Prototype (TP3)

Par :

Bénard Desjardins, Yann - 111 114 105
Bruère, Sébastien Samuel - 111 244 646
Dion, Charles-Etienne - 111 155 401
Dupe, Guillaume - 111 244 690
Lauzon, Marc - 111 183 330
Leroy, Maxime - 111 244 596
Mori, Nicolas - 111 244 583

Équipe 06

Réalisé dans le cadre du cours :

IFT-3113 - Projet de jeu vidéo

Rapport présenté à :

Chéné, François

Remis le :

03 mars 2019

UNIVERSITÉ LAVAL

Table des matières

The Echoes of Thalass

Aventure, exploration et combat sous-marin.

Expérience de jeu

Rescapé d'une attaque spatiale sur la planète la plus proche, Bob doit explorer les profondeurs de cette planète océanique pour trouver les pièces manquantes et les ressources nécessaires pour reconstruire son vaisseau.

Par chance, un sous-marin contrôlable par télécommande survie aussi l'incident et permet à Bob d'explorer cette planète inconnue tout en restant à la surface. Cependant, pour accumuler les ressources nécessaires à la reconstruction de son vaisseau spatial, le sous-marin devra être amélioré autant pour atteindre des niveaux de profondeurs de plus en plus extrêmes que pour vaincre les dangereuses créatures marines qui habitent cette planète.

Lorsque Bob commence son exploration, le sous-marin ne comporte qu'une caméra, des armes minimalistes et une pile avec une capacité très faible. Il doit donc commencer par explorer près de la surface et trouver des ressources pour améliorer son sous-marin. Déjà près de la surface, il s'aperçoit que cette planète est remplie de créatures aquatiques agressives et que reconstruire son vaisseau ne sera vraiment pas simple. Après avoir éliminé différentes créatures et amassé des ressources et quelques débris de son vaisseau, Bob peut maintenant équiper son sous-marin d'une meilleure pile et d'armes plus efficaces.

Pendant son exploration, Bob aperçoit une cave sombre qui semble mener vers une partie encore plus profonde de la planète. Maintenant que son sous-marin est mieux équipé, il se sent prêt à refaire face à l'inconnu et pénétrer dans les abysses océaniques.

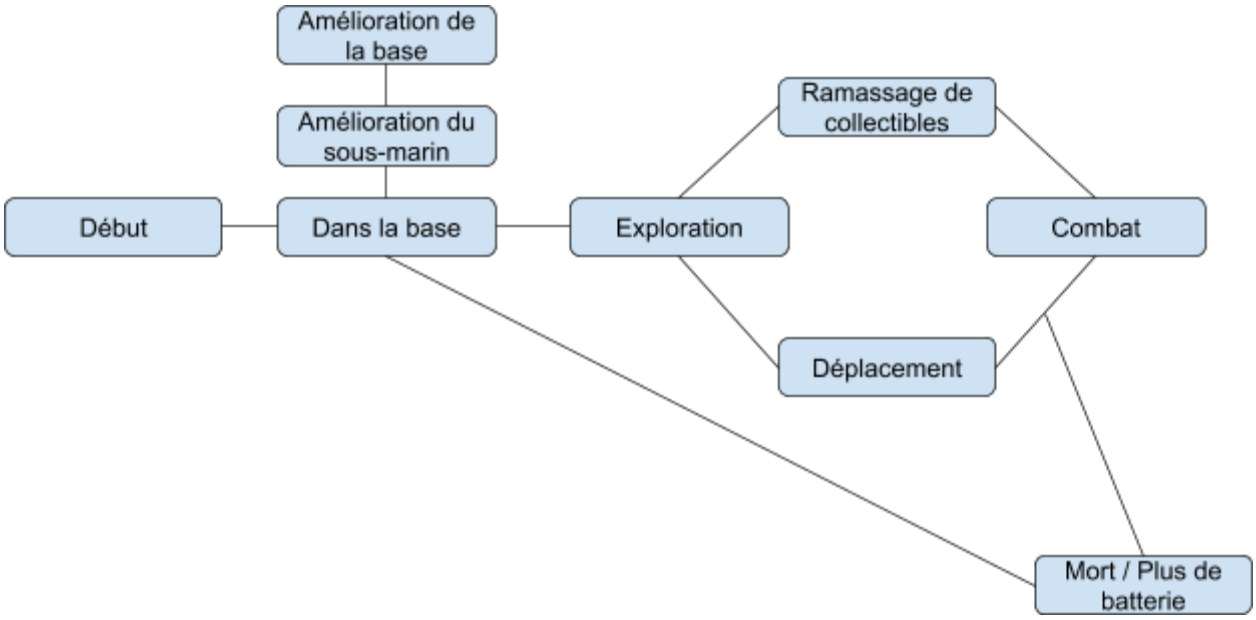
Intégration du thème

Le thème étant “Sous la surface” nous avons choisi l'atmosphère sous-marine.

Inspirations

Le jeu est principalement inspiré de Subnautica pour l'histoire d'arrière-plan et l'exploration sous-marine, tandis que la partie gestion des ressources se range du côté des jeux tel que Learn to Fly où on doit améliorer son équipement entre chaque exploration à temps limités pour progresser dans le jeu. Le système de combat et les contrôles du sous-marin télécommandé sont inspirés par des jeux de tir en première personne dans l'espace comme Descent.

Boucles de jeu



Mécaniques prévues

Il y a une mécanique primordiale qui concerne l'expérience d'exploration sous-marine : Le contrôle sous-marin sur tous les axes. Ensuite, il y a les mécaniques de combat, d'inventaire et d'augmentation des capacités qui viennent agrémenter la mécanique primordiale.

Déplacement

Déplacement sous l'eau donc en 3 dimensions sans contraintes, c'est la direction de la caméra qui sert de référence.
Si aucun déplacement, on est attiré vers le fond de l'eau suivant l'axe **Y**.

Combat

Pour vaincre les différents ennemis présent sous-l'eau le joueur pourra utiliser plusieurs types d'armes pour se défendre. Un système d'expérience et de niveau style RPG permettra au joueur d'améliorer ses statistiques pour vaincre des créatures sous-marines de plus en plus féroce et menaçante.

Inventaire

L'inventaire permet au joueur de voir les ressources qu'il a accumulé. Chaque type de ressource est identifiable par une image unique et le nombre de cette ressource accumulé est affiché pour que le joueur puisse savoir quel ressources il est mieux de ramasser durant son exploration.

Augmentation

L'augmentation des capacités concernent les éléments suivants :

- *La pile (Battery)* : La pile est le réservoire d'énergie du sous-marin télécommandé. Lorsque le sous-marin est attaqué ou si il se déplace, la pile perd de son énergie. Lorsque la pile est vidée, le sous-marin remonte d'urgence.
- *L'armure (Armor)* : L'armure est le rempart contre les dommages qui pourraient être infligé à la pile. Celui-ci est endommagé avant que la pile ne commence à perdre son énergie.
- *L'entreposage (Storage)* : L'entreposage concerne l'espace que le sous-marin possède pour conserver les objets qu'il récupère.
- *Le taser (Taser)* : Le taser est l'arme du sous-marin télécommandé qui permet de neutraliser les vies sous-marines agressives. Celle-ci utilise de l'énergie de la pile.
- *La propulsion (Propulsion)* : La propulsion concerne la force de poussée maximale du sous-marins.

L'augmentation se fait dans la scène de gestion du sous-marin entre chaques épisodes d'exploration. Lorsqu'on entre dans la scène de gestion, une fenêtre s'affiche pour nous informer des différentes augmentations disponibles. Un simple bouton ordonne l'amélioration de l'élément.

Il y a 4 niveaux d'augmentation possibles au-delà du niveau de base et chacune d'elle demande plus de ressources que le niveau précédent. Ces ressources sont consommés à partir de l'entreposage du sous-marin.

Ressources

Agents

- Sous-marin (joueur invisible)
- Poissons
  - Passifs
  - Défensifs
  - Agressifs
- Collectibles
  - Plastique
  - Métal
  - Circuit électronique

Effets

- Bulles
- Arc électrique
- Rayon de soleil

Environnement

- Textures du sol
- Roches
- Algues
- Radeau

Menu

- Menu d’accueil
- Menu d’option
- Menu d’augmentation
- Interface du sous-marin
- Menu de pause
- 3 polices de caractères
  - Titre
  - Texte
  - Icones

Plateforme technologique

Le jeu est réalisé en 3D avec le moteur de jeu Unity. Le code consiste en script C# conçu avec l'IDE Visual Studio. Une gestion de version Git avec GitHub.

