utilizzata la tecnica di **override** per l'implementazione del metodo *Clone()* dei pokémon, mentre è stato utilizzato l'**overload** per implementare più costruttori delle eccezioni da utilizzare in base alle necessità.

Sono stati utilizzati inoltre alcuni costrutti del linguaggio, quali:

- **Gestione delle eccezioni** (utilizzate principalmente nella lettura dei pokémon e skill da file)
- **Gestione di strutture dati di alto livello offerte dalla libreria standard** (sono state utilizzate delle *List* in cui sono state memorizzati i pokémon)

Come scelta di progetto si è voluta implementare la lettura dei dati dei pokémon da dei file .csv.

I file sono divisi in *skill* e *pokemon*, così formattati:

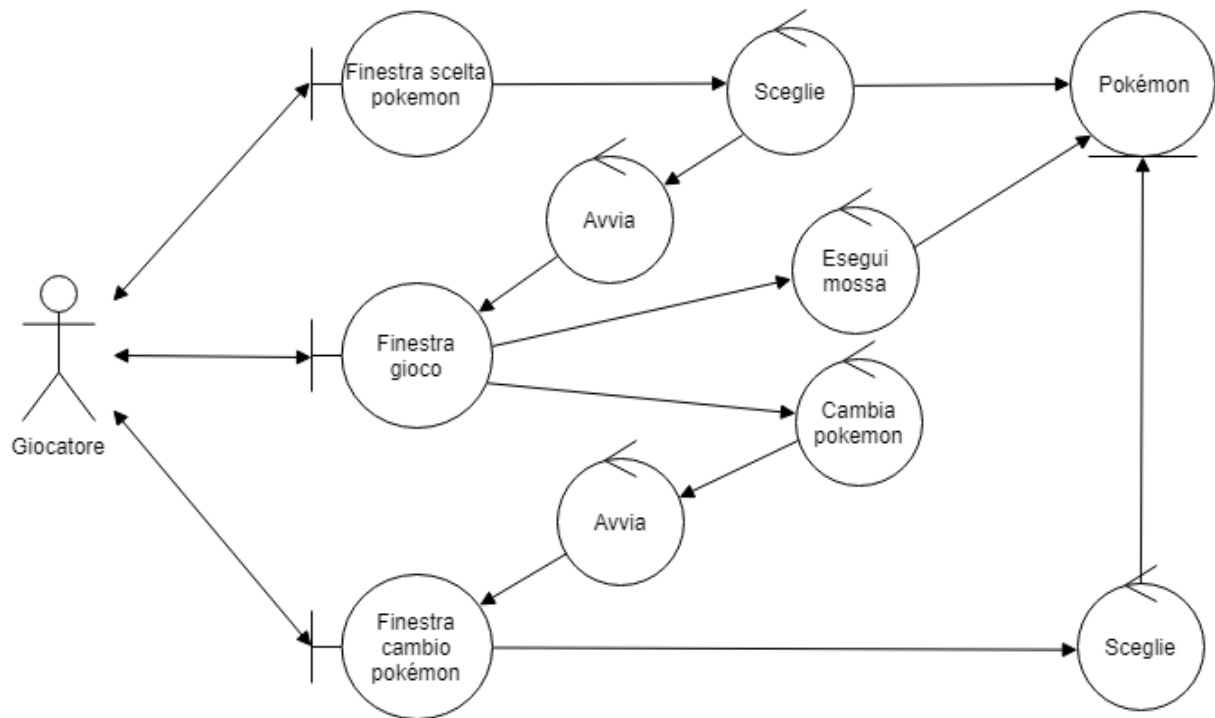
- ***skill.csv:***
  - **tipo attacco:** attack;[nome];[danno];[punti\_esperienza]
  - **tipo difesa:** defence;[nome];[salute];[punti\_esperienza]
- ***pokemon.csv:***
  - **Livello 1:** 1;[attributo];[nome];[punti\_attacco];[punti\_difesa];[nome\_mossa1];[nome\_mossa2]
  - **Livello 2:** 2;[nome\_livello\_precedente];[nome];[punti\_attacco]; [punti\_difesa];[nome\_mossa3]
  - **Livello 3:** 3;[nome\_livello\_precedente];[nome];[punti\_attacco];[punti\_difesa];[nome\_mossa4]

Altra scelta di progetto è stata quella di utilizzare eccezioni personalizzate, in particolare:

- **ChangeException :**  
L'eccezione verrà lanciata nel momento in cui un giocatore chiederà il cambio pokémon, ma effettivamente selezionerà quello già presente in campo.
- **PokemonNotFoundException:**  
L'eccezione verrà lanciata quando nel caricamento dei pokémon non verrà trovato il riferimento all'eventuale pokémon di livello precedente.
- **SkillNotFoundException**  
L'eccezione verrà lanciata quando nel caricamento dei pokémon non verrà trovato il riferimento alla skill del pokémon.

### 3.1 Diagramma di robustezza

Per illustrare il meccanismo di interazione tra i vari elementi del software, viene illustrato un diagramma di robustezza.

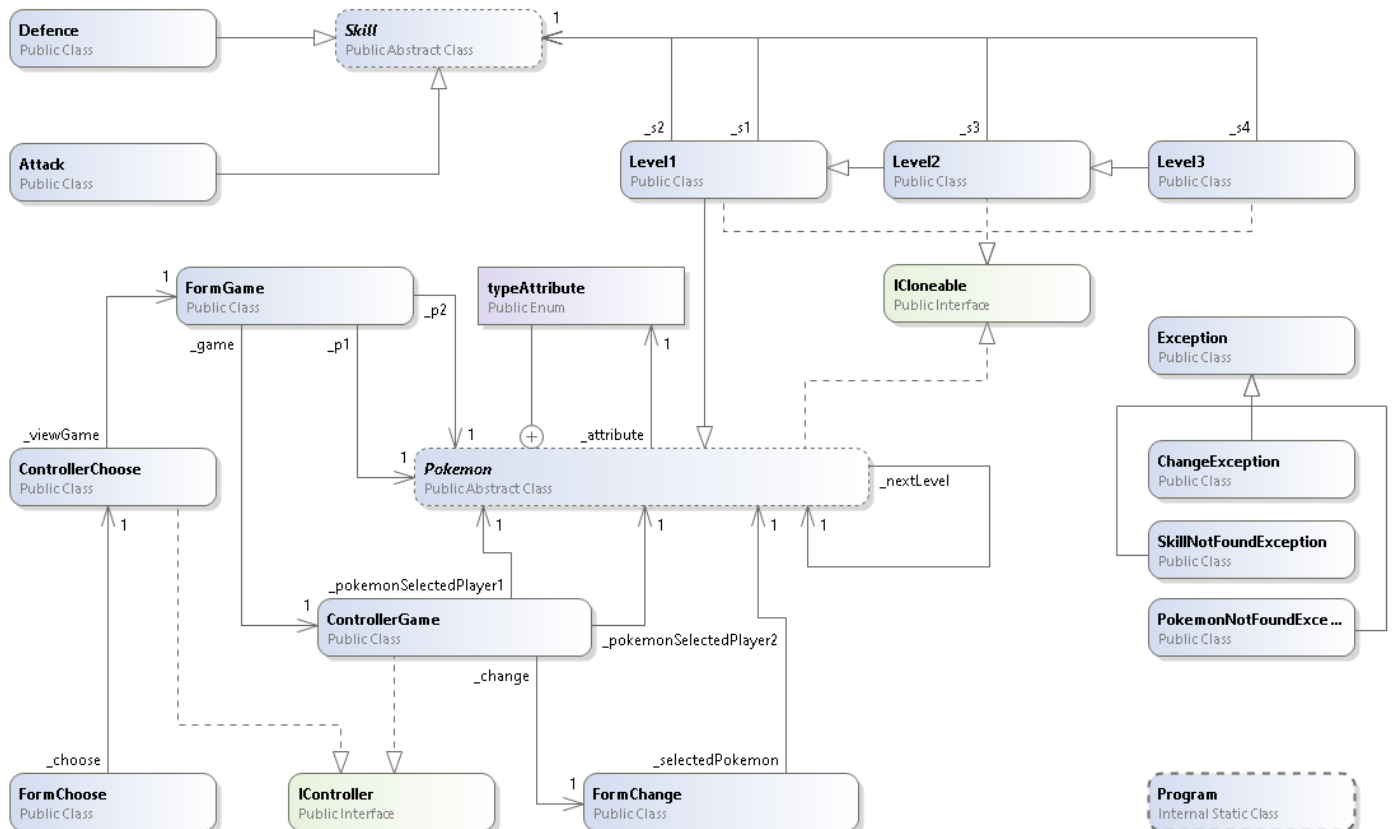


Il comportamento del giocatore può essere quindi descritto da questi passi:

1. Il giocatore sceglie i pokémon da utilizzare.
2. Il giocatore avvia la battaglia.
3. il giocatore può effettuare il suo turno, con la possibilità di :
  - Effettuare una mossa del pokémon.
  - Effettuare un cambio del pokémon fra quelli scelti nel primo passo.

Il punto 3 verrà ripetuto fino a quando uno dei due giocatori non avrà sconfitto tutti i pokémon dell'avversario.

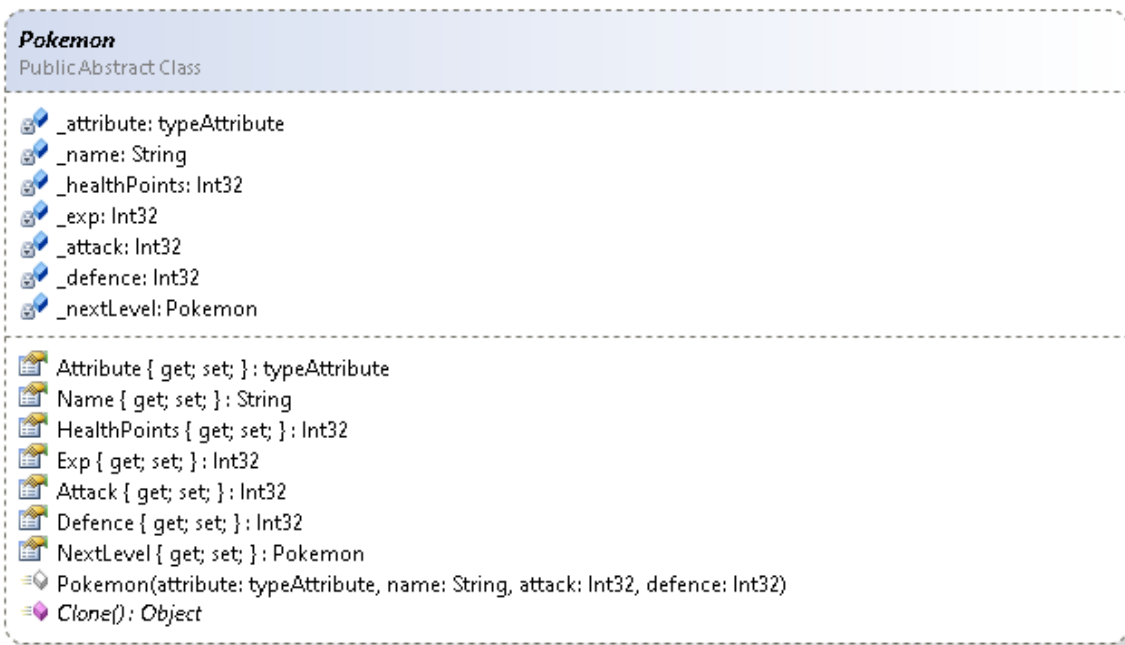
Il diagramma delle classi costituisce lo strumento principale per rappresentare graficamente le classi utilizzate, ed illustrare la struttura del programma. Questo diagramma mostra anche gli attributi ed i metodi (mostrati nei capitoli seguenti) con i vincoli fra gli oggetti.



