Documentation pour le Projet Chamois&Toi

SOMMAIRE

I) SCÈNES

Premièrement, le jeu est composé de 3 scènes, “Menu”, “Game” et”StarWars”. Pour lancer le jeu, il faut passer par “Menu” car il nous permet de choisir le rôle et donc d’initialiser les paramètres comme il le faut.

Dans l’architecture du projet, pour accéder à ces scènes, se diriger vers :

Assets > Scenes > Menu > Menu (pour la scène du menu)
Assets > Scenes > Jeu > Game (pour la scène du jeu)
Assets > Scenes > Jeu > StarWars (pour la scène du générique des crédits)

II) SCENE MENU

1) *Le MainMenu*

“Menu” est composée d’un Objet “Canvas”.
Cet objet contient l’image de fond : “Background”, qui contient aussi le titre du jeu.
Il contient aussi un GameObject nommé “MainMenu”, qui contient les 3 boutons permettant de jouer, de lancer les options, ou encore de lancer le générique.

2) *Le ”Bouton Play”*

Utiliser le bouton “Play” activera le menu de sélection de personnages, le boutons “Options” ouvrira les options, enfin, le bouton générique lancera la scène nommée “StarWars” permettant d’observer les crédits.

“Play” va donc activer SelectionPersonnages qui contient l’objet “Personnages” contenant 3 boutons : Chamois, Chasseur et Randonneur.
Cliquer sur un de ces boutons va appeler la fonction OnClickButton de l’objet EventSystem en transmettant le nom du rôle choisi.

Cet objet contient un script “OnButtonPress”, demandant un nom de scène, ici Game. Quand on clique sur un des rôles, on va donc passer sur la scène “Game” avec le paramètre du personnage choisi.

3) *Le Bouton “Options”*

Le Bouton “Options” amène le joueur vers les options de jeu, lui proposant de régler le volume, ainsi que 3 options à cocher, permettant d’activer / désactiver les aides de jeu, les effets sonores et les vibrations.
Ces options sont stockées dans les PlayerPrefs.
De base, quand on lance l’application pour la première fois, le son est à 100% et toutes les options sont cochées. Si le joueur choisit de les changer, ces changements seront sauvegardés pour les prochains lancements du jeu.
Ces valeurs sont récupérées et enregistrées dans les PlayerPrefs dans le fichier StockOptions.cs, attaché au GameObject “OptionsMenu”.

4) *Le Bouton “Generique”*

Enfin, “Menu” contient le bouton “Generique”, permettant de lancer la scène “StarWars”.

III) SCENE STARWARS

Ce qu'il faut savoir sur cette scène est très simple : Dans l'objet CanvasTexte > TextCredits, existe un script nommé CreditsScroll.cs proposant une variable nommée "Speed" qui va permettre au texte de défiler vers le haut à la vitesse (speed) définie dans l'inspecteur.

La rotation du texte permettant de faire l'effet "star wars" est dû à la rotation x du rectTransform de l'objet "TextCredits".

IV) SCENE GAME

1) *Organisation des catégories*

Tous les objets de jeu ont étés classés, pour plus de clarté, dans de grandes catégories étant des GameObjects de type “empty”

Voici comment sont organisées ces catégories :

>> Brouillard de Guerre
>> Carte de Jeu
>> Haut Faits
>> Personnages Non Joueurs
>> Guide de Jeu
>> Joueur
>> Menu de Jeu
>> Carte
>> Encyclopédie
>> Managers
>> Décor
>> Cameras
>> Système de Dates
>> Système de Jour & Nuit
>> Autres
>> Autres Informations

2) *Brouillard de Guerre*

a) *Adapter le brouillard de guerre à la carte*

Pour cette fonctionnalité, si la carte vient à changer de taille ou de proportions, ce qu’il faut faire c’est d’adapter la taille des caméras “FogMainCamera” et “FogSecondCamera” ainsi que la taille de l’objet “FogOfWar”, se trouvant dans “FogOfWarCanvas”, aux contours de la carte de jeu (“OptimisationWorld”).
Les deux caméras sont placées au centre de la carte, le fait de modifier le paramètre “size”, adaptera facilement les caméras à la taille de la carte (il faudrait de préférence garder une carte de taille carrée afin d’éviter la déformation du brouillard de guerre.

b) *Cercles de clarification du brouillard de guerre*

Le Chamois et le Randonneur sont exposés à ce brouillard de guerre. Ils possèdent tous deux 2 cercles permettant de clarifier le brouillard de guerre et de définir le brouillard de guerre déjà découvert mais hors de portée.

Ces éléments se trouvent dans : Game > Joueur > Players > PlayerChamois / PlayerRandonneur > Sprites
Leurs noms sont : FogOfWarMainCircleChamois, FogOfWarMainCircleChamois2, FogOfWarMainCircleRandonneur, FogOfWarMainCircleRandonneur2.

3) *Carte de Jeu*

a) *Collider boundaries et fonctionnalité d'optimisation de la carte*

Carte de Jeu contient un préfab "OptimisationWorld" qui correspond ni plus ni moins au fond de carte.
Ce dernier contient un objet "ItemActivatorObject" composé de 4 Box Colliders 2D étant les colliders de bord de map.
Le GameObject contient aussi de multiples images, correspondant au morceaux de carte.
Ces derniers sont placés les uns collés aux autres et sont cachés ou bien montrés en fonction de la proximité du joueur de ces dernières, grâce au script "DisableIfFarAway.cs" que contiennent chaque images.

b) *Ajout / modification des images composant la carte*

Pour ajouter une nouvelle "case" vous pouvez dupliquer une des cases déjà existantes, puis la placer (à la main) à côté des autres.

