TP RPG Visuel

Objectif

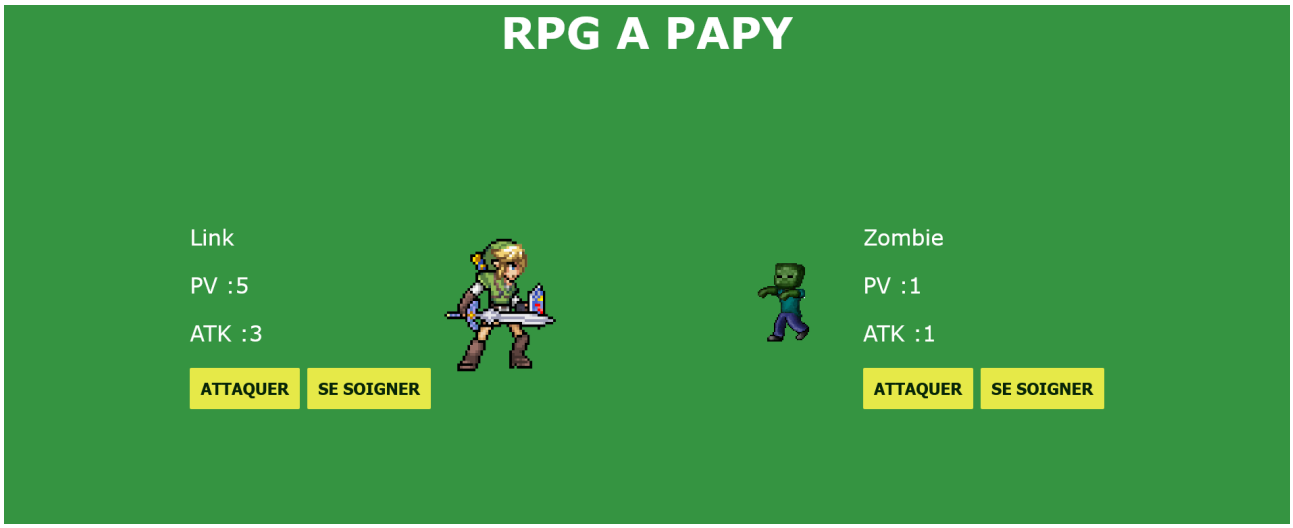
- Se familiariser avec la programmation événementielle et la méthode addEventListener
- Se familiariser avec les fonctions *callback et setTimeout*
- Se familiariser avec l'objet document et l'accession au *DOM* via la méthode *querySelector*

Vue dans ce TP :

- setTimeout
- querySelector
- addEventListener
- Class

Résultat final

Ce projet est l'aboutissement visuel du TP RPG textuel, nous allons pouvoir utiliser nos objets *hero* et *ennemie* et leurs méthodes en réaction au clic de l'utilisateur sur certain bouton.



Mettre en place les fichiers

Dans un dossier nommé *rpg_visuel*, créez un fichier ***index.html***, ***style.css*** et ***script.js*** contenant le code suivant. Puis ouvrez la page html.

index.html

```
<!DOCTYPE html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <h1>RPG A PAPY</h1>
    <main>
        <section id="game">

            <div id="hero" class="character_card">
                <div>
                    <p class="name">[name]</p>
                    <p class="pv">[pv]</p>
                    <p class="attack">[attack]</p>
                    <button class="btn_attack">ATTAQUER</button>
                    <button class="btn_soin">SE SOIGNER</button>
                </div>
                <div>
                    <img src="link.webp" alt="">
                </div>
            </div>

            <div id="ennemy" class="character_card">
                <div>
                    <img src="zombie.gif" alt="ennemy">
                </div>
                <div>
                    <p class="name">[name]</p>
                    <p class="pv">[pv]</p>
                    <p class="attack">[attack]</p>
                    <button class="btn_attack">ATTAQUER</button>
                    <button class="btn_soin">SE SOIGNER</button>
                </div>
            </div>

        </section>
    </main>

    </div>
    <script src="script.js"></script>
</body>
</html>
```

