Time Quest 42

Un projet réalisé par l'équipe :

Time's Masters

Composé de Aïssa Sohane, Danycan Dané, DiSanto Alexandre et Royer Julia.

Sommaire

I-Introduction

Pour cette deuxième partie du projet, il ne s'agit plus de savoir ce que nous allons produire mais comment nous allons le faire.
En effet, il est maintenant question de conception.

Il a fallu dans un premier temps trouver les outils (logiciels), indispensables à la création de Time Quest 42, et apprendre à les utiliser. Une fois le matériel connu, nous avons fait face aux premières contraintes, des contraintes imprévues, qui nous ont amenés à revoir nos exigences.

Nous avons pris conscience pendant cette conception que nous n'avions pas choisi le jeu le plus facile, en effet, peu de documentation, d'aide est disponible sur internet pour ce type de jeu, nous avons donc appris de nos erreurs, découvert des fonctionnalités en se perdant sur UNITY et internet.

Malgré cette phase de flou, qui nous a pas mal occupé, nous avons adoré travailler sur Time Quest 42, voir le jeu prendre forme, le premier personnage bouger, le premier "Build" sur UNITY réussi, le menu qui fonctionne.

Au-delà de l'aspect codage, la création d'un jeu demande tellement de ressources supplémentaires, il faut penser à tout, bien choisir la musique, les personnages, la bonne couleur de case…

Pour ce qui est du code, nous avions fait l'erreur de ne pas coder directement sur UNITY, mais en parallèle.
Nous l'avons beaucoup regretté quand il a fallu les utiliser, le code sur UNITY étant bien différent.

Ce deuxième rapport de soutenance va donc retracer l'intégralité de notre avancée de notre projet, comment nous avons réussi à réaliser les différentes tâches, quels ont été nos choix pour se faire et pourquoi.

I-Avancées par tâches

A/ Bande son et choix des personnages

Après la première soutenance, il nous semblait important de trouver les différents personnages ainsi que la musique, afin que le jeu devienne plus concret, réel.

Pour ce faire, nous avons voulu utiliser le logiciel BLENDER, qui permet de créer nous même nos personnages, en pixel art.

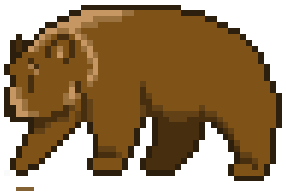
Mais après plusieurs tentatives, nous n'avons pas réussi à obtenir le personnage attendu.
Suite à ce constat, nous nous sommes lancés à la recherche de personnages libres de droits qui puissent coller un maximum avec nos attentes.

Voici donc les différents personnages que nous avons retenus :

Voici notre soldat, brave, beau, et vaillant.
C'est le premier personnage qu'aura chaque joueur en début de partie.

Pour différencier les deux équipes, elles n'auront pas les mêmes couleurs.

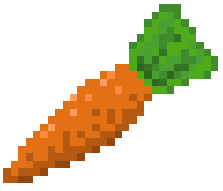
Voici notre barbare et notre ours, les deux ia de notre jeu :



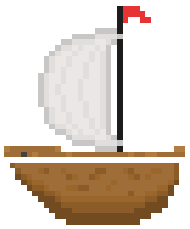
Notre roi, un cavalier ayant réussi à capturer un cheval (le personnage le plus fort).

Symbole des vies

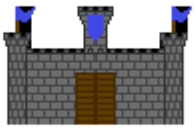
Ressources qui apparaîtront de manière aléatoire et éphémère sur le plateau.



Le bateau :



Le château :



Animation d'un crabe sur la plage :

Pour ce qui est de la musique, nous avons décidé, comme pour les personnages/objets, de chercher une musique libre de droit. (voir annexe).

Pour ce qui est des bruits/effets sonores du jeu, « La Sonothèque » et « Sound Fishing » ont été les deux principaux sites où nous sommes allés chercher nos bandes sons. Parmi ces sons on peut retrouver par exemple les grognements d'ours qui se trouveront sur le plateau, mais aussi les coups d'épées lorsqu'un combat sera engagé ainsi que les bruissements de l'eau quand un personnage sera sur un bateau.

B/ Modélisation de la carte

Pour la seconde soutenance nous avions prévu de finir la modélisation de la carte, des différents personnages et objets. Nous avons donc grâce au design des personnages précédemment choisis créé des Sprits (type de gameobject utile pour gérer les personnages en 2D) pour chacun d'entre eux : game objects manipulables par Unity.