Dans le cas où l'on veut déplacer les morceaux de carte, pas besoin de déplacer les cases, changez juste les sprites contenus dans les cases.
Ces images se trouvent dans Assets > Resources > Image > IMGCarte > imgCarte.

c) *Éléments de décor*

OptimisationWorld contient aussi un objet "elementJeu", qui contient tous les éléments de décor, comme les arbres, les roches, etc...
Il faudrait que chaque objet de décor contienne un composant "Sprite Mask" afin que la surbrillance derrière les objets puisse être appliquée.

d) *Caméras de joueur*

Enfin, le préfab est composé d'un GameObject "CameraReg", contenant les caméras pour chacun des 3 rôles jouables.

4) Haut Faits

Haut Faits contient un GameObject “AchievmentMenu” qui correspond aux éléments visuels du menu des haut faits en jeu.

Il contient aussi un GameObject “AchievmentManager” auquel est attaché un script AcheivmentManager.cs
Ce script demande une liste d’images correspondant aux images qui seront attachées à un haut fait.
Il demande aussi un “JsonFile” qui correspond au fichier JSON qui contient les informations de tous les hauts faits.

a) Récupérer les données des hauts faits

Au lancement de la partie, une fonction Start() du script AchievmentManager.cs récupère les informations dans le JSON et crée automatiquement des objets de préfab de type “Achievment” qui sont automatiquement ajoutés dans les bonnes catégories (Chasseur, Chamois et Randonneur).

b) Insérer un nouveau Haut Fait dans le fichier de stockage

Dans les cas où un Haut Fait doit être ajouté, il faut, dans le fichier “AchievmentInfo.json”, ajouter un nouvel élément.
L’élément “joueur” doit contenir “Chamois”, “Randonneur” ou bien “Chasseur”, en fonction du rôle auquel on veut ajouter le haut fait.
L’élément “nomAch” correspond au titre du haut fait, comme par exemple “Survivre II”.
L’élément “descAch” correspond à la description plus ou moins détaillée du haut fait créé.
L’élément “points” correspond au nombre de points de haut fait que rapportera l’accomplissement du haut fait.
L’élément “image” correspond au numéro d’index de la liste d’image insérée dans la liste d’images du script attaché à l’objet “AchievmentManager”
Enfin, les éléments “dependance x” correspondent aux hauts faits nécessitant d’être accomplis afin que le haut fait décrit soit accompli aussi.
Par exemple, pour accomplir le haut fait “Terminer le jeu” on pourrait par exemple devoir au préalable avoir accompli les haut faits “Terminer une chasse”, “Terminer une Randonnée” et “Terminer une partie de chamois”. Le haut fait sera automatiquement accompli lorsque ces 3 conditions sont remplies.

c) ***Gérer l'obtention du haut fait***

Mais créer le haut fait n'est pas suffisant pour pouvoir l'obtenir. Il faut ajouter une ligne de code,
AchievmentManager.Instance.EarnAchievment("nomDeL'achievment) :

```
if (!score1000 && carteActive == false)
{
    if (score > 999)
    {
        AchievmentManager.Instance.EarnAchievment("Score Chamois I");
        score1000 = true;
    }
}
```

Cette ligne peut être utilisée dans n'importe quel script, mais de préférence, il faudrait effectuer les détections d'accomplissement de haut faits dans les fichiers suivants :

- DataStorer.cs pour le Chamois
- DataStorerRandonneur.cs pour le Randonneur
- DataStorerChasseur.cs pour le Chasseur

Voici comment il serait conseillé de l'utiliser :

```
// Update is called once per frame
void Update()
{
    carteActive = GameObject.Find("Game Map").GetComponent<SwitchPlayerMap>().isActive;

    if (PersonageJoue.Personnage == "Chasseur")
    {
        if (!quete1 && carteActive == false)
        {
            if (nbQuetes > 1)
            {
                AchievmentManager.Instance.EarnAchievment("Quête Chasse I");
                quete1 = true;
            }
        }
```

d) Tester le rôle joué

Il faut donc premièrement bien s'assurer du rôle qu'on joue avec :

```
if (PersonageJoue.Personnage == "Chamois")
```

```
if (PersonageJoue.Personnage == "Chasseur")
```

```
if(PersonageJoue.Personnage == "Randonneur")
```

(On peut évidemment utiliser un switch(PersonageJoue.Personnage)).

(Il faut garder en mémoire que cette ligne est très importante partout dans le code, quand on développe une nouvelle fonctionnalité bien avoir cette idée en tête : cette fonctionnalité est-elle unique à un rôle seulement (ou deux), les objets en question dans cette partie de code sont-ils actifs seulement pour un ou deux rôles ?
→ Dans ce cas, il peut être judicieux d'utiliser cette ligne.
Il faut aussi savoir que cette ligne a souvent corrigé des erreurs survenant après avoir oublié de se poser ces questions).

e) Activation de la carte

Il faut vérifier si la "mini map" est active, car comme cette dernière change de caméra et désactive donc des éléments. Comme on place la détection dans un Update(), cela peut engendrer des erreurs. Ici, avec le boolean **carteActive.**
Il faut aussi vérifier si le haut fait n'a pas déjà été obtenu afin d'éviter de le gagner de multiples fois, ici avec le boolean **quete1**.

On vérifie si la condition de complétion est remplie et ensuite on fait gagner le haut fait.