style.css

```
/* Style de placement des balises */
body{
    background-color: rgb(53, 148, 65);
    color:white;
    font-family: Verdana;
```

```css
    font-size: 2rem;
    min-height : 10vh;
}
h1{
    text-align: center;
}
main{
    display: flex;
    align-items: center;
    justify-content: center;
    min-height : 70vh;

}
#game{
    display:grid;
    grid-template-columns: 1fr 1fr;
    gap:200px;
}
.character_card{
    display:flex;
    gap : 20px;
    align-items: center;
}

/* Style de design des balises */
button{
    border-radius: 2px;
    font-size: 1.5rem;
    padding : 1rem;
    border:none;
    background-color: rgb(230, 233, 72);
    color:rgb(7, 37, 8);
    font-weight: bold;
}
button:active{
    scale: 0.90;
}

/* Style d'animation */
img{
    max-height : 200px;
    transition: scale 0.25s;        /* Fluidifie l'animation d'attaque */
}
#ennemy img{
    transform: rotateY(180deg); /* Le gif n'est pas dans le bon sens, on le retourne à 180deg
**/
}

/* Classe css appliqué en JavaScript à une image lorsque un personnage attaque */
.attaquer{
    scale : 1.25;
}
```

script.js

```js
// vide
```

Activité - Déclarer les objets *hero* et ennemie

Déclarez les objets *hero* et *ennemie*, ces objets proviennent du TP RPG – Textuel

Ces objets doivent posséder les mêmes méthodes et attributs, seul la valeur de leurs attributs changes.

On voit apparaître l'attribut n_attaque qui représente les points d'attaque du hero et de l'ennemie

Tester la présence des objets via la console dans le navigateur.

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution

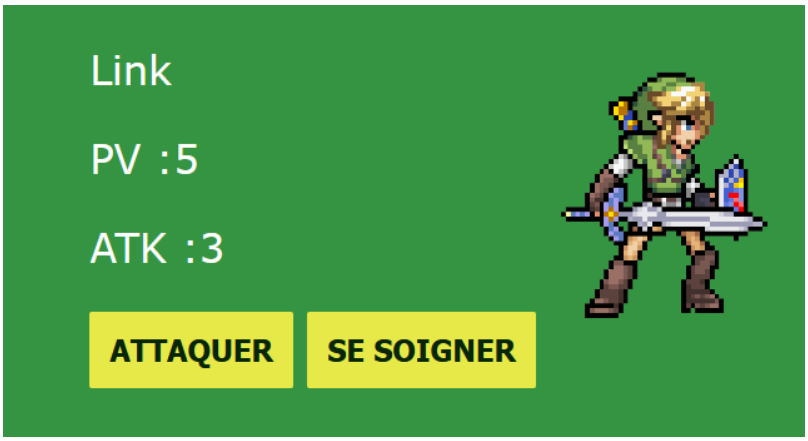
script.js

```
let hero = {
    s_nom : "Link",
    n_pv : 5,
    n_attaque : 3,
    saluer(personnage){
        console.log("Salutation "+personnage.s_nom+" je suis "+this.s_nom+" !");
    },
    ficheDePersonnage(){
        return "Nom : "+this.s_nom+" | PdV : "+this.n_pv;
    },
    soin(n_soin){
        this.n_pv = this.n_pv + n_soin;
    },
    attaquer(ennemie){
        ennemie.n_pv -= this.n_attaque;
    }
};

let ennemie = {
    s_nom : "Zombie",
    n_pv : 4,
    n_attaque : 1,
    saluer(personnage){
        console.log("Salutation "+personnage.s_nom+" je suis "+this.s_nom+" !");
    },
    ficheDePersonnage(){
        return "Nom : "+this.s_nom+" | PdV : "+this.n_pv;
    },
    soin(n_soin){
        this.n_pv = this.n_pv + n_soin;
    },
    attaquer(ennemie){
        ennemie.n_pv-=this.n_attaque;
    }
};
```

Afficher les informations du *hero* dans le HTML

Nous voulons afficher le nom, les points de vie et les points d'attaque du hero dans les balise <p> HTML de la div à l'id « #hero ». Les balises HTML sont de la classe *HTMLElement*.



document.querySelector(cssSelector)

querySelector est une méthode de l'objet *document*, elle permet de récupérer une balise html sous la forme d'une objet JS.

