# Table des matières

# Liste des tableaux





# Listes des figures

## Abréviations

ALM : application lifecycle management

APA : *American Psychological Association*

CCU : concurrent users – joueurs concurrents

CORS : Cross-origin resource sharing

CREALP : Centre de recherche sur l'environnement alpin

IDE : *Integrated development environment* – Environement de développement

Json: JavaScript Object Notation – format de données

SP : story point

US : user stories

US : User stories

VCS : Version control source

WebGL : Bibliothèque (Library) de Graphismes Web

XAMPP: distribution de logiciel libre : X (cross) Apache MariaDB Perl PHP

XML: Extensible Markup Language – format de données

# 1. Introduction

Ce travail de Bachelor a été réalisé à la suite d'une demande émise par le CREALP - Centre de recherche sur l'environnement alpin - afin d'élaborer le *proof of* concept d'un jeu sérieux. Son but est de sensibiliser la population à la problématique de l'eau de manière ludique et pédagogique.

Ce document présente l'état de l'art des jeux sérieux qui ont une orientation développement durable. Il explique les réflexions, le déroulement du projet ainsi que le cheminement du travail fourni pour mener à bien la conception de ce jeu.

## 1.1. Contexte

Le CREALP est une fondation à but non lucratif basée à Sion qui « mène des projets de recherche appliquée dans le domaine des géosciences orientées principalement vers les problématiques de surveillance environnementale et de risques naturels. » (« Présentation - CREALP - Centre de recherche sur l'environnement alpin », s. d.) Dans le contexte de réchauffement climatique, les glaciers valaisans sont amenés à disparaitre dans le siècle à venir. Le glacier de la Plaine-Morte aura totalement fondu d'ici 2100. C'est à ce niveau que le problème de l'approvisionnement en eau des Valaisans se pose.

Afin de préparer et de sensibiliser le public et les politiciens à cette problématique, nous avons pensé à développer un jeu sérieux.

## 1.2. Objectif

L'objectif de ce projet est d'imaginer et concevoir un outil pédagogique sous la forme d'un jeu afin de sensibiliser la population aux problématiques citées ci-dessus. Il doit toucher un public varié. La technologie et la plateforme de développement sont à choix.

La modélisation 3D n'étant pas de notre domaine, le mandant a accepté que nous fassions abstraction de l'aspect graphique, l'important étant de délivrer un jeu fonctionnel.

# 2. Serious Game

## 2.1. Définition

Avant tout, il faut noter que le *serious game* ou jeu sérieux en français, est adapté à tous types de support, bien qu'aujourd'hui, il ait une notion presque exclusivement vidéoludique (Djaouti, 2012). Les jeux vidéo ne sont plus seulement destinés au pur divertissement du grand public. Aujourd'hui, nous voyons une multitude d'acteurs qui ciblent divers domaines. Ceux-ci vont garder l'aspect ludique du jeu vidéo et y ajouter une couche « sérieuse », essentiellement à des fins pédagogiques. Selon les concepteurs de jeu Chen & Michael, un *serious game* est « tout jeu dont la finalité première est autre que le simple divertissement » (Djaouti, 2012).

Le terme de *serious game* est assez méconnu. Lorsque nous avons parlé à nos proches de notre projet, ils nous ont tous demandé la signification de ce terme. Il y a une certaine confusion entre le terme *serious*, de l'anglais sérieux et *game* pour jeux (de Roquefeuil,

2017). Certaines personnes ont du mal à concevoir un jeu vidéo avec d'autre but que celui du pur divertissement.

Il ne faut pas confondre *serious game* et *serious gaming*. Ces deux termes ont un sens très différent l'un de l'autre. Le *serious gaming* consiste à détourner un jeu vidéo classique, dédié au divertissement, vers une utilisation pédagogique. Par exemple, le jeu *Sim City* , un simulateur de construction de villes, peut être utilisé comme support par les professeurs pour leur cours de géographie et urbanisme (Djaouti, 2012). Un jeu sérieux est nommé comme tel, si lors de sa conception, il a un autre but que de divertir l'utilisateur.

## 2.2. Type de serious game

Il existe divers genres de *serious game*. Chaque type a un objectif particulier. Julian Alvarez et Olivier Rampnoux proposent cinq classifications selon le but du produit : Advergaming, edutainment, edumarket game, jeux engagés et jeux d'entrainement et simulation (« Jeux sérieux, mondes virtuels - Typologie », 2018).

- Advergaming

Ce type de jeu a comme intention de promouvoir l'image d'une marque. Il a été particulièrement développé depuis l'émergence d'Internet et des smartphones qui permettent d'atteindre plus facilement le public cible.

Ce type de jeu existait également avant la démocratisation d'Internet. Certaines marques ont développé ou adapté des jeux sur des consoles de salon. En 1983, Coca-Cola a ainsi produit un jeu afin de dénigrer la concurrence Pepsi : *Pepsi Invader*. Il a repris le principe du jeu *Space Invader* (jeu consistant à tirer sur des aliens depuis un vaisseau spatial), mais a remplacé les aliens par des lettres formant le mot « Pepsi ». (Isatis, 2016)

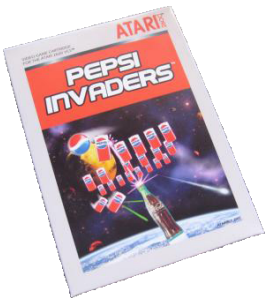



Figure 1: Pepsi Invader
Source : Tiré de https://level-1.fr/2016/04/advergames-quand-les-marques-font-leur-pub-avec-des-jeux-video/

- Edutainment

Ce genre a une visée éducative. Nous pouvons la retrouver sous diverses formes : Internet, console de salon, smartphone, etc. Cette catégorie de jeu s'est particulièrement bien développée sur la console portable de Nintendo : la Nintendo DS. Nous retrouvons beaucoup d'exemple, comme *Brain Age*, consistant à résoudre des exercices afin de stimuler son cerveau (« Brain Age », s. d.).

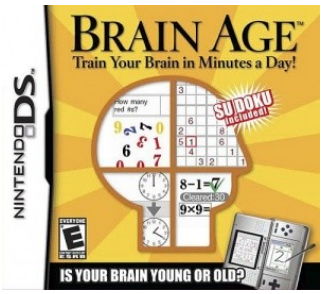



Figure 2: Jeu éducatif Brain Age
Source : Tiré de http://www.jeuxvideo.com/jeux/nintendo-ds/00029654-brain-age.htm

- Edumarket game

Ces jeux sont dédiés à la stratégie de communication. Le mot anglais “Edumarket game” signifie : « edu pour "education", de market "marché " et de game, "jeu" et pourrait se traduire par “jeu dont l’intention est d’éduquer sur un type de marché” » (« Jeux sérieux, mondes virtuels - Typologie », 2018). Ce genre est particulièrement destiné aux entreprises afin de familiariser leurs collaborateurs à un nouvel outil ou à l’apprentissage de nouvelles techniques de vente.

La marque de voiture Renault a chargé sa filière Renault Academy de mettre en place un *serious game* afin de former 15’000 commerciaux et parfaire leur technique de vente (2014).

