[Déroulement du projet 1](#_Toc160532338)

[Amorçage du projet 1](#_Toc160532339)

[Début du projet 1](#_Toc160532340)

[Partie 1.1 : Création des cibles 1](#_Toc160532341)

[Bugs : 2](#_Toc160532342)

[Partie 1.2 : Destruction des cibles 2](#_Toc160532343)

[Partie 2.0 : Introduction 2](#_Toc160532344)

[Partie 2.1 : Chronomètres 3](#_Toc160532345)

[Bug rencontrer : 3](#_Toc160532346)

[Partie 2.2 : Générer les cibles multiples 3](#_Toc160532347)

[a) Cibles restantes 3](#_Toc160532348)

[b) Génération des cibles 3](#_Toc160532349)

[c) Condition de victoire 4](#_Toc160532350)

[Les problèmes/bug : 4](#_Toc160532351)

[Bug rencontrer : 4](#_Toc160532352)

[Annexes 4](#_Toc160532353)

# Déroulement du projet

## Amorçage du projet

Lecture des objectifs à réaliser, visualisation de la vidéo.

Analyse des fichier fournis

Recherche des ressources que je pense être nécessaire à la création « from scratch » du projet.

Réflexion de l’orientation à prendre pour la réalisation du projet. Par quoi commencer.

Jouer avec l’environnements afin de comprendre les détails de ce qui a été fournis : différence entre les class et les id fournis, leurs utilités.

## Début du projet

Etant donné qu’il y a un bouton générant une cible « d’entrainement », c’est un moyen parfait de réaliser les différentes étapes du projet finales de façon contrôler. Pas 10 000 cibles à générer, pas de gestion de victoire, pas de chronomètre. Par contre, les fonctionnalités primaires sont présente : création, destruction des cibles.

Commençons par là.

## Partie 1.1 : Création des cibles

Je me dis que si j’ai une fonction qui génère UNE cible, je pourrais la réappeler lors de la création des cibles multiples.

Je commence donc une tentative de création de cibles. J’ai vu plutôt que les class target était déjà toute faite, il n’y a plus qu’as les générer en JS.

### Bugs

* Création de plusieurs cibles si on clique plusieurs fois sur le bouton [Annexe 1].

Ensuite il faut s’occuper des différents évènements pouvant avoir lieu sur la cible en elle-même.

D’après le sujet et la vidéo, j’ai cru comprendre qu’il y avait 3 événements a gérer :

* La destruction de la cible lors du clique (click) [Annexe 2]
* La modification de la class de la cible lors d’un mousehover. (Mouseenter, mouseleave) [Annexe 3-4]

Une fois que les cibles sont créées et les événements sont appliquer aux cibles, il va falloir les détruire lors du clique.

## Partie 1.2 : Destruction des cibles

Afin de détruire la cible précédemment crée il faut simplement sélectionner la cible sur laquelle on a cliqué et la supprimer.

Le problème est le suivant : sur la vidéo on voit bien que l’opacité est passée a 0 progressivement. C’est le comportement du CSS lors d’un opacity=1 qui passe a opacity=0. Or, cette même classe CSS (target on hit) est construit avec un z-index a 0, envoyant ducoup instantanément la cible derrière le terrain et la cachant. Je n’aime pas cette façon de gérer la disparition de la cible, elle peut causer des problèmes futurs : multitudes de cibles avec une class target présente même si invisible, etc…

Nous allons donc jouer avec un delay sur l’application de la classe afin de coller avec l’énoncer et ne pas modifier le CSS, mais nous allons ajouter un remove() tout de même histoire de supprimer la cible crée. [Annexe 5].

## Partie 2.0 : Introduction

La gestion d’une cible est faite. Théoriquement, il ne me faudrait pas grand-chose de plus afin de gérer plusieurs cibles en même temps.

La différence ici vas être de gérer :

* Le chronomètre
* La génération de cibles multiples
  + Récupération de l’input
  + Génération de plusieurs cibles
* Affichage des cibles restante en temps réel
* Fin du jeu
  + Condition de fin (cibles restantes == 0)
  + Affichage résultat final.

J’ai décidé de le réaliser dans cet ordre. Allons y.

## Partie 2.1 : Chronomètres

Ici il a fallu analyser ou été écrit les informations pour l’utilisateur. En l’occurrence nous avions 3 div avec des classes associées au minutes, secondes et dixième de seconde.

Pour les remplir il va falloir trouver une fonction gérant le temps. Date.now() est suffisant pour ce qu’on veut faire. **On récupère deux valeurs de temps que l’on va soustraire pour avoir la valeur du chronomètre** : une première valeur de la date au moment du clique sur le bouton démarrer, et une valeur lors de la réinitialisation du chronomètre. [Annexe 6]

Ainsi nous avons une valeur associée au temps passer depuis le clique du bouton (en milliseconde)

Afin de réinitialiser le chronomètre régulièrement, nous allons utiliser un setinterval et stocker son id dans une variable globale afin de l’arrêter quand nécessaire.