Les modèles proviennent d'album web libre de droit ou réalisé avec Blender. Les éléments 2D des menus et interfaces seront conçus avec Adobe Illustrator CS6 et Adobe Photoshop CS6.

Les effets sonores seront trouvés dans des bibliothèques de son et modifié s'il y a lieu avec Audacity.

La documentation est produite avec Google Docs.

Architecture logicielle

Le jeu consiste en un seul joueur, alors toute la gestion de l'application sera locale. La gestion d'entrée utilise le système déjà en place dans le moteur de jeu Unity. Aucun système ou technique particulière n'est présentement envisagée.

Présentation de l'équipe

- Sébastien Bruere - Programmeur Gameplay
- Yann Bénard Desjardins - Programmeur Graphique/Gameplay
- Guillaume Dupe
- Marc Lauzon - Programmeur GUI / XUI
- Nicolas Mori
- Maxime Leroy
- Charles-Etienne Dion

Annexe 1 - Analyse technologique

Défi 1 – Objet programmable du sous-marin et UI. (Marc Lauzon)

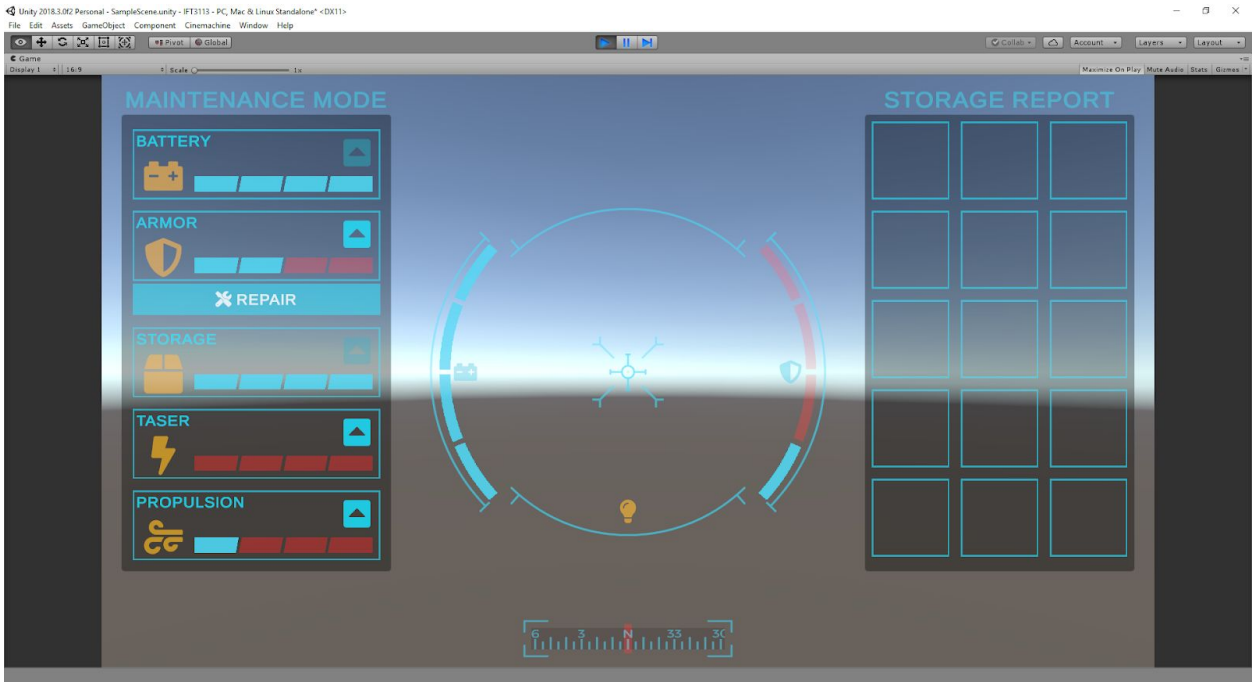
Objet programmable

Une partie du défi, l'objet programmable du sous-marin. N'ayant pas encore défini toutes les capacités du sous-marins, je m'en suis tenu à quelques éléments.

Avant tout, une classe imbriquée qui représente un élément qui possède un niveau, une valeur de base et une valeur actuelle. Cette classe possède une méthode qui retourne le maximum étant la valeur de base fois le niveau, un ratio étant la valeur courante sur le maximum, ainsi qu'une fonction «Upgrade» qui augmente le niveau de l'élément.

Jumelé avec l'interface d'augmentation, ce dernier possède des boutons qui commandent à un script d'appeler la fonction «Upgrade», ensuite de mettre à jour l'interface pour représenter les nouvelles valeurs.

Interface

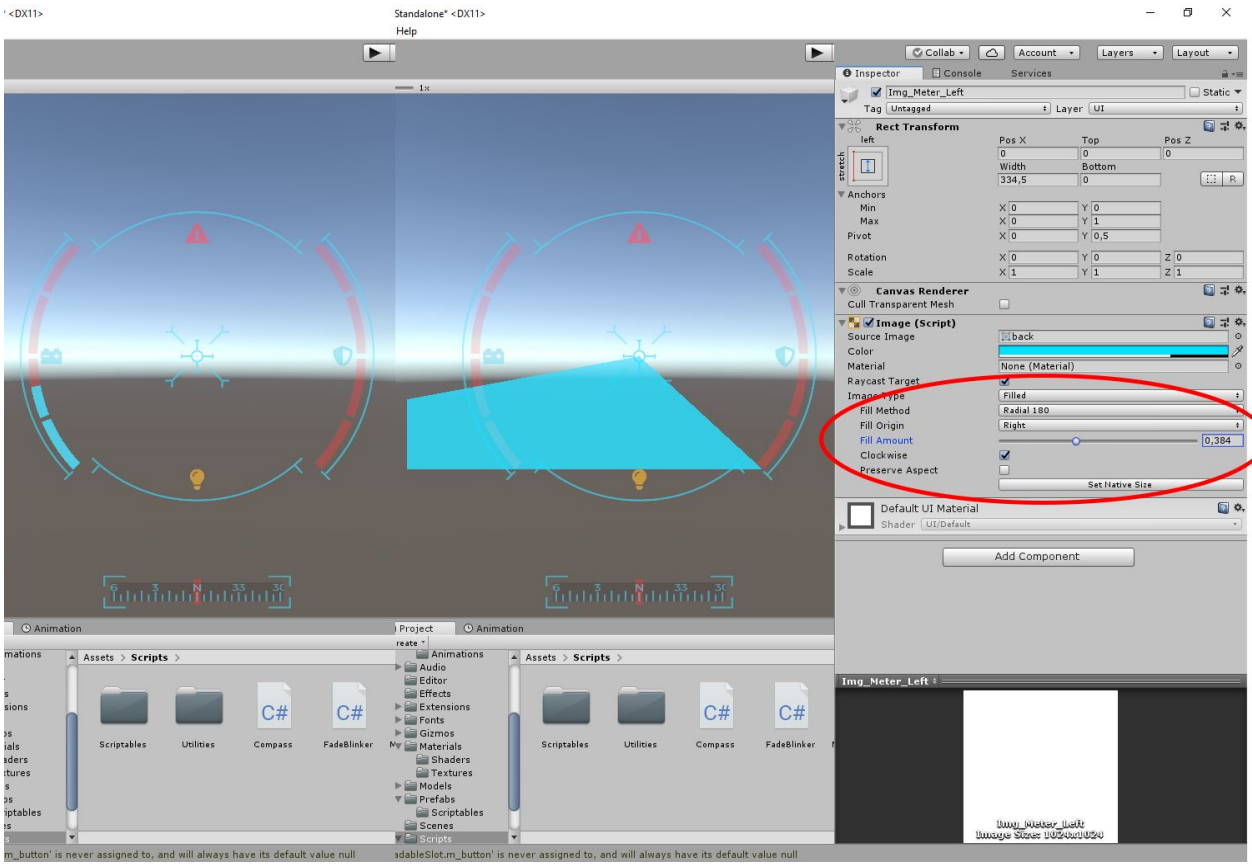


Interface du menu d'augmentation de la scène de gestion du sous-marin

Ayant comme hobby la conception de jeu vidéo et ayant aussi un côté artistique prononcé, développer d'une interface est une habitude. Le tout commence toujours par la recherche d'inspiration sur Google et Pinterest.

Parmis les difficultés, avant tout, Adobe Illustrator. D'ordinaire, j'utilise Adobe Photoshop, mais j'ai choisi de prendre cette opportunité pour m'habituer à ce logiciel. Unity se devait de prendre nativement le format SVG, mais il semble encore en développement, alors je me suis rabattu sur le format PNG. Habituellement, je favorise Adobe Photoshop, car Unity assure l'importation des fichiers PSD, mais durant un projet collaboratif si un collaborateur n'a pas l'application, Unity n'affiche rien de bon.