5) Personnages Non Joueurs

a) Le prefab "predateurs"

La catégorie predateurs contient une liste de prefabs correspondant à la faune dangereuse pour le chamois. Un prédateur peut être ajouté en ajoutant à cette catégorie de prédateurs le prefab "predateurs".
Ce prefab contient un prefab "ennemi" ainsi qu'une liste de points appelée "Patrol Points", chaque point peut être placé où bon nous semble.
Le prefab de "l'ennemi" contient 3 scripts : Enemy_Degat, Patrol et Killable.

b) Paramètres du prédateur

Enemy_Degat va nous permettre de modifier la vitesse de déplacement, les dégâts de saignement, les dégâts de base d'une attaque, le score que retire une attaque au joueur chamois (variable Sc), le temps que dure une griffure / morsure / morsure grave (les 3 types de saignement aléatoires)

c) *Mouvements aléatoires de la faune*

Patrol est le script qui permet de gérer les mouvements aléatoires de la faune PnJ. Ce qui est important à savoir sur ce script est que la variable "Chase Radius" détermine à partir de quelle proximité avec le joueur ce dernier va l'attaquer. Aussi, il faut glisser, dans la variable path, tous les points de patrouille du PNJ (ceux placés dans "Patrol Points"). "Rounding Distance" détermine l'approximation d'approche du point de patrouille, afin qu'une fois que le prédateur est aux environs du point de destination, il choisisse automatiquement un autre point de destination.

d) *Le script "Killable.cs"*

Enfin, le script "Killable" nous permet juste de savoir si la cible est possible d'être tuée par la balle d'un chasseur.
Ce script demande un string, qui n'est cependant pas obligatoire, mais on peut définir dans celui-ci le type de cible abattue, ce qui va notamment nous être utile si la cible abattue est une cible importante pour le déroulement de l'aventure : par exemple, pour un Pnj de quête, on mettra dans ce champ "PnjImportant", et si cette cible est tuée, le joueur sera prévenu (si cela est activé dans les options) qu'il a abattu un PnJ de quête.

e) *CircleCollider2D de stress*

Aussi, chaque prédateur possède un CircleCollider2D en isTrigger, qui va permettre de faire augmenter le stress du chamois si ce dernier se trouve dans ce cercle, sans que le chamois soit forcément dans la zone d'agression du prédateur. (Si cette zone doit être modifiée, changez la valeur du radius).

f) *Le préfab "proies"*

La catégorie proies contient la liste de préfabs correspondant à la faune non agressive, qui va même fuire les humains.
Chacun de ces éléments contient donc un script "Prey" dans lequel on définit la vitesse de l'objet, ainsi que la distance à laquelle l'objet va fuir (variable Chase Radius).
Comme les prédateurs, les proies contiennent le script Killable, leur permettant d'être abattues.

g) Personnages de "discussion"

L'objet NPC Collection contient tous les personnages pouvant avoir une conversation avec le joueur.
A l'intérieur se trouve un autre GameObject appelé "New NPC" qui contient les PnJ donnant les infos n'ayant pas d'utilité dans la quête principale (comme par exemple ceux qui posent des questions de connaissances), ainsi que les GameObjects contenant les Checkpoints des randonnées. (New NPC pourrait ne pas exister et on pourrait mettre les objets contenus à l'intérieur dans proies, mais on a choisi de les mettre dans un GameObject précis car selon nous, ils ne devraient pas être changés comparés aux autres).

h) Créer un nouveau personnage

Afin de créer un nouveau NPC, simplement dupliquer un NPC.

i) Gestion de la discussion avec les PnJ

Voici comment on gère les conversations avec les NPC :

Premièrement, on trouvera les scripts contenant les quêtes et les informations dans
Assets > Script > Jeu > PNJ > PNJQuete

Dans ces scripts, on déclare tous les NPC qui ont une action dans cette "quête"
Ces scripts sont attachés au GameObject "NPC Collection", il faudra glisser les éléments PNJ ayant des discussions dans l'inspecteur.

```
public GameObject infosDeplacementRandonneur;
public GameObject infosBrucellose;
public GameObject infosVitesseSentiers;
public GameObject infosChallengeChamois;
public GameObject infosBarriereComportementale;
public GameObject infosAlimentationEspeces;
public GameObject infosPlantesBuffet;
public GameObject infosVigilance;
public GameObject infosDeplacements;
public GameObject infosGestationChamois;
public GameObject infosVitesseCourse;
public GameObject infosDureeVie;
```

On déclare les fichiers JSON grâce auxquels nous allons récupérer les informations des conversations. (ici jsonFile et jsonFileEncy)

```
public TextAsset jsonFile;
public TextAsset jsonFileEncy;
public List<ConversationInfos> data;
public List<EncyInfo> data2;
```

j) Récupérer les données du JSON

Et on récupère les informations dans les JSON :

```
void Start()
{
    ConversationInfosList infosInJson = JsonUtility.FromJson<ConversationInfosList>(jsonFile.text);
    data = new List<ConversationInfos>();

    foreach (ConversationInfos convinfo in infosInJson.convinfos)
    {
        data.Add(convinfo);
    }


    EncyInfoList infosInJson2 = JsonUtility.FromJson<EncyInfoList>(jsonFileEncy.text);
    data2 = new List<EncyInfo>();

    foreach (EncyInfo encyinfo in infosInJson2.encyinfos)
    {
        data2.Add(encyinfo);
    }
```

(on récupère en premier les informations des conversations, puis de l'encyclopédie).

k) Code de la discussion avec un PnJ

Voici comment gérer les discussions avec les pnj donnant les informations quelconques (ceux qui posent les questions par exemple) :

```
//infosDeplacementRandonneur;
ConversationOption o1 = new ConversationOption() { text = data[0].branche1Reponse, targetId = data[0].branche1ID };
ConversationOption o2 = new ConversationOption() { text = data[0].branche2Reponse, targetId = data[0].branche2ID };
ConversationPiece c1 = new ConversationPiece() { id = data[0].ID, text = data[0].texte, options = new List<ConversationOption>(), hint = data[0].hint };
c1.options.Add(o1);
c1.options.Add(o2);
infosDeplacementRandonneur.GetComponent<ConversationScript>().Add(c1);
ConversationPiece c2 = new ConversationPiece() { id = data[1].ID, text = data[1].texte, options = new List<ConversationOption>(), hint = data[1].hint };
infosDeplacementRandonneur.GetComponent<ConversationScript>().Add(c2);
ConversationPiece c3 = new ConversationPiece() { id = data[2].ID, text = data[2].texte, options = new List<ConversationOption>(), hint = data[2].hint };
infosDeplacementRandonneur.GetComponent<ConversationScript>().Add(c3);
infosDeplacementRandonneur.GetComponent<ConversationScript>().OnAfterDeserialize();
```

Dans cet exemple, on gère la conversation du PnJ :
Ici, ConversationPiece c1 définit ce que dit le PnJ au départ, ConversationOption o1 et o2 définissent les réponses proposées au joueur.
On ajoute les options de conversation à la pièce de conversation avec c1.options.Add(o1).
On ajoute ensuite la pièce de conversation au personnage.