S’en suit un certain temps afin de comprendre exactement comment obtenir chacune de ces valeurs sans se planter ( milisecondes / 1000 = secondes, etc….).

Puis on rempli les div vue precedemment avec ces informations. [Annexe 7]

### Bug rencontrer :

* Le chronometre n’ajoute pas les 0 devant les chiffre <10 [Annexe 9]
* Le SetInterval n’est pas dans le bon scope pour le clearInterval finale
  + Solution : Passage en globale
* Problemes de comparaison des 2 variables de temps.
  + Solution : Un peu de temps et de logique

## Partie 2.2 : Générer les cibles multiples

Afin de faire les choses complètement et de surtout ne pas créer de bug sur la partie antérieur, j’ai préféré faire une 2nd fonction générant les cibles lors du clique de ce bouton. J’ai fait 2 parties distinctes : l’entrainement et la partie. C’est pourquoi il y a une autre fonction pour la gestion du bouton démarré, une autre fonction pour la génération des cibles.

(Enfaite c’est une erreur dans la vidéo qui m’as mis la puce à l’oreille : si la gestion des cible est identiques, on va décrémenté le nombre de cible restante dans l’entrainement. Ce n’est pas voulu dans la finalité, autant l’empêcher dès le début, j’ai choisit cette façon.)

### Cibles restantes

Tout d’abord nous allons récupérer l’input de l’utilisateur. Ainsi on pourra crée un nombre de cible égales a la valeur rentrer et nous pourrons aussi gérer l’affichage du nombre de cible restante. [Annexe 8].

### Génération des cibles

Ensuite, l’idée reste la même, mais nous allons enrober le code dans une boucle while qui s’arrêteras en fonction du nombre passer en argument. Ce nombre étant le nombre de cible voulu.

La plus grande différence avec la partie entrainement est que la valeur de cibles restante se décrémente de 1 a chaque fois qu’une cible est détruite. [Annexe 10]

### Condition de victoire

Cette fois, lorsque le joueur clique sur une cible, nous allons verifier si le nombre de cibles restantes est égale a 0. Si c’est le cas on peut lancer la séquence de victoire : Arret du chronometre, affichage du message de victoire. [Annexe 12]

### Les problèmes/bug :

* Les cibles ne se retire pas si on ne clique pas dessus = possibilité de régénérer des cibles par-dessus celle de la partie précédant
  + Rajouter une partie supprimant toute les cibles sur le terrain lors du clique sur le bouton [Annexe 11]
* Le time se réinitialise au clique du bouton, mais pas les 2 valeurs. Ce qui entraine un comportement indescriptible du chronomètre (affiche la différence entre 2 valeurs différentes en même temps)
  + Encore un peu de logique, ajout de Date.now() lors du clique du bouton au bonne endroit.

# Conclusion

05/03/2024 : J’écris cette conclusion après un premier jet après environs 10h de code, le programme est fonctionnel et l’ensemble des bugs rencontrer sont corriger, le fonctionnement est totalement contrôlé. Avant d’attaquer les points bonus, j’aimerais nettoyer mon code car autant je le comprends sans soucis, autant je comprends qu’il soit illogique sur certains points. Pourquoi ne pas réutiliser un maximum les fonctions que je peux réutiliser ? Pourquoi garder les console.log//commentaires stupides alors que c’est un rendu final ? etc…

# Bug rencontrer :

* Les dixièmes de second été enfaite les millièmes, ce qui affichais constamment 2 digits au compteur.
* Les 0 n’était pas ajouter au compteur devant les chiffres des nombres inferieur a 10.
* Target on hit ne s’applique pas correctement
  + La modification par le CSS est instantanée, passant le z-index en même temps que l’opacity=0. De ce fait, la cible n’est pas dégradée vers la couleur du ‘background color’ de target on hit.
* Chronomètre qui deconne si on appuis sur le bouton démarrer a la suite.
* Impossible de relancer le jeu si la partie est en cours
* Multi clique sur une seule cible.
* Le CSS ne s’applique pas en id mais en classe.
* Les cibles ne se retire pas si on ne clique pas dessus = possibilité de régénérer des cibles par-dessus celle de la partie précédante
  + Rajouter une partie supprimant toute les cibles sur le terrain lors du clique sur le bouton [Annexe 11]
* Le time se réinitialise au clique du bouton, mais pas les 2 valeurs. Ce qui entraine un comportement indescriptible du chronomètre (affiche la différence entre 2 valeurs différentes en même temps)
  + Encore un peu de logique, ajout de Date.now() lors du clique du bouton au bonne endroit.
* Le chronomètre n’ajoute pas les 0 devant les chiffres <10 [Annexe 9]
* Le SetInterval n’est pas dans le bon scope pour le clearInterval finale
  + Solution : Passage en globale
* Problèmes de comparaison des 2 variables de temps.
  + Solution : Un peu de temps et de logique
* Création de plusieurs cibles si on clique plusieurs fois sur le bouton [Annexe 1].