Certaines ressources ont été utilisés en dehors de leur contexte. Les icônes sont en fait des caractères en provenance de la police de caractère FontAwesome[1]. Les jauges de la pile et de l'armure utilise une propriété des images de Unity qui permet un remplissage rotatif et un masque. Les jauges de niveau des augmentations sont des glissières de la trousse de base UI de Unity agrémentés de masques.



Propriété d'image : Remplissage rotatif (Radial Fill)

[1] FontAwesome : https://fontawesome.com

Une petite difficulté d'un élément de l'interface fut la boussole horizontale à ruban qui a nécessité quelques essais et erreurs pour bien calibrer une rotation 360 en utilisant toujours la même texture. Notamment la conversion d'une rotation 360 sur une échelle de [-180, 180[applicable sur le déplacement du ruban.

Boussole horizontale à ruban

- Lorsque l'angle est assigné, la valeur est modulé par 360 et ensuite échelonné entre -180 et 180.
    - Si la valeur de l'angle est plus grand que 180, on soustrait 360.
    - Si la valeur de l'angle est plus petite que -180, on ajoute 360.
    - Autrement, l'angle est entre les bornes, on ne modifie pas la valeur.

Exemple : Le nord est à 0 degré. Je fais faire deux tours et trois quart anti-horaire à mon sous-marin, soit -360+-360+-270. L'assignation ne va que retenir -270. -270 qui est plus petit que -180 se voit ajouter 360, et donc 90. 90 dans le sens horaire c'est l'est. Lorsque une translation positive de cette valeur est appliqué au ruban, multiplié par 2 de par sa taille, il va afficher un E pour l'est.

Menu d'augmentation

Le menu d'augmentation est uniquement disponible lorsque le joueur se trouve dans la scène de gestion du sous-marin. On présente les cinq éléments dans cinq boîtes à leur nom et une petite icône décorative. Un bouton est disponible lorsque l'élément n'est pas complètement augmenté. Suffit d'appuyer ce bouton pour augmenter le niveau de l'élément. Dans l'avenir, un critère de coût pour la disponibilité du bouton sera ajouté.

Boîte d'augmentation de la propulsion

Vidéos de démonstration

Jauge et boussole : https://www.youtube.com/watch?v=8AW3x8rvSUw
Menu d'augmentation : https://www.youtube.com/watch?v=Ft_HDXDOLIU

Défi 2 et 3 – Objets à collectionner et système de récupération, Gestion de l'inventaire
(Guillaume DUPE)

Tâche à effectuer

Une des composantes essentielles de notre jeu est l'amélioration du sous-marin.

Afin de mettre en place cette fonctionnalité, nous devons passer par certains pré-requis comme:
1. La création d'équipements avec leur caractéristique liée.
2. La création de ressources afin de pouvoir améliorer, acquérir nos équipements.
3. Un système de récolte pour récupérer les ressources en jeu.
4. Un inventaire pour visualiser les ressources récoltées.

1, 2. Équipements et ressources

Pour créer les équipements et les ressources, j'ai commencé par créer une classe mère définissant la base d'un objet:

- Un nom.
- Une description.
- Une catégorie (Ressources, équipements, consommables).
- Une rareté (Commun, rare, épique, légendaire).

Suite à cela, pour avoir un équipement, j'ai rajouté les propriétés suivantes:

- Des caractéristiques qui serviront à faire évoluer le sous-marin selon le niveau et l'équipement équipé (Batterie, Armure, Stockage, Taser, Propulseur).
- Des pré-requis de ressources selon le niveau de l'objet (Liste de ressources selon le niveau).

En effet, plus le niveau d'un équipement est haut, plus il nécessitera de ressources que ce soit en quantité, qu'en rareté.

Les ressources quant à eux n'eurent besoins que de la propriété:

- Vie correspondant à l'état de l'objet.

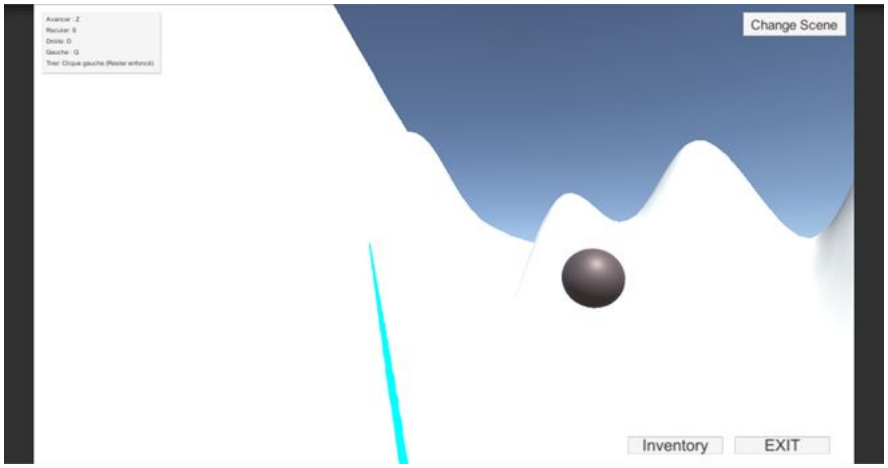
Pour faciliter la création d'équipement et de ressources dans notre projet, j'ai mis en place deux "Usines", dans lequel sont déclarés tous les équipements/ressources du jeu, chacun possédant un identifiant unique.

3. Système de récolte

Lors de la simulation les ressources à récolter sont sous la forme de boule marron dans la scène.
Celles-ci contiennent l'identifiant et la vie de la ressource qu'elles représentent.
De plus, celles-ci possèdent une quantité correspondant aux nombres de ressources récoltées lorsque celles-ci sont détruites.

Boule marron représentant la ressource en jeu / Information de la ressource

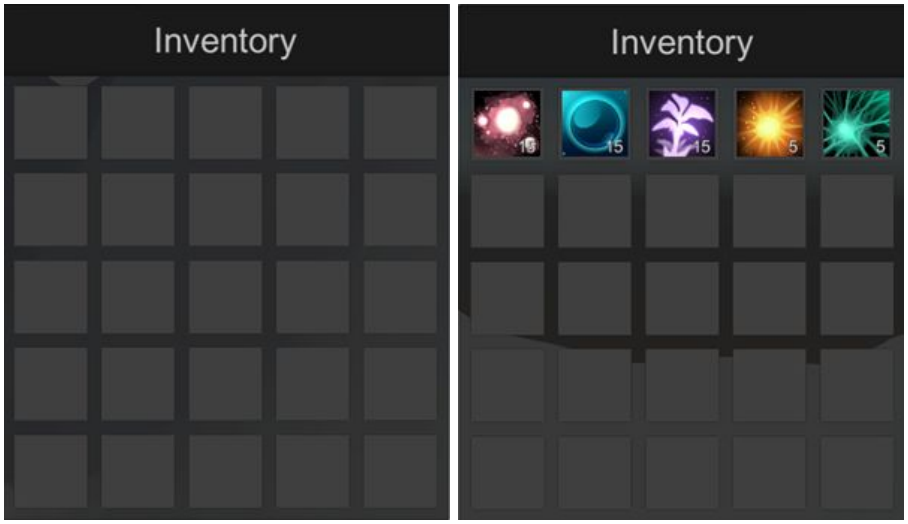
Pour récupérer les ressources en jeu, j'ai récupéré le laser d'un asset Unity et adapté à notre projet (https://assetstore.unity.com/packages/vfx/particles/spells/volumetric-3d-lasers-104580).
Il suffit de rester appuyer sur le clique gauche et sélectionnant une ressource pour la récolter.



Laser permettant de récupérer les ressources

4. L'inventaire

Pour visualiser les récoltes effectuées, j'ai mis en place un inventaire, accessible par le bouton "Inventory" ou encore la touche I. Celui-ci affichant l'image correspondant à l'identifiant de la ressource et la quantité récoltée de la ressource dans un slot.