Comme on a 2 réponses possibles, on crée 2 autres pièces de conversations correspondant à ce que nous dira le PNJ en fonction de notre réponse.

Une fois que toute la conversation est traitée, on doit désérialiser la conversation.

l) Stockage des informations de discussion

Pour ce qui est des informations tirées du Json, voici comment sont stockées les données :


```
{
  "_comment": "DonneurInfo1 id0",
  "ID": "1",
  "hint": "",
  "texte": "Eh toi !?",
  "branche1ID": "1.1",
  "branche2ID": "1.2",
  "branche3ID": "",
  "branche1Reponse": "Oui ?",
  "branche2Reponse": "Non...",
  "branche3Reponse": ""
},
{
  "_comment": "DonneurInfo1 id1",
  "ID": "1.1",
  "hint": "deplacementRandonneur",
  "texte": "Nous, les chamois, on adapte nos déplacements en fonction de la présence de randonneurs",
  "branche1ID": "",
  "branche2ID": "",
  "branche3ID": "",
  "branche1Reponse": "",
  "branche2Reponse": "",
  "branche3Reponse": ""
},
{
  "_comment": "DonneurInfo1 id2",
  "ID": "1.2",
  "hint": "",
  "texte": "Reviens !?",
  "branche1ID": "",
  "branche2ID": "",
  "branche3ID": "",
  "branche1Reponse": "",
  "branche2Reponse": "",
  "branche3Reponse": ""
},
```


Ci-dessus se trouvent les informations des 3 pièces de conversation données en exemple précédemment.
Chaque pièce possède un ID, ainsi que des ID de branchage, qui correspondent à l'ID de la pièce suivant le choix effectué.
Dans la première pièce, si on répond "Oui ?", on fait appel à la pièce d'ID "1.1", ce qui va faire avancer la discussion du PNJ.

m) L'utilisation du message "hint"

On peut voir que si on répond oui, la pièce d'ID "1.1" possède une variable nommée "hint" qui n'est pas vide.
Cette "hint" signifie que nous avons fait les bons choix afin d'obtenir l'information que l'on peut avoir grâce à ce PNJ.
Elle va notamment nous permettre, si on retourne parler au PnJ après avoir reçu son information, d'avoir une autre discussion avec le PNJ, car il est, selon nous, inintéressant d'avoir la même discussion avec un même PnJ.

Mais comment cette "hint" fonctionne-t-elle ?

Simplement, cette hint fonctionne comme une sonnette d'alarme, dans le fichier ShowConversation.cs.
Dans la fonction Execute() de ce fichier, il existe 3 switch, en fonction du rôle joué.

```
if (PersonageJoue.Personnage == "Chamois")
{
    /// <summary>
    /// Voir le script chamoisInfosPNJ
    /// </summary>
    var script = collec.GetComponent<chamoisInfosPNJ>();


    switch (ci.hint)
    {
        case "deplacementRandonneur":
            script.deplacementRandonneur();
            GameObject.Find("PlayerChamois").GetComponent<DataStorer>().nbInfos += 1;
            break;
        case "brucellose":
            script.brucellose();
            GameObject.Find("PlayerChamois").GetComponent<DataStorer>().nbInfos += 1;
            break;
        case "vitesseSentiers":
            script.vitesseSentiers();
            GameObject.Find("PlayerChamois").GetComponent<DataStorer>().nbInfos += 1;
            break;
```

Comme on peut le voir dans le screen du JSON, on a une "hint" : "deplacementRandonneur".

Quand le switch détecte cette hint, on appelle la fonction deplacementRandonneur() du script chamoisInfoPNJ.cs que voici :

```
public void deplacementRandonneur()
{
    ency.addInfoToList(data2[0].hint, ency.pagesDynamic);
    //SELF
    infosDeplacementRandonneur.GetComponent<ConversationScript>().items.Clear();
    ConversationPiece c = new ConversationPiece() { id = data[48].ID, text = data[48].texte, options =
        new List<ConversationOption>(), hint = data[48].hint };
    infosDeplacementRandonneur.GetComponent<ConversationScript>().items.Add(c);
    infosDeplacementRandonneur.GetComponent<ConversationScript>().OnAfterDeserialize();
}
```

Ce qui mettra à jour la discussion du PNJ.