- Cette méthode renvoi un objet de la classe *Element* en fonction du *selecteur css* utilisé en paramètre.
- Le *sélecteur* est une String, il doit être un sélecteur css valide.
- La classe *Element* représente un élément HTML présent dans votre document.

Voir la documentation de la MDN.

Element.innerText

innerText est une attribut de la classe *Element*.

- *innerText* est une String
- *innerText* contient le texte présent dans l'element (la balise) HTML
- si l'on modifie la valeur de l'attribut *innerText* le contenu de son élément va se mettre à jour sur la page.

Le mot clé const

Le mot clé *const* permet de déclarer une variable de la même façon que le mot clé *let*. Cependant cette variable ne peut être modifiée, c'est une constance.

Je recommande de toujours créer des constances sauf si la donnée stockée est vraiment destinée à changer. Ainsi votre code deviendra plus clair vis à vis des éléments « variables » et « statiques » de votre code.

Activité - Afficher le nom du *hero*

Objectif :

Afficher le nom du hero dans la balise <p> correspondant grâce au méthodes précédemment décrite.

Visualisez le changement dans le navigateur.

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution - Afficher le nom du hero

script.js

```
const element_hero_nom = document.querySelector("#hero .name");
element_hero_name.innerText = hero.s_name;
```

Activité - Afficher les points de vie du *hero*

Objectif :

Afficher les points de vie du hero selon le format «PV: 5 » pour des points de vie égal à 5.

Visualisez le changement dans le navigateur.

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution - Afficher les points de vie du hero

script.js

```
const element_hero_pv = document.querySelector("#hero .pv");
element_hero_pv .innerText = "PV : "+hero.n_pv;
```

Activité - Afficher les points d’attaque du *hero*

Activité :

Dans *script.js* ajoutez le code nécessaire pour afficher les points d'attaque du *hero*.

Résultat attendu:

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution – affichage de l’attaque du *hero*.

script.js

```
const element_hero_attack = document.querySelector("#hero .attack");
element_hero_ attack .innerText = "ATK : "+hero.n_attaque;
```

Activité - Afficher les informations de l’ennemi dans le HTML

Rajoutez le code nécessaires pour que les informations de l’ennemie soit correctement affichées

Résultat attendu :

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution – Afficher les informations de l’ennemi dans le HTML.

script.js

```
    const element_ennemy_name = document.querySelector("#ennemy .name");
     const element_ennemy_pv = document.querySelector("#ennemy .pv");
     const element_ennemy_attack = document.querySelector("#ennemy .attack");

     element_ennemy_name.innerText = ennemie.s_nom;
     element_ennemy_pv.innerText = "PV :"+ennemie.n_pv;
     element_ennemy_attack.innerText = "ATK :"+ennemie.n_attaque;
```

Soigner le *hero* avec le bouton soin

Objectif

Lorsque l'utilisateur clique sur le bouton « *SOIGNER* » du *hero* les points de vie du *hero* doivent augmenter et les informations doivent être misent à joueur sur l'affichage HTML du *hero*.

Pour ceci nous aurons besoin de faire de la programmation événementielle, c'est à dire écouter l'événement « click » de la balise <button> SOIGNER du hero et appeler une fonction lorsque l'événement se déclenche.

AddEventListener - Explications

HTMLElement.addEventListener est une méthode de la classe *HTMLElement*, soit une méthode que l'on retrouve sur les éléments retournés par la fonction document.querySelector.
Cette méthode possède deux arguments obligatoire:

- Une string décrivant l'événement à écouter parmi la liste des événements disponible
- Une fonction callback que le navigateur appellera lorsque l'événement sera déclenché.