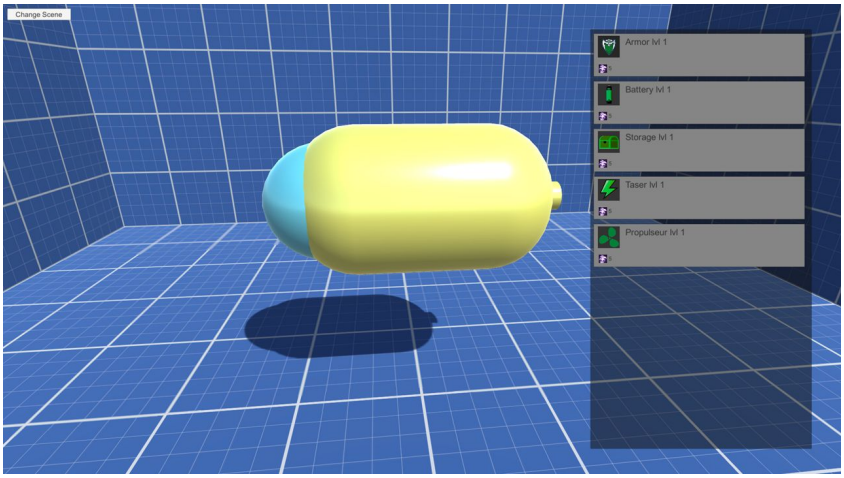


Inventaire vide / Inventaire remplis de différentes ressources

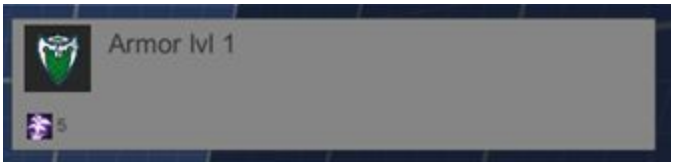
Une fois tout ceci mis en place, j'ai pu me consacrer sur l'amélioration du sous-marin.

Amélioration du sous-marin

Pour aller améliorer le sous-marin, il faut cliquer sur le bouton "Change Scene" qui amène sur la scène d'amélioration.
Sur cette scène se trouve plusieurs éléments:
- Un aperçu du sous-marin.
- Une interface où l'on peut voir tous les équipements disponibles, les améliorations possibles, ainsi que les ressources qu'il faut disposer pour les acquérir.

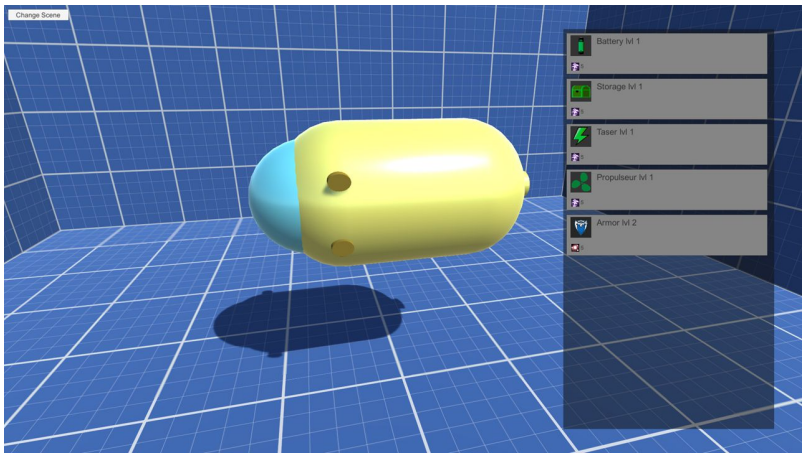


Scène d'amélioration du sous-marin

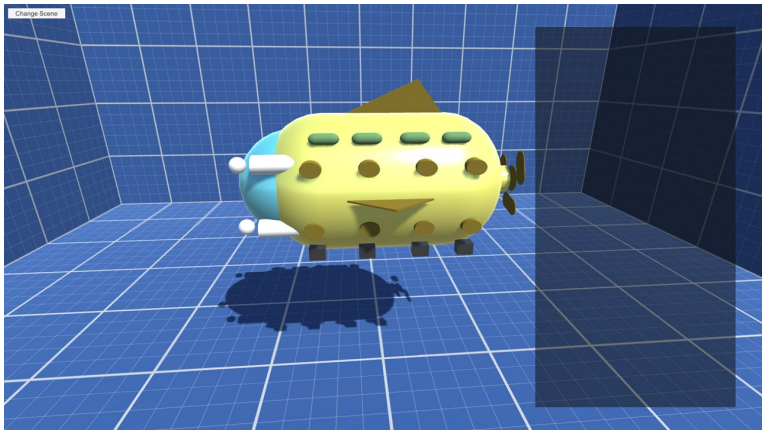


Équipement disponible pour 5 fleurs

À chaque amélioration, l'apparence du sous-marin change ainsi que les caractéristiques des équipements.



Sous-marin avec une armure lvl 1



Sous-marin avec toutes les améliorations

Vidéo de démonstration

Youtube: https://youtu.be/Srd0tWMEFIQ

Défi 4 – Contrôle du sous-marin et gestion de l’énergie (Sébastien Bruere)

Contrôle du sous-marin

La partie exploration du jeu se passe à bord d’un sous-marin télécommandé. Pour cela, j’ai mis un place un système complet de déplacement en essayant de reproduire au plus fidèle un déplacement sous l’eau.
Les contrôles sont en 3 dimensions et peuvent sans contraintes s'exécuter en suivant les 3 axes. La caméra étant en vue 1ère personne, j’ai choisi de baser tous les mouvements par rapport à celle-ci:

- Si on appuie sur la touche **AVANT**, on va droit vers la direction de la caméra
- Si on appuie sur la touche **GAUCHE** ou **DROITE**, on se déplace sur l’axe perpendiculaire à la direction de la caméra.
- Si on appuie sur la touche **ARRIÈRE**, on va à l’inverse de la direction de la caméra.

La physique de Unity étant utilisé, une diminution progressive de la vélocité quand aucune touche n’est appuyée est possible et permet de ne pas brusquement arrêter le déplacement.

Si aucune touche n’est enfoncée, le sous-marin est naturellement attiré vers le fond des eaux dû à la gravité définie de l’environnement sur l’axe **Y**.
Si on appuie sur la touche **Échap**, la caméra est verrouillée et le curseur de la souris apparaît. Au prochain clic sur l’écran, la caméra revient dans son état verrouillé et le curseur de la souris disparaît.

Gestion de l’énergie

Quand le sous-marin de déplace, donc quand un bouton de déplacement est enfoncé, l'énergie du sous-marin télécommandé diminue (La vitesse de diminution peut être réglée via une variable publique).
A l’écran ceci est représentée par un Slider vert en haut à gauche.

Défi 5 – Définition visuel de l’environnement sous-l’eau (Yann Bénard Desjardins)

Pour donner l’impression d’être sous-l’eau j’ai choisi comme défi d’implémenté différent shaders et effet visuels comme prototype.

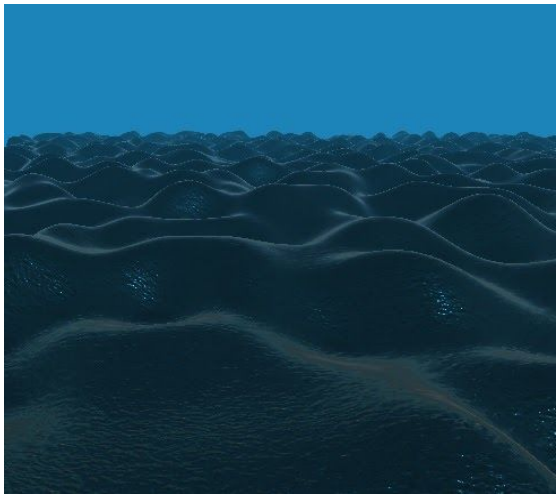
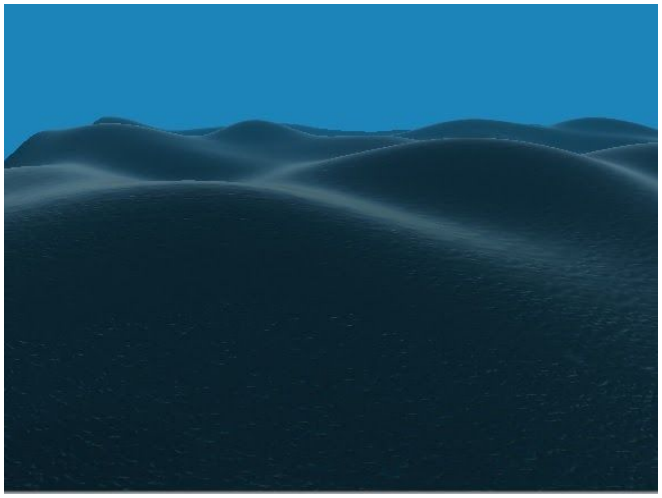
Contrôles du prototype

- WASD pour déplacer la caméra.
- Tenir SHIFT enfoncé pour déplacer la caméra plus vite.
- Utiliser la souris pour tourner la caméra.
- ESC ouvre un menu pour continuer la simulation ou quitter.