Pour le déroulement des quêtes, comme par exemple celle du chasseur, c'est exactement la même méthode qui est utilisée :

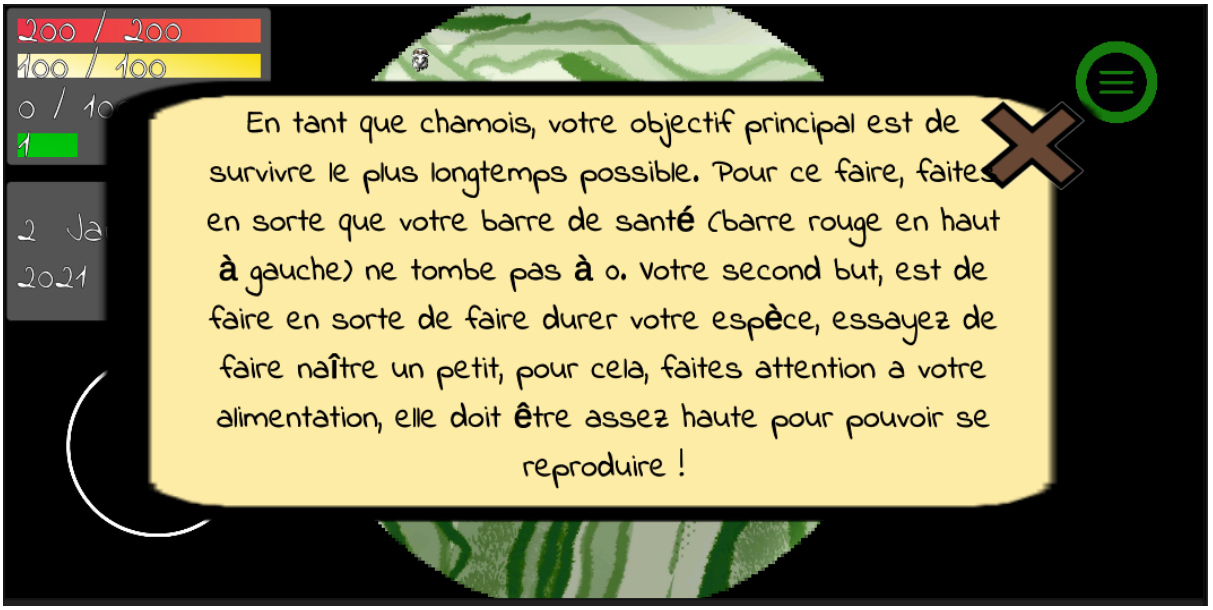
Dans le Start(), on met la discussion de base de chaque PNJ, et quand on a parlé au premier PNJ de quête, le hint fera mettre à jour la discussion de ce PNJ ainsi que du PNJ suivant dans la quête :

```
public void donnerQuete()
{
    ency.addInfoToList(data2[0].hint, ency.quete);
    //GameObject.Find("GameControl").GetComponent<GameControlScript>().setTrue();
    //SELF
    donneurDeQuete.GetComponent<ConversationScript>().items.Clear();
    ConversationPiece c = new ConversationPiece() { id = data[11].ID, text = data[11].texte, options = new List<ConversationOption>(), hint = data[11].hint };
    donneurDeQuete.GetComponent<ConversationScript>().items.Add(c);
    donneurDeQuete.GetComponent<ConversationScript>().OnAfterDeserialize();

    // GARDE
    gardeForestier.GetComponent<ConversationScript>().items.Clear();

    ConversationOption o1 = new ConversationOption() { text = data[12].branche1Reponse, targetId = data[1].branche1ID };
    ConversationOption o2 = new ConversationOption() { text = data[12].branche2Reponse, targetId = data[12].branche2ID };
    ConversationPiece c1 = new ConversationPiece() { id = data[12].ID, text = data[12].texte, options = new List<ConversationOption>(), hint = data[12].hint };
    c1.options.Add(o1);
    c1.options.Add(o2);
    gardeForestier.GetComponent<ConversationScript>().Add(c1);

    ConversationOption o3 = new ConversationOption() { text = data[13].branche1Reponse, targetId = data[13].branche1ID };
    ConversationPiece c2 = new ConversationPiece() { id = data[13].ID, text = data[13].texte, options = new List<ConversationOption>(), hint = data[13].hint };
    c2.options.Add(o3);
    gardeForestier.GetComponent<ConversationScript>().Add(c2);

    ConversationPiece c3 = new ConversationPiece() { id = data[14].ID, text = data[14].texte, options = new List<ConversationOption>(), hint = data[14].hint };
    gardeForestier.GetComponent<ConversationScript>().Add(c3);

    ConversationPiece c4 = new ConversationPiece() { id = data[15].ID, text = data[15].texte, options = new List<ConversationOption>(), hint = data[15].hint };
    gardeForestier.GetComponent<ConversationScript>().Add(c4);
    gardeForestier.GetComponent<ConversationScript>().OnAfterDeserialize();

}
```

Cette fonction, appelée après avoir accepté la quête du chasseur, va mettre à jour la discussion du PNJ donneur de quête, ainsi que celle du prochain PNJ auquel nous devons parler, ici "gardeForestier".
(La partie //SELF correspond au personnage auquel on vient de parler, // GARDE au personnage suivant).

6) *Guide de Jeu*

L'objet "Guide de Jeu" est un objet visuel permettant de donner au joueur des information de type Texte au joueur, comme par exemple, au début de la partie, les information sur le rôle qu'on joue :



a) *Utilisation du Guide de Jeu*

Cependant, cet objet peut être appelé pour d'autres raisons lors d'une partie, alors voici comment faire apparaître ce guide et changer le texte :

```
guide.GetComponent<Canvas>().enabled = true;
GameObject.Find("CanvasGuideJeu").GetComponent<GuideManager>().guideText.SetText("Mettre son texte ici");
```

Ici, l'objet "guide" est un GameObject que l'on doit glisser dans l'inspecteur de l'objet sur lequel le script est attaché. Cet Object que l'on doit glisser est "CanvasGuideJeu"

Il faut juste mettre le texte voulu là où il est écrit "Mettre son texte ici".

b) *Le script "GuideManager.cs"*

Sur l'objet "CanvasGuideJeu" est attaché le script "GuideManager.cs".
Ce dernier va permettre de mettre les valeurs de base du texte selon le rôle joué.