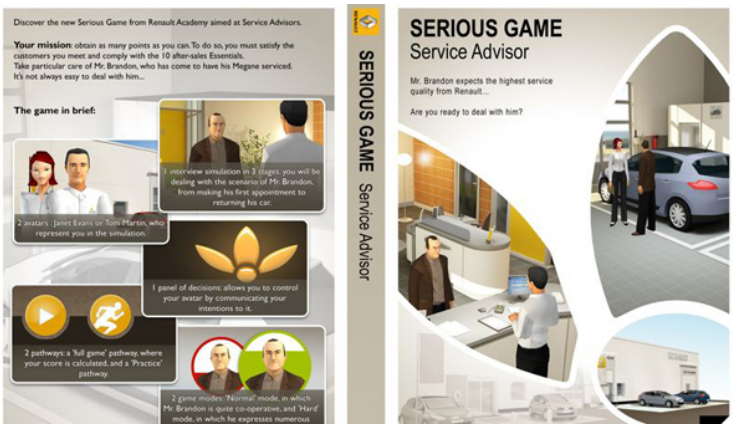



Figure 3: Edumarket Renault
Source : Tiré de http://myseriousgame.com/renault-forme-15000-commerciaux-serious-games

- Jeux engagés

Ces jeux sérieux vont utiliser le jeu vidéo comme support afin de dénoncer des problèmes politiques, sociaux, environnementaux, etc., soutenir une cause ou signaler des abus (« Jeux Engagés », s. d.). Par exemple, *Ice Flow* a été développé afin d’attirer l’attention du public sur le changement climatique. Le joueur doit garder en vie trois manchots alors que leur banquise fond.


Figure 4: Jeu engagé Ice Flow
Source : Tiré de http://www.serious-game.fr/ice-flows-sauvons-lantarctique/

Nous pouvons également mentionner un exemple de jeu engagé dans le but de sensibiliser à une cause humanitaire. Le Haut-Commissariat des Nations unies pour les réfugiés en Malaisie a sorti un jeu sérieux : *Finding Home*. Il met en lumière la cause des réfugiés Rohingyas (minorité musulmane persécutée et apatride de Birmanie) et des réfugiés de manière globale.

Dans ce jeu, nous nous retrouvons dans la peau d'un jeune réfugié Rohingays âgé de 16 ans, Kathijah. Grâce à un smartphone, nous devons trouver un refuge.(Mérancourt, 2017)

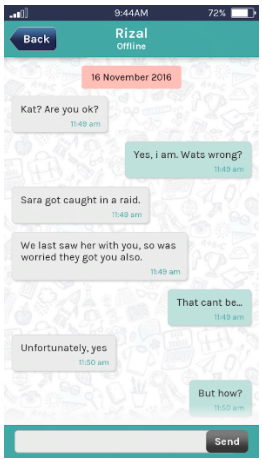



Figure 5: Jeu engagé Finding Home
Source: Tiré de (« Finding Home – Applications sur Google Play », s. d.)

- Jeux d'entrainement et simulation

Ce dernier type de jeu permet d'apprendre et de s'entrainer à l'exécution d'une nouvelle tâche. Le joueur va reproduire des mouvements, ou des actes s'inspirant du monde réel à l'instar de CiNACity.(« Jeux sérieux, mondes virtuels - Typologie », 2018)

CiNACity est un jeu de formation aux premiers secours conçu par des experts du monde de la santé. L'utilisateur va devoir adopter différents comportements en cas d'accidents et apprendre les gestes de premiers secours. Le jeu va simuler diverses situations où l'utilisateur sera amené, de manière virtuelle, à effectuer des massages cardiaques ou à réagir à un cas d'hémorragie (« CiNACity », s. d.).

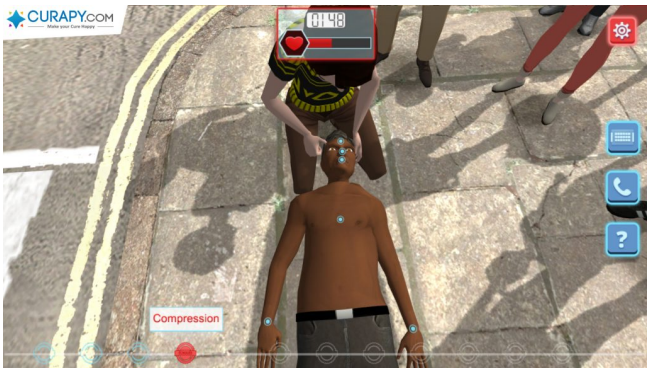



Figure 6: Jeu d'entrainement et de simulation CiNACity
Source : Tiré de https://www.curapy.com/jeux/cinacity/

## 2.3. Utilité

Nous pouvons nous demander si la conception d'un *serious game* est pertinente. En effet, le développement d'un jeu demande énormément de ressources tant humaines que financières (voir chapitre 5). Il est donc important de savoir si cette démarche ne doit pas être effectuée à travers des canaux basiques, comme la publicité, journée d'information et de sensibilisation, etc.

Les *serious game* ne sont pas forcément destinés à un public jeune. Si le jeu est bien conçu et simple, ce support a un impact sur toutes les classes d'âge. De plus, l'important n'est pas de produire un jeu aux graphismes sublimes et au *gameplay* (jouabilité) révolutionnaire, mais de donner envie aux joueurs de refaire des parties afin de les sensibiliser davantage (Février, 2015).

Dans une période où de nouveaux concepts pédagogiques apparaissent, il existe beaucoup d'outils d'apprentissages et il ne parait donc pas pertinent de se limiter uniquement aux anciennes méthodes. Durant notre cursus à l'HEVS, nous avons notamment fait l'expérience de l'enseignement via des MOOC (massive open online course). Une nouvelle classe sans prof et sans note a également vu le jour dans la filière Économie de l'HEVS (Gabbud, s. d.). Les *serious games* font partie de ces nouvelles méthodes. Ils permettent l'apprentissage de manière ludique à travers les jeux vidéo (Cohard, 2015). Ils représentent un atout important comme support pédagogique et de sensibilisation.

## 2.4. Conclusion type de serious game

Il existe plusieurs types de *serious games*, chacun ayant un but bien précis autre que celui du simple divertissement. Mise à part, l'*advertgaming*, utilisé à des fins publicitaires, nous pouvons souligner que tous les autres genres sont utilisés à des fins pédagogiques. Que ce soit l'*edutainment*, l'*edumarket game*, les jeux engagés ou les jeux de simulations, ils ont tous la volonté de faire apprendre à l'utilisateur le sujet du jeu de manière ludique.

Après analyse de ces différents genres et la discussion des besoins de notre client, détaillés au chapitre 3.1, nous avons décidé que notre jeu sérieux serait engagé. Le CREALP veut un jeu sérieux pédagogique, lié au développement durable, et plus précisément, à la problématique de l'eau en Valais. L'état de l'art va s'inspirer d'autres jeux engagés et plus particulièrement ceux qui traitent des problèmes environnementaux.

# 3. Analyse des besoins

La conception d'un *serious game* est un travail long et coûteux. Aux États-Unis, une version de démonstration coute 200 000 dollars, un jeu complet un million de dollars. En France, les chiffres s'articulent autour de 10'000-15'000 euros pour une version de test et 100'000-300'000 euros pour une version définitive (Méli, 2018).

En Suisse, la création d'un tel jeu varie entre 20'000 et 500'000 CHF (« Les jeux vidéo au service de la formation », 2017). Les montants dépendent bien évidemment des attentes du client.

Afin d'éviter tout risque d'échec, et de préparer le développement de l'application de la meilleure manière qui soit, il est impératif de préciser quelques points importants avant la réalisation :

- Comprendre les besoins du client.
- Établir un état de l'art.
- Choisir la technologie et la plateforme.