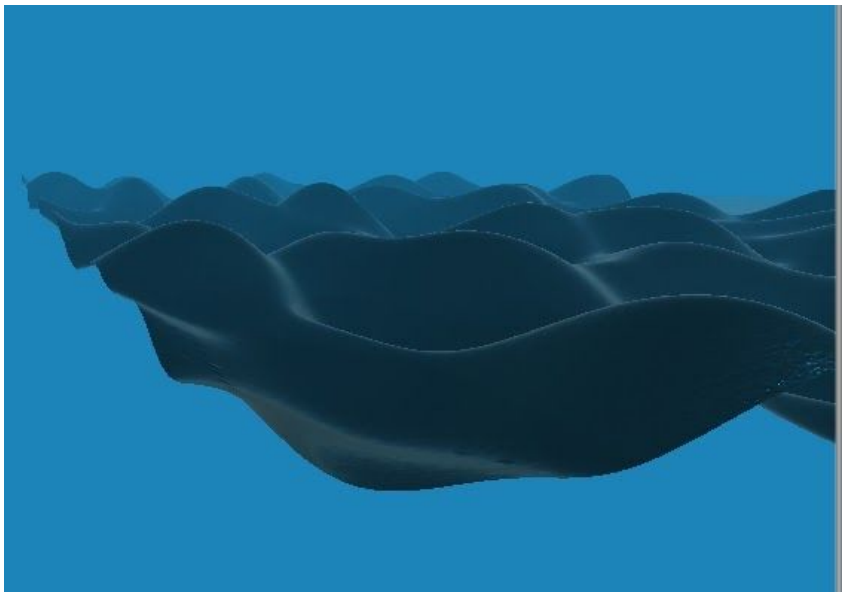
Shader de surface avec tessellation (NoiseGround)

Ce shader permet de créer des surfaces varié en utilisant du bruit de perlin (un port de la librairie noiseSimplex pour Unity). Le niveau de tessellation augmente le nombre de triangles et la qualité du rendu mais demande plus de performance.

Il est possible de modifier la hauteur et la fréquence du bruit pour créer différent type de surfaces:

Un des problèmes pour utiliser ce shader dans notre projet sera de gérer les transitions entre différentes surfaces puisque le shader affecte complètement un objet (dans notre cas une plane). Ceci créer du bruit sur l’ensemble du contour de l’objet:



Il faudra donc soit trouver un moyen d'appliquer le shader seulement sur des polygons spécifique d'une surface ou soit utiliser ce shader de manière plus limiter en cachant les extrémités.

Source:

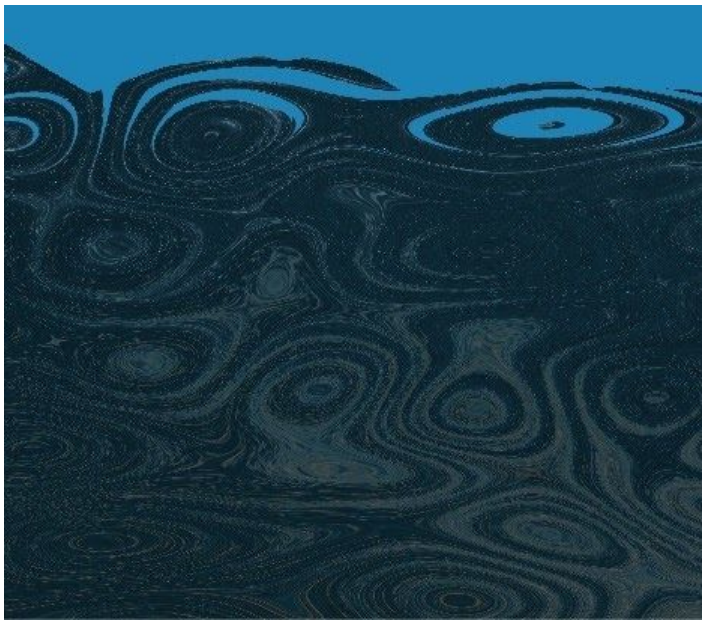
Port de noiseSimplex pour Unity
https://forum.unity.com/threads/2d-3d-4d-optimised-perlin-noise-cg-hlsl-library-cginc.218372/
Implémentation du shader
https://www.youtube.com/watch?v=f-emLwDhI2E

Shader de distorsion sur la caméra (UnderwaterCameraEffect)

Pour que le joueur semble être sous-l'eau un effet de distorsion est appliqué sur la caméra principal. Cette effet dans le prototype est assez subtile et est principalement visible lorsque le joueur ne se déplace pas. L'effet créer une simple distorsion qui semble faire bouger les objets prêt du joueur et donne l'impression d'être sous-l'eau.

Il faudra décider si cette effet pourra bien fonctionner dans notre jeu surtout durant les combats. Il ne faudrait pas implémenter un effet visuel qui pourrait déranger au gameplay (ou au moins permettre au joueur de ne pas l'utiliser).

Plusieurs variante beaucoup plus exagéré de ce shader son aussi possible et pourrait être utilisé:



Par exemple: un ennemi qui attaque le sous-marin et ferait réduire la vision du joueur pendant quelques fractions de seconde.

Ce shader utilise aussi le bruit de Perlin et le port de noiseSimplex pour unity.

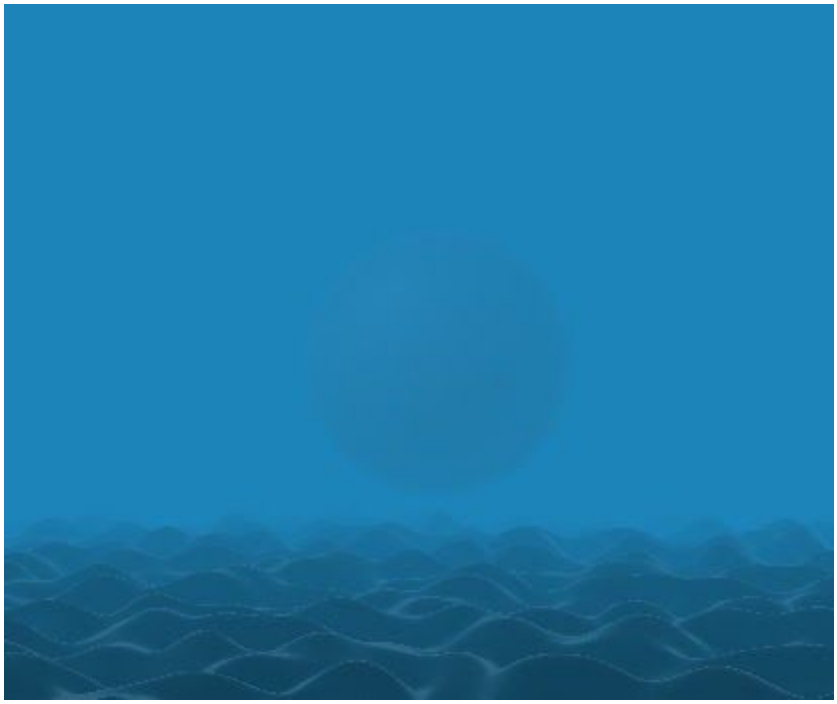
Source:

Implémentation du shader
https://www.youtube.com/watch?v=yXu55U_rRLw

Shader de fog sur la caméra (FogCameraEffect)

Puisque la visibilité est réduite sous-l'eau il est important de réduire la visibilité du joueur. Un shader de fog a donc été implémenté pour permettre de tranquillement réduire la visibilité des objets d'après la distance entre l'objet et le joueur. Le shader se sert de la fonction Linear01Depth pour capturer la distance entre la caméra et l'ensemble de la géométrie de l'environnement et ainsi réduire la visibilité de chaque objet d'après la distance entre celui-ci et le joueur.

Une sphère blanche qui est loins du joueur et est donc très peu visible à cause de la brume:



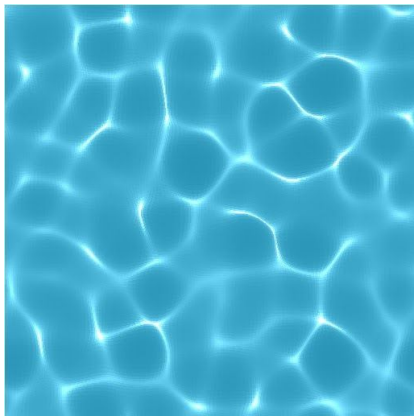
La distance à laquelle la brume commence à affecter les objets ainsi que sa couleur peuvent être modifié. Nous allons donc pouvoir créer des variations du shaders pour des environnement de couleur et d'ambiance différente.