On peut voir la fonction pour fermer le guide dans lequel on met du délai après avoir fermé le guide afin de permettre au joueur d'avoir le temps de replacer ses doigts sur le joystick avant de reprendre la partie.

7) ***Joueur***

a) ***Les Joysticks***

Le premier GameObject de Joueur est “Joystick”, qui contient les joysticks de chaque rôle.
Chaque JoystickCanvas doit avoir comme RenderCamera la camera du rôle en question.
On doit aussi adapter le Sorting Layer en fonction de si le rôle est soumis ou non au brouillard de guerre (Pour le chamois et le randonneur c’est le cas, le Sorting Layer est donc FogOfWar)

Chacun des joueurs possèdent les scripts JoystickLink, auquel on lie le FixedJoytick de chaque rôle, ils possèdent aussi un script JoueurChamois, JoueurRandonneur et JoueurChasseur, dans lequel sont stockés la vitesse du joueur, la caméra, et d’autres informations comme le tir (pour le chasseur), l’adrénaline quand on est attaqué (pour le chamois) ou des informations sur les randonnées (pour le randonneur).

b) ***Les 3 rôles***

Le second GameObject est “Players” qui contient les 3 personnages jouables.
Il contient donc PlayerChasseur, PlayerChamois et PlayerRandonneur.

Chacun de ces objets contient des Sprites et des Colliders.

c) ***Le stock des données des rôles***

Chaque rôle possède aussi un script de type “DataStorer” permettant de stocker les informations utiles pour l’accomplissement des hauts faits, mais aussi la gestion du score pour le chamois.

d) ***Le tir et le ramassage de déchets du Chasseur***

Pour le Chasseur, il y a un Objet “Boutons”.
Le bouton “ModePewPew” est le bouton permettant au chasseur de passer en mode de tir.
Appuyer sur ce bouton fait apparaître le bouton “Pew” permettant de tirer une balle.
L’objet Munitions contient l’image de balle ainsi que le nombre de balles restantes, permettant d’avoir l’information visuellement.

De la même manière que Munitions, l’objet Dechets contient les informations visuelles à propos des déchets ramassés.

Le chasseur possède un script Munition, gérant les munitions du joueur, mais aussi ChasseurDechet permettant de gérer le ramassage de déchets.

e) Les Jauges du Chamois

Pour le Chamois, il existe un GameObject “Jauges”. Il contient un objet “decorations” qui contient lui-même les 4 jauges de vie, de faim, de stress et d’expérience, ainsi que les textes permettant d’afficher visuellement les valeurs des jauges.

Sont donc attachés à l’objet “Jauges” les scripts correspondant aux 4 jauges.
Ces 4 scripts permettent de modifier directement dans l’inspecteur les valeurs des jauges :
VieActuelle, StressActuel, FaimActuelle, ExpActuelle correspondent aux valeurs des jauges à un instant t. Les valeurs entrées de base correspondent aux valeurs de base des jauges.
Pour PVMax, StressMax, FaimMax, ExpMax, ces valeurs correspondent aux valeurs maximum que peuvent atteindre les jauges.
Les valeurs de palier correspondent aux valeurs dont seront augmentées ou bien réduites les jauges par intervalles de temps, ces intervalles sont déterminées dans les valeurs TimerStress, TimerFaim, WaitTimerStress, WaitTimerFaim, WaitTimerExp.

f) Avoir une influence sur les jauges depuis un autre script

Pour faire varier la jauge dans un autre script voici comment faire :

```
vie.setImage(vie.image, vie.vieActuelle, vie.pvMax);
```

Il faut juste instancier un GameObject (ici vie) dans lequel on glissera l’objet de jauge dans l’inspecteur, ou en effectuant un GameObject.Find(“Vie”).
(cela marche aussi avec les autres jauges on peut autant faire un vie.setImage qu’un stress.setImage tant qu’on a déclaré vie et stress dans le script en question).

Dans ces 4 scripts sont gérés les éléments ayant une influence sur les jauges et comment les jauges ont une influence les unes sur les autres.

g) Augmentation de la taille du cercle de brouillard

Enfin, dans le script “Experience”, se trouve une variable “AugmentScale”. Cette valeur est utilisée à chaque montée de niveau afin d’augmenter la taille de la vision dans le brouillard de guerre (dans le cas ou l’on veut modifier l'agrandissement de ce rayon, il faut changer le x et le y d’AugmentScale, mais faire en sorte que ces deux valeurs soient égales).

h) Les bonnes / mauvaises zones de randonnée

Enfin, pour le randonneur, un script nommé “TestTriggerZone” est attaché.
Ce dernier sert à déterminer si une zone est une bonne ou une mauvaise zone lors d’une randonnée. Cependant, ici, ce n’est une fonctionnalité qui a été développée, mais pas encore utilisée dans le gameplay encore.
Ces zones se trouvent dans Game > Décor > Zones Randonnees

8) Menu de Jeu

a) Le bouton de Menu

La catégorie “Menu de Jeu” contient 2 choses : MenuRegroupement, soit disant le bouton permettant de faire apparaître les boutons de jeu (encyclopédie, pause, haut faits), mais aussi MenuManager, qui correspond au bouton de Pause.

Afin de faire apparaitre / disparaitre les boutons en appuyant sur le bouton de regroupement, on utilise le script RegroupementMenu, qui est attaché à l’objet “Button”, se trouvant dans l’objet MenuRegroupement.

b) Mettre le jeu en Pause

Le script permettant de gérer le menu de pause se trouve sur l’objet “PauseButton”, se trouvant dans “MenuManager”. Ce script se nomme “PauseMenu”.