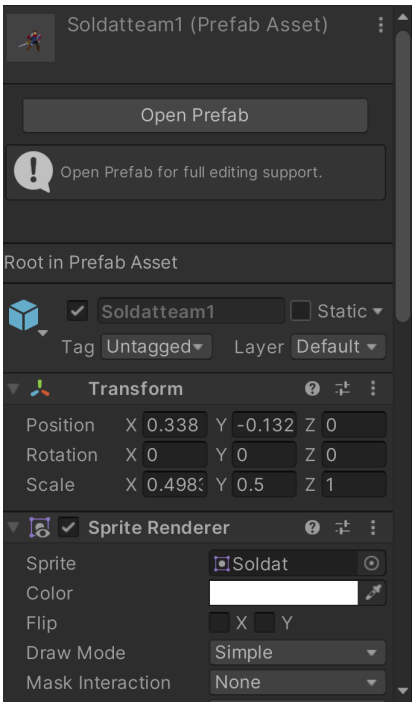
Pour chaque personnage nous avons dû adapter leur niveau d'importance pour éviter que les soldats des deux joueurs disparaissent sous les ours ou les carottes par exemple.

Pour les gameobjects des soldats nous leur avons attribué des collider (collisionneur), pour pouvoir cliquer dessus à l'aide de fonction prédéfini de UNITY, chose que nous verrons plus en détail dans une prochaine partie. Nous ferons de même pour les autres personnages des joueurs, et les autres objets afin que le joueur puisse choisir les interactions possibles avec chacun des objets présents sur la carte .

Pour la prochaine soutenance nous devrons alors ajouter le château et le bateau aux objets de la carte. Nous avons d'abord décidé de nous concentrer sur les petits objets pour éviter de s'éparpiller avec des objets encombrants et mieux comprendre les concepts que nous pourrons ensuite étendre aux autres objets.

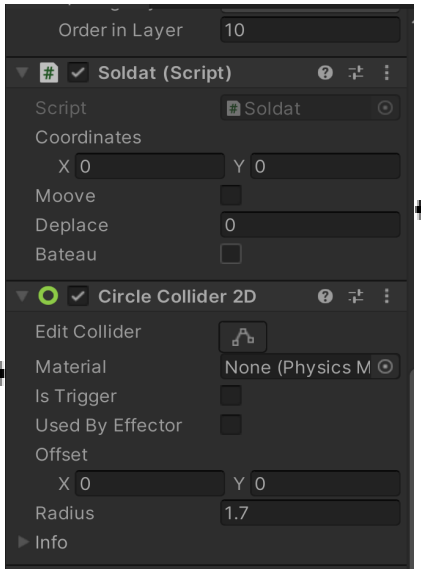
A chaque objet, nous avons également attribué un script pour pouvoir leur attribuer des fonctions et des attributs propres, chose plutôt utile pour les contrôler.

Voici un exemple de la composition d'un Gameobject avec celui du soldat :



Pour gérer la position dans l'espace de l'objet : taille, rotation …..

Ici on peut gérer l'aspect du sprite, on lui a attribué le design : soldat



Ici c'est le script du personnage, à chaque étape du jeu l'on peut suivre les attributs que nous lui avons attribués(Pour le moment)

Voici le collider pour gérer le clic souris

Voici un exemple de comment nous avons organisé la fonction permettant de tout faire apparaître sur la carte et de les organiser dans les premières structures :

```
//objets
//carottes
carottes = new Carotte[2];
GameObject carotte1 = (GameObject) Instantiate((GameObject) Resources.Load("perso/Carotte"));
GameObject carotte2 = (GameObject) Instantiate((GameObject) Resources.Load("perso/Carotte"));
carottes[1] = carotte1.AddComponent<Carotte>();
carottes[0] = carotte2.AddComponent<Carotte>();
carottes[1].transform.position = new Vector3(12, 5, 0);
carottes[0].transform.position = new Vector3(12, 19, 0);
```

Notre jeu contient 2 carottes, nous avons alors créé une liste de deux carottes. Ensuite nous créons les deux GameObject carotte à partir du bon sprits. Comme ces derniers ont encore le type GameObject et non Carotte, pour les rentrer dans les liste nous indiquons que carotte1 et carotte2 sont des carottes pour les mettre dans la liste ensuite, nous modifions leur position de départ pour qu'elles s'affichent correctement sur la carte.