Source:

Implémentation du shader
https://www.youtube.com/watch?v=XD2ryuF6leA

Effet de caustique en utilisant un projector (Caustics Projector)

Les caustiques créer par les rayons de lumière sur la surface de l'eau sont généralement visible près de la surface. Pour implémenter cette effet j'ai utilisé un projector du standard asset de Unity et j'ai générés 16 images de caustics différents avec le programme Caustics Generator. Exemple d'une des images utilisés:



C’est images sont alterner à chaque changement de frame ce qui créer un effet de mouvement. Il est possible de modifier la vitesse en modifiant le ratio d’image par frame.

Puisqu’un projector de Unity est utilisé il est très facile de seulement projeter les caustiques sur les objets voulu. Il sera donc facile de seulement affecter les objets désiré et d’éviter les objets qui pourrait potentiellement mal réagir à cette effet visuel qui est principalement conçu pour les surface plane.

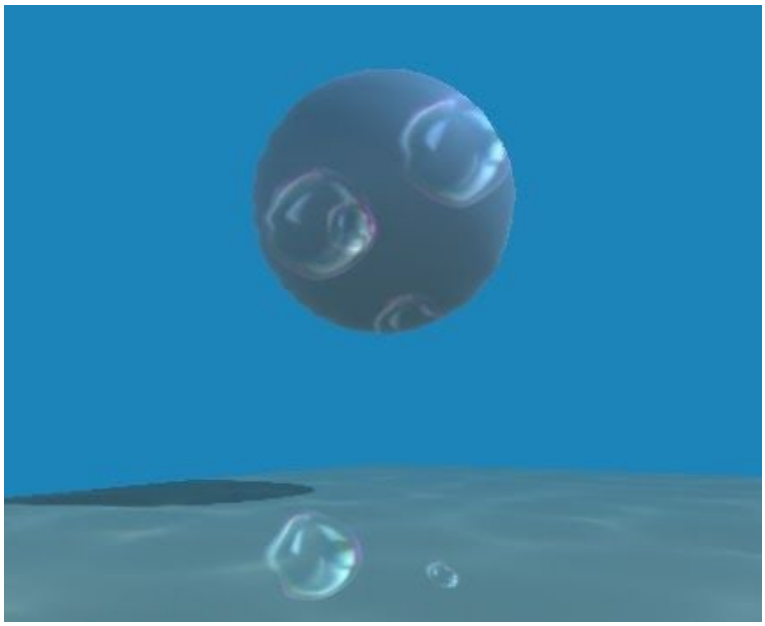
Source:

Implémentation du projector
https://www.youtube.com/watch?v=GHYUJO8P4_Y
Caustics Generator
https://www.dualheights.se/caustics/

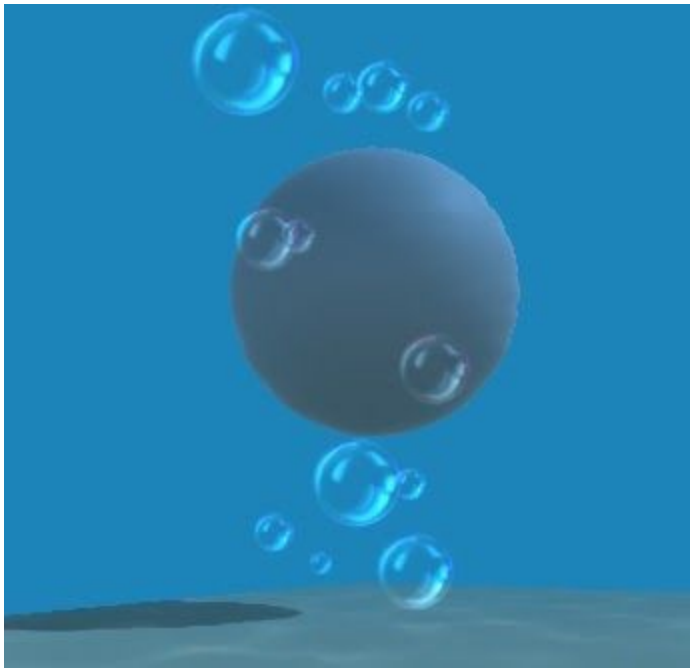
Effet de particules (Bubbles Particles)

J’ai téléchargé le package jiggly bubble free sur le Unity store pour avoir une image de bulle déjà prêt pour faire un effet de particules. J’ai essayé les prefabs contenu dans ce package mais j’ai préféré implémenté l’effet pour avoir une meilleur idée du fonctionnement et pour plus facilement modifier l’effet d’après l’utilisation que l’équipe pourrait vouloir en faire.

Le premier problème majeur que j’ai eu est l'interaction entre le shader de fog et les particles. Puisque le shader de fog retourne seulement les objets proche de la caméra, les particules sont coupé lorsqu’elles embarquent sur le fond de la simulation:



J’ai décidé d’ajouter une deuxième caméra qui est enfant à la caméra principal et qui traite uniquement des effets de particules. J’ai créer une layer de caméra additionnel pour les particules qui a réglé le problème:



Cependant, j’ai du utilisé Don`t Clear de Clear Flags dans les options de ma 2ième caméra ce qui entraîne d’autre problème visuel. J’ai du ajuster les Clipping Planes pour arriver à un juste milieu mais cette solution n’est pas idéal puisque les bulles disparaissent/apparaissent lorsqu’ils

ne devraient pas. J'ai essayé d'utilisé le Clear Flags Depth Only mais l'Occlusion Culling ne fonctionne pas avec les objets des autres layer de caméra.

Une autre solution serait de modifier le shader de fog pour toujours retourner les particules complète mais je suis incertain si cela est possible ou comment faire.

Je vais continuer d'explorer sur ce sujet en espérant trouver une meilleur solution soit par le shader ou par la deuxième caméra.

Source:

Texture de bulle:
https://assetstore.unity.com/packages/vfx/particles/environment/jiggly-bubble-free-61236

Pour aider au fonctionnement du prototype

J'ai trouver un script sur les forum de Unity pour permettre le déplacement de la caméra et faciliter l'utilisation du prototype

J'ai aussi ajouté un menu pour résumer/quitter le prototype.

Source:
https://forum.unity.com/threads/fly-cam-simple-cam-script.67042/

Conclusion

Je pense que plusieurs éléments de mon prototype pourront être utilisés dans notre projet après avoir été modifiés pour convenir à l'ensemble de l'équipe. Malgré le problème des particules avec le shader de fog, le reste du prototype à bien fonctionné et donne une bonne impression d'être sous-l'eau.

Autre Source

Texture de sable:
https://3dtextures.me/2017/03/23/sand-002/

Défi 6 - Intelligences artificielle des créatures sous-marines. (Nicolas Mori)

Rappel du défi

Pour que le joueur trouve la simulation stimulante, les créatures rencontrées devront avoir des comportements travaillés.
Différentes intelligences artificielles seront alors mises en place pour que les créatures, en fonction de la profondeur, se comportent plus ou moins agressivement.

Contexte du prototype

Dans le but d'avoir une simulation intéressante, un des défis est de réaliser les comportements des créatures rencontrées. Pour faire un prototype présentant cette fonctionnalité, le déplacement de modèle 3D dans l'espace est suffisant, cependant quelques éléments ont été ajoutés pour rendre la démonstration plus représentative.
De ce fait, un fond marin a été créé (terrain Unity) avec une texture de roche, le sous-marin représenté par un cylindre suit les rotations et les déplacements de la caméra guidé par l'utilisateur. Des modèles 3D libre de droit sont utilisés pour les entités sous-marins mais seront remplacés dans le futur par des modèles en adéquation avec l'univers de notre jeu.

Les contrôles

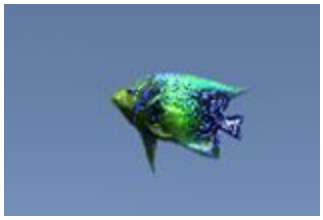
L'utilisateur peut orienter la caméra utilisant la souris.
Le déplacement se fait en utilisation les touches Z (avancer), S (Reculer), Q (Gauche), D (Droite) ou se servant des flèches directives.
La touche « Echap » permet de quitter la simulation.

Les différentes intelligences artificielles

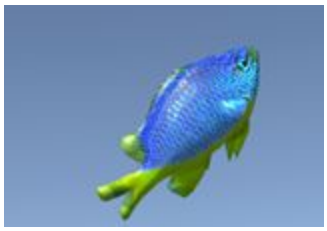
Chaque entité retourne à sa position d'apparition lorsque elle s'en éloigne trop (20 unité) et, si le joueur s'éloigne de manière importante d'elle (50 unité) leurs comportement est désactivé et le modèle n'est plus affiché de manière à optimiser les performances.