## 3.3 Modelli

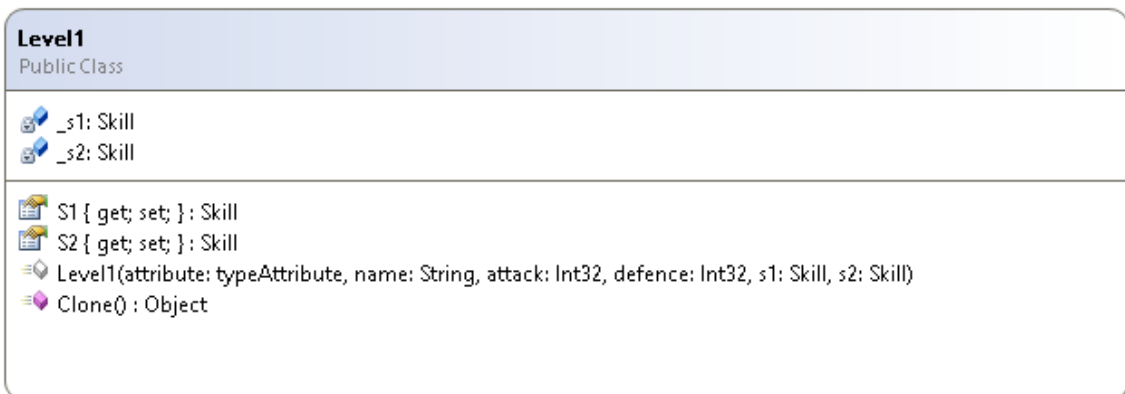
### 3.3.1 Classe Pokemon

Classe astratta, il padre di tutti i pokémon. Contiene tutti gli attributi e i metodi base.



### 3.3.2 Classe Level1





Classe che estende *Pokemon*, rappresenta un pokémon nella sua prima evoluzione. Questa classe aggiunge due *Skill* al pokémon.









### 3.3.3 Classe Level2

Classe che estende *Level1*, rappresenta un pokémon nella sua seconda evoluzione. Questa classe aggiunge una *Skill* al pokémon.

Level2
Public Class
 _s3: Skill
 S3 { get; set; } : Skill  Level2(attribute: typeAttribute, name: String, attack: Int32, defence: Int32, s1: Skill, s2: Skill, s3: Skill)  Clone() : Object

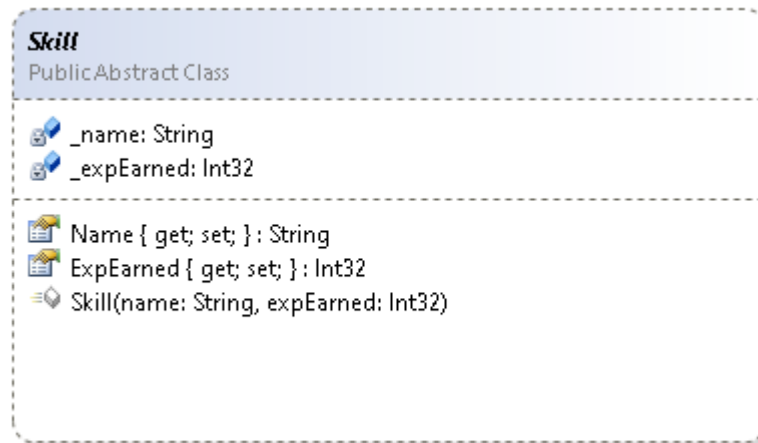
### 3.3.4 Classe Level3

Classe che estende *Level2*, rappresenta un pokémon nella sua terza evoluzione. Questa classe aggiunge una *Skill* al pokémon.

Level3
Public Class
 _s4: Skill
 S4 { get; set; } : Skill  Level3(attribute: typeAttribute, name: String, attack: Int32, defence: Int32, s1: Skill, s2: Skill, s3: Skill, s4: Skill)  Clone() : Object

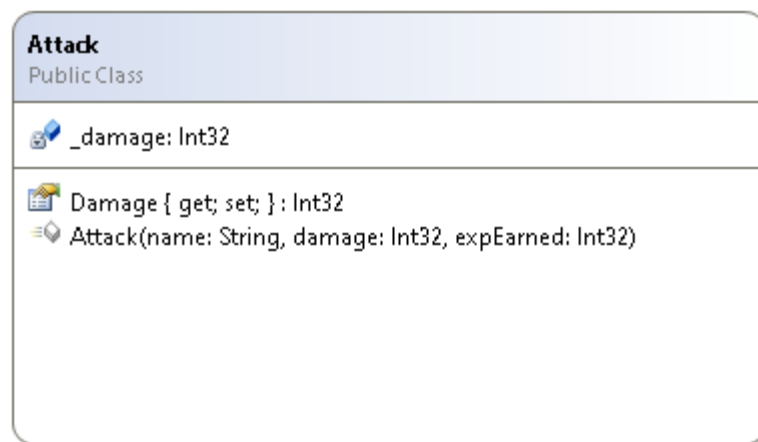
### 3.3.5 Classe Skill

Classe astratta, il padre di tutti le skill. Contiene tutti gli attributi e i metodi base.



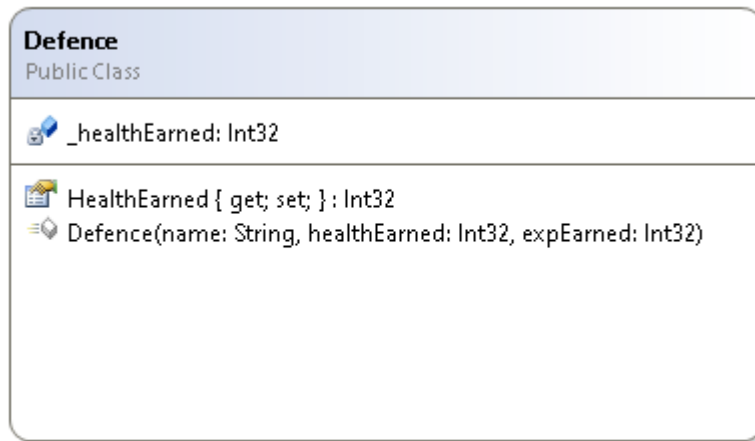
### 3.3.6 Classe Attack

Classe che estende *Skill*, rappresenta una skill di attacco.



### 3.3.7 Classe Defence

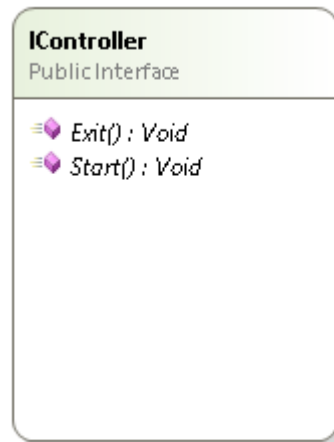
Classe che estende *Skill*, rappresenta una *skill di difesa*.



## 3.4 Controller

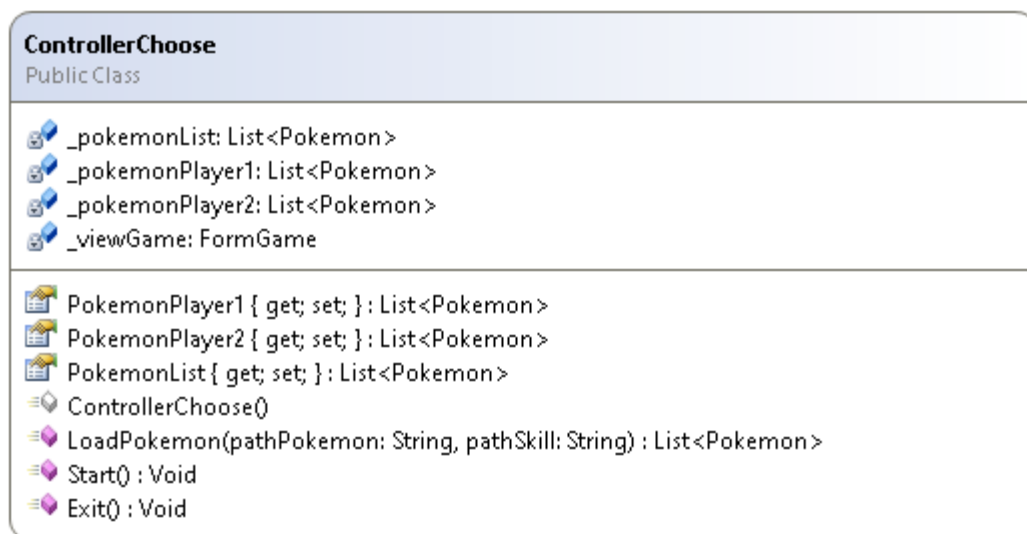
### 3.4.1 Interfaccia IController

Interfaccia per i controller. Contiene tutti i metodi che un controller deve implementare.



### 3.4.2 Classe ControllerChoose

Classe che implementa *IController*, è il controller per la selezione dei pokémon. Contiene attributi e metodi usati per interagire con i modelli.



### 3.4.3 Classe ControllerGame

Classe che implementa *IController*, è il controller per il gioco vero e proprio. Contiene attributi e metodi per gestire il gioco.

**ControllerGame**  
Public Class

\_numRound: Int32  
 \_isRoundPlayer1: Boolean  
 \_change: FormChange  
 \_pokemonSelectedPlayer1: Pokemon  
 \_pokemonSelectedPlayer2: Pokemon  
 \_pokemonPlayer1: List<Pokemon>  
 \_pokemonPlayer2: List<Pokemon>





















PokemonSelectedPlayer1 { get; set; } : Pokemon  
 PokemonSelectedPlayer2 { get; set; } : Pokemon  
 NumRound { get; set; } : Int32  
 IsRoundPlayer1 { get; set; } : Boolean  
 ControllerGame(pokemonPlayer1: List<Pokemon>, pokemonPlayer2: List<Pokemon>)  
 ChoosePokemon(p: Pokemon) : Void  
 ChangePokemon() : Boolean  
 CheckDefeat() : Boolean  
 DoSkill(s: Skill) : Boolean  
 Evolve() : Boolean  
 CalculatesPossibility(p: Pokemon) : Boolean  
 CalculatesDamage(p1: Pokemon, p2: Pokemon, s: Attack) : Int32  
 LevelOf(p: Pokemon) : Int32  
 Start() : Void  
 Exit() : Void  
 Restart() : Void







## 3.5 Viste

### 3.5.1 Classe FormChoose

Classe che estende *Form*, è la vista per la scelta dei pokémon personali.

**FormChoose**  
Public Class

 `_choose: ControllerChoose`  
 `components: IContainer`  
 `checkedListBox: CheckedListBox`  
 `buttonStart: Button`  
 `labelPlayer: Label`  
 `label3: Label`  
 `panel1: Panel`  
 `picture: PictureBox`  
 `labelAttack: Label`  
 `label6: Label`  
 `labelAttribute: Label`  
 `label4: Label`  
 `labelName: Label`  
 `label1: Label`  
 `labelSkill2: Label`  
 `label9: Label`  
 `labelSkill1: Label`  
 `label11: Label`  
 `labelDefence: Label`  
 `label13: Label`

 `FormChoose()`  
 `Button1_Click(sender: Object, e: EventArgs) : Void`  
 `CheckedListBox_SelectedIndexChanged(sender: Object, e: EventArgs) : Void`  
 `FormChoose_FormClosed(sender: Object, e: FormClosedEventArgs) : Void`  
 `Dispose(disposing: Boolean) : Void`  
 `InitializeComponent() : Void`

### 3.5.2 Classe FormChange

Classe che estende *Form*, è la vista per la scelta del pokémon principale.

**FormChange**  
Public Class

 `_pokemonList: List<Pokemon>`  
 `_selectedPokemon: Pokemon`  
 `components: IContainer`  
 `panel1: Panel`  
 `labelSkill2: Label`  
 `label9: Label`  
 `labelSkill1: Label`  
 `label11: Label`  
 `labelDefence: Label`  
 `label13: Label`  
 `picture: PictureBox`  
 `labelAttack: Label`  
 `label6: Label`  
 `labelAttribute: Label`  
 `label4: Label`  
 `labelName: Label`  
 `label1: Label`  
 `buttonChange: Button`  
 `labelHealtPoints: Label`  
 `label8: Label`  
 `labelExperience: Label`  
 `label7: Label`  
 `labelLevel: Label`  
 `label3: Label`  
 `labelSkill4: Label`  
 `labelTxtSkill4: Label`  
 `labelSkill3: Label`  
 `labelTxtSkill3: Label`  
 `comboBox: ComboBox`  
 `label10: Label`

 `SelectedPokemon { get; set; } : Pokemon`  
 `FormChange(pokemonList: List<Pokemon>)`  
 `ButtonChange_Click(sender: Object, e: EventArgs) : Void`  
 `ComboBox_SelectedIndexChanged(sender: Object, e: EventArgs) : Void`  
 `FormChange_FormClosing(sender: Object, e: FormClosingEventArgs) : Void`  
 `Dispose(disposing: Boolean) : Void`  
 `InitializeComponent() : Void`

### 3.5.3 Classe FormGame

Classe che estende *Form*, è la vista per la battaglia tra i pokémon dei due giocatori.