Et c'est comme ça que pour chaque objet, personnage ou encore IA, nous les avons modélisés sur la carte.

C/ Menu principal/pause

Une fois la carte de début de partie prête, nous avons réalisé le menu principal/pause/action et nous sommes en train de finaliser l'inventaire de chaque personnage.

Pour tous ces menus, nous nous sommes appuyés sur les vidéos proposées par « Tuto Unity FR ».

La principale difficulté a été d'arranger le code proposé par le vidéaste pour l'implémenter à notre projet, plus particulièrement pour le menu d'action et l'inventaire.

Car en effet, la structure du code Csharp sur UNITY est très différente et demande des connaissances supplémentaires, par exemple pour les «GameObjects» ou les « Transforms » qui n'existent pas dans le code habituel.

Le premier sur lequel nous avons travaillé est le menu principal.

Après le visionnage de plusieurs vidéos (Voir annexe), nous avons pu recréer un menu "basique": avec les boutons "play" et "quit".

Nous envisagions déjà à ce moment-là de travailler sur la sauvegarde, nous avons donc rajouté un bouton "save" (que nous mettrons en marche plus tard).

Puis le bouton "setting", qui nous a pris le plus de temps, il fallait en effet faire le lien entre les différentes scènes, gérer les réglages du sons présents sur UNITY et le code.

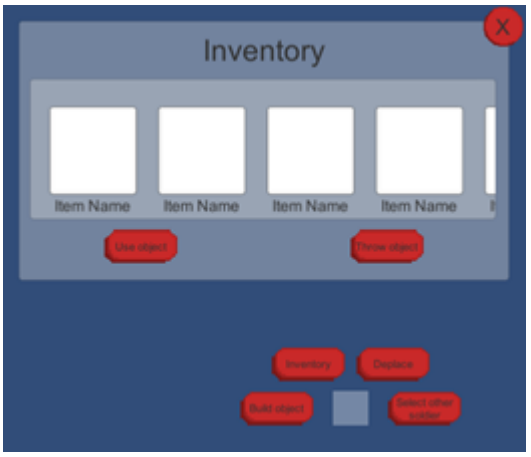
Pour les textures, nous avons choisi un thème médiéval, nous nous sommes donc tourné vers une texture papier-bois, texture qui rappelle d'après nous la nature, et qui nous fait penser à Times Quest 42.

Retrouvez le code qui nous a permis de mettre en place le menu dans les annexes.

Pour ce qui est du menu pause, il a été plus facile à faire, puisqu'il n'a fallu implémenter que trois boutons, "resume", "save" & "quit".

Nous sommes actuellement en train de finaliser l'inventaire ainsi que le lien entre celui-ci et le menu d'action. Pour l'inventaire, nous nous sommes aussi servis d'une vidéo de « Akbar Project » sur les menus défilants pour que cela ait plus de style soit plus agréable. Nous avons également cherché à changer la police d'écriture, nous n'y sommes pas encore parvenus mais cela ne saurait tarder.

Inventaire des personnages :



D/ Sauvegarde

Nous avons conçu ,dans le projet Csharp, une fonction qui écrit toutes les données de la partie sur un fichier texte tous les différents objets, personnages. Sans oublier tous les attributs qui vont avec : leur position, leurs apparences ou bien encore les scripts qui vont avec.

Il ne faut pas oublier le contenu des possessions de chaque joueur et du jeu contenu dans les listes.

Après pour les charger depuis le menu principal et donc pour les utiliser, nous avons codé une fonction qui lit dans ces fichier et qui donne à un jeu vidéo les attributs de la partie déjà enregistrée.

En effet les jeux de stratégies dans ce genre pouvant durer longtemps, nous nous sommes dit que cela pouvait être une bonne façon pour ne pas arrêter une partie quand on a un empêchement.

Mais pour utiliser ces fonctions qui écrivent les données et les lisent. Nous avons eu besoin d'une fonction qui crée ces fichiers.

Une fois que le jeu sera fini nous pourrons enfin les implémenter avec le reste, mais nous préférons pour l'instant nous consacrer à la finition de chaque étape du jeu et aux interactions avec le plateau, chose essentielle pour notre jeu.