9) Carte

La catégorie Carte correspond à la carte que l’on peut ouvrir pour avoir une vue générale sur notre position et notre environnement.

a) La Caméra de la Carte

A l’intérieur, existe un objet Camera qui correspond à la caméra utilisée pour regarder la carte. Son script “CamMov” va permettre de déplacer cette carte au simple glissement de doigt sur l’écran.

b) Ouvrir la Carte

ButtonMap, objet aussi présent dans la catégorie, correspond au bouton permettant d’ouvrir la carte, en faisant changer la caméra actuelle, grâce au script SwitchPlayerMap.

c) Objet visuel de la Carte

Enfin, fondCarte est l’objet qui correspond à la carte simplifiée. l’objet Camera est fixé sur cet objet, quand on clique sur le bouton, on switch de caméra et on voit donc cette carte.

10) Encyclopédie

a) La Notification d'entrée dans l'Encyclopédie

Cette catégorie contient un objet "EncyclopédieManager", qui contient un objet "Notification", qui sera activé lorsqu'une nouvelle information sera ajoutée à l'encyclopédie.

L'objet contient aussi "Livre" qui l'objet visuel sur lequel seront "écrites" les informations.

b) Les objets composant l'Encyclopédie

Il contient aussi un objet "Ouvre" étant composé de 3 boutons : OuvreChasseur, OuvreChamois et OuvreRandonneur, permettant d'ouvrir l'encyclopédie appropriée, en fonction du rôle joué.
Ces 3 boutons permettront d'activer les objets d'EncyclopédieManager nommés "ChapitreChasseur", "ChapitreChamois" et "ChapitreRandonneur" qui sont composés des chapitres pour chaque rôle (infos de bases, infos obtenues et infos de quêtes).

Pour revenir sur l'objet Livre, il contient "Contenu" définissant la page gauche et la page droite, ainsi que les boutons pour tourner les pages, et fermer l'encyclopédie.

c) Le contenu de l'Encyclopédie des rôles

Dans l'objet principal, "EncyclopédieManager" se trouvent 3 scripts : EncycloContentChasseur, EncycloContentChamois et EncycloContentRandonneur, on récupère les informations que l'on peut obtenir au cours de notre aventure, de la même manière que précisée dans la partie **NPC Collection** (c.f page ?).
C'est aussi dans ces scripts que l'on gère l'action de tourner les pages et le choix du chapitre d'informations sélectionné.

d) Créer une nouvelle information d'Encyclopédie

Pour créer une nouvelle information, il faut aller dans les fichiers EncyinfosDynamicChasseur, EncyinfosDynamicChamois, EncyinfosDynamicRandonneur (en fonction du rôle pour lequel on veut ajouter une information.

Chaque information doit posséder une "hint" (comme précisé dans la partie **NPC Collection**), un texte, correspondant à l'information qui sera affichée dans l'encyclopédie, et la taille du texte, ne devant pas être supérieure à 6.
Cette "taille" correspond à la place qui sera réservée sur la page pour cette info, sachant que le cumul de la taille de toutes les infos sur une page ne doit pas excéder 6. Pour donner un exemple, si on ne possède que 3 infos de taille 2, toutes les infos seront affichées sur la page de gauche. Si l'info fait plus que la taille d'une page, le programme ne saura pas ou afficher le texte et une erreur risque d'être rencontrée.

e) Ajouter une information dans l'Encyclopédie (en jeu)

Il faut cependant savoir qu'ajouter l'information dans le JSON ne permettra pas instantanément d'observer l'information dans l'encyclopédie.
En effet l'information existe pour le programme, mais vu qu'elle n'est pas obtenue par le joueur, elle n'est pas présente dans son encyclopédie.

Afin que l'information soit ajoutée à l'encyclopédie, voici ce qu'il faut faire :

```
public static void addToEncy()
{
    if (!activateOnce)
    {
        EncycloContentChamois ency = GameObject.Find("EncyclopedieManager").GetComponent<EncycloContentChamois>();
        ency.addInfoToList("danger", ency.pagesDynamic);
        activateOnce = true;
    }
}
```

On crée une fonction qui va permettre d'ajouter l'information à l'encyclopédie.
Cette fonction doit ensuite être appelée dans le code, au moment où la condition d'obtention est réalisée.
Dans le cas ci-dessus, on obtient l'information dès que le chamois se trouve dans la zone de détection d'un prédateur.
Cependant, si on l'ajoute ainsi dans un Update(), l'information va être ajoutée à chaque frame où l'on se trouve dans le rayon du prédateur, c'est pour cela que nous testons un booléen "activateOnce" qui dès que l'information a été ajoutée, va passer à vrai pour éviter de re-rentrer dans la fonction.

11) Managers

a) Notification du Bouton de Menu

Passons maintenant à la catégorie “Managers”.
Cette catégorie contient un objet “GameControl” auquel est attaché un script “GameControlScript.cs”. Ce dernier va juste gérer l’activation des notifications lorsqu’une nouvelle information entre dans l’encyclopédie.
Ces notifications se représentent visuellement par des points rouges se trouvant sur le bouton du MenuRegroupement, et, quand ce menu est ouvert, sur le bouton de l’encyclopédie.

b) La vibration sur les boutons

On y trouve aussi un objet “GameManager”.
A cet objet sont attachés plusieurs scripts, comme “vibrate.cs”, qui possède la fonction pour faire vibrer le téléphone.
Pour l’instant cette fonction est appelée sur chaque bouton :
dans les OnClick() des boutons, on ajoute une fonction, on link l’objet “GameManager” et on appelle la fonction vibrate.vibration.

c) Panneaux de fin de partie & haut faits de fin de partie

Un autre script de GameManager est “FinPartie.cs”, qui va gérer l’activation des panneaux de fin de partie, ces panneaux se trouvant à l’intérieur du GameObject “GameManager”.
Dans ce script sont aussi gérés l’accomplissement de certains hauts faits (ceux qui s’accomplissent seulement sur 1 seule partie.