**FormGame**  
Public Class

- \_game: ControllerGame
- \_p1: Pokemon
- \_p2: Pokemon
- components: IContainer
- picture1: PictureBox
- panel1: Panel
- buttonSkill3: Button
- buttonSkill4: Button
- buttonSkill1: Button
- buttonSkill2: Button
- ButtonChangePokemon: Button
- panel3: Panel
- progressBar1: ProgressBar
- label1: Label
- labelLevel1: Label
- labelName1: Label
- panel2: Panel
- labelName2: Label
- labelLevel2: Label
- label4: Label
- progressBar2: ProgressBar
- picture2: PictureBox
- labelPlayer: Label
- label5: Label
- labelExp1: Label
- label3: Label
- label6: Label
- labelExp2: Label
- label8: Label
- progressBar3: ProgressBar
- progressBar4: ProgressBar
- hp2: Label
- hp1: Label
- label2: Label
- label7: Label
- labelMessage: Label

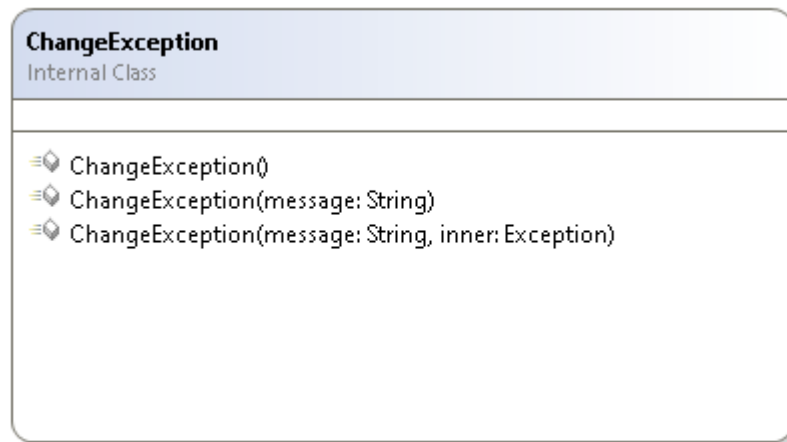
- FormGame(pokemonPlayer1: List<Pokemon>, pokemonPlayer2: List<Pokemon>)
- UpdateGraphics() : Void
- Change\_round() : Void
- WriteMessage(message: String) : Void
- ChangeDeadPokemon(p: Pokemon) : Void
- ButtonChangePokemon\_Click(sender: Object, e: EventArgs) : Void
- FormGame\_FormClosed(sender: Object, e: FormClosedEventArgs) : Void
- ButtonSkill1\_Click(sender: Object, e: EventArgs) : Void
- ButtonSkill2\_Click(sender: Object, e: EventArgs) : Void
- ButtonSkill3\_Click(sender: Object, e: EventArgs) : Void
- ButtonSkill4\_Click(sender: Object, e: EventArgs) : Void
- Dispose(disposing: Boolean) : Void
- InitializeComponent() : Void



## 3.6 Altre classi

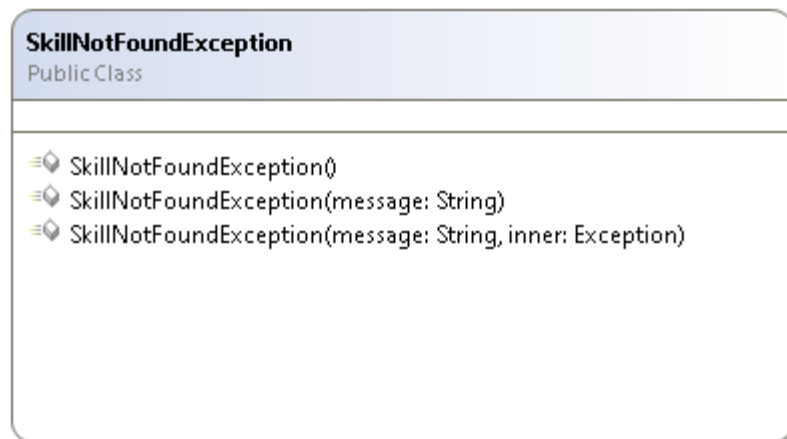
### 3.6.1 Classe ChangeException

Classe che estende *Exception*. Eccezione generata quando viene scelto un pokémon già presente nel campo di battaglia.



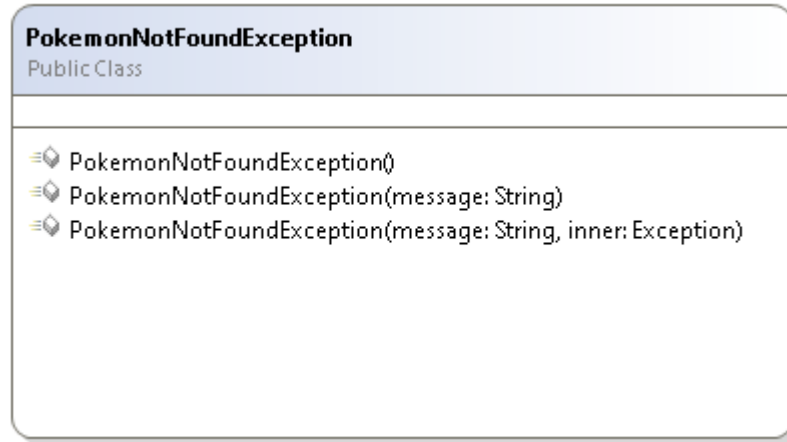
### 3.6.2 Classe SkillNotFoundException

Classe che estende *Exception*. Eccezione generata quando una skill non è stata trovata.



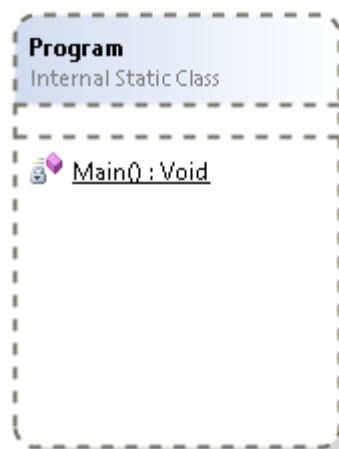
### 3.6.3 Classe PokemonNotFoundException

Classe che estende *Exception*. Eccezione generata quando un pokémon non è stato trovato.



### 3.6.4 Classe Program

Classe principale, contiene il metodo *main()*, è l'oggetto di avvio del programma.



## 4. Implementazione

### 4.1 Modelli

#### 4.1.1 Classe Pokemon

```
using System;

namespace PokemonGame.Model
{
    /// <summary>
    /// The father of all Pokemon.
    /// Contains all the basic attributes and methods of a pokemon.
    /// </summary>
    public abstract class Pokemon : ICloneable
    {
        // Definition of public enumerators.
        #region Public Enumerator
        /// <summary>
        /// Enumerator for the Pokemon attribute.
        /// </summary>
        public enum typeAttribute
        {
            /// <summary>Fire attribute.</summary>
            Fire,
            /// <summary>Water attribute.</summary>
            Water,
            /// <summary>Grass attribute.</summary>
            Grass
        }
        #endregion

        // Definition of private internal attributes.
        #region Private
        private typeAttribute _attribute;
        private string _name;
        private int _healthPoints;
        private int _exp;
        private int _attack;
        private int _defence;
        private Pokemon _nextLevel;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Pokemon type attribute.</summary>
        public typeAttribute Attribute { get => _attribute; set => _attribute = value; }
        /// <summary>Name of the pokemon.</summary>
        public string Name { get => _name; set => _name = value; }
        /// <summary>Health points of the pokemon.</summary>
        public int HealthPoints
        {
            get => _healthPoints;
        }
    }
}
```

```

        set
        {
            if (value > 0)
            {
                if (value > 100)
                {
                    _healthPoints = 100;
                }
                else
                {
                    _healthPoints = value;
                }
            }
            else
            {
                _healthPoints = 0;
            }
        }
    }

    /// <summary>Current experience points of the pokemon.</summary>
    public int Exp
    {
        get => _exp;
        set
        {
            if (value > 0)
            {
                if (value > 100)
                {
                    _exp = 100;
                }
                else
                {
                    _exp = value;
                }
            }
            else
            {
                _exp = 0;
            }
        }
    }
}

/// <summary>Points of defence of the pokemon.</summary>
public int Attack { get => _attack; set => _attack = value; }

/// <summary>Reference to the evolution of the pokemon.</summary>
public int Defence { get => _defence; set => _defence = value; }

/// <summary>Reference to the evolution of the pokemon.</summary>
public Pokemon NextLevel { get => _nextLevel; set => _nextLevel = value; }
#endregion

```

```

// Definition of class methods.
#region Methods

/// <summary>
/// Constructor method of the <c>Pokemon</c> class.
/// </summary>
/// <param name="attribute">Pokemon attribute.</param>
/// <param name="name">Pokemon name.</param>
/// <param name="attack">Value of the Pokemon attack.</param>
/// <param name="defence">Value of the Pokemon defence.</param>
/// <exception cref="System.ArgumentException">Negative attack or defense.</exception>
public Pokemon(typeAttribute attribute, string name, int attack, int defence)
{
    if (attack < 0)
    {
        throw new ArgumentException(name + ": attack must be positive.");
    }
    if (defence < 0)
    {
        throw new ArgumentException(name + ": defence must be positive.");
    }
    else
    {
        _attribute = attribute;
        _name = name;
        _healthPoints = 100;
        _exp = 0;
        _attack = attack;
        _defence = defence;
    }
}

/// <summary>
/// Create a new instance of the class with the same value as an existing instance.
/// </summary>
/// <returns>New object which is a copy of the current instance.</returns>
public abstract object Clone();

#endregion
}
}

```

## 4.1.2 Classe Level1

```
using System;
using PokemonGame.Exceptions;

namespace PokemonGame.Model
{
    /// <summary>
    /// Class that extends <c>Pokemon</c>.
    /// Represents a Pokemon on its first evolution.
    /// </summary>
    /// <remarks>
    /// This class adds two skills to the Pokemon.
    /// </remarks>
    public class Level1 : Pokemon, ICloneable
    {
        // Definition of private internal attributes.
        #region Private
        private Skill _s1;
        private Skill _s2;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Reference to skill number one.</summary>
        public Skill S1 { get => _s1; set => _s1 = value; }
        /// <summary>Reference to skill number two.</summary>
        public Skill S2 { get => _s2; set => _s2 = value; }
        #endregion

        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>Level</c> class.
        /// </summary>
        /// <param name="attribute">Pokemon attribute.</param>
        /// <param name="name">Pokemon name.</param>
        /// <param name="attack">Value of the Pokemon attack.</param>
        /// <param name="defence">Value of the Pokemon defence.</param>
        /// <param name="s1">Skill number one of the Pokemon.</param>
        /// <param name="s2">Skill number two of the Pokemon.</param>
        /// <exception cref="SkillNotFoundException">Reference to the skill not found.</exception>
        public Level1(typeAttribute attribute, string name, int attack,
            int defence, Skill s1, Skill s2)
            : base(attribute, name, attack, defence)
        {
            if (s1 != null && s2 != null)
            {
                _s1 = s1;
                _s2 = s2;
            }
            else
            {
                throw new SkillNotFoundException(name + ": skill not found.");
            }
        }
    }
}
```

```

    /// <summary>
    /// Create a new instance of the class with the same value as an existing instance.
    /// </summary>
    /// <returns>New object which is a copy of the current instance.</returns>
    public override object Clone()
    {
        Pokemon p = new Level1(Attribute, Name, Attack, Defence, _s1, _s2);
        p.NextLevel = NextLevel;
        return p;
    }

    #endregion
}

```

### 4.1.3 Classe Level2

```

using System;
using PokemonGame.Exceptions;

namespace PokemonGame.Model
{
    /// <summary>
    /// Class that extends <c>Level1</c>.
    /// Represents a Pokemon on its second evolution.
    /// </summary>
    /// <remarks>
    /// This class adds one skill to the level one pokemon.
    /// </remarks>
    public class Level2 : Level1, ICloneable
    {
        // Definition of private internal attributes.
        #region Protected
        private Skill _s3;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Reference to skill number three.</summary>
        public Skill S3 { get => _s3; set => _s3 = value; }
        #endregion
    }
}