## 3.1. Besoin client

Lors de notre entretien liminaire avec le CREALP, nous avons été sollicités afin de concevoir un prototype, une version de démonstration d'un *serious game*. Après discussion, nous avons dressé une liste des diverses attentes et requêtes de notre client :

- Traiter de la problématique de l'eau en Valais.
- Sensibiliser et éduquer les utilisateurs.
- S'adresser à un public varié (toucher les personnes jeunes et moins jeunes).
- Être disponible en plusieurs langues du jeu.
- Texte parlé à l'introduction du jeu.
- Ajout de nouveaux scénarios de jeu.
- Répondre à des problématiques locales (les problèmes et solutions futurs de l'eau ne sont pas les mêmes par exemple dans la région de Crans-Montant, ou du Bas-Valais).
- Avoir une interaction entre les joueurs, et inciter à la collaboration.
- Recevoir des bonus/malus selon les décisions prises par les joueurs.

Ce *proof of concept* va permettre d'avoir un premier aperçu et les retours des populations cibles afin de débloquer éventuellement des fonds pour la réalisation « professionnelle » d'un *serious game*. Comme nous le verrons plus loin, bon nombre de corps de métier sont nécessaires à la conception d'un jeu sérieux et nous ne sommes qu'un « maillon de la chaîne ».

## 3.2. État de l'art

Afin de concevoir un scénario pertinent et de développer un *serious game*, nous avons effectué un état de l'art afin d'analyser les différents développements déjà effectués en rapport à notre type de jeu sérieux, pédagogique et engagé. Nous avons, dans un premier

temps, analysé la conception du graphisme, du gameplay, de la plateforme de développement, de la technologie et du type de jeu : quiz, puzzle, construction, etc.

Cette analyse nous a permis de poser des bases solides pour la conception de notre jeu sérieux et de sélectionner l'outil de développement ainsi que la plateforme qui est explicités plus loin. Pour une meilleure compréhension et analyse plus fine**,** nous nous sommes concentrés uniquement sur les jeux francophones. Les jeux sérieux analysés sont tous orientés développement durable/économie de ressource, mis à part le jeu *Datak* qui, lui, sensibilise les joueurs sur l'utilisation de leurs données. Cette recherche a eu lieu lors du *sprint* 0. Elle nous a permis de sortir les éléments-clés et nous a orienté vers une technologie de conception ou un type de *gameplay*. L'état de l'art nous a également servis comme source d'inspiration afin de concevoir notre *serious game*.

Nous avons consulté bon nombre de *serious games*, la plupart ont été découverts sur le blog dédié aux jeux sérieux, serious-game.fr : blog-catalogue regroupant et décrivant les jeux sérieux. Dans les paragraphes suivants, nous avons décidé de développer les quatre jeux qui nous ont servi d'inspiration par leur technologie, leur gameplay ou leurs graphismes.

### 3.2.1. Datak

*Datak* est le jeu qui nous a fait découvrir les *serious games* bien avant le début de ce projet. Il a été développé par DNA Studios en collaboration avec la RTS et lancé le 13 décembre 2016. Il a pour but de sensibiliser les utilisateurs à l'utilisation de leurs données (« La RTS lance Datak, le jeu qui interroge notre gestion des données », 2016).

Le jeu est conçu en 2D et dispose d'un graphisme « BD » très agréable à l'œil. L'utilisateur incarne un nouvel assistant stagiaire du maire et doit le conseiller sur la sécurité informatique, les attaques de hackeurs, les données personnelles, etc. À chaque choix, le jeu nous donne le pourcentage de joueur ayant répondu de la même manière.

Il s'agit d'un jeu WebGL. Ne connaissant pas les outils de développement, nous avons pris contact avec le studio DNA via leur formulaire internet. Mr David Hofer nous a expliqué les différents outils utilisés pour le développement de *Datak* : AngularJS, HTML5/CSS, VanillaJS, Phaser.

À cette étape de notre projet, ne connaissant rien du développement de jeu vidéo, ce premier contact nous a fait découvrir les *frameworks* jeu de JavaScript avec Phaser et d'effectuer des recherches dans cette direction. Phaser n'a pas retenu notre attention, car il est orienté jeu vidéo en 2D. Mais cela nous a fait connaitre Babylon.js, Three.js et Playcanvas qui seront analysés plus bas.

### 3.2.2. Wasteblasterz

*Wasterblasterz* est un jeu éducatif développé par la société DOWiNO, spécialisée dans la conception d'outils pédagogiques, de sensibilisation et de formation. Le but de ce jeu est de sensibiliser l'utilisateur à l'impact de son comportement sur l'environnement. Le jeu met en scène des « Wastivores » qui se nourrissent du gaspillage énergétique. Le joueur doit les

découvrir et les éliminer en répondant correctement à un quiz. En cas de mauvaise réponse, une nouvelle question est proposée jusqu'à ce que l'utilisateur réponde correctement.

Multiplateforme, ce jeu est disponible sur iOS, Android et navigateur web. Il a été développé grâce à Unity3D. L'icône d'agrandissement de la fenêtre de la version web est celle d'Unity3D. De plus, dans son code source, il utilise le fichier UnityLoader.js afin de charger le contenu web (« Unity - Manual », s. d.-b) .


Figure 7: Capture d'écran du jeu Wasterblasterz
Source : Adapté du jeu web Wasterblasterz - http://game.wasteblasterz.com/

Il propose un *gameplay* et une histoire simple afin de sensibiliser les gens aux problèmes liés au gaspillage sous toutes ces formes. Mis à part quelques interactions par l'utilisation d'outil, l'élément central de ce jeu est le quiz. Les questions sont variées et touchent diverses thématiques : consommation électrique, gaspillage d'eau, gestion des déchets.

Comme nous le verrons plus loin dans ce rapport, notre scénario est inspiré de divers jeux et simulations. *Wasterblasterz* a orienté notre décision de concevoir un *serious game* centré sur les quiz. Il nous a montré qu'avec quelques interactions simples, un jeu de questions peut devenir très intéressant. Il nous a également montré toute la puissance d'Unity 3D, qui permet un développement multiplateforme.

### 3.2.3. Compost Challenge

Ce jeu sérieux est également orienté sur le développement durable. La page de lancement annonce sa conception par Unity3D, il est disponible sur iOS et Android. Proposé par la

société OrgaNeo, spécialisée dans le conseil de compostage et d'écologie, il a été développé par Lozange Lab (« Lozange Lab », 2015).

Il a comme objectif d'enseigner les bonnes pratiques du recyclage. Des éléments du quotidien, canettes d'aluminium, trognons de pomme, marc de café, paquets de chips, bouteilles en verre et en pet, défilent sur un écran et le joueur doit les placer dans le bon seau : compost, ordure ménagère, verre, ou recyclable. Le but est surtout d'apprendre au joueur ce que nous pouvons mettre dans le compost, comment le griffer, le récolter et l'utiliser pour l'entretenir. Le matériel est ensuite utilisé pour concevoir et entretenir un jardin virtuel.

*Compost Challenge* a la particularité d'être collectif. Selon le code postal des utilisateurs, le jeu peut faire remonter les informations des utilisateurs. Ainsi, les différentes villes partenaires, comme Metz en France, peuvent utiliser *Compost Challenge* comme moyen supplémentaire de communication dans la sensibilisation de la population au recyclage (« Collectivités », s. d.).

Ce jeu nous a particulièrement touchés par ses graphismes. Simples et beaux, ils sont en parfaite adéquation avec la philosophie véhiculée. Pour notre projet, nous avons repris la position surélevée de la caméra. Cette vue, dite *bird eye*, permet aux joueurs de voir l'ensemble du plateau ainsi que la position des participants.

### 3.2.4. Les Maitres de l'Eau

*Les Maitres de l'Eau* est un jeu développé par la mairie de Paris. Son but est de sensibiliser les citoyens aux problématiques liées à la gestion de l'eau : comment est-elle captée, traitée, distribuée, consommée et finalement épurée ? (« Serious game "Les Maîtres de l'eau" », s. d.).