Intelligence passive : Les entités se déplacent dans l'espace dans des directions aléatoires et changent leurs orientations toutes les 3 à

10 secondes. Elles ne changent pas leurs comportements lorsque le joueur se trouve à proximité.



Intelligence fuyarde : Les entités se déplacent dans l'espace dans des directions aléatoires et changent leurs orientations toutes les 3 à 10 secondes. Si le joueur se trouve à moins de 5 unités, il s'éloigne le plus possible de lui en augmentant sa vitesse.



Intelligence agressive : Les entités se déplacent dans l'espace dans des directions aléatoires et changent leurs orientations toutes les 3 à 10 secondes. Si le joueur se trouve à moins de 12 unités, il le prend en chasse en se dirigeant vers lui de façon rapide.

Vidéo de démonstration

Youtube : https://youtu.be/QoZQrDpp39M

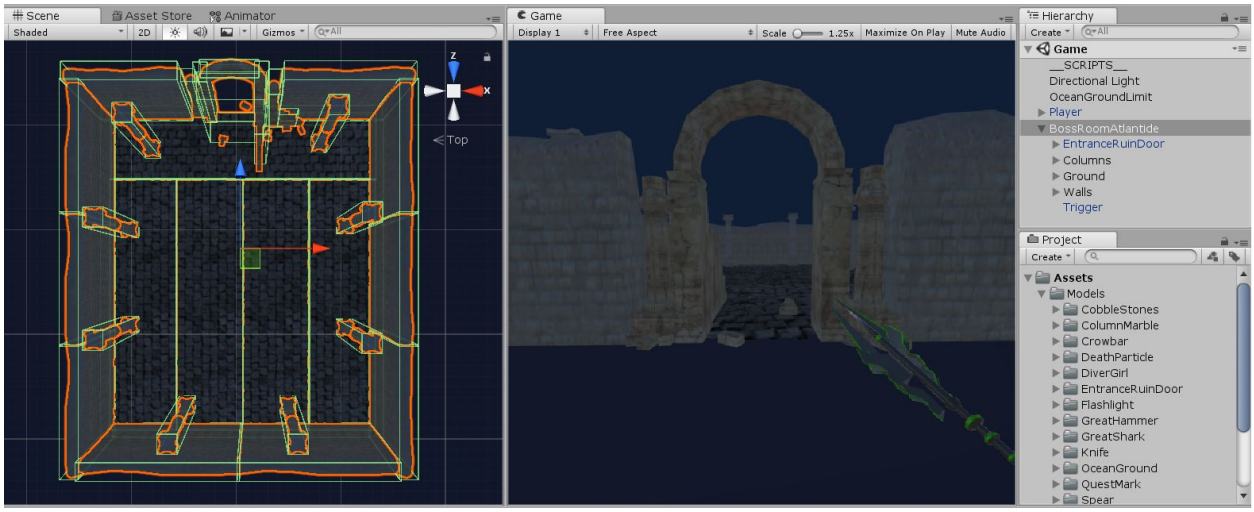
Défi 7 - Système de combat.
(Maxime Leroy)

Démonstration et lien de téléchargement dans la description :
https://www.youtube.com/watch?v=wWvcW78KS04

Environnement

Concernant le choix du décor dans lequel le joueur va se confronter à des monstres, il se trouve que l'Atlantide étant une île mythique qui fut engloutie dans un cataclysme selon les dires de Platon, m'a permis de baser toutes mes décisions sur un style plutôt grec voire antique.

Dans le prototype réalisé, nous nous retrouvons dans les profondeurs de l'océan aux abords d'une salle de boss où tous les assets utilisés ont été dotés de "box collider":



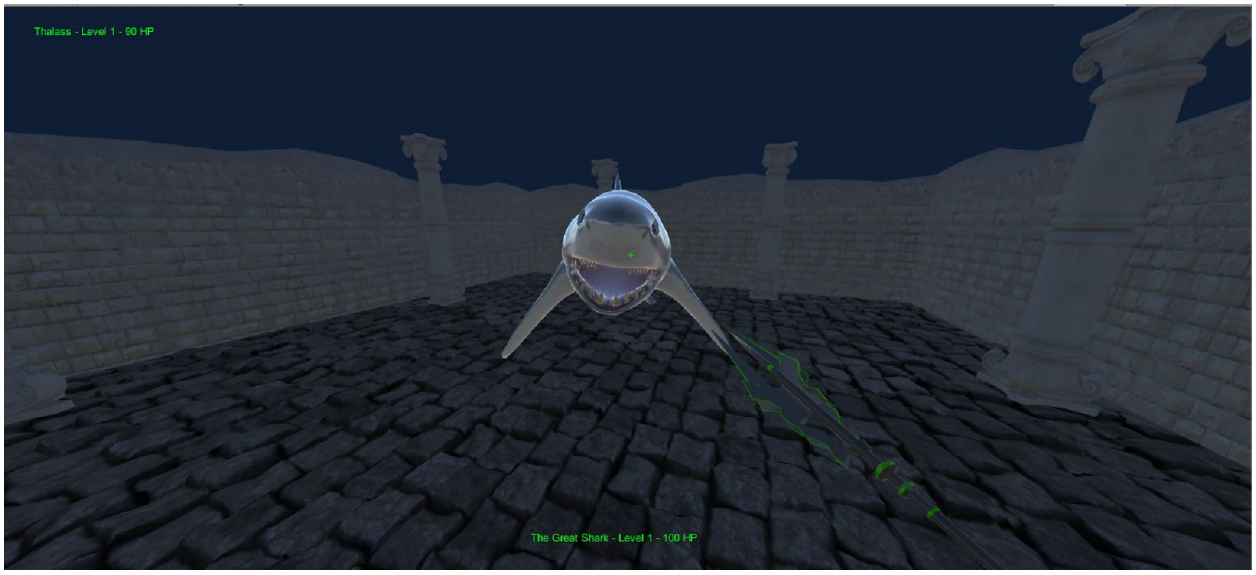
Interface

Pour guider notre personnage principal dans sa quête pour la survie, des éléments visuels (HUD) ont été implémentés afin de disposer d'informations sur l'environnement dans lequel il se trouve. Notamment les noms de ses adversaires, leur niveau, et points de vie.

Les détails du joueur sont affichés en permanence afin que celui-ci ne se retrouve pas malencontreusement dans une situation où le monstre qu'il combat soit d'une difficulté trop élevée.

Un crosshair destiné à la visée est présent au milieu de l'écran du joueur, toujours dans une optique de confort. Il est utilisé pour le moment afin d'afficher les détails de l'entité (ennemie) où le viseur est orienté. Quand il regarde dans une autre direction sans importance, les détails se retirent pour ne pas afficher d'éléments qui pourraient "flooder" le contenu de l'écran ou être inutiles. Ci-dessous, une idée de ce à quoi pourrait ressembler l'interface finale:

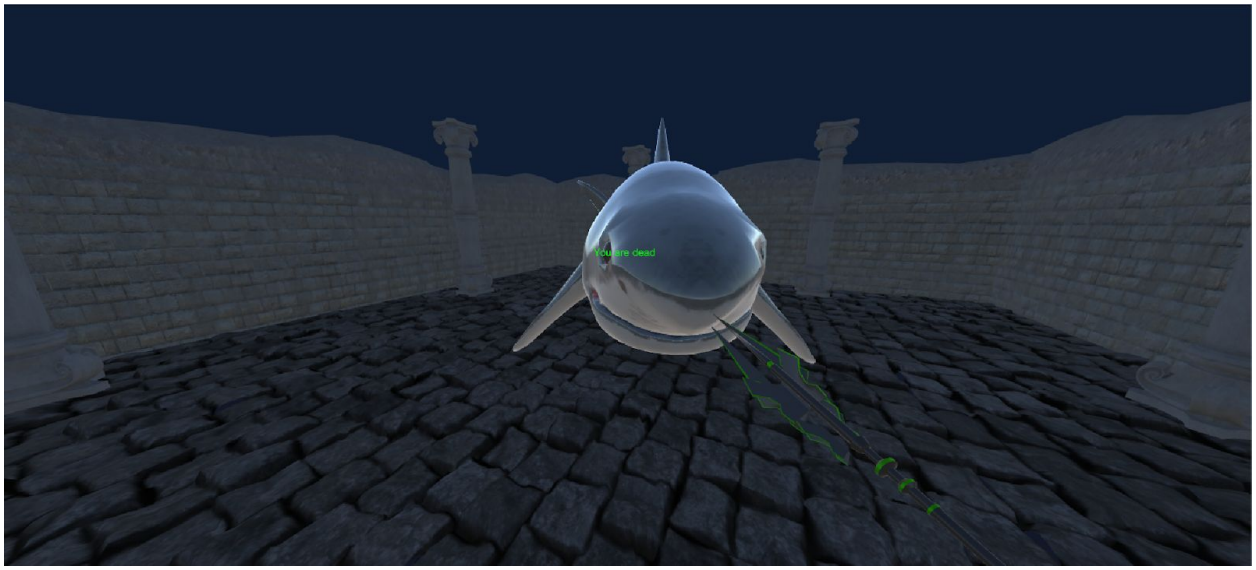
Combat



Les informations accessibles se retrouvent dans la classe “EntityController”, qui est parente de chaque classe d’agents: “PlayerController”, “EnemyController”, “WeaponController”...