```

```

// Definition of class methods.
#region Methods

/// <summary>
/// Constructor method of the <c>Level2</c> class.
/// </summary>
/// <param name="attribute">Pokemon attribute.</param>
/// <param name="name">Pokemon name.</param>
/// <param name="attack">Value of the Pokemon attack.</param>
/// <param name="defence">Value of the Pokemon defence.</param>
/// <param name="s1">Skill number one of the Pokemon.</param>
/// <param name="s2">Skill number two of the Pokemon.</param>
/// <param name="s3">Skill number three of the Pokèmon.</param>
/// <exception cref="SkillNotFoundException">Reference to the skill not found.</exception>
public Level2(typeAttribute attribute, string name, int attack,
              int defence, Skill s1, Skill s2, Skill s3)
    : base(attribute, name, attack, defence, s1, s2)
{
    if (s3 != null)
    {
        _s3 = s3;
    }
    else
    {
        throw new SkillNotFoundException(name + ": skill not found.");
    }
}

/// <summary>
/// Create a new instance of the class with the same value as an existing instance.
/// </summary>
/// <returns>New object which is a copy of the current instance.</returns>
public override object Clone()
{
    Pokemon p = new Level2(Attribute, Name, Attack, Defence, S1, S2, _s3);
    p.NextLevel = NextLevel;
    return p;
}

#endregion
}
}

```



## 4.1.4 Classe Level3

```
using System;
using PokemonGame.Exceptions;

namespace PokemonGame.Model
{
    /// <summary>
    /// Class that extends <c>Level2</c>.
    /// represents a Pokemon on its third evolution.
    /// </summary>
    /// <remarks>
    /// This class adds one skill to the level two pokemon.
    /// </remarks>
    public class Level3 : Level2, ICloneable
    {
        // Definition of private internal attributes.
        #region Private
        private Skill _s4;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Reference to skill number three.</summary>
        public Skill S4 { get => _s4; set => _s4 = value; }
        #endregion

        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>Level3</c> class.
        /// </summary>
        /// <param name="attribute">Pokemon attribute.</param>
        /// <param name="name">Pokemon name.</param>
        /// <param name="attack">Value of the Pokemon attack.</param>
        /// <param name="defence">Value of the Pokemon defence.</param>
        /// <param name="s1">Skill number one of the Pokemon.</param>
        /// <param name="s2">Skill number two of the Pokemon.</param>
        /// <param name="s3">Skill number three of the Pokemon.</param>
        /// <param name="s4">Skill number four of the Pokemon.</param>
        /// <exception cref="SkillNotFoundException">Reference to the skill not found.</exception>
        public Level3(typeAttribute attribute, string name, int attack,
            int defence, Skill s1, Skill s2, Skill s3, Skill s4)
            : base(attribute, name, attack, defence, s1, s2, s3)
        {
            if (s4 != null)
            {
                _s4 = s4;
            }
            else
            {
                throw new SkillNotFoundException(name + ": skill not found.");
            }
        }
    }
}
```

```

    /// <summary>
    /// Create a new instance of the class with the same value as an existing instance.
    /// </summary>
    /// <returns>New object which is a copy of the current instance.</returns>
    public override object Clone()
    {
        Pokemon p = new Level3(Attribute, Name, Attack, Defence, S1, S2, S3, _s4);
        p.NextLevel = NextLevel;
        return p;
    }
    #endregion
}
}

```

## 4.1.5 Classe Skill

```

using System;

namespace PokemonGame.Model
{
    /// <summary>
    /// The father of all Skills.
    /// contains all the basic attributes and methods of a skill.
    /// </summary>
    public abstract class Skill
    {
        // Definition of private internal attributes.
        #region Private
        private string _name;
        private int _expEarned;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Name of the skill.</summary>
        public string Name { get => _name; set => _name = value; }
        /// <summary>Experience points earned by the skill.</summary>
        public int ExpEarned { get => _expEarned; set => _expEarned = value; }
        #endregion

        // Definition of class methods.
        #region Methods
        /// <summary>
        /// Constructor method of the <c>Pokemon</c> class.
        /// </summary>
        /// <param name="name">Name of the skill.</param>
        /// <param name="expEarned">Experience earned.</param>
        public Skill(string name, int expEarned)
        {
            if (expEarned < 0)
            {
                throw new ArgumentException(name + ": exp. earned must be positive.");
            }
            _name = name;
            _expEarned = expEarned;
        }
        #endregion
    }
}

```

## 4.1.6 Classe Attack

```
using System;

namespace PokemonGame.Model
{
    /// <summary>
    /// Class that extends Skill.
    /// Represents an attack skill.
    /// </summary>
    public class Attack : Skill
    {
        // Definition of private internal attributes.
        #region Private
        private int _damage;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Damage inflicted by the skill.</summary>
        public int Damage { get => _damage; set => _damage = value; }
        #endregion

        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>Attack</c> class.
        /// </summary>
        /// <param name="name">Name of the attack skill.</param>
        /// <param name="damage">Damage that the skill performs.</param>
        /// <param name="expEarned">Experience earned.</param>
        public Attack(string name, int damage, int expEarned) : base(name, expEarned)
        {
            if (damage < 0)
            {
                throw new ArgumentException(name + ": damage must be positive.");
            }
            _damage = damage;
        }

        #endregion
    }
}
```

## 4.1.7 Classe Defence

```
using System;

namespace PokemonGame.Model
{
    /// <summary>
    /// Class that extends Skill.
    /// Represents a defense skill.
    /// </summary>
    public class Defence : Skill
    {
        // Definition of private internal attributes.
        #region Private
        private int _healthEarned;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Health points earned.</summary>
        public int HealthEarned { get => _healthEarned; set => _healthEarned = value; }
        #endregion

        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>Defence</c> class.
        /// </summary>
        /// <param name="name">Name of the attack skill.</param>
        /// <param name="healthEarned">Health points that the skill increases.</param>
        /// <param name="expEarned">Experience earned.</param>
        public Defence(string name, int healthEarned, int expEarned) : base(name, expEarned)
        {
            if (healthEarned < 0)
            {
                throw new ArgumentException(name + ": health earned must be positive.");
            }
            _healthEarned = healthEarned;
        }

        #endregion
    }
}
```

## 4.2 Controller

### 4.2.1 Interfaccia IController

```
namespace PokemonGame.Controller
{
    /// <summary>
    /// Interface for controllers.
    /// </summary>
    public interface IController
    {
        // Definition of class methods.
        #region Methods

        /// <summary>
        /// End program execution.
        /// </summary>
        void Exit();

        /// <summary>
        /// Method that starts the form.
        /// </summary>
        void Start();

        #endregion
    }
}
```

### 4.2.2 Classe ControllerChoose

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Windows.Forms;
using PokemonGame.Exceptions;
using PokemonGame.Model;
using PokemonGame.View;

namespace PokemonGame.Controller
{
    /// <summary>
    /// Controller class for the pokemon selection.
    /// Contains methods and attributes used to interact with models.
    /// </summary>
    /// <remarks>
    /// It implements the "IController" interface.
    /// See <see cref="IController"/> For more information.
    /// </remarks>
    public class ControllerChoose : IController
    {
        // Definition of private internal attributes.
        #region Private
        private List<Pokemon> _pokemonList;
        private List<Pokemon> _pokemonPlayer1;

```

```

private List<Pokemon> _pokemonPlayer2;
private FormGame _viewGame;
#endregion

// Definition of public attributes, for the "get/set" methods.
#region Public
/// <summary>List of pokemon chosen by player one.</summary>
public List<Pokemon> PokemonPlayer1
{
    get => _pokemonPlayer1;
    set => _pokemonPlayer1 = value;
}

/// <summary>List of pokemon chosen by player one.</summary>
public List<Pokemon> PokemonPlayer2
{
    get => _pokemonPlayer2;
    set => _pokemonPlayer2 = value;
}

/// <summary>List of pokemon to choose.</summary>
public List<Pokemon> PokemonList
{
    get => _pokemonList;
    set => _pokemonList = value;
}
#endregion

// Definition of class methods.
#region Methods

/// <summary>
/// Constructor method of the <c>ControllerChoose</c> class.
/// </summary>
public ControllerChoose()
{
    _pokemonPlayer1 = new List<Pokemon>();
    _pokemonPlayer2 = new List<Pokemon>();

    _pokemonList = LoadPokemon(Properties.Settings.Default.pathPokemon,
                                Properties.Settings.Default.pathSkill);
}

/// <summary>
/// Method that reads data from files to load pokemon in the list.
/// </summary>
/// <remarks>
/// Reading takes place on two ".csv" files, one for pokemon and one for skills.
/// </remarks>
/// <param name="pathPokemon">File path with pokemon data.</param>
/// <param name="pathSkill">file path with skills data.</param>
/// <returns>List of pokemon with the appropriate skills.</returns>
public List<Pokemon> LoadPokemon(string pathPokemon, string pathSkill)
{
    Pokemon tmpPokemon, prevPokemon;
    Skill tmpSkill;

    List<Pokemon> listPokemon = new List<Pokemon>();
    List<Skill> listSkill = new List<Skill>();

```

```

StreamReader readerPokemon, readerSkill;
Pokemon.typeAttribute attribute;

#region Reading/loading skill
try
{
    // Reading skills from file.
    readerSkill = new StreamReader(pathSkill);
    while (!readerSkill.EndOfStream)
    {
        string line = readerSkill.ReadLine();
        string[] values = line.Split(';');
        if (!string.IsNullOrEmpty(line))
        {
            // Switch on skill type.
            switch (values[0].ToLower().Trim())
            {
                case "attack":
                    tmpSkill = new Attack(values[1],
                                           Int32.Parse(values[2]),
                                           Int32.Parse(values[3]));
                    break;
                case "defence":
                    tmpSkill = new Defence(values[1],
                                           Int32.Parse(values[2]),
                                           Int32.Parse(values[3]));
                    break;
                default:
                    throw new FormatException("Attribute: " + values[0] +
                                             " is invalid.");
            }
            listSkill.Add(tmpSkill);
        }
    }
}
catch (FormatException formatEx) // The type of skill is invalid.
{
    Console.WriteLine(formatEx);
    MessageBox.Show(formatEx.Message, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    Exit();
}
catch (ArgumentNullException argNullEx) // A value is missing.
{
    Console.WriteLine(argNullEx);
    MessageBox.Show("A value of a skill is missing.", "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    Exit();
}
catch (ArgumentException argEx) // A value is incorrect.
{
    Console.WriteLine(argEx);
    MessageBox.Show(argEx.Message, "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
    Exit();
}
catch (OverflowException overfEx) // A value is overflowed.
{
    Console.WriteLine(overfEx);
    MessageBox.Show("A value of a skill is overflowed.", "Error",
                    MessageBoxButtons.OK, MessageBoxIcon.Error);
}

```

```

        Exit();
    }
    catch (SystemException sysEx)           // Capture the StreamReader exceptions.
    {
        Console.WriteLine(sysEx);
        MessageBox.Show("Error reading skill.", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);

        Exit();
    }
    #endregion

    #region Reading/loading pokemon
    try
    {
        // Reading pokemon from file
        readerPokemon = new StreamReader(pathPokemon);
        while (!readerPokemon.EndOfStream)
        {
            string line = readerPokemon.ReadLine();
            string[] values = line.Split(';');

            if (!string.IsNullOrEmpty(line))
            {
                // Switch on level.
                switch (Int32.Parse(values[0]))
                {
                    case 1:
                        // Definition of pokemon attribute.
                        switch (values[1].ToLower().Trim())
                        {
                            case "grass":
                                attribute = Pokemon.typeAttribute.Grass;
                                break;
                            case "fire":
                                attribute = Pokemon.typeAttribute.Fire;
                                break;
                            case "water":
                                attribute = Pokemon.typeAttribute.Water;
                                break;
                            default:
                                throw new ArgumentException(values[2] +
                                    ": attribute not found.");
                        }

                        // Pokemon instance creation.
                        tmpPokemon = new Level1(attribute, values[2],
                            Int32.Parse(values[3]),
                            Int32.Parse(values[4]),
                            listSkill.Where(s =>
                                s.Name == values[5]).FirstOrDefault(),
                            listSkill.Where(s =>
                                s.Name == values[6]).FirstOrDefault());

                        break;

                    case 2:
                        // Creation of links between levels 1 and 2.
                        prevPokemon = (Level1)listPokemon.Where(p =>
                            p.Name == values[1]).FirstOrDefault();

```



```

        if (prevPokemon == null)
        {
            throw new PokemonNotFoundException(values[2] +
                                                ": prev pokemon not found.");
        }

        // Pokemon instance creation.
        tmpPokemon = new Level2(prevPokemon.Attribute, values[2],
                                Int32.Parse(values[3]),
                                Int32.Parse(values[4]),
                                ((Level1)prevPokemon).S1,
                                ((Level1)prevPokemon).S2,
                                listSkill.Where(s =>
                                                s.Name == values[5]).FirstOrDefault());

        prevPokemon.NextLevel = tmpPokemon;
        break;

    case 3:
        // creation of links between levels 2 and 3.
        prevPokemon = (Level2)listPokemon.Where(p =>
                                                p.Name == values[1]).FirstOrDefault();

        if (prevPokemon == null)
        {
            throw new PokemonNotFoundException(values[2] +
                                                ": prev pokemon not found.");
        }