Ce serious game a été développé sous Flash, logiciel propriété d'Adobe. L'entreprise ayant annoncé la fin de leur logiciel emblématique des années 2000, cette technologie n'a pas été prise en compte pour la création de notre jeu (Lellouche, 2017). Le déclin et la fin de Flash Player s'expliquent par les différentes failles de sécurités du logiciel, certains malwares imitant le plugin de Flash afin de s'installer sur les navigateurs Internet des utilisateurs. L'HTML5 est également l'une des raisons, ce dernier fonctionnant nativement sur les navigateurs, il ne nécessite aucune installation particulière (Leblal, 2017). Enfin, Flash a très mal fonctionné sur les terminaux mobiles. Apple ne l'a jamais proposé sur iOS et Google a mis fin en 2012 au support de Flash sur Android (Lellouche, 2017).

*Les Maitres de l'Eau* nous a particulièrement plu par sa pédagogie. Nous en avons retenu l'alternance entre jeu et quiz. Au début de chaque manche, il propose un petit jeu sur un thème, par exemple un casse-tête comme le démineur.


Figure 8: Tour deux du jeu « Les maîtres de l'eau »
Source : Tiré du jeu « Les maîtres de l'eau » - http://www.eau.paris/game/

Le joueur accède ensuite à une petite encyclopédie qui servira de référence pour une série de questions. Il doit cliquer sur les différentes cases afin de découvrir les informations nécessaires dans chaque domaine lié à la gestion de l'eau.

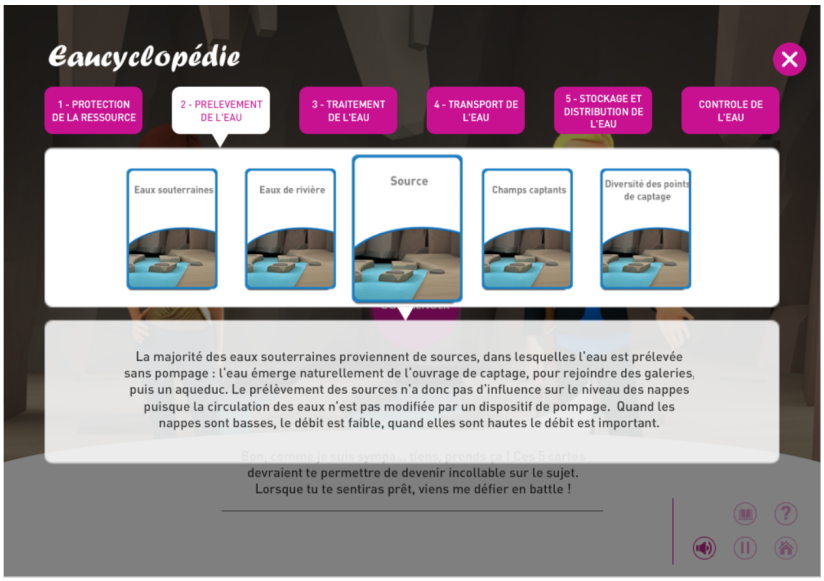



Figure 9: Mini-encyclopédie du jeu « Les maîtres de l'eau »
Source : Tiré du jeu « Les maîtres de l'eau » - http://www.eau.paris/game/


Figure 10: Quizz du jeu « Les maîtres de l'eau »
Source : Tiré du jeu « Les maîtres de l'eau » - http://www.eau.paris/game/

### 3.2.5. AquaCity Game

*AquaCity Game* est aussi un jeu dédié à la gestion de l'eau. Il a été développé par Interactive4D, et conçu pour le SICASIL - syndicat intercommunal de l'eau potable de l'agglomération cannoise. Il permet aux habitants de la région de comprendre la manière dont l'eau est acheminée jusqu'à leur robinet.

Ce jeu a également été développé avec Unity3D. Il est disponible sur iOS, Android et WebGL. Il faut noter que ce jeu ne fonctionne pas sur un navigateur mobile ou tablette, nous renvoyant à la version Android du jeu disponible sur le store comme le montre l'image ci-dessous.

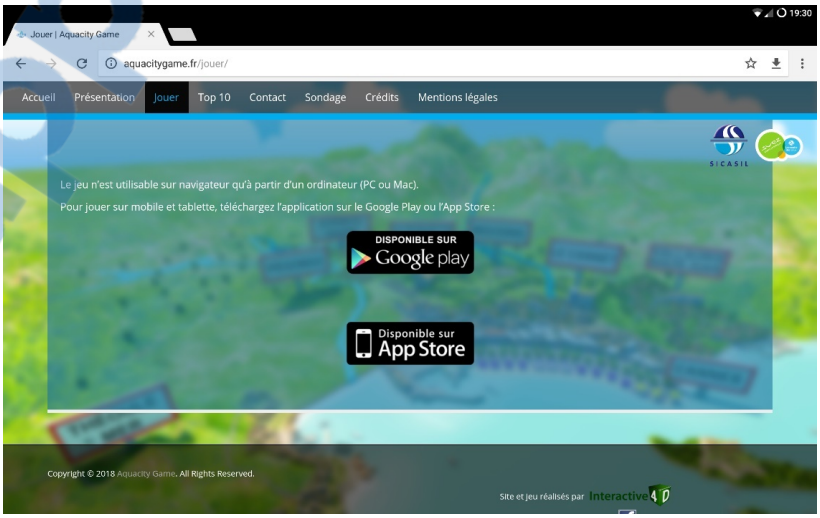



Figure 11: AquaCity WebGL sur tablette
Source : Tiré du jeu web AquaCity Game - http://aquacitygame.fr/jouer/

AquaCity Game propose au joueur de remettre en place les différentes étapes de la filtration de l'eau par un simple puzzle divisé en 8 cases.


Figure 12: Puzzle AquaCity
Source : Tiré du jeu web AquaCity Game - http://aquacitygame.fr/jouer/

Nous avons également apprécié l'utilisation de noms de villes familiers. Comme nous l'avons énoncé plus haut, ce jeu a été commandé par le SICASIL. Afin que le joueur puisse s'identifier davantage et s'immerger totalement dans le jeu, les développeurs ont utilisé des agglomérations existantes : Cannes, Mougins, Pégomas, etc.

En Valais, les problématiques de l'eau étant différentes d'une région à une autre et même d'une commune à une autre, il pourrait être intéressant et pertinent d'utiliser le nom des localités.

## 3.3. Plateforme de développement

Il est important de cibler la technologie et la plateforme de développement. Il ne faut pas lancer un développement sans réfléchir aux utilisateurs finaux :

- Qui sont-ils ?
- Comment consomment-ils ce type d'application ?
- Sur quel support ?

Comme nous l'avons énoncé au point 3.1, l'objectif de notre jeu sérieux et de toucher un public varié. Afin de cibler la meilleure plateforme possible, nous nous sommes demandé s'il était plus judicieux d'effectuer un développement mobile (application iOS/Android) ou web. De plus, comme nous l'avons vu au point 3.2, il y a autant de *serious games* sur une interface web que sur une application mobile.

En 2017, les utilisateurs consomment leur « temps digital » essentiellement depuis un smartphone (Coëffé, 2017). Nous pourrions donc légitimement penser qu'un développement mobile serait le plus adéquat.