Activité - Récupérer la balise <*button*>

Récupérez la balise « *SOIGNER* » via la fonction *document.querySelector()*.

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution - Récupérer la balise <*button*>

script.js

```
const hero_btn_soin = document.querySelector("#hero .btn-soin");
```

La programmation événementielle

Écouter l'événement « click »

JavaScript est un langage de programmation événementiel, ce qui signifie qu'il permet d'appeler des fonctions lorsqu'un certain événement se produit. Parmi les événements les plus connus ont retrouve : *click*, *input*, *change* ...

C'est l'événement « click » qui nous intéressent ici.

Écouter l'événement « click »

script.js

```
hero_btn_soin.addEventListener("click",function(){/* code à executer */});
// Quand le joueur "click" sur le bouton soigner : la fonction passée en deuxième
// paramètre est appelée.
```

Activité - Appel de la fonction *hero.soin*

La méthode à utiliser pour soigner le *hero* est *hero.soin*

Appelez la méthode soin et rafraîchissez l'affichage lorsque l'utilisateur clique sur le bouton SOIGNER

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution – Appel de la fonction *hero.soin*

script.js

```
hero_btn_soin.addEventListener("click",function(){
	hero.soin(1); // Le joueur va se soigner de 1 point de vie par clique.
	Draw();	// Mise à jour de l'affichage avec les nouvelles données des objets
});
```

Activité - Le clic du bouton attaquer du *hero*

Objectif

Lorsque l'utilisateur clique sur le bouton «ATTAQUER» du *hero* les points de vie de l'*ennemie* doivent baisser du nombre de point d'attaque du *hero.*

Pour ça vous aurez besoin :

- HTMLElement.querySelector
- document.addEventListener
- et la méthode hero.attaquer

Mettez en place le bouton « ATTAQUER » du héro

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution - Le clic du bouton attaquer du hero

script.js

```
const hero_btn_attack = document.querySelector("#hero .btn_attack");
hero_btn_attack.addEventListener("click",function(){
    hero.attaquer(ennemie); // Passage par reference de l'ennemie dans la méthode attaquer
    Draw();                 // Mise à jour de l'affichage
});
```

Activité - Les boutons soin et attaquer de l’ennemie

Objectif

Lorsque l’utilisateur clique sur le bouton «ATTAQUER» de l’ennemie les points de vie du hero doivent baisser du nombre de point d’attaque de l’ennemie.

Lorsque l’utilisateur clique sur le bouton «SOIGNER» de *l’ennemie* les points de vie de *l’ennemie* doivent augmenter.

Mettez en place les boutons « ATTAQUER » et « SOIGNER » de l’ennemie

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

script.js

```
const ennemy_btn_soin = document.querySelector("#ennemy .btn_soin");
ennemy_btn_soin.addEventListener("click",function(){
    ennemie.soin(1);
     Draw();
});

const ennemy_btn_attack = document.querySelector("#ennemy .btn_attack");
ennemy_btn_attack.addEventListener("click",function(){
    ennemie.attaquer(ennemie);
    Draw();

});
```

Animer le *hero* lorsqu'il attaque

Objectif :

Effectuer une animation CSS sur l'image du *hero* lorsqu'il clique sur le bouton *ATTAQUER*.

L'animation voulu est un zoom.

> *Pour effectuer un zoom en CSS il faut utiliser l'attribut scale ce qui changera instantanément la taille de l'image. Pour pouvoir lisser l'animation on ajoute l'attribut CSS transition avec comme valeur scale.*

style.css

```css
/* Style d'animation */
img{
    max-height : 200px;
    transition: scale 0.25s;        /* Fluidifie l'animation d'attaque */
}

/* Classe css ajouter en JavaScript à une balise image lorsque un personnage attaque */
.attaquer{
    scale : 1.25;
}
```