        // Pokemon instance creation.
        tmpPokemon = new Level3(prevPokemon.Attribute, values[2],
                                Int32.Parse(values[3]),
                                Int32.Parse(values[4]),
                                ((Level2)prevPokemon).S1,
                                ((Level2)prevPokemon).S2,
                                ((Level2)prevPokemon).S3,
                                listSkill.Where(s =>
                                                s.Name == values[5]).FirstOrDefault());

        prevPokemon.NextLevel = tmpPokemon;
        break;

    default:
        throw new FormatException(values[2] +
                                    ": level must be between 1 and 3.");
    }
    if (tmpPokemon != null)
    {
        listPokemon.Add(tmpPokemon);
    }
}
}
catch (FormatException formatEx)
{
    Console.WriteLine("Error reading from pokemon file.");
    MessageBox.Show(formatEx.Message, "Error", MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
    Exit();
}
catch (PokemonNotFoundException pnfEx) // Pokemon not found.

```

```

    {
        Console.WriteLine(pnfEx);
        MessageBox.Show(pnfEx.Message, "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        Exit();
    }
    catch (SkillNotFoundException snfEx)        // Skill not found.
    {
        Console.WriteLine(snfEx);
        MessageBox.Show(snfEx.Message, "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        Exit();
    }
    catch (ArgumentNullException argNullEx)    // Missing argument.
    {
        Console.WriteLine(argNullEx);
        MessageBox.Show("A value of a pokemon is missing.", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Error);
        Exit();
    }
    catch (ArgumentException argEx)           // Errors during instance creation.
    {
        Console.WriteLine(argEx);
        MessageBox.Show(argEx.Message, "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        Exit();
    }
    catch (SystemException sysEx)            // Capture the StreamReader exceptions.
    {
        Console.WriteLine(sysEx);
        MessageBox.Show("Error reading pokemon.", "Error", MessageBoxButtons.OK,
            MessageBoxIcon.Error);
        Exit();
    }
    #endregion

    return listPokemon;
}

/// <summary>
/// Method that starts the choose form.
/// </summary>
/// <remarks>
/// It instantiates a <c>FormGame</c> object and shows it.
/// </remarks>
public void Start()
{
    _viewGame = new FormGame(_pokemonPlayer1, _pokemonPlayer2);
    _viewGame.Show();
}

/// <summary>
/// End program execution.
/// </summary>
public void Exit()
{
    Environment.Exit(0);
}
#endregion
}
}

```

## 4.2.3 Classe ControllerGame

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using PokemonGame.Exceptions;
using PokemonGame.Model;
using PokemonGame.View;

namespace PokemonGame.Controller
{
    /// <summary>
    /// Controller for the game.
    /// Contains all the basic attributes and methods to control the game.
    /// </summary>
    /// <remarks>
    /// It implements the "IController" interface.
    /// See <see cref="IController"/> For more information.
    /// </remarks>
    public class ControllerGame : IController
    {
        // Definition of private internal attributes.
        #region Private
        private int _numRound;
        private bool _isRoundPlayer1;
        private FormChange _change;
        private Pokemon _pokemonSelectedPlayer1;
        private Pokemon _pokemonSelectedPlayer2;
        private List<Pokemon> _pokemonPlayer1;
        private List<Pokemon> _pokemonPlayer2;
        #endregion

        // Definition of public attributes, for the "get/set" methods.
        #region Public
        /// <summary>Pokemon selected by player one.</summary>
        public Pokemon PokemonSelectedPlayer1
        {
            get => _pokemonSelectedPlayer1;
            set => _pokemonSelectedPlayer1 = value;
        }

        /// <summary>Pokemon selected by player two.</summary>
        public Pokemon PokemonSelectedPlayer2
        {
            get => _pokemonSelectedPlayer2;
            set => _pokemonSelectedPlayer2 = value;
        }

        /// <summary>Number of the current round.</summary>
        public int NumRound
        {
            get => _numRound;
            set
            {
                _numRound = value;
                _isRoundPlayer1 = (_numRound % 2 == 0) ? true : false;
            }
        }
    }
}
```

```

/// <summary>Boolean that identifies if it is the round of the player one.</summary>
public bool IsRoundPlayer1
{
    get => _isRoundPlayer1;
    set => _isRoundPlayer1 = value;
}
#endregion

// Definition of class methods.
#region Methods

/// <summary>
/// Constructor method of the <c>ControllerGame</c> class.
/// </summary>
/// <param name="pokemonPlayer1">List of <c>Pokemon</c> chosen by player 1.</param>
/// <param name="pokemonPlayer2">List of <c>Pokemon</c> chosen by player 2.</param>
public ControllerGame(List<Pokemon> pokemonPlayer1, List<Pokemon> pokemonPlayer2)
{
    _pokemonPlayer1 = pokemonPlayer1;
    _pokemonPlayer2 = pokemonPlayer2;
    NumRound = -1;
}

/// <summary>
/// Select the main Pokemon of current player.
/// </summary>
/// <param name="p">Pokemon that will become the main one.</param>
public void ChoosePokèmon(Pokemon p)
{
    if (_isRoundPlayer1)
    {
        _pokemonSelectedPlayer1 = p;
    }
    else
    {
        _pokemonSelectedPlayer2 = p;
    }
}

/// <summary>
/// Show the form for choosing the pokemon to be the main one,
/// and make it effective.
/// </summary>
/// <returns>Boolean for the success of the exchange.</returns>
/// <exception cref="PokemonGame.Exceptions.ChangeException">
/// Pokemon already in the battlefield.
/// </exception>
public bool ChangePokemon()
{
    bool success = false;
    Pokemon selectePokemon;

    if (!CheckDefeat())
    {
        if (_isRoundPlayer1)
        {
            _change = new FormChange(_pokemonPlayer1, _isRoundPlayer1);
        }
        else
    }

```

```

    {
        _change = new FormChange(_pokemonPlayer2, _isRoundPlayer1);
    }

    _change.ShowDialog();
    selectePokemon = _change.SelectedPokemon;

    if (selectePokemon == ((_isRoundPlayer1) ? _pokemonSelectedPlayer1
                                                : _pokemonSelectedPlayer2))
    {
        throw new ChangeException();
    }

    if (selectePokemon != null)
    {
        ChoosePokèmon(selectePokemon);
    }

    success = true;
}
return success;
}

/// <summary>
/// Check if the current player has been defeated.
/// </summary>
/// <remarks>
/// Check the health points of all pokemon of the current player,
/// to verify the defeat.
/// </remarks>
/// <returns>Boolean for the outcome of the defeat check.</returns>
public bool CheckDefeat()
{
    bool success = true;

    if (_isRoundPlayer1)
    {
        foreach (Pokemon p in _pokemonPlayer1)
        {
            if (p.HealthPoints > 0)
            {
                success = false;
            }
        }
    }
    else
    {
        foreach (Pokemon p in _pokemonPlayer2)
        {
            if (p.HealthPoints > 0)
            {
                success = false;
            }
        }
    }

    return success;
}
}

```

```

/// <summary>
/// It try to execute the skill.
/// </summary>
/// <remarks>
/// Calculate and make changes to pokemon values.
/// </remarks>
/// <param name="s">Skill to try to perform.</param>
/// <returns>Boolean for the success of the skill.</returns>
public bool DoSkill(Skill s)
{
    bool success = CalculatesPossibility((_isRoundPlayer1) ? _pokemonSelectedPlayer1 :
                                                                    _pokemonSelectedPlayer2);

    int damage;

    if (success)
    {
        if (s.GetType() == typeof(Attack))
        {
            if (_isRoundPlayer1)
            {
                damage = CalculatesDamage(_pokemonSelectedPlayer1,
                                          _pokemonSelectedPlayer2,
                                          (Attack)s);
                _pokemonSelectedPlayer2.HealthPoints -= damage;
                _pokemonSelectedPlayer1.Exp += s.ExpEarned;
            }
            else
            {
                damage = CalculatesDamage(_pokemonSelectedPlayer2,
                                          _pokemonSelectedPlayer1,
                                          (Attack)s);
                _pokemonSelectedPlayer1.HealthPoints -= damage;
                _pokemonSelectedPlayer2.Exp += s.ExpEarned;
            }
        }
        else if (_isRoundPlayer1)
        {
            _pokemonSelectedPlayer1.HealthPoints += ((Defence)s).HealthEarned;
        }
        else
        {
            _pokemonSelectedPlayer2.HealthPoints += ((Defence)s).HealthEarned;
        }
    }

    return success;
}

/// <summary>
/// Try to evolve the current pokemon.
/// </summary>
/// <returns>Boolean for the success of evolution.</returns>
public bool Evolve()
{
    bool success = false;
    Pokemon next;

    // A pokemon evolves after using a skill, so it is checked
    // if it has the max. experience points on its turn.
    if (_isRoundPlayer1)
    {

```

```

        if (_pokemonSelectedPlayer1.Exp == 100 && _pokemonSelectedPlayer1.NextLevel != null)
        {
            next = (Pokemon)_pokemonSelectedPlayer1.NextLevel.Clone();
            _pokemonPlayer1.Remove(_pokemonSelectedPlayer1);
            _pokemonPlayer1.Add(next);
            _pokemonSelectedPlayer1 = next;
            success = true;
        }
    }
    else
    {
        if (_pokemonSelectedPlayer2.Exp == 100 && _pokemonSelectedPlayer2.NextLevel != null)
        {
            next = (Pokemon)_pokemonSelectedPlayer2.NextLevel.Clone();
            _pokemonPlayer2.Remove(_pokemonSelectedPlayer2);
            _pokemonPlayer2.Add(next);
            _pokemonSelectedPlayer2 = next;
            success = true;
        }
    }
    return success;
}

/// <summary>
/// Calculate the possibility that a pokemon fails the skill.
/// </summary>
/// <param name="p">Pokemon for which you want to calculate the chance of success.</param>
/// <returns>Boolean for calculation success.</returns>
private bool CalculatesPossibility(Pokemon p)
{
    bool success = false;
    Random rnd = new Random();

    if (((LevelOf(p) == 1) && (rnd.Next(1, 4) < 3)) ||
        ((LevelOf(p) == 2) && (rnd.Next(1, 6) < 5)) ||
        ((LevelOf(p) == 3) && (rnd.Next(1, 11) < 10)))
    {
        success = true;
    }

    return success;
}

/// <summary>
/// Calculate the damage of an attack.
/// </summary>
/// <remarks>
/// It is calculated based on the attacking pokemon, attacked pokemon and the skill.
/// </remarks>
/// <param name="p1">Attacking pokemon.</param>
/// <param name="p2">Attacked pokemon.</param>
/// <param name="s">Attack skill used.</param>
private int CalculatesDamage(Pokemon p1, Pokemon p2, Attack s)
{
    double bonusAttribute = 1;
    double totalDmg = 0;

    // Check for attribute bonus damage.
    // ...-> Fire -> Grass -> Water -> Fire ->...
    switch (p1.Attribute)
    {

```

```

        case Pokemon.typeAttribute.Fire:
            if (p2.Attribute == Pokemon.typeAttribute.Grass)
            {
                bonusAttribute = 2;
            }
            else if (p2.Attribute == Pokemon.typeAttribute.Water)
            {
                bonusAttribute = 0.5;
            }
            break;

        case Pokemon.typeAttribute.Water:
            if (p2.Attribute == Pokemon.typeAttribute.Fire)
            {
                bonusAttribute = 2;
            }
            else if (p2.Attribute == Pokemon.typeAttribute.Grass)
            {
                bonusAttribute = 0.5;
            }
            break;

        case Pokemon.typeAttribute.Grass:
            if (p2.Attribute == Pokemon.typeAttribute.Water)
            {
                bonusAttribute = 2;
            }
            else if (p2.Attribute == Pokemon.typeAttribute.Fire)
            {
                bonusAttribute = 0.5;
            }
            break;
    }

    // Calculation of actual damage.
    totalDmg = (s.Damage + (s.Damage * p1.Attack / 100)) * bonusAttribute;
    totalDmg -= totalDmg * p2.Defence / 100;

    return (int)totalDmg;
}

/// <summary>
/// Determines the level of a pokemon.
/// </summary>
/// <param name="p">Pokemon that wants to determine the level.</param>
/// <returns>Level in integer format.</returns>
public static int LevelOf(Pokemon p)
{
    int level;

    if (p.GetType() == typeof(Level1))
    {
        level = 1;
    }
    else if (p.GetType() == typeof(Level2))
    {
        level = 2;
    }
    else if (p.GetType() == typeof(Level3))
    {
        level = 3;
    }
}