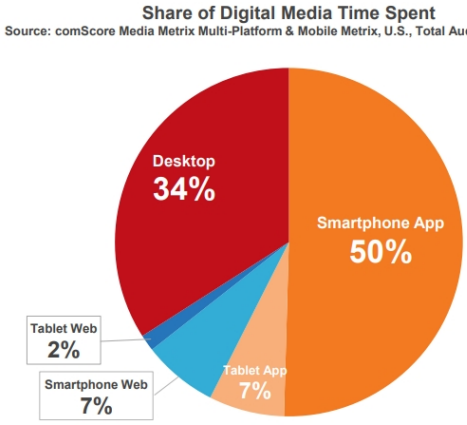



Figure 13: Utilisation "temps digitaux" 2017 selon les terminaux
Source : Tiré de (Coëffé, 2017)

Or, plus de la moitié de ces mêmes utilisateurs ne téléchargent en général que peu d'applications, seulement celles dont ils ont un réel besoin, comme les réseaux sociaux : Snapchat, Facebook, Uber. (Degeorges, 2016) Cet état de fait peut s'expliquer par la faible notoriété des autres applications, et le manque d'espace de stockage sur le téléphone.

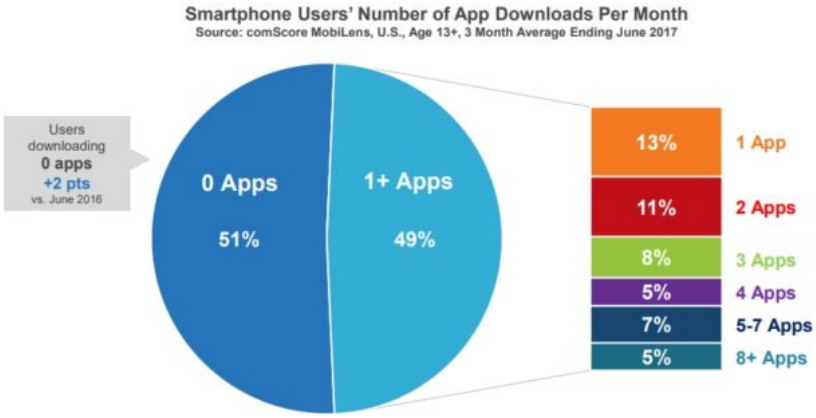



Figure 14: Pourcentage d'utilisateurs téléchargeant des applications par mois
Source : Tiré de (Coëffé, 2017)

Pour ces raisons, nous avons logiquement privilégié une version web. Bien que la population favorise plus les applications mobiles, elle ne délaisse pas encore les *desktops*, qu'elle utilise à hauteur de 34% (Coëffé, 2017). Par ailleurs, les personnes âgées, qui représentent une partie de notre public cible, recourent encore majoritairement aux appareils *desktops*, comme le montre la figure ci-dessous (voir Figure 15). Pour le reste de la population, nous remarquons que les 18-24 ans utilisent beaucoup plus les applications sur smartphone que leurs ainés. Il existe une corrélation entre l'âge et les plateformes d'accès à Internet. En effet, la tendance indique que plus les utilisateurs sont jeunes, moins ils utilisent d'appareils desktop. La domination du mobile peut s'expliquer par la forte mobilité de la population mondiale actuelle. Comme nous voyons sur la figure ci-dessous, mis à part pour la tranche d'âge 55-64 ans, plus nous avançons dans les tranches d'âge, plus l'augmentation de l'utilisation d'un *desktop* se fait sentir.

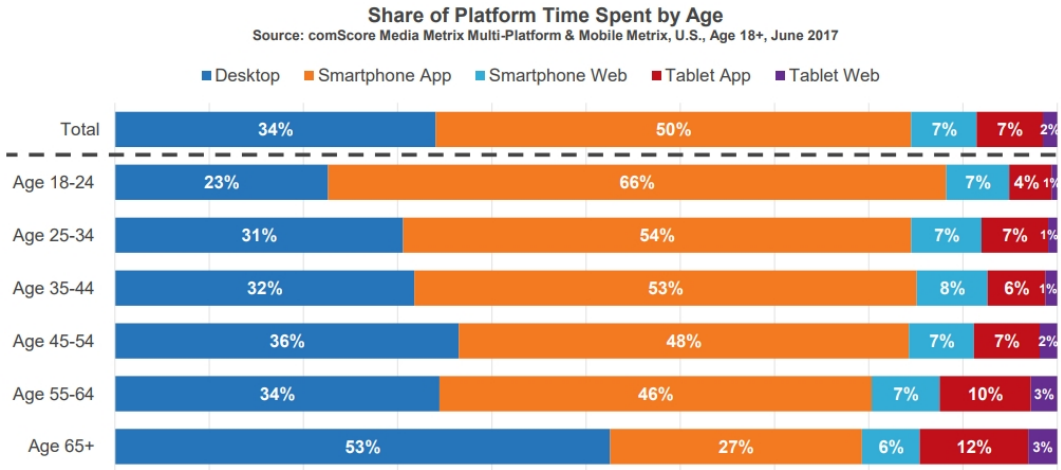



Figure 15: Utilisation mobile/desktop selon la tranche d'âge
Source : (Coëffé, 2017)

Après analyse de la situation, nous avons décidé de nous concentrer sur un développement web, qui offre une meilleure expérience utilisateur. Il est bien plus agréable de jouer sur un ordinateur avec un écran, un clavier et une souris que sur un smartphone. Comme nous le verrons dans le chapitre suivant, il est possible de mettre en place une application multiplateforme, afin d'offrir le choix à l'utilisateur. Faute de temps, nous avons dû nous contenter d'une seule plateforme.

La plateforme choisie, il reste à déterminer la technologie idéale. La conception d'un jeu au format web est possible en utilisant plusieurs outils différents, seuls ou de manière combinée.

## 4. Technologie web - WebGL

Notre application va être développée pour le format web, basé sur du WebGL (Bibliothèque de Graphismes Web). Le WebGL est une API, parente du OpenGL ES 2.0, qui permet de communiquer directement avec les cartes graphiques (« WebGL », 2018) via l'utilisation de la balise *canvas* d'HTML5. Grâce à la combinaison du WebGL et l'HTML5, les navigateurs Internet peuvent accéder aux ressources graphiques de l'appareil et générer du rendu 3D par l'intermédiaire du WebGL. (Beaucage, 26.02) Cet outil permet d'ajouter de l'animation à une page internet, de développer un site, un film interactif, comme *3 Dreams of Black* de Chris Milk (Beaucage, 26.02) et de créer des jeux vidéo.

WebGL est open source et multiplateforme. Il a pour seule contrainte, de disposer de l'utilisation d'un navigateur internet supportant le WebGL, ainsi que d'un appareil compatible OpenGL. Nous pouvons résumer le WebGL en du code écrit en JavaScript, exécuté dans une balise *canvas* d'une page HTML5. Les différents scripts vont être exécutés à l'intérieur de la balise *canvas*, comme le montre la figure ci-dessous.

```
<!DOCTYPE HTML>
<html>
    <head>
      <style>
        #mycanvas{border:1px solid red;}
      </style>
    </head>
    <body>
      <canvas id = "mycanvas" width = "800" height = "600"></canvas>

      <script>
        ...
      </script>

    </body>
</html>
```

Figure 16: WebGL, exemple balise canvas
Source : Tiré de (« WebGL Html5 Canvas Overview », s. d.)

Le WebGL est un langage très verbeux et complexe. C'est pourquoi une multitude de frameworks se sont développés afin de faciliter la conception. Pour notre *serious game*, nous avons analysé deux axes afin de choisir l'outil de développement le plus pertinent :

1. Un développement pur JavaScript.
2. L'utilisation d'un moteur de jeu qui va transpiler (traduire) en JavaScript.

Nous avons également dû faire un choix : développer le jeu en 2D ou en 3D. Nous avons opté pour une solution trois dimensions qui permet une plus grande dynamique de jeu lorsque celui-ci est joué à plusieurs comme notre *serious game*.