E/ Moteur du jeu

Pour la seconde soutenance nous voulions finir de faire les fonctions à part sur un IDE comme rider ou visual studio. Nous nous sommes rendus compte que le Csharp Unity était assez spécial avec des spécificités à lui conçues pour le jeu. Et nous voulions aussi être en mesure de montrer quelque chose de concret pour cette soutenance intermédiaire.

La première chose que nous avons alors commencé à faire, c'est de mettre en place les personnages et objets dont nous avions déjà choisis leurs aspects pour nous en servir . Avant la seconde soutenance, nous avions par exemple décidé de faire une liste contenant tous les personnages quelque soit leur équipe ou leur type. Cette fois, n'ayant pas compris comment l'on faisait hériter d'une classe à une autre, nous avons décidé de faire des listes séparées pour les soldats, cavaliers mais aussi selon leur équipe. Cela pour plus facilement gérer leur design ou comment ils réagissent.

(voir annexe)

Nous avons ensuite fait en sorte de pouvoir sélectionner les personnages avec la souris, ceci ne fut pas la chose la plus évidente, en effet nous voulions d'abord faire cela grâce à la position du clic souris, mais nous sommes rendus comptes que celle-ci était transcrite en pixels, alors que celle des différents personnages était en unité.

Nous avons essayé de convertir mais malheureusement cela était trop compliqué et pas précis. C'est alors qu'avec plus de recherches nous avons trouvé un tuto qui nous a beaucoup aidé :

(Voir annexe)

Une fois le personnage sélectionné il ne restait plus qu'à le déplacer mais seulement le nombre de déplacements auquel il avait le droit. Ici trois comme nous avons d'abord testé avec avec le soldat. La fonction

déplacement et la Validposition n'étaient ensuite qu'une adaptation de ce que nous avions déjà fait sur rider.

Voici comment elles marchent :

```
public void Depalce(List<Soldat> equip)
{
    foreach (var soldat in equip)
    {
        if (soldat.moove == false)
        {
            if (Input.GetKeyDown(KeyCode.UpArrow))
            {

                if (Validsoldat(soldat.transform.position.x, soldat.transform.position.y + 1))
                {
                    Debug.Log("bougehaut");
                    soldat.transform.position = new Vector3(soldat.transform.position.x, soldat.transform.position.y + 1, 0);
                    soldat.deplace--;
                }
            }
            else if (Input.GetKeyDown(KeyCode.LeftArrow))
            {
                if (Validsoldat(soldat.transform.position.x - 1, soldat.transform.position.y))
                {
                    Debug.Log("bougehaut");
                    soldat.transform.position = new Vector3(soldat.transform.position.x - 1, soldat.transform.position.y, 0);
                    soldat.deplace--;
                }
            }
            else if (Input.GetKeyDown(KeyCode.DownArrow))
            {
                if (Validsoldat(soldat.transform.position.x, soldat.transform.position.y - 1))
                {
                    Debug.Log("bougehaut");
                    soldat.transform.position = new Vector3(soldat.transform.position.x, soldat.transform.position.y - 1, 0);
                    soldat.deplace--;
                }
            }
            else if (Input.GetKeyDown(KeyCode.RightArrow))
            {
                if (Validsoldat(soldat.transform.position.x + 1, soldat.transform.position.y))
                {
                    Debug.Log("bougehaut");
                    soldat.transform.position = new Vector3(soldat.transform.position.x + 1, soldat.transform.position.y, 0);
                    soldat.deplace--;
                }
            }
            else
            {

                //mettre un message d'erreur : Vous ne pouvez pas aller lÃ
            }

            if (soldat.deplace == 0)
            {
                soldat.moove = true;
            }
        }
```

Pour **Deplace**, elle prend en paramètres une liste de soldats. Elle va tous les parcourir jusqu'à trouver un soldat qui peut bouger. Comment sait-on qu'il peut bouger ?

Dans la classe soldat, on a une fonction qui change son statut, qui permet de le sélectionner :

```
void OnMouseDown(){

Debug.Log(this.gameObject.name);



    GameObject request = (GameObject) Instantiate((GameObject) Resources.Load("perso/action"));
    this.deplace = 3;
    request.name = "request";
    request.transform.position = new Vector3(-10,12,0);
    this.moove = false;
}
```

Lorsque l'on clique sur ce soldat, son paramètre moove devient vrai : il peut bouger.