```



```

    }
    else
    {
        level = -1;    // Error.
    }

    return level;
}

/// <summary>
/// Method that starts the form.
/// </summary>
public void Start()
{
    NumRound++;
    ChangePokemon();
    NumRound++;
    ChangePokemon();
}

/// <summary>
/// End program execution.
/// </summary>
public void Exit()
{
    Environment.Exit(0);
}

/// <summary>
/// Restart program execution.
/// </summary>
public void Restart()
{
    System.Diagnostics.Process.Start(Application.ExecutablePath);
    Exit();
}

#endregion
}
}

```

## 4.3 Viste

### 4.3.1 Classe FormChoose

```
using System;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using PokemonGame.Controller;
using PokemonGame.Model;
using static System.Windows.Forms.CheckedListBox;

namespace PokemonGame.View
{
    /// <summary>
    /// Form for choosing your own pokemon.
    /// </summary>
    /// <remarks>
    /// Extends the <c>Form</c> class.
    /// <see cref="Form"/>
    /// </remarks>
    public partial class FormChoose : Form
    {
        // Definition of private internal attributes.
        #region Private
        private ControllerChoose _choose;
        #endregion

        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method pf the <c>FormChange</c> class.
        /// </summary>
        public FormChoose()
        {
            InitializeComponent();

            this.FormBorderStyle = FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;

            _choose = new ControllerChoose();

            foreach (Pokemon p in _choose.PokemonList)
            {
                if (p.GetType() == typeof(Level1))
                {
                    checkedListBox.Items.Add(p.Name, false);
                }
            }

            checkedListBox.SelectedIndex = 0;
        }
    }
}
```

```

/// <summary>
/// Action to take when "Button1" is clicked.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Button1_Click(object sender, EventArgs e)
{
    CheckItemCollection checkedPlayer = checkedListBox.CheckedItems;

    if (checkedPlayer.Count != 3)
    {
        MessageBox.Show("Select 3 pokemon!", "Error",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    else if (_choose.PokemonPlayer1.Count == 0)
    {
        // Select pokemon player 1.
        foreach (string pName in checkedPlayer)
        {
            _choose.PokemonPlayer1.Add((Pokemon)_choose.PokemonList.Where(p =>
                p.Name == pName).FirstOrDefault().Clone());
        }

        labelPlayer.Text = "Player2";
        buttonStart.Text = "Start game";

        // Clear selected.
        for (int i = 0; i < checkedListBox.Items.Count; i++)
        {
            checkedListBox.SetItemCheckState(i, CheckState.Unchecked);
        }
    }
    else
    {
        // Select pokemon player 2.
        foreach (string pName in checkedPlayer)
        {
            _choose.PokemonPlayer2.Add((Pokemon)_choose.PokemonList.Where(p =>
                p.Name == pName).FirstOrDefault().Clone());
        }

        _choose.Start();
        this.Hide();
    }
}

/// <summary>
/// Action to take when changing a selection in the "CheckedListBox".
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void CheckedListBox_SelectedIndexChanged(object sender, EventArgs e)
{
    //detail pokemon on the right side
    string pokemonName = (string)checkedListBox.SelectedItem;
    Level1 pokemonSelected = (Level1)(_choose.PokemonList.Where(p =>
        p.Name == pokemonName).FirstOrDefault());
}

```

```

        if (pokemonSelected != null)
        {
            labelName.Text = pokemonSelected.Name;
            labelAttribute.Text = pokemonSelected.Attribute.ToString();
            labelAttack.Text = pokemonSelected.Attack.ToString();
            labelDefence.Text = pokemonSelected.Defence.ToString();
            labelSkill1.Text = pokemonSelected.S1.Name;
            labelSkill2.Text = pokemonSelected.S2.Name;
            picture.Image = Image.FromFile(Properties.Settings.Default.pathSprites + "/front/"
                                                + pokemonSelected.Name + ".gif");
        }
    }

    /// <summary>
    /// Action to take when the form is closed.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void FormChoose_FormClosed(object sender, FormClosedEventArgs e)
    {
        _choose.Exit();
    }

    #endregion
}
}

```

### 4.3.2 Classe FormChange

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using PokemonGame.Controller;
using PokemonGame.Model;

namespace PokemonGame.View
{
    /// <summary>
    /// Form for changing the main pokemon.
    /// </summary>
    /// <remarks>
    /// Extends the <c>Form</c> class.
    /// <see cref="Form"/>
    /// </remarks>
    public partial class FormChange : Form
    {
        // Definition of private internal attributes.
        #region Private
        private List<Pokemon> _pokemonList;
        private Pokemon _selectedPokemon;
        #endregion
    }
}

```

```

// Definition of public attributes, for the "get/set" methods.
#region Public
/// <summary>SPokemon that was selected.</summary>
public Pokemon SelectedPokemon { get => _selectedPokemon; set => _selectedPokemon = value; }
#endregion

// Definition of class methods.
#region Methods

/// <summary>
/// Constructor method of the <c>FormChange</c> class.
/// </summary>
/// <param name="pokemonList">List of pokemon of which one can choose.</param>
/// <param name="isRoundPlayer1">
/// Boolean that identifies if it is the round of the player one.
/// </param>
public FormChange(List<Pokemon> pokemonList, bool isRoundPlayer1)
{
    InitializeComponent();

    FormBorderStyle = FormBorderStyle.FixedSingle;
    MaximizeBox = false;

    labelPlayer.Text = (isRoundPlayer1) ? "Player1" : "Player2";

    _pokemonList = pokemonList;

    foreach (Pokemon p in _pokemonList)
    {
        if (p.HealthPoints > 0)
        {
            comboBox.Items.Add(p.Name);
        }
    }

    comboBox.SelectedIndex = 0;
}

/// <summary>
/// Action to take when "ButtonChange" is clicked.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonChange_Click(object sender, EventArgs e)
{
    string selected = (string)comboBox.SelectedItem;
    _selectedPokemon = _pokemonList.Where(p => p.Name == selected).FirstOrDefault();
    Close();
}

/// <summary>
/// Action to take when changing a section in the "ComboBox".
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ComboBox_SelectedIndexChanged(object sender, EventArgs e)
{
    // Detail pokemon on the right side.
    string pokemonName = (string)comboBox.SelectedItem;
    Level1 pokemonSelected = (Level1)_pokemonList.Where(p =>
        p.Name == pokemonName).FirstOrDefault();
}

```

```

        int level = ControllerGame.LevelOf(pokemonSelected);

        picture.Image = Image.FromFile(Properties.Settings.Default.pathSprites + "/front/" +
            pokemonSelected.Name + ".gif");

        labelName.Text = pokemonSelected.Name;
        labelAttribute.Text = pokemonSelected.Attribute.ToString();
        labelAttack.Text = pokemonSelected.Attack.ToString();
        labelDefence.Text = pokemonSelected.Defence.ToString();
        labelLevel.Text = level.ToString();
        labelExperience.Text = pokemonSelected.Exp.ToString();
        labelHealthPoints.Text = pokemonSelected.HealthPoints.ToString();
        labelSkill1.Text = pokemonSelected.S1.Name;
        labelSkill2.Text = pokemonSelected.S2.Name;

        if (level == 1)
        {
            labelSkill3.Visible = false;
            labelTxtSkill3.Visible = false;
            labelSkill4.Visible = false;
            labelTxtSkill4.Visible = false;
        }

        if (level >= 2)
        {
            labelSkill3.Text = ((Level2)pokemonSelected).S3.Name;
            labelSkill3.Visible = true;
            labelTxtSkill3.Visible = true;
        }

        if (level == 3)
        {
            labelSkill4.Text = ((Level3)pokemonSelected).S4.Name;
            labelSkill4.Visible = true;
            labelTxtSkill4.Visible = true;
        }
    }

    /// <summary>
    /// Action to take when the form is closed.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void FormChange_FormClosing(object sender, FormClosingEventArgs e)
    {
        if (_selectedPokemon == null)
        {
            string selected = (string)comboBox.SelectedItem;
            _selectedPokemon = _pokemonList.Where(p => p.Name == selected).FirstOrDefault();
        }
    }

    #endregion
}
}

```

### 4.3.3 Classe FormGame

```
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using PokemonGame.Controller;
using PokemonGame.Exceptions;
using PokemonGame.Model;

namespace PokemonGame.View
{
    /// <summary>
    /// Form for fighting between the pokemon of the two players.
    /// </summary>
    /// <remarks>
    /// Extends the <c>Form</c> class.
    /// <see cref="Form"/>
    /// </remarks>
    public partial class FormGame : Form
    {
        // Definition of private internal attributes.
        #region Private
        private ControllerGame _game;
        private Pokemon _p1;
        private Pokemon _p2;
        #endregion

        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>FormGame</c> class.
        /// </summary>
        /// <param name="pokemonPlayer1">List of pokemon chosen by player one.</param>
        /// <param name="pokemonPlayer2">List of pokemon chosen by player two.</param>
        public FormGame(List<Pokemon> pokemonPlayer1, List<Pokemon> pokemonPlayer2)
        {
            InitializeComponent();

            this.FormBorderStyle = FormBorderStyle.FixedSingle;
            this.MaximizeBox = false;

            labelMessage.Text = "";

            _game = new ControllerGame(pokemonPlayer1, pokemonPlayer2);

            _game.Start();

            Change_round();
        }

        /// <summary>
        /// Update view items.
        /// </summary>
        private void UpdateGraphics()
        {
            int p1_level, p2_level;
```

```

// The correct values are assigned to the graphic elements based on the round.

progressBar1.Value = _game.PokemonSelectedPlayer1.HealthPoints;
progressBar4.Value = _game.PokemonSelectedPlayer1.HealthPoints;
progressBar2.Value = _game.PokemonSelectedPlayer2.HealthPoints;
progressBar3.Value = _game.PokemonSelectedPlayer2.HealthPoints;

if (_game.IsRoundPlayer1)
{
    labelPlayer.Text = "Player 1";
    _p1 = _game.PokemonSelectedPlayer1;
    _p2 = _game.PokemonSelectedPlayer2;

    progressBar1.BringToFront();
    progressBar2.BringToFront();
}
else
{
    labelPlayer.Text = "Player 2";
    _p1 = _game.PokemonSelectedPlayer2;
    _p2 = _game.PokemonSelectedPlayer1;

    progressBar3.BringToFront();
    progressBar4.BringToFront();
}

hp1.Text = _p1.HealthPoints.ToString();
hp2.Text = _p2.HealthPoints.ToString();

p1_level = ControllerGame.LevelOf(_p1);
p2_level = ControllerGame.LevelOf(_p2);

picture1.Image = Image.FromFile(Properties.Settings.Default.pathSprites +
    "/back/" + _p1.Name + ".gif");
picture2.Image = Image.FromFile(Properties.Settings.Default.pathSprites +
    "/front/" + _p2.Name + ".gif");

labelLevel1.Text = p1_level.ToString();
labelLevel2.Text = p2_level.ToString();

labelName1.Text = _p1.Name;
labelName2.Text = _p2.Name;

labelExp1.Text = _p1.Exp.ToString();
labelExp2.Text = _p2.Exp.ToString();

// Different actions are performed based on the level of the pokemon of the indicator.
if (p1_level == 1)
{
    buttonSkill11.Text = ((Level1)_p1).S1.Name;
    buttonSkill12.Text = ((Level1)_p1).S2.Name;
    buttonSkill13.Visible = false;
    buttonSkill14.Visible = false;
}
else if (p1_level == 2)
{
    buttonSkill11.Text = ((Level2)_p1).S1.Name;
    buttonSkill12.Text = ((Level2)_p1).S2.Name;
    buttonSkill13.Text = ((Level2)_p1).S3.Name;
    buttonSkill13.Visible = true;
}

```



```

        buttonSkill4.Visible = false;
    }
    else
    {
        buttonSkill1.Text = ((Level3)_p1).S1.Name;
        buttonSkill2.Text = ((Level3)_p1).S2.Name;
        buttonSkill3.Text = ((Level3)_p1).S3.Name;
        buttonSkill4.Text = ((Level3)_p1).S4.Name;
        buttonSkill3.Visible = true;
        buttonSkill4.Visible = true;
    }
}

/// <summary>
/// Change the game round and then the form too.
/// </summary>
private void Change_round()
{
    _game.NumRound++;

    // If the pokemon has been defeated it will be asked to change it.
    if (_game.PokemonSelectedPlayer1.HealthPoints == 0)
    {
        ChangeDeadPokemon(_game.PokemonSelectedPlayer1);
    }

    if (_game.PokemonSelectedPlayer2.HealthPoints == 0)
    {
        ChangeDeadPokemon(_game.PokemonSelectedPlayer2);
    }

    UpdateGraphics();
}

/// <summary>
/// Write a new message.
/// </summary>
/// <param name="message">The new message.</param>
private void WriteMessage(string message)
{
    labelMessage.Text = message;
}

/// <summary>
/// Allows you to change the pokemon that was defeated.
/// </summary>
/// <param name="p">Pokemon to change.</param>
private void ChangeDeadPokemon(Pokemon p)
{
    WriteMessage("Pokemon " + p.Name + " died!");
    UpdateGraphics();

    if (!_game.ChangePokemon())
    {
        if (MessageBox.Show("Player " + ((_game.IsRoundPlayer1) ? "2" : "1") +
            " win! \n Do you want restart?", "End game",
            MessageBoxButtons.YesNo, MessageBoxIcon.Information) == DialogResult.Yes)
        {
            _game.Restart();
        }
    }
}

```

```

        else
        {
            _game.Exit();
        }
    }
}

/// <summary>
/// Action to do when "ButtonChangePokemon" is clicked.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonChangePokemon_Click(object sender, System.EventArgs e)
{
    try
    {
        _game.ChangePokemon();
        Change_round();
    }
    catch (ChangeException ex)
    {
        MessageBox.Show(ex.Message, "Pokemon change",
            MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
}

/// <summary>
/// Action to do when the form is closed.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void FormGame_FormClosed(object sender, FormClosedEventArgs e)
{
    _game.Exit();
}

#region BUTTON SKILL

/// <summary>
/// Action to do when the skill button one was clicked.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void ButtonSkill1_Click(object sender, System.EventArgs e)
{
    Level1 p = (Level1)_p1;

    if (!_game.DoSkill(p.S1))
    {
        WriteMessage(p.Name + " failed " + p.S1.Name + "!");
    }
    else
    {
        WriteMessage(p.Name + " uses " + p.S1.Name + "!");
    }

    if (_game.Evolve())
    {
        WriteMessage(p.Name + " evolved!");
    }
}

```

```

        Change_round();
    }

    /// <summary>
    /// Action to do when the skill button two was clicked.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonSkill2_Click(object sender, System.EventArgs e)
    {
        Level1 p = (Level1)_p1;

        if (!_game.DoSkill(p.S2))
        {
            WriteMessage(p.Name + " failed " + p.S2.Name + "!");
        }
        else
        {
            WriteMessage(p.Name + " uses " + p.S2.Name + "!");
        }

        if (_game.Evolve())
        {
            WriteMessage(p.Name + " evolved!");
        }

        Change_round();
    }

    /// <summary>
    /// Action to do when the skill button three was clicked.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonSkill3_Click(object sender, System.EventArgs e)
    {
        Level2 p = (Level2)_p1;

        if (!_game.DoSkill(p.S3))
        {
            WriteMessage(p.Name + " failed " + p.S3.Name + "!");
        }
        else
        {
            WriteMessage(p.Name + " use " + p.S3.Name + "!");
        }

        if (_game.Evolve())
        {
            WriteMessage(p.Name + " evolved!");
        }

        Change_round();
    }

    /// <summary>
    /// Action to do when the skill button four was clicked.
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
    private void ButtonSkill4_Click(object sender, System.EventArgs e)

```

```

{
    Level3 p = (Level3)_p1;

    if (!_game.DoSkill(p.S4))
    {
        WriteMessage(p.Name + " failed " + p.S4.Name + "!");
    }
    else
    {
        WriteMessage(p.Name + " use " + p.S4.Name + "!");
    }

    if (_game.Evolve())
    {
        WriteMessage(p.Name + " evolved!");
    }

    Change_round();
}

#endregion

#endregion
}
}

```

## 4.4 Altre classi

### 4.4.1 Classe ChangeException

```
using System;

namespace PokemonGame.Exceptions
{
    /// <summary>
    /// Exception generated when a pokemon is already chosen on the battlefield.
    /// </summary>
    /// <remarks>
    /// The exception extends directly <c>Exception</c>.
    /// </remarks>
    public class ChangeException : Exception
    {
        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>SkillNotFoundException</c> class.
        /// It already includes a standard message.
        /// </summary>
        public ChangeException()
            : base("Pokemon already in the battlefield.") { }

        /// <summary>
        /// Constructor method of the <c>SkillNotFoundException</c> class.
        /// Add a specified message.
        /// </summary>
        /// <param name="message">Error message.</param>
        public ChangeException(string message)
            : base(message) { }

        /// <summary>
        /// Constructor method of the <c>SkillNotFoundException</c> class.
        /// Add a specified message and reference to the internal exception,
        /// which is the cause of the current exception.
        /// </summary>
        /// <param name="message">Error message.</param>
        /// <param name="inner">Reference to the internal exception</param>
        public ChangeException(string message, Exception inner)
            : base(message, inner) { }

        }

    #endregion
}
```

## 4.4.2 Classe SkillNotFoundException

```
using System;

namespace PokemonGame.Exceptions
{
    /// <summary>
    /// Exception generated when a skill was not found.
    /// </summary>
    /// <remarks>
    /// The exception extends directly <c>Exception</c>.
    /// </remarks>
    public class SkillNotFoundException : Exception
    {
        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Contructor method of the <c>SkillNotFoundException</c> class.
        /// It already includes a standard message.
        /// </summary>
        public SkillNotFoundException()
            : base("Skill not found!") { }

        /// <summary>
        /// Contructor method of the <c>SkillNotFoundException</c> class.
        /// Add a specified message.
        /// </summary>
        /// <param name="message">Error message.</param>
        public SkillNotFoundException(string message)
            : base(message) { }

        /// <summary>
        /// Contructor method of the <c>SkillNotFoundException</c> class.
        /// Add a specified message and reference to the internal exception,
        /// which is the cause of the current exception.
        /// </summary>
        /// <param name="message">Error message.</param>
        /// <param name="inner">Reference to the internal exception</param>
        public SkillNotFoundException(string message, Exception inner)
            : base(message, inner) { }

        #endregion
    }
}
```

### 4.4.3 Classe PokemonNotFoundException

```
using System;

namespace PokemonGame.Exceptions
{
    /// <summary>
    /// Exception generated when a pokemon was not found.
    /// </summary>
    /// <remarks>
    /// The exception extends directly <c>Exception</c>.
    /// </remarks>
    public class PokemonNotFoundException : Exception
    {
        // Definition of class methods.
        #region Methods

        /// <summary>
        /// Constructor method of the <c>PokemonNotFoundException</c> class.
        /// It already includes a standard message.
        /// </summary>
        public PokemonNotFoundException()
            : base("Pokemon not found!") { }

        /// <summary>
        /// Constructor method of the <c>PokemonNotFoundException</c> class.
        /// Add a specified message.
        /// </summary>
        /// <param name="message">Error message.</param>
        public PokemonNotFoundException(string message)
            : base(message) { }

        /// <summary>
        /// Constructor method of the <c>PokemonNotFoundException</c> class.
        /// Add a specified message and reference to the internal exception,
        /// which is the cause of the current exception.
        /// </summary>
        /// <param name="message">Error message.</param>
        /// <param name="inner">Reference to the internal exception.</param>
        public PokemonNotFoundException(string message, Exception inner)
            : base(message, inner) { }

        #endregion
    }
}
```

#### 4.4.4 Classe Program

```
using System;
using System.Windows.Forms;
using PokemonGame.View;

namespace PokemonGame
{
    static class Program
    {
        /// <summary>
        /// Punto di ingresso principale dell'applicazione.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FormChoose());
        }
    }
}
```



## 5. Testing

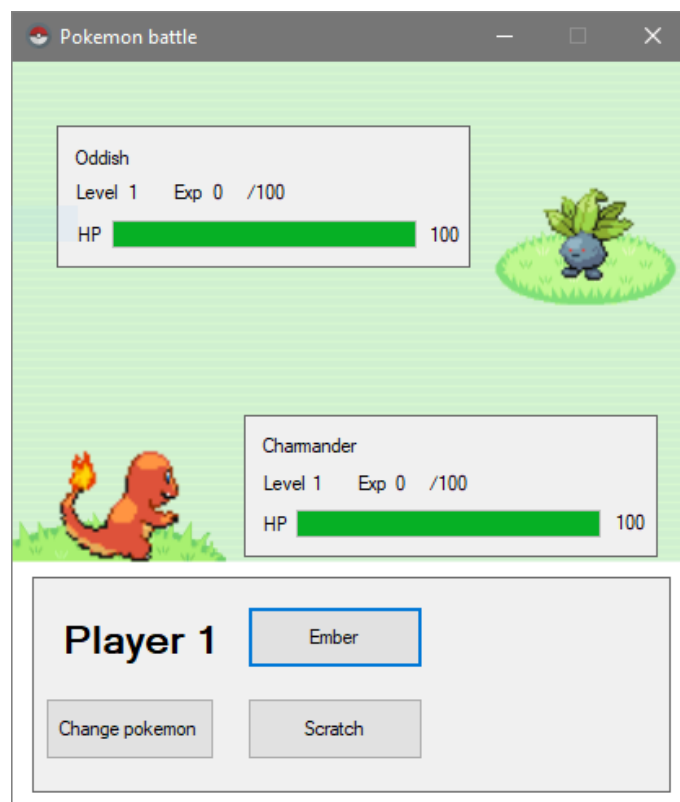
Per garantire il corretto funzionamento del gioco, ogni input dell'utente (quindi ogni caso d'uso) viene testato in ogni sua parte. I test effettuati sono sia di tipo white-box (basati sulla conoscenza del codice) che di tipo black-box (test effettuati senza considerare il codice).

Verranno di seguito illustrati alcuni dei test degli di nota.

### 5.1 Test sull'efficacia del calcolo sul danno

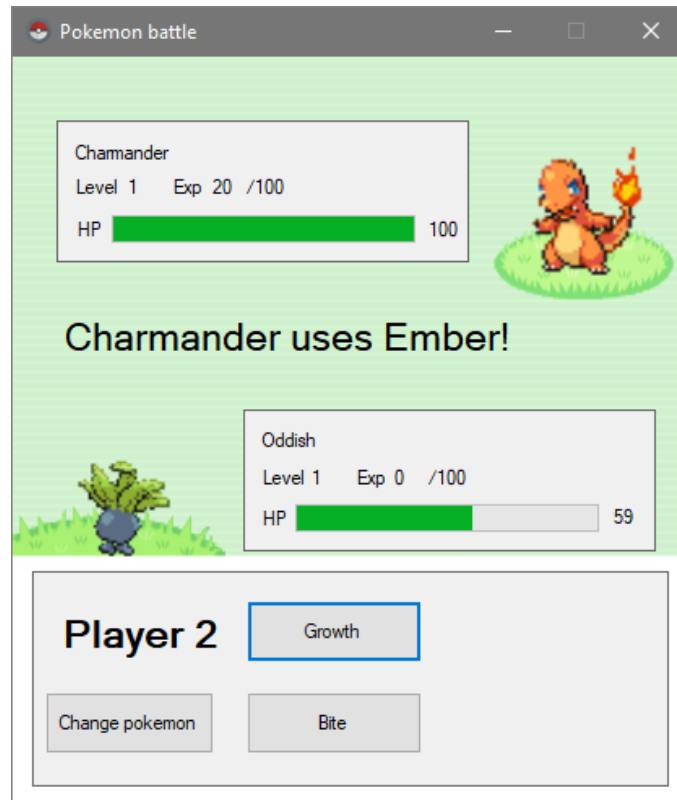
Si vuole verificare il calcolo del danno, mediante un test black-box.

Come pokémon attaccante viene scelto *"Charmander"* (attributo: *Fuoco*, attacco: *10*), come pokémon attaccato viene scelto *"Oddish"* (attributo: *Erba*, difesa: *5*) e viene scelta come mossa *"Ember"* (danno: *20*).



Secondo il calcolo descritto nella specifica:  $(20 + (10\% 20)) * 2 = 44$   
 $44 - (5\% 44) = 41,8$

I punti vita di *"Oddish"* dopo l'attacco dovranno essere 59, in quanto il danno viene sempre approssimato per difetto.



Come si può notare i punti vita di *"Oddish"* sono quelli previsti.

## 5.2 Test sulla lettura dei pokémon da file csv

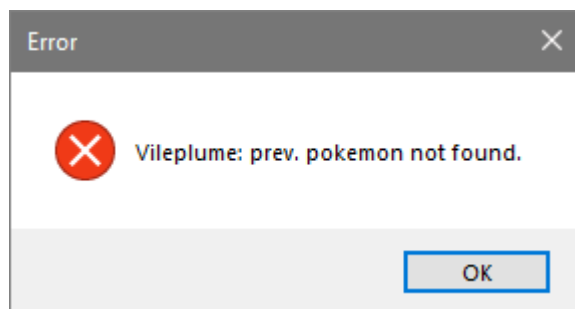
Si vogliono testare i controlli sulla correttezza dei file csv, mediante un test black-box. Per la struttura dei file si rimanda alla sezione [Analisi e progettazione](#).

Di seguito verranno mostrate delle righe errate nei file csv, con una breve spiegazione e l'immagine riportante l'errore.

*pokemon.csv:*

```
1;Grass;Oddish;5;5;Growth;Bite  
2;Oddish;Gloom;10;40;Spore  
3;G100m;Vileplume;25;75;Repose
```

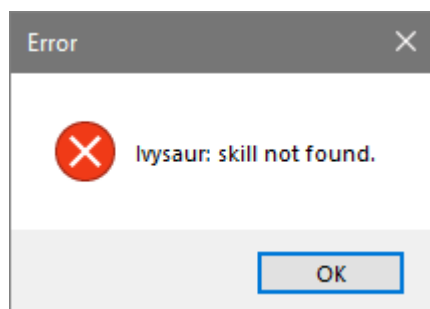
il pokémon "G100m" non esiste, si ottiene quindi il seguente messaggio di errore:



*pokemon.csv:*

```
2;Bulbasaur;Ivysaur;25;25;L4ssshh
```

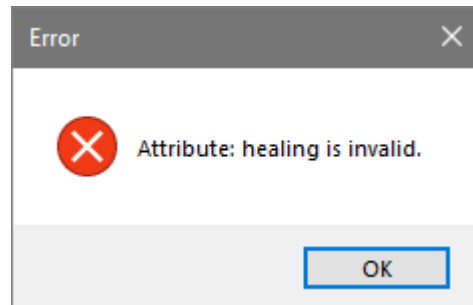
la mossa "L4ssshh" non esiste, si ottiene quindi il seguente messaggio di errore:



*skill.csv:*

healing;Growth;10;25

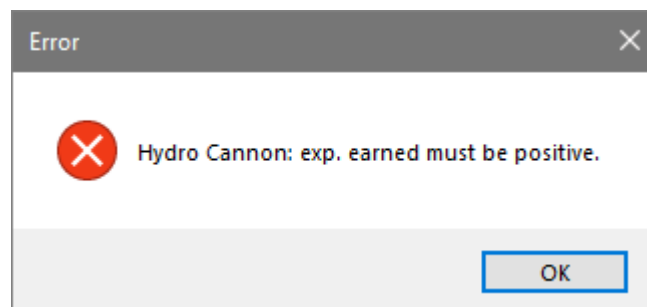
il tipo di mossa "healing" non è valido, si ottiene quindi il seguente messaggio di errore:



*skill.csv:*

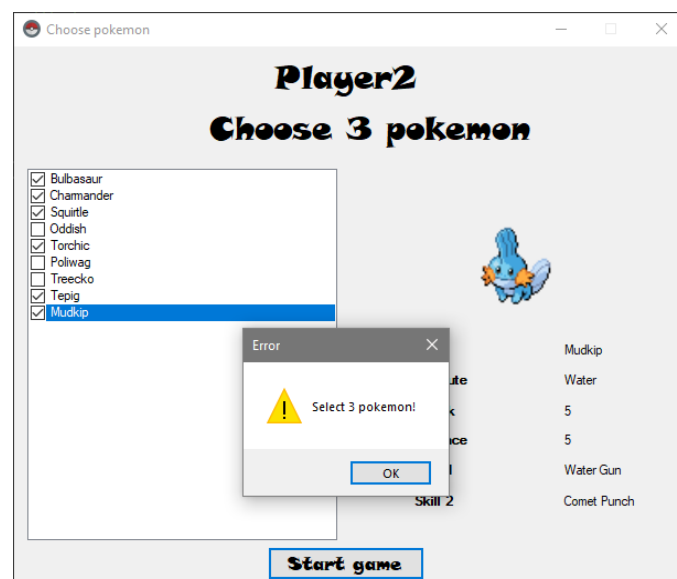
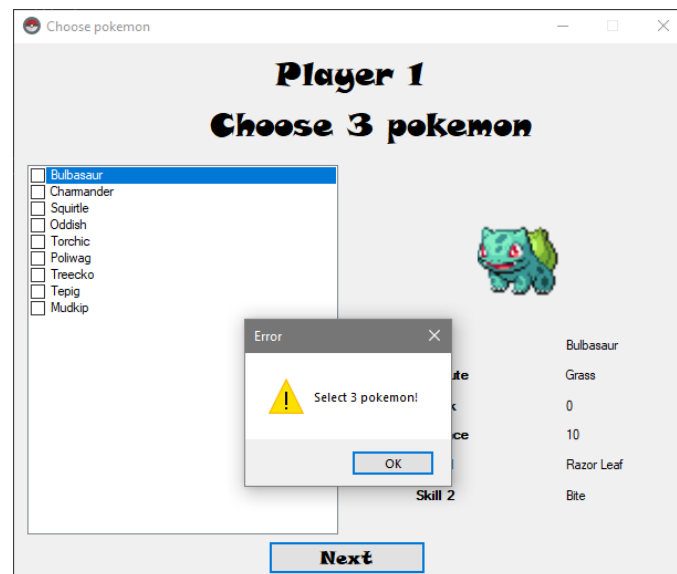
attack;Hydro Cannon;50;-35

il valore dell'esperienza guadagnata dalla mossa è negativo, si ottiene quindi il seguente messaggio di errore:



### 5.3 Test sulla scelta dei pokémon in numero diverso da tre

Si vuole verificare che non si possa scegliere un numero di pokémon diverso da tre, mediante un test black-box.



Come si può notare dalle immagini il programma mostra un avvertenza nel caso in cui si scelgano un numero non consentito di pokémon, per entrambi i giocatori.

## 5.4 Test sulle evoluzioni

Si vuole verificare la correttezza delle evoluzioni dei pokémon mediante un test white-box.

In particolare, sappiamo che ogni volta che viene eseguita una mossa, viene richiamata la funzione *evolve()* per controllare se un pokémon può effettuare una evoluzione.

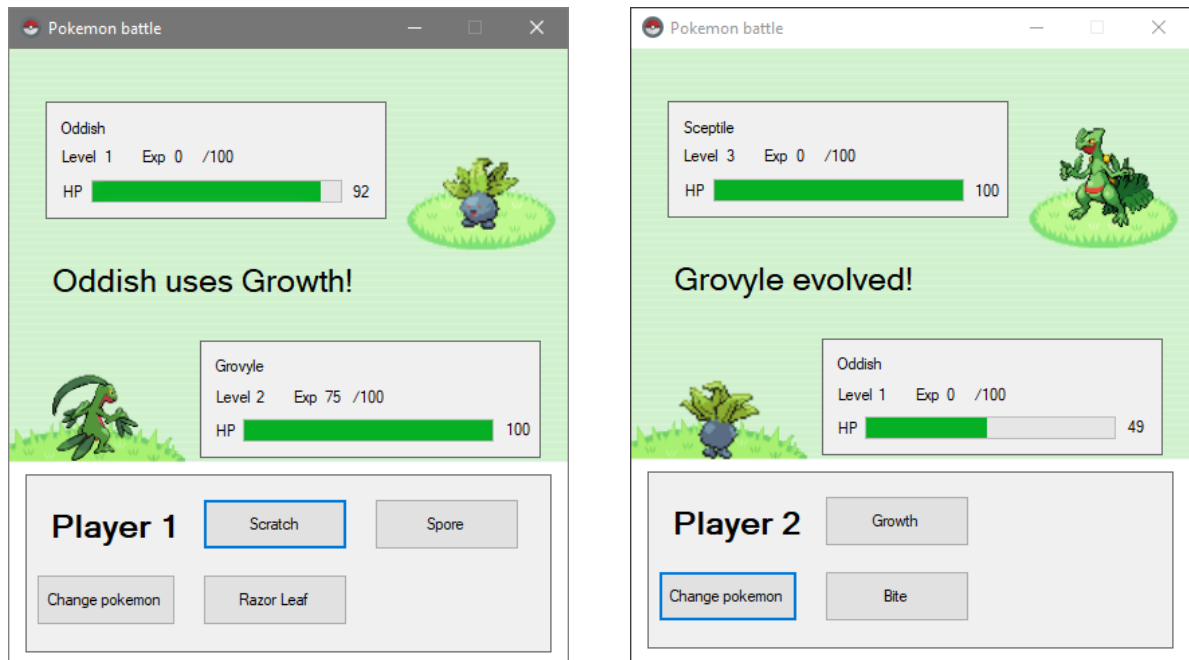
La funzione è così implementata:

```
public bool Evolve()
{
    bool success = false;
    Pokemon next;

    // A pokemon evolves after using a skill, so it is checked
    // if it has the max. experience points on its turn.
    if (_isRoundPlayer1)
    {
        if (_pokemonSelectedPlayer1.Exp == 100 && _pokemonSelectedPlayer1.NextLevel != null)
        {
            next = (Pokemon)_pokemonSelectedPlayer1.NextLevel.Clone();
            _pokemonPlayer1.Remove(_pokemonSelectedPlayer1);
            _pokemonPlayer1.Add(next);
            _pokemonSelectedPlayer1 = next;
            success = true;
        }
    }
    else
    {
        if (_pokemonSelectedPlayer2.Exp == 100 && _pokemonSelectedPlayer2.NextLevel != null)
        {
            next = (Pokemon)_pokemonSelectedPlayer2.NextLevel.Clone();
            _pokemonPlayer2.Remove(_pokemonSelectedPlayer2);
            _pokemonPlayer2.Add(next);
            _pokemonSelectedPlayer2 = next;
            success = true;
        }
    }
    return success;
}
```

Come si può notare dal codice, l'evoluzione viene effettuata solo nel caso in cui il pokémon abbia, il valore dei punti esperienza a 100 e un livello successivo.

Si inizia il test avviando una nuova partita e giocando alcuni turni.



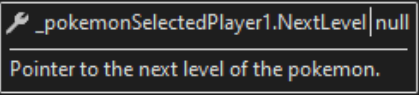
Come si può notare, "Grovyle" si è evoluto in "Sceptile", in quanto la mossa che ha eseguito lo ha portato ad avere 100 punti esperienza.

Continuando la partita, "Sceptile" anche avendo 100 punti esperienza non si evolve.



L'evoluzione non viene eseguita in quanto quel pokémon non possiede un livello successivo.

```
if (_pokemonSelectedPlayer1.Exp == 100 && _pokemonSelectedPlayer1.NextLevel != null)
{
    next = (Pokemon)_pokemonSelectedPlayer1.NextLevel;
    _pokemonPlayer1.Remove(_pokemonSelectedPlayer1);
    _pokemonPlayer1.Add(next);
    _pokemonSelectedPlayer1 = next;
    success = true;
}
```



L'immagine mostra il controllo su una espressione da cui si nota che non si ha un riferimento al livello successivo.



## 6. Compilazione ed esecuzione

Gli strumenti utilizzati per la compilazione del programma sono i seguenti :

- **Ambiente di sviluppo** : Visual Studio Community 2019
- **Versione** : 16.2.5
- **Framework**: .NET Framework 4.6

Per la compilazione mediante ambiente di sviluppo, nella barra dei menù selezionare *Compila > Compila soluzione*.

Per eseguire l'applicazione fare doppio click sull'eseguibile nel percorso "*PokemonGame\bin\Release\Pokemon.exe*"

I requisiti minimi per l'esecuzione del programma sono i seguenti :

- **Sistema operativo** : Windows Vista SP2
- **Architettura** : 32 Bit & 64 Bit
- **Framework**: .NET Framework 4.6

Non vi sono requisiti di performance particolari, tuttavia si rimanda a consultare i requisiti minimi per il Framework nel sito :

<https://docs.microsoft.com/dotnet/framework/get-started/system-requirements>

tenendo in considerazione che il programma utilizza al più 50 MB di memoria RAM.

Il software è stato testato su un computer avente le seguenti caratteristiche:

- **CPU** : Intel Core i7-8550U @ 1.80 GHz
- **RAM** : 16 GB
- **GPU** : NVIDIA GeForce 930MX
- **S.O** : Windows 10 Pro, Build 1809
- **Architettura** : 64 Bit