### 4.1. JavaScript

JavaScript est créé en 1995. Il n'est plus cantonné à une simple utilisation web. Depuis 2010, il est utilisé comme un langage de programmation au même titre que Java. Nous le retrouvons aussi bien du côté client que du serveur avec notamment Node.js (Ratignier, 2017).


Figure 17: Logo JavaScript
Source : Tiré de (« JavaScript », 2018)

De plus grâce à ses nombreux frameworks et l'HTML5, il permet la modélisation 3D et, par analogie, le développement de jeux vidéo, comme 3D City : une simulation de gestion de la ville comme Sim City, mais réalisé en pur JavaScript.


Figure 18: Jeu JavaScript- 3D City
Source : Tiré de (*3D.CITY*, s. d.)

Ce rendu est possible par l'utilisation de la balise *canvas* d'une page HTML standard. Les différents scripts vont s'exécuter à l'intérieur de cette balise. Nous trouvons deux catégories dans l'usage du JavaScript, les *frameworks* « purs code » et ceux où il n'est pas nécessaire de coder (une interface web permet une modélisation et une conception facilitée avec la possibilité d'effectuer des glisser-déposer) (Weinberg, 2016). Il existe une multitude d'outils et/ou *frameworks* basés sur JavaScript. Nous avons analysé les possibilités de concevoir notre jeu sur les deux *frameworks* les plus connus : Babylon.js et Three.js, ainsi que sur PlayCanvas, un outil/*framework* de conception online.

### 4.1.1. « Pure » code

#### 4.1.1.1. Babylon.js

Babylon.js est un moteur 3D, développé tout d'abord par des employés de Microsoft en 2013. (Mallet, s. d.). Il utilise JavaScript et TypeScript. Nous pouvons intégrer de l'audio et des éléments de physique. L'interface réseau n'est pas directement compatible, mais possible, avec une utilisation combinée de NodeJs et Socker.io.

Il est entièrement gratuit. Pour l'utiliser, il suffit de télécharger la bibliothèque selon nos besoins, et d'utiliser notre éditeur JavaScript favori.

#### 4.1.1.2. Three.js

Three.js est également une bibliothèque, open source et gratuite, JavaScript orienté WebGL. Il suffit de le télécharger et d'utiliser notre IDE Web de prédilection, comme WebStorm. Tout comme Babylon.js, cette bibliothèque intègre l'audio. Par contre, elle ne permet pas d'intégrer d'interface réseau, et ne prend pas en charge la notion de physique. Mais elle est disponible via une autre bibliothèque, Physijs.

Ces deux bibliothèques JavaScript ne permettent pas une visualisation directe des éléments développés, à moins d'utiliser des testeurs JavaScript comme « babylonjs-playground.com ». Comme nous devions, en plus de la programmation, concevoir l'aspect graphique de l'application sans aucune notion dans ce domaine, cela nous aurait demandé un temps bien plus long que celui à notre disposition.

### 4.1.2. Avec éditeur

#### 4.1.2.1. PlayCanvas

PlayCanvas propose un éditeur de code online. Le développement graphique et la construction graphique du jeu sont ainsi facilités. En effet, il permet d'importer facilement divers éléments graphiques et dispose également d'un *store* bien fourni où les éléments sont téléchargés directement dans le projet souhaité. Nous avons à notre disposition une documentation bien fournie comprenant des tutoriels, ainsi que des références à l'API.

Le moteur de jeu est open source, mais le stockage cloud et la plateforme web sont sous licence propriétaire. Cet éditeur permet de développer des jeux web que nous pouvons retrouver sur diverses plateformes. Ainsi, un jeu développé sur PlayCanvas peut être exporté selon la licence achetée au format HTML et installé sur nos propres serveurs ou déployé sur Facebook par exemple.

PlayCanvas permet également pour les licences payantes d'exporter et de compiler vers une version mobile à travers l'utilisation de CocoonJS (« Cocoon », s. d.)..

Il possède plusieurs types de licences. La version gratuite permet un usage limité : projet public seulement, 200Mb d'espace de stockage, et l'obligation d'utiliser la plateforme PlayCanvas sans aucun moyen d'exportation. La version « Personal » à 15 dollars le siège et par mois permet la création de projets privés, 1Gb d'espace de stockage, l'exportation HTML de notre jeu et l'héberger où nous le souhaitons. Enfin, la licence « Organisation » offre toutes les fonctionnalités de la plateforme : exportation du projet, accès à des APIs REST et, surtout, cette version permet un travail collaboratif : plusieurs personnes, sur plusieurs machines, peuvent effectuer des modifications.

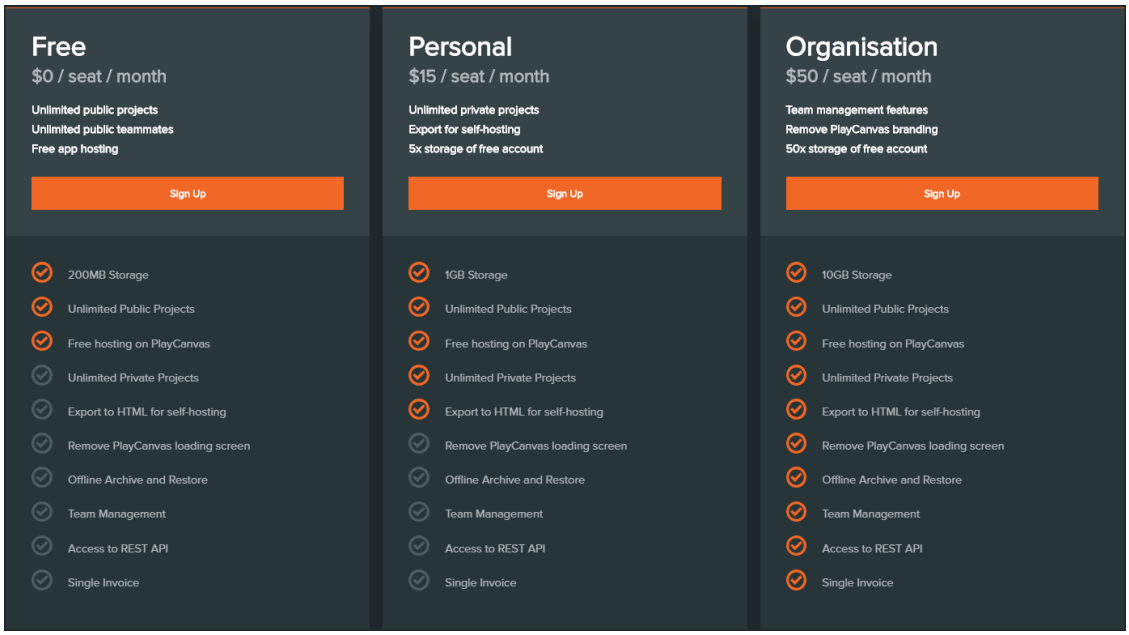



Figure 19: Licence PlayCanvas
Source : Capture d'écran tiré de https://playcanvas.com/plans

## 4.2. Moteur de jeu transpilé

### 4.2.1. Unreal Engine

Unreal Engine, ou UE4, à sa version 4 actuellement, est le moteur de jeu développé par Epic Game. Il est très connu actuellement pour son jeu *free-too-play*, Fortnite. C'est un moteur 3D utilisé par les plus grandes maisons de production.