Les données sont les suivantes: Base initiale de vie, Base initiale de dégâts, Niveau, Vie, Dégâts, Pseudo, État de vie et de mort.

Mort du joueur



Évènement

Pour permettre l’apparition du boss qui est un grand requin blanc (voir images ci-dessus), une hit box invisible placée à l’entrée de la salle sert de trigger unique à son instanciation.

Lorsque les points de vie du joueur tombent en dessous de la barre des zéros, la caméra se bloque et toutes les informations du HUD sont remplacées en son centre, par un message indiquant le décès de celui-ci. (voir images ci-dessus)

À l'inverse, quand le joueur réussit à vaincre son adversaire, il gagne de l'expérience et peut potentiellement passer au niveau supérieur, ce qui signifie que toutes ses statistiques sont augmentées proportionnellement. Par exemple, les stats actuelles sont définies de la sorte:

- Vie = niveau * base de vie initiale
- Dégâts = niveau * base de dégats initial

Camera

La vue est à la première personne avec un angle de 90° pour se rapprocher le plus possible d'une sensation de réalité (FPS). Le joueur a la possibilité de régler son angle d'orientation de sa vue verticale entre deux valeurs (minimale et maximale).

Aussi, il peut régler la sensibilité de déplacement de la caméra que ça soit sur l'angle vertical ou horizontal. Le contrôle de celle-ci peut également être bloqué, ce qui est utile notamment lorsque le joueur décède pour l'empêcher de bouger et rendre la simulation la plus réaliste.

Animation

- Joueur:
  - Nage normale
- Grand requin blanc:
  - Nage normale
  - Nage rapide (pourchasse)
  - Attaque
  - Mort
- Décès d'une entité:
  - Concentration d'énergie (Disparition graduelle)

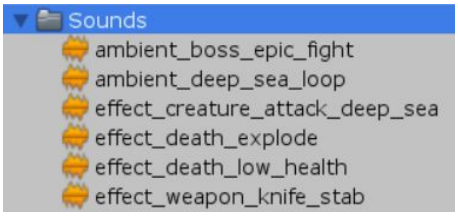
  - Explosion de particules d'énergie

Arme

Comme vu précédemment, chaque agent dispose de la classe qui découle de “EntityController”, et bien c’est exactement la même chose pour les armes (HP, Level, Damage).
Chacune dispose d’une portée et d’un son spécifique joué lorsque l’on attaque.
Les armes actuellement disponibles sont les suivantes:

Son

Chaque son a un type spécifique relié à sa propre source de son:
Combat contre un boss = musique épique (combat)
N’importe quel moment = musique des fonds marins angoissante (ambient)
Attaque à l’arme blanche = poignarder de la chair (effet)
Attaque d’une créature = grondement animal sous-marin (effet)
Animation de mort = perte de points de vie (effet)
Animation de disparition de l’entité = concentration d'énergie et de relâchement (effet)



Défi 8 - Ambiance sonore
(Charles-Etienne Dion)

Bien que l'ambiance sonore soit très importante dans un jeu vidéo, il est très compliqué de la développer tant que le reste du jeu n'est pas terminé. De ce fait, un prototype sur l'ambiance sonore de notre jeu est assez simple. En effet, tous les bruitages reliés au combat, à l'interface utilisateur ou aux IA de notre jeu ne peuvent être faits, puisque dans mon prototype, ils n'existent pas encore. J'ai donc dû réaliser une ébauche de ce que ressemblerait l'ambiance sonore de notre jeu terminé. D'abord, une musique rappelant l'ambiance sous-marine se met à jouer dès le début du prototype. Ensuite, il est possible d'entendre des sons d'ambiance de déplacements d'eau en tout temps, ce qui rappelle au joueur qu'il est sous l'eau. Dès que le joueur se déplace avec les touches w, a, s et d ainsi que LeftShift et Space pour monter et descendre, un bruit de moteur se met à jouer pour rappeler au joueur qu'il se déplace en sous-marin. Quelques sphères sont disposés dans le niveau, lorsque le joueur entre en collision avec l'une d'entre elles, un bruit sourd d'impact métallique se fait entendre. Finalement, lorsque le joueur s'approche trop près de la grosse capsule rouge (Considéré comme un ennemi), une musique intense de combat se met à jouer. La musique douce revient lorsque le joueur s'en éloigne suffisamment. Pour quitter le prototype, il suffit d'appuyer sur échappe. Je crois que c'est le mieux que je puisse faire pour un prototype malheureusement. Évidemment, le son sera bien mieux géré lors du jeu final. De plus, par manque de musicien dans l'équipe, j'ai dû utiliser des banques de sons libres de droits trouvés sur internet.

Annexe 2 - Rapport d’avancement 1

Tâches complétés

Rapport d’avancement 1 : 3 mars 2019

Première version du document : 3 mars 2019

Prototypes : 3 mars 2019

Obstacles rencontrés

Tâches à venir

● Vérification des prototypes et choix des éléments à conserver : 4 mars (Sébastien Bruère / Marc Lauzon)

● Décision du style visuel et liste des ressources à créer et/ou trouver : 5 mars (Guillaume Dupe / Yann Bénard Desjardins)

● Intégration des prototypes au premier jouable : 8 mars (Nicolas Mori / Sébastien Bruère)

● Test et évaluation du premier jouable (1) : 15 mars (Maxime Leroy / Nicolas Mori)

● Rapport d’avancement 2 : 18 mars (Charles-Etienne Dion / Guillaume Dupe)

● Test et évaluation du premier jouable (2) : 25 mars (Yann Bénard Desjardins / Maxime Leroy)

● Vérification des mises à jour du document de design pour TP4: 25 mars (Marc Lauzon / Charles-Etienne Dion)

Quatrième livrable : Premier jouable (TP4) : 29 mars (Vérification en équipe)

Révision du document : 29 mars

Premier jouable : 29 mars

Rapport d'avancement 3 : 29 mars

● Révision de la liste des ressources à créer et/ou trouver : 30 mars (Sébastien Bruère / Guillaume Dupe)

● Test et évaluation du premier jouable publiable (1) : 10 avril (Guillaume Dupe / Nicolas Mori)

● Préparation à la présentation : 11 avril (En équipe)

Cinquième livrable : présentation des projet (TP5) : 12 avril

● Rapport d'avancement 4 : 19 avril (Nicolas Mori / Marc Lauzon)

● Test et évaluation du premier jouable publiable (2) : 21 avril (Charles-Etienne Dion / Yann Bénard Desjardins)

● Vérification des mises à jour du document de design pour TP6: 1 mai (Maxime Leroy / Sébastien Bruère)

● Test et évaluation du premier jouable publiable (3) : 1 mai (En équipe)

● Début Post-mortem : 1 mai (En équipe)

Livrable final : Premier jouable publiable (TP6) : 3 mai (Vérification en équipe)

Révision du document : 3 mai

Premier jouable publiable : 3 mai

Post-mortem : 3 mai

Annexe 3 - Galerie de production

Puisque nous n’avons pas encore décider ce que nous allons garder comme élément visuel de nos prototypes nous avons ajouter des images d’inspiration que nous allons remplacer au prochain livrable par les différents éléments visuels que nous avons trouvés et/ou créés.

Cité de l’Atlantide (https://hitek.fr/img/up_o/1111061455/hitek_2ddfc38ee55883d736bd29d22daa84ac_1542385580.jpeg)

Grand requin blanc (https://cdnfr1.img.sputniknews.com/images/103737/72/1037377230.jpg)

Lance de poséidon dans la mythologie grecque
(https://www.tribality.com/wp-content/uploads/2015/01/poseidon.jpg)