# Annexes

1. Gestion bug : max 1 cible par terrain.

    if (target.length == 0){

        targetCreator();

    } else {

        console.log("ERROR : 1 seul cible a la fois.");

    }

1. Evènement gérer : click cible

    target.addEventListener('click', function(){ //ajoute l'eventListener au la cible crée

        onClickTarget();

        clicked = true;

    });

1. Evènement gérer : mouseenter cible

    target.addEventListener('mouseenter', function(event) {

        target.classList.add("on");

    });

1. Evènement gérer: mouseleave

    target.addEventListener('mouseleave', function(event) {

        if (clicked == false){

            target.classList.remove("on");

        }

    });

1. Destruction de cible après clique.

//Fonction faisant disparaitre la cible une fois cliquer.

function onClickTarget(){

    var target = document.querySelectorAll(".target");

    if (target.length != 0){

        for (var i = 0; i < target.length; i++) {

            //target[i].setAttribute('class', 'target on hit');

            removeElementWithDelay(target[i], 300); //le z-index de .target.on.hit pas ce que je veux, on vas pas y toucher...

        }

    }

}

//Fonction permettant d'attendre le $delay avant de remove l'element. (utiliser pour le onClickTarget)

function removeElementWithDelay(element, delay) {

    element.setAttribute('class', 'target on hit');

    setTimeout(() => {

        element.remove();

    }, delay);

}

1. Comparaison des valeurs afin de générer le chronometre.

    currentTime = Date.now() //recupere le temps en milisec maintenant

    chronoTimer = currentTime - buttonTime;

1. Fonction remplissant les div du chronomètre

function updateTimer(buttonTime){

    currentTime = Date.now() //recupere le temps en milisec maintenant

    chronoTimer = currentTime - buttonTime;  //fait la difference entre le temps initial et le temps recup a l'instant pour avoir le temps durant lequel le chronometre a fonctionner

    //converti les valeurs en Int arrondis.

    var totalSeconds = Math.floor(chronoTimer / 1000);

    var minutes = Math.floor(totalSeconds / 60);

    var milisecond = Math.floor((chronoTimer % 1000) / 10);// =>Useless

    var tenth = Math.floor((chronoTimer % 1000)/100)

    //Ajoute les 0 manquant :

    totalSeconds = totalSeconds < 10 ? '0'+totalSeconds : totalSeconds;

    minutes = minutes < 10 ? '0'+minutes :  minutes;

    //rempli les span associé.

    document.getElementById("minutes").textContent = minutes;

    document.getElementById("seconds").textContent = totalSeconds;

    document.getElementById("tenth").textContent = tenth;

}

1. Récupération de l’input de l’utilisateur et affichage des cibles restantes

function onClickButtonStart(){

    var inputvalue = document.getElementById("nbtargets");

    var nbrtarget = inputvalue.value;

    var targetRemaining  = document.getElementById("remaining");

    targetRemaining.innerHTML = nbrtarget;

1. Ajout des 0 manquant dans le chronometre

    //Ajoute les 0 manquant :

    totalSeconds = totalSeconds < 10 ? '0'+totalSeconds : totalSeconds;

    minutes = minutes < 10 ? '0'+minutes :  minutes;

1. Gestion du panel « cibles restante »

                if (!event.target.classList.contains("hit")) {

                    targetRemaining.innerHTML = targetRemaining.innerHTML -1 ;

                }

1. Supprime toute les cibles du terrain pour rejouer.

    var target = document.querySelectorAll(".target");

    if (target.length != 0){

        for (var i = 0; i < target.length; i++) {

            target[i].remove();

        }

    }

1. Vérifie le nombre de cible restante // Lance la sequence de victoire

                if (targetRemaining.innerHTML == 0){

                    stopTimer();

                    win();

                }

function stopTimer(){

    clearInterval(intervalId);

}

function win(){

    var minute = document.getElementById("minutes").textContent;

    var seconds = document.getElementById("seconds").textContent;

    var tenth = document.getElementById("tenth").textContent;

    var finalTime = minute + "' " + seconds + "'' " + tenth;

    var winnerSentence = "Vous avez gagnez ! vous avez detruit toute les cibles en \n "+minute+"min "+seconds+ "sec " + tenth+"dizaine";

    alert(winnerSentence);

}

↓↑←↓↑→↓↑