Donc La boucle va détecter qu'on veut faire bouger ce perso, et tant que l'on ne l'a pas bougé 3 fois (c'est un soldat), on pourra choisir la direction qu'il doit prendre gauche/droite/devant/derrière/ grâce aux flèches directionnelles. Pour l'instant, l'on peut bouger plusieurs fois le même soldat dans un seul tour, pour la prochaine soutenance, nous ferons en sorte que ce ne soit pas possible.

Après certe le personnage peut bouger tout autour de lui d'une case, mais nous ne voudrions pas que ce dernier puisse sauter dans la lave ou bien se noyer dans l'eau sans bateau. C'est pourquoi **ValidPosition** est utile.

```
public static bool Validsoldat(float x, float y)
  {
      bool res = false;
      if ((x <= 24 && y <= 24) && (x >= 1 && y >= 1))
      {
          res = true;
          if (x ==24 || y==24 || x==1 || y==1)
          {
              res = false;
          }
          if(y<18 && y>5 && x==22)
          {
              res=false;
          }
          if(y>6 && y<17 && x==21)
          {
              res= false;
          }
          if(y>9 && y<14 && x==20)
          {
              res= false;
          }
      }
      return res;
  }
```

Cette fonction renvoie un bool : oui le personnage peut aller là, ou non il ne peut pas aller là. Cela bien-sûr grâce à sa position : deux float dans Unity, x et y , c'est pourquoi nous les mettons en paramètres. Ainsi dans Deplace, nous pouvons les faire bouger et décrémenter leur nombre de déplacements restant si Valid position renvoie true, sinon cela doit renvoyer un message comme quoi le joueur ne peut pas déplacer à cet endroit son personnage (nous ne l'avons pas encore créé).

Nous avons aussi eu l'occasion de mettre en place le moteur du jeu, c'est-à-dire le tour par tour, que nous agrémenterons au fur et à mesure

des différentes fonctions prévues . Mais désormais, le joueur 1 joue puis peut indiquer qu'il a fini son tour, ensuite c'est au joueur deux, et comme ça pendant 42 tours.

Voici donc la fonction Update qui permet de mettre en place le système de tour par tour :

```
void Update()
{
    if (Turn < 42)
    {
        if (TurnEquip1)
        {
            Depalce(soldats1);
        }

        if (TurnEquip2)
        {
            Depalce(soldats2);
        }
    }

}
```

Ici nous pouvons voir qu'elle va juste appeler Déplace pour les soldats de l'équipe 1 tant que TurnEquip1 est vraie. Ou Déplace pour les soldats de l'équipe 2 si TurnEquip2 est vrai. Pour permettre d'abord au joueur 1 de jouer on initialise le bool correspondant à son tour à vraie et TurnEquip2 à false.

C'est pourquoi pour faire passer TurnEquip1 à false et TurnEquip2 à vraie, nous avons mis en place le Bouton FinTour et son script :

```
public void Fintour()
{
    if (TurnEquip1)
    {
        TurnEquip1 = false;
        TurnEquip2 = true;
    }
    else if (TurnEquip2)
    {
        TurnEquip2 = false;
        touria();
        SpawnObject();
        Turn++;
        TurnEquip1 = true;
    }
}
```

Quand le joueur 1 cliquera sur ce bouton fin tour, ça passera TurnEquip1 à false et permettra au joueur 2 de jouer en passant TurnEquip2 à true. et quand les deux joueurs auront déclarer la fin de leur tour, on incrémente le nombre de tours de 1, et on repasse les bool TurnEquip1 et TurnEquip2 à leur état initial.

Nous avons ensuite pu faire la liaison avec les différents menus : principaux et de pause. Et faire la vidéo pour le site web :

(Voir annexe)

Nous avons également commencé à regarder comment marchait l'hérédité sur unity pour simplifier notre code.

F/ Site web

Nous n'avons pas eu besoin de travailler beaucoup sur le site web, en effet dès la première soutenance nous avions déjà découvert comment gérer le css et l'html pour en faire la création.

Un menu avait déjà été fait avec différentes parties : état de l'art, origine et le jeu sont déjà remplis, ils contiennent les règles de notre jeu ainsi que là d'où il vient.

Nous avions fait les onglets Tuto qui reste à remplir où nous aimerions afficher une vidéo pour expliquer le but du jeu et comment y jouer.

L'onglet Installation lui servira quand notre jeu sera abouti, ou nous mettrons en place le système d'installation du fichier.exe pour pouvoir jouer au jeu depuis n'importe quel pc sous linux.