Figure 20: Logo Unreal Engine
Source : Tiré de https://commons.wikimedia.org/wiki/File:Unreal_Engine_Logo.svg

Avec ce type de logiciel, nous sommes capables de développer un jeu vidéo multiplateforme. Unreal Engine permet de transpiler un développement et le rendre compatible avec Windows, Mac OS, Linux, WebGL, PlayStation, Xbox, etc. Il nécessite l'utilisation du langage C++, qui va être transpilé en JavaScript dans notre cas, afin de compiler une version WebGL. Il permet également le développement d'un jeu simple sans avoir à taper une seule ligne de code. En effet, son outil « Blueprint » offre la possibilité de faire du script visuel (Demay, s. d.). Il sert également à développer des applications en réalité virtuelle.

Auparavant payant, Unreal Engine 4 est maintenant gratuit et open source. Les entreprises voulant commercialiser un jeu doivent reverser 5% de *royalties* sur leurs chiffres d'affaires bruts s'ils gagnent plus de 3000 dollars par trimestre. S'ils ne souhaitent pas reverser de dividendes, ils ont également la possibilité de s'entendre sur une somme avec Epic Games et d'acquérir une licence personnalisée (« Unreal Engine Frequently Asked Questions », s. d.).

Unreal Engine 4 dispose d'un *store* et la documentation est très bien fournie, avec de nombreux tutoriels présents sur leur site Internet. La communauté s'est grandement développée depuis le passage à l'open source.

Bien qu'étant un excellent logiciel de développement, Unrel Engine utilise un langage de programmation méconnu du développeur. Nous aurions pu utiliser leur système *blueprint*, mais le but de ce projet est de développer avec un langage de programmation. De plus, il n'est pas forcément plus simple de travailler avec des scripts visuels. L'apprentissage et la maitrise de ce nouvel outil aurait nécessité plus de temps que celui dont nous disposons.

### 4.2.2. Unity 3D

Unity 3D est le second moteur de jeu multiplateforme qui permet de transpiler le code C# vers d'autres langages afin de le rendre compatible pour le WebGL par exemple. Auparavant, il disposait de son propre environnement de développement, monodevelopp.

Aujourd'hui, le développement C# peut s'opérer via deux IDE : Visual Studio Code et JetBrains Rider.

Tout comme Unreal Engine, il est utilisé par bon nombre de studios de production de jeux vidéo. Il permet la conception sur plusieurs plateformes : Windows, Mac OS, Linux, Android, iOS, réalité virtuelle, etc. et également en WebGL, même si dans leur application, il est mentionné HTML5, qui désigne un « ensemble de technologie web » : HTML5, CSS3 et JavaScript (« HTML5 », 2018).

Unity 3D est le seul des outils présentés qui n'est pas open source. Unity a récemment publié une partie du code source de leur moteur. Il l'a protégé par une licence très stricte qui ne permet que la lecture et aucune utilisation ou modification (goss, 2018). Il propose diverses licences : *Personal*, *Plus* et *Pro*.

La version *Personal* est gratuite. Elle permet d'utiliser les principales fonctionnalités d'Unity et des gains à hauteur de 100'000 dollars. Il limite les jeux multijoueur à 20 personnes simultanées.

La version *Plus* coute 35 dollars par mois et par ordinateur. Elle offre des avantages supplémentaires comme 20 % de réductions sur les articles payant de leur *store.* Il permet de modifier le logo au démarrage de l'application. (« Unity Plus », s. d.)

La version *Pro* coute 125 dollars par mois et par ordinateur. Elle permet 200 joueurs en mode multijoueur. Elle ne possède aucune limite de revenus. Elle inclut durant une année la fonctionnalité *Cloud Build*, service qui permet la compilation de l'application dans un *cloud* et sa mise à disposition des autres collaborateurs directement depuis l'interface web.

Unity propose également une version *Enterprise*, une licence *Pro* personnalisée, à destination des entreprises disposant plus de 21 membres.

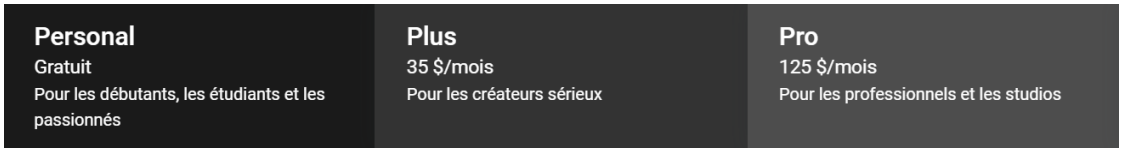



Figure 21: Licence Unity 3D
Source : Tiré de https://unity3d.com/fr/unity

Unity n'est pas open source, mais propose comme le montre l'image ci-dessous d'acquérir le code source sous condition très stricte et à leur bon vouloir, moyennant une somme que nous pouvons imaginer non-négligeable :

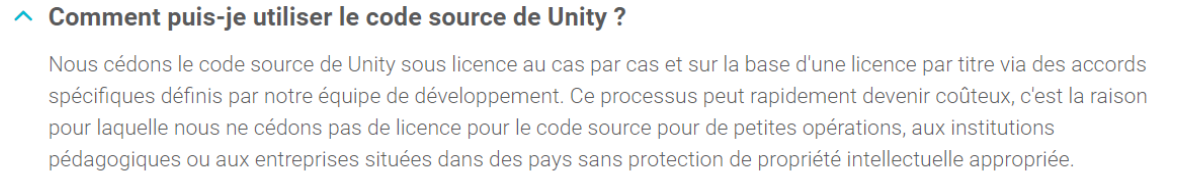



Figure 22: Information code source Unity 3D
Source : Tiré de https://unity3d.com/fr/unity/faq/2491

Tout comme PlayCanvas et Unreal Engine, il dispose d'un *store* où nous trouvons des personnages, décors, outils, et même des solutions complètes, payantes ou gratuites. La documentation est très bien établie et la communauté, de grande taille, est très réactive. Leur site Internet propose des tutoriels vidéo ou texte.

## 4.3. Technologie choisie

Après l'analyse des divers outils de développement, nous avons dû effectuer un choix. Les éléments qui ont été déterminants sont :

- La mise à disposition d'une interface graphique permettant une conception facilitée des graphismes.
- Un *store* : la modélisation 3D n'étant pas notre domaine, un magasin permettant l'intégration de différents éléments graphiques est déterminant, si possible gratuit.
- Une bonne documentation, des tutoriels, ainsi qu'une grande communauté.
- La légèreté du produit compilé.
- La connaissance du langage de programmation.

Les deux frameworks Babylon.js et Three.js, ne disposant pas d'une interface graphique directe, ont été écartés de notre choix. Il nous reste donc Playcanvas, Unreal Engine et Unity. Ils disposent tous trois d'une interface de production et d'un magasin. Certains sites internet proposent l'achat d'éléments graphiques comme turbosquid.com ou gamedevmarket.net qui peuvent être compatibles avec ces logiciels. Lors du développement d'un jeu sérieux, il peut être utile et nécessaire d'acheter des éléments graphiques afin d'accélérer la production ou si nous n'avons pas les ressources nécessaires : animateur et modeleur 3D. Mais nous avons jugé que l'achat d'éléments graphiques n'était pas adéquat pour la conception d'un prototype.

Le *store* de Playcanvas est relativement pauvre, moins d'une cinquantaine d'objets. Mais il renvoie vers d'autres sites de vente. Unreal Engine dispose de son propre magasin bien fourni. Nous trouvons des éléments de décors, des personnages, ainsi que des paquets de démarrage. Mais la plupart des éléments sont payants. Unity, quant à lui, possède un *store* très riche où nous trouvons aussi bien des éléments graphiques que des outils/plug-ins pour implémenter rapidement un mode multijoueur ou la traduction du jeu. Une large gamme de ces éléments est intégrable gratuitement.