d) Le script “PlayerType.cs” & les objets à désactiver en début de partie

Y est attaché aussi le script “PlayerType.cs”, qui, en fonction du rôle choisi, va activer ou désactiver les bons objets en début de partie. Donc si une fonctionnalité est développée pour un rôle en particulier il serait judicieux d’activer / désactiver les gameObjects appropriés dans ce script.

e) Changer la date de départ du jeu

Enfin est attaché à GameManager le script “DayNight.cs”, script permettant de gérer le système de jour et de nuit, ainsi que le système de dates. Dans le cas où l’on veut changer la date de départ de jeu, on peut changer les valeurs de base dans l’inspecteur.

f) La musique du jeu

Enfin, dans “Managers” se trouve un gameObject nommé “Controllers”.
La seule chose intéressante à savoir ici est la présence d’une “AudioSource”. Cette “AudioSource” contient la musique du jeu dans la variable “AudioClip”.

12) Décor

La catégorie "Décor" contient premièrement un GameObject "Nourriture". Il contient 2 sortes de choses : "EventNourriture" et tous les préfabs correspondant aux objets de nourriture.

a) Alimentation du chamois & repousse de la flore

"EventNourriture" possède 2 scripts : "NourritureMangee.cs" qui va gérer le fait de manger une plante ; Il possède aussi le script "NourritureRepousse.cs", qui va gérer la repousse des plantes qui ont commencé à être mangées. On peut choisir ici, dans l'inspecteur, l'intervalle à laquelle les plantes repoussent (variable Temps Repousse).

b) Caractéristiques de la nourriture

Ensuite, chaque élément "Nourriture" possède un script "CaracNourriture.cs". Ce script permet de définir les caractéristiques de chaque élément de nourriture. On peut choisir dans l'inspecteur les valeurs suivantes :

Nourrissant, qui va avoir une influence sur la jauge de faim (la faire augmenter),
Destressant, qui va avoir une influence sur la jauge de stress (la faire baisser),
Sante, qui va avoir une influence sur la jauge de vie (la faire augmenter),
Score, qui va améliorer le score de fin de partie du chamois.

Bien évidemment, on peut mettre une valeur négative, ce qui aura l'effet inverse sur la jauge (dans le cas d'une plante toxique par exemple).

c) Évolution visuelle de l'état de la plante

Enfin CaracNourriture demande un "Sprite Array" qui correspond aux sprites de tous les états de la plante (entière, mangée un peu, bien mangée).

d) Stockage des bonnes & mauvaises zones de randonnée

"Décor" contient un GameObject "ZonesRandonnees", qui contient toutes les zones définies pour les randonnées comme précisé auparavant. (c.f page ?, avec le script TestTriggerZone.cs).

13) Cameras

La catégorie “Cameras” ne contient qu’une caméra nommée UI Camera, cette dernière nous permet de pouvoir voir les discussions avec les PNJ.
De préférence, il ne faudrait pas la modifier car celle-ci provient de l’asset.

14) Système de Dates

La catégorie Système de Dates ne contient que les éléments visuels permettant d’afficher la date.
Comme précisé précédemment, pour changer la date de départ du jeu, il faut changer les valeurs dans le script “DayNight.cs” attaché à l'objet “GameManager” de la catégorie Managers.

15) Système de Jour & Nuit

La catégorie Système de Jour & Nuit contient un prefab qui correspond à une image noire.

a) Nécessités du système de Jour et Nuit

Cette dernière doit absolument recouvrir toute la surface de jeu (elle peut dépasser, c’est pas grave).

b) Fluctuation de l’alpha de l’image : Simulation du jour et de la nuit

Pour simuler le jour et la nuit, on fait fluctuer l’alpha (soit la transparence) de cette image.
Cette fluctuation est gérée dans le script “DayNight.cs” attaché à l'objet “GameManager” de la catégorie Managers.
Cette simulation est gérée par les fonctions Sunset() et Sunrise().

16) Autres

Dans cette catégorie se trouvent les objets InventoryItemCollection, EventSystem, UI et StoryItemCollection, qui sont des objets provenant de l’asset de discussion avec les personnages, il est donc conseillé de ne pas y toucher.

a) Les colliders des parois rocheuses

Il y a aussi l’objet “ParoisRocheuses” qui contient tous les colliders correspondant aux parois rocheuses.

b) Ajouter un collider "personnalisé"

S’il faut rajouter un collider pour une paroi, dupliquer une paroi déjà existante, il faut ensuite cliquer sur Clear Points dans le script “Draw2D”, ensuite effectuer sur la scène “shift + clic gauche” pour placer les points et dessiner la forme définissant la paroi.

17) Autres Informations

a) Ajouter un effet sonore

Si l'on veut jouer un son lors d'un événement, il faut ajouter une audio source à un gameObject (comme par exemple dans notre gameobject "predateurs", qui contient une audio source). Dans la variable AudioClip, on glisse notre son et on décoche tout ce qu'on peut décocher (surtout PlayOnAwake et Loop).
Ensuite, dans le code, quand on veut jouer le son, voici comment faire :

```
GameObject.Find("predateurs").GetComponent<AudioSource>().Play();
```

Il faut juste remplacer, dans l'exemple, "predateurs" par le nom du GameObject dans lequel on a mis notre son (et donc l'audio source).

Pour respecter le choix de l'utilisateur, vérifiez quand même, avant de jouer l'effet, que le joueur a bien activé les effets sonores :

```
if (PlayerPrefs.GetInt("soundEffects") == 1)
{
    GameObject.Find("predateurs").GetComponent<AudioSource>().Play();
}
```

b) Système d'inventaire en construction

Il existe aussi un système d'inventaire, qui est en construction. Pour le trouver, il se trouve dans une scène à part, qui se trouve selon ce chemin :
Assets > Scenes > Jeu > DragAndDrop