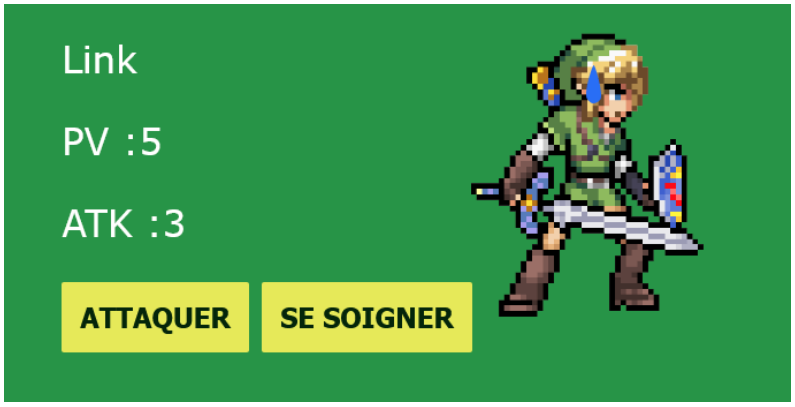
> *Pour ajouter la classe CSS « attaquer » à la balise <img> il faut utiliser la méthode HTMLElement.classList.add, elle prend en paramètre une string contenant le nom de la classe à rajouter.*

```javascript
const image = document.querySelector("#hero img"); // Je récupére la balise img
image.classList.add("attaquer");                    // J'ajoute la classe attaquer à la balise img
```

Précision

Pour supprimer une classe CSS il faut utiliser la fonction *HTMLElement.classList.remove*, où le premier argument est une string contenant le nom de la classe à retirer.

Activité - A l'aide ! Mon *hero* reste coincé !



Si vous avez coder l'ajout de la classe attaquer au clic du bouton vous allez voir votre hero grandir mais jamais revenir à sa taille normal. Il y a plusieurs solutions pour ce soucis, j'ai décidé de vous faire utiliser la fonction setTimeout.

window.setTimeout permet d'appeler une fonction après un certain nombres de millisecondes passés en paramètre.

Ajoutez le code suivant au bonne endroit pour faire revenir le *hero* à une taille normal 250ms après l'attaque.

script.js

```
setTimeout(function(){
       image.classList.remove("attaquer");
},250);
```

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution - A l’aide ! Mon *hero* reste coincé !

```
const hero_btn_attack = document.querySelector("#hero .btn_attack");
hero_btn_attack.addEventListener("click",function(){
    hero.attaquer(ennemie);
    const image = document.querySelector("#hero img");
    image.classList.add("attaquer");

    setTimeout(function(){
        image.classList.remove("attaquer");
    },250);

    Draw();
});
```

Activité - Animer l’ennemie lorsqu’il attaque

Objectif :

Animer l’ennemie lorsqu’il attaque le hero.

[APPELEZ MOI QUAND C'EST FINI ! :D]

ATTENTION - LA SOLUTION EST A LA PAGE SUIVANTE ! NE LA REGARDEZ PAS AVANT D'AVOIR REUSSI ;-)

Solution - Animer l’ennemie lorsqu’il attaque

```
const ennemy_btn_attack = document.querySelector("#ennemy .btn_attack");
ennemy_btn_attack.addEventListener("click",function(){
    ennemie.attaquer(ennemie);
    const image = document.querySelector("#ennemy img"); //Notez bien l'id html n'est pas le même
    image.classList.add("attaquer");

    setTimeout(function(){
        image.classList.remove("attaquer");
    },250);

    Draw();
});
```

Projet en autonomie

Objectif : Animer le soin du héro et le l’ennemie.

<u>En parfaite autonomie et sans aide de ma part vous allez devoir animer le héro et l’ennemie lorsqu’il se soigne.</u>

ATTENTION *je veux évidement une animation différente du zoom de l’attaque ;)*

Vous avez carte blanche vous pouvez animer l’image, les textes, le fond de couleur, changer de gif pendant l’attaque, changer le code source HTML, CSS, JS.

Vous êtes libre.S

Faite preuve d’imagination et bonne chance !