Les trois outils disposent d'une très bonne documentation et de leurs références API. Ils proposent tous de très bons tutoriels afin d'avoir une bonne prise en main de leur outil. Bien

que l'anglais écrit ne nous pose pas de problème, Unity possède, en plus, une très grande communauté francophone (Herrenschneider, 2016).

La fluidité du jeu produit, ainsi que les tests de performance sont des éléments très importants afin de garantir la meilleure expérience de jeu à l'utilisateur final. Playcanvas propose un code JavaScript WebGL natif. Unreal Engine et Unity vont transpiler le code en WebGL. Nous nous retrouvons avec la transpilation de leur code respectif, C++ pour UE4 et C# pour Unity, avec une application lourde et des performances amoindries face au développement natif JavaScript permis par Playcanvas. Cette lourdeur se retrouve lors du chargement du jeu, ainsi que dans la taille de l'application. Selon Playcanvas, un projet réalisé sur leur plateforme pèse 0.22MB et le temps de chargement est d'une seconde. Sur Unity version 5.3.2, le même projet pèse 4.72MB, et met treize secondes à se charger. (« PlayCanvas versus Unity WebGLPlayCanvas », 2016) Un autre projet développé par Playcanvas pour se différencier d'Unreal Engin montre que le même type de projet pèse 0.22MB sur leur plateforme, contre 10MB avec l'application développée sur UE4 avec des temps de chargement sur Firefox 48 de 1.4 et 11 secondes respectivement (« PlayCanvas versus Unreal WebGLPlayCanvas », 2016).

Le tableau ci-dessous note les différents éléments de notre test. Pour le développement de notre application, nous avons opté pour Unity. Son principal défaut est la lourdeur de l'application une fois compilée en HTML5 en comparaison avec un développement natif. Mais ce problème n'est pas rédhibitoire. Un jeu WebGL développé avec Unity est tout à fait jouable. Nous avons finalement opté pour cet outil surtout pour son asset *store* qui propose une multitude d'éléments et d'outils gratuitement. Il est très bien documenté et de nombreux tutoriels sont proposés par les développeurs et leur communauté. Enfin, nous maitrisons bien le langage de programmation C#, l'apprentissage d'un nouveau langage aurait demandé bien trop de temps.

Dans un cadre professionnel avec des graphistes, des animateurs 3D, modeleurs 3D et d'autres personnes nécessaires à la conception d'un jeu vidéo (voir chapitre 5), nous aurions préféré utiliser Playcanvas. Mais, comme le résume le tableau ci-dessous, dans notre cas, Unity s'est révélé être la meilleure solution.

| Technologie | *Store* | Documentation | Légèreté de l'application | Connaissance langage du développeur |
|---|---|---|---|---|
| **PlayCanvas (JavaScript)** | ★☆☆☆☆ | ★★★☆☆ | ★★★★☆ | ★☆☆☆☆ |
| **Unreal Engine (C++, Blueprint)** | ★★☆☆☆ | ★★★☆☆ | ★★☆☆☆ | ☆☆☆☆☆ |
| **Unity 3D (C#)** | ★★★★☆ | ★★★★☆ | ★★☆☆☆ | ★★★★☆ |

Tableau 1: Comparatif technologie
Source : Données personelles

# 5. Conception de notre *serious game*

Comme nous l'avons vu, un jeu vidéo et un jeu sérieux ne se distinguent que par leur finalité, ludisme pur pour l'un et apprentissage, pour l'autre. Leur conception se révèle donc relativement similaire. La création s'effectue par la collaboration de plusieurs corps de métier. Nous retrouvons les métiers artistiques ainsi que ceux de la création, de la technique, de la postproduction et de la communication (« Les métiers du jeu vidéo », s. d.).

Dans les métiers artistiques, nous trouvons le directeur artistique, chargé de l'harmonie du jeu : « s'assurer que toutes les idées émises et appliquées s'assemblent de façon harmonieuse et réalisable. » (« Le Directeur Artistique - Dossier », 2012). Les graphistes s'occupent de l'aspect visuel défini par le directeur artistique (« Graphiste - Dossier », s. d.). Les animateurs sont chargés de donner vie à différentes animations pour la création des graphistes (« Animateur - Dossier », 2012). Enfin, les scénaristes imaginent l'histoire du jeu. Dans le cas de notre *serious game*, aux scénaristes, nous pouvons ajouter les experts dans les différents domaines abordés : géologues, hydrologues, glaciologues et politiciens, ainsi que des personnes issues des milieux pédagogiques afin de parfaire la sensibilisation.

Les métiers de la création comportent le directeur créatif, chargé de coordonner le travail entre les équipes, d'établir une charte graphique afin de définir l'univers du jeu vidéo (« Anthony Lejeune, directeur artistique dans les jeux vidéo », s. d.). Nous trouvons également les *game* designers, chargés d'élaborer la mécanique de jeu, les règles et le *gameplay* (la manière de jouer)(« Game designer », 2017).

Ensuite, nous avons les métiers techniques. C'est à ce niveau que nous nous trouvons, avec les programmeurs (intégrer les différents objets graphiques, sonores, jouabilité, etc.) et les testeurs (tester le travail des programmeurs et faire remonter les bugs). Nous avons également le *producer* qui est chargé de faire respecter le délai de la mise en production et de s'assurer que le jeu remporte un succès (« En quoi consiste le métier de producer de jeux vidéo ? », s. d.).

La postproduction est chargée des doublages des différents personnages, de la traduction du jeu ou encore de la musique. Les métiers de la communication s'occupent, quant à eux, de la promotion, de la diffusion et de la vente du jeu.

Comme nous le voyons, il existe une multitude de métiers gravitant autour de la création d'un *serious game*. Selon la taille du projet, nous avons des personnes spécialisées dans chaque domaine. Il est bien entendu probable qu'une personne ait plusieurs tâches qui lui soient attribuées. Dans notre cas, nous avons dû endosser plusieurs casquettes afin de concevoir notre *serious game*. N'ayant pas de base graphique, ni artistique et ne connaissant aucunement l'animation des personnages, nous avons été contraints d'utiliser des *assets* disponibles sur le *store* d'Unity.

Dans la suite de ce travail, nous traiterons, dans un premier temps, de la vision de notre jeu sérieux puis du scénario et des raisons qui nous ont amenées à le choisir, des règles du jeu, du graphisme envisagé et des animations. Nous exposerons ensuite le développement de l'application au moyen de l'outil choisi, Unity 3D.

## 5.1. Scénario et mode de jeu

### 5.1.1. Scénario

Afin de pouvoir intégrer différents moyens de sensibilisation et de pouvoir toucher autant les jeunes que les moins jeunes, nous avons proposé à notre client de développer un jeu de plateau du type « jeu de l'oie ».

Ce type de jeu répond à toutes les attentes énoncées plus haut au point 3.1. Il est intemporel et touche donc toutes les tranches d'âges. Il est facilement actualisable en y intégrant de nouvelles thématiques de sensibilisation à l'aide de cases supplémentaires. De plus, c'est un jeu qui peut se jouer à plusieurs et nous pouvons imaginer certaines interactions-collaborations entre les joueurs. Le mode multijoueur permet une interaction, de l'émulation et de la compétition stimulante. Notre *serious game* étant à destination des valaisans, notre scénario reprend des éléments familiers aux habitants de ce canton. L'histoire débute ainsi :

- Dans un futur proche, durant un hiver sans neige, un habitant de la plaine veut boire de l'eau, prendre un bain/douche. Quand il ouvre le robinet plus une goutte n'en sort. Lorsqu'il essaie de contacter le service des eaux, personne ne répond. Il décide donc de partir s'enquérir lui-même