En lisant le document sur ce qu'il devait contenir, nous nous sommes rendus compte qu'il manquait des choses, notamment un onglet Avancement que nous avons ajouté.

Celui-ci contient un résumé de nos péripéties dont la vidéo présentant ce qui a été fait jusqu'à maintenant concrètement que nous mettrons à jour à la prochaine soutenance.

Et enfin nous l'avons mis en ligne grâce à un hébergeur gratuit.

Ainsi, pour la prochaine soutenance, nous n'aurons plus qu'à mettre le système d'installation que ce soit du jeu ou bien des différents fichiers demandés pour que le site web soit finalisé.

(voir annexe pour obtenir le lien du site web)

IV-Conclusion

A/ Tâches à effectuer pour la prochaine fois

Nous attaquons maintenant la dernière phase de notre jeu.

Pour la soutenance finale, Time Quest 42 devra être entièrement fini. Pour ce faire, nous allons devoir nous concentrer sur les interactions entre les personnages et le plateau.

C'est à dire qu'il nous est encore nécessaire de pouvoir ramasser des carottes ou des cannettes de coca. Ou bien alors d'attrapper des chevaux ou d'attaquer des ennemis, qu'ils soient IA ou bien de l'équipe adverse. nous aurons alors de nouveaux attributs à donner à chaque classe et devrons rajouter la classe cavalier et la classe roi qui ne sont pas encore là.

Cela veut dire que nous devrons être capable de créer de nouveaux personnages à la demande du joueurs si et seulement s' il a les ressources pour le faire. Donc nous devrons implémenter un système d'argent .

Cela veut aussi dire que nous allons devoir gérer l'affichage des ressources/ des vies, pour chaque personnage, adapter les fonctions déjà faites sur UNITY, et coder les autres.

Nous avons réussi à mettre en place la notion de tours par tours mais reste un problème : celui d'empêcher les personnages qui ont déjà réalisé des actions à ce tour de rebouger.

En effet, rien ne nous empêche avant de cliquer sur le bouton “end of your turn” de recliquer sur le soldat pour le faire agir encore et encore sans que l’autre joueur ne puisse rien faire.

En plus de cela, il va falloir implémenter les IA pour qu’ils puissent attaquer les personnages et se déplacer vers eux lorsqu’ils sont proches.

Nous allons aussi devoir mettre en place la possibilité d’acheter un bateau pour naviguer sur l’eau. Pour pouvoir pêcher du poisson, mais aussi traverser plus vite la carte sans rencontrer d’Ia.

Nous n’avons pas encore créé les poissons, nous devrons donc aussi s'occuper d’eux pour qu’ils soient disponibles dans la mer .

Ainsi nous pourrons associer les bruitages que nous avons commencé à réaliser aux bons endroits ainsi que la bande son.

Une fois tout ça fini nous pourrons alors mettre en place correctement le système de sauvegarde pour permettre au joueur de faire une pause.

Ainsi nous pourrons implémenter le téléchargement sur notre site web.

Malgré qu’il reste encore pas mal de choses à réaliser, nous avons réussi à prendre en main nos outils et aussi nous savons comment chercher les ressources dont nous avons besoin sur internet : tutos, personnages, objets ….

Finalement, ces tâches sont faisables dans les semaines à venir, nous ne sommes pas spécialement en retard sur notre planning.

B/ Annexes

Lien du site web :
https://timequest42.github.io/TimeQuest42/

Choix musique :
https://www.musicscreen.be/musique-libre-de-droit/Catalogue/L-erreur.html

Choix personnages :
https://sventhole.itch.io/

Création du menu :
https://www.youtube.com/watch?v=4LkiX_XioXg
https://www.youtube.com/watch?v=ABWt1ipTAMg&t=948s
https://www.youtube.com/watch?v=GURPmGoAOoM

Modélisation de la carte :
https://www.youtube.com/watch?v=MOehmikBox8&list=PLHpC3gVKYuvfcxrThC4arYTjOyU2mKWH8

Implémentation de la souris :
https://www.youtube.com/watch?v=VN1gryMOpWo&t=591s

Sauvegarde :
TP5-Registre de Dumbledor (cours de programmation)
https://drive.google.com/file/d/1l-_4uM_3njsA0sxw1Ad34xFuDxNRs0iG/view?usp=sharing

Vidéo d'avancement :
https://youtu.be/SufTeJ7d4Ug