

# Pokemon Game Lite

Ein Pokemon-Spiel (ohne Grafik)



# Pokemon Game Lite



## Das Inhaltsverzeichnis:

1.- Spielbeschreibung. Anforderungen und entwickelte.

2.- OOP Javascript Klassendiagramm.

2.1.- Die Klasse "Attack".

2.2.- Die Klasse "Pokemon".

2.3.- Die Klasse "UserPokemon".

2.4 - Die Pokemon-Klasse "Fight".

3.- Überraschung und Verbesserungen.



# 1.- Spielbeschreibung. Anforderungen und entwickelte.

**Game development with JS:** I think (and I have checked it) that the best way to "learn" a language is by developing games, because you "play" everything and it's fun.

**1.- "Pokemon Game Lite":** as a **pure OO development**, no matter if the language is JS, you will see the final result.

**2.- "Pokemon Game Lite",** I didn't care about the **"graphical aspect"**, but the analysis and implementation of the OO **"as pure"** as possible to the project.

**3.- Why two projects?** As I saw that everybody was developing the applications "graphically" (**DOM**) I thought **I was wrong**, so I decided to develop a graphical game, **"Bubbles and Fish"**, but I think that the Pokemon game, although less **"showy"**, is more **"valuable"**.

**"Brain" OO**



**"Visual" - DOM**





## 2.1.- Die Klasse "Attack".

**Attack class:** Objects that representing the "attacks possible, created and available" in the game, with properties and methods:

### 1.- Properties of attack class.

1.1- idAttack

1.2.- imgAttack

1.3.- nameAttack

1.4.- acquiValueAttack

1.5.- typeAttack

1.6.- damageOfAttack

1.7.- useOfMagic

1.8.- arrayIdAttacks

### 2.- Attack methods:

2.0.- constructor (imgAttack, nameAttack, typeAttack, damageOfAttack, useOfMagic)

2.1.- attackEfficiency ()

2.2.- addAttackToArray ()

2.3.- showAttack (parTypeArrayAttack = 'avaibleAttacks')

## 2.1.- Die Klasse "Attack".

Attack class

```
{  
  imgAttack      : 'imgAttack',  
  idAttack       : 1,  
  nameAttack     : 'Wing Steel',  
  acquiValueAttack : 35,  
  forTypePokemon : ['Bug', 'Flying'],  
  IniDamage      : 35,  
  amountMagic    : 10  
}
```

IdAttack: 1

"Wing Steel"

I dont like "Bug" and "" Flying Pokemon

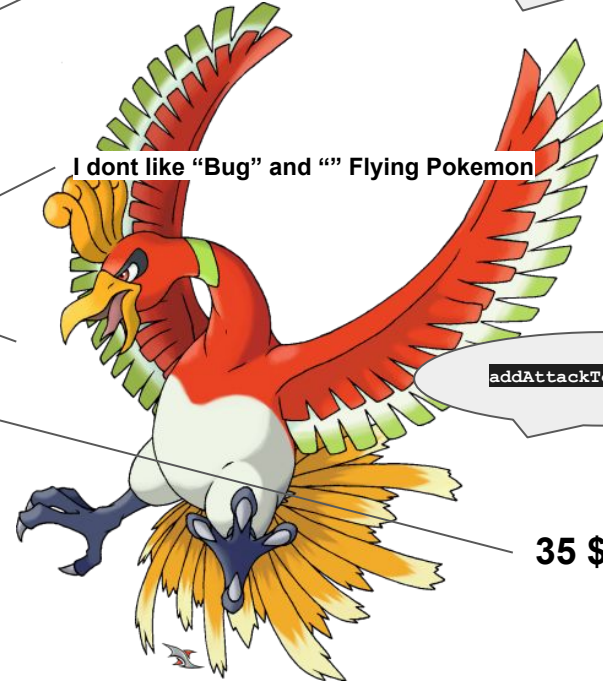
addAttackToArray ()

35 \$

showAttack (parTypeArrayAttack = 'avaibleAttacks')

constructor

attackEfficiency ()





## 2.2.- Die Klasse "Pokemon".

**Pokemon class:** Objects that representing the "pokemons" possible "created and available" in the game, with the properties and methods:

### 1.- Properties of Pokemon class.

1.1.- imagePokemon

1.2.- idPokemon

1.3.- namePokemon

1.4.- acquisitionValue

1.5.- level

1.6.- typePokemon

1.7.- weakness

1.8.- evolution

1.9.- healthPokemon

1.10.- amountMagic

1.11.- skillsOfPokemon

1.12.- userPro

### 2.- Pokemon methods:

2.0.- constructor (namePokemon, health, magic, skills = [])

2.1.- dataAdquisitionPokemon (parName)

2.2.- learnAttackSkill (...parObjectAttack)

2.3.- attack (idAttack, pokemonObjectiv)

2.4.- setAmountMagic (parAmountMagic)

2.5.- showSkill()

2.6.- showStatus ()

2.7.- static showArrayAvaibleOfPokemon (parArrayOfPokemons  
= arrayOfAvaibleRandomPokemons)

2.8.- static showGeneralOfPokemon (parArrayOfPokemons =  
arrayGeneralOfPokemons)

2.9.- static isPokemonLive (parPokemon)

## 2.2.- Die Klasse "Pokemon".



```
{  
  imagePokemon : '⚡ (●●_ ●●) ♂',  
  idPokemon : 25,  
  namePokemon : 'Pikachu',  
  acquisitionValue : 500,  
  level : 1,  
  type : ['Electric'],  
  weakness : ['Earth'],  
  evolution : ['Raichu'],  
}
```

```
1.9.- healthPokemon  
1.10.- amountMagic  
1.11.- skillsOfPokemon  
1.12.- userPro
```



constructor

dataAdquisitionPo  
kemon

25

setAmountMagic

I am Pikachu

I dont like "Earth" Pokemon

Electric

500 \$

learnAttackSkill

{ "attacks" Objects }

attack

"Raichu" in the Future

showStatus ()





## 2.3.- Die Klasse "UserPokemon".

**UserPokemon class:** Objects that representing the "pokemons" possible "created and available" in the game, with the properties and methods:

### 1.- Properties of Pokemon class.

1.1.- idUserPokemon

1.2.- \_imagesUser

1.3.- nameUser

1.4.- experience

1.5.- pokemonsOfUser

1.6.- capacityLearnAttacks

1.7.- capacityToAcquirePokemons

1.8.- \_attacksLearndSkill

1.9.- AcquisitionIniCapacity

### 2.- Pokemon methods:

2.0.- constructor (idUserPokemon, nameUser, experience, pokemonsOfUser = [])

2.1.- completeUserProperties (parExperience)

2.2.- pokemonBelongsToUser (parPokemonObject)

2.3.- addOutPokemonToUser (parObjectPokemon)

2.4.- selectPokemon (parTypeSearch, parIdPokemon)

2.5.- learnAttackSkill (...parObjectAttack)

2.6.- passToPokemonAttack (parNameAttack, parIdPokemon)

2.7.- giveMagicToPokemon (parAmountMagic, parIdOwnTargetPokemon)

2.8.- showSkill()

2.9.- showOwnPokemon ()

2.10.- showStatus ()





## 2.3.- Die Klasse "UserPokemon".

UserPokemon class

constructor

completeUserProperties (parExperience)

Jose Antonio Fernández

```
{
  idUserPokemon : 2,
  imagesUser : '',
  nameUser : 'Jose Antonio',
  experience : ['very experienced', 8],
  pokemonsOfUser: ['Pikachu', 'Meow'],
  capacityLearnAttacks : 35,
  capacityToAcquirePokemons : 435,
  attacksLearndSkill : 250,
  AcquisitionIniCapacity : 800
}
```

Id: 2

['very experienced', 8]

Own "Pokemon"  
(Pikachu)

showStatus ()

35

2500 \$

pokemonBelongsToUser (parPokemonObject)

selectPokemon (parTypeSearch, parIdPokemon)

attacksLearndSkill

capacityToAcquirePokemons

learnAttackSkill (...parObjectAttack)

passToPokemonAttack (parNameAttack, parIdPokemon)





## 3.- Überraschung und Verbesserungen

**1.- Überraschungen:** OO in Javascript does not work "the same" as other languages (Java, C++, etc).

1.- Disadvantages of JS in OO: *meine Meinung nach ....*

0.- As JS is multi-typed, everything is "strange", the developments can start with OO, continue in a functional way and finish in a sequential way.

1.- The "inheritance" is rare, by prototyping, delegating to objects, ...

2.- There are neither methods nor "private" properties and methods

3.- Problem in the design of communication by messages between objects ("It is a serious problem").

4.- There is "sugar syntax", the concept of class does not exist, the concept of "OO inheritance by Classes" does not exist, in "OO with theatre", only the "object" exists, the object is the "king".

5.- How to kill an object in a "controlled" way???? Garbage Collector worries (another "Java")

....

2.- Advantages of OO in JS:

1.- Functional concepts in OO work very well though, they help the low level design of the methods.

2.- For not very big developments OO in JS works very good.

Anyway, in general I'm quite satisfied with the results, I still need to study and go deeper, but I think you can develop OO with JS.



## 3.- Überraschung und Verbesserungen

### 1.- Verbesserungen:

1.- Develop the **"classes"** **"Fight"** and **"Environment"**.

2.- Rethink the **"intelligence"** of the **"machine"** users in a program (just random and other variables is not enough).

3.- Move to graphical environment (**DOM development**)

4.- Think about the method **"Pokemon training"** and development of **"minigames"** for the Pokemon to acquire and improve their attacks and skills.

5.- I like **JSON mit OO**, but a lot, keep it up.

### My letter to the St. Nikolas of JS (St. Jan and St. Christoph)

*Dear "Santa Nikolases", I have been a "good developer" and I would like to:*

*1.- When the code "grows and grows" and the "bugs" appear, it is complicated to be able to develop: how the "modules" work in JS, or how to "cut" in different files a program in console, (in Brower with DOM I already know).*

*2.- Debugging tools.*

*3.- "Try" and "catch" of code errors.*

*4.- "Typescript" and "Babel", would it be the same thing that Sass represented with CSS for us?*

*Thank you very much.*

*Mit freundlichen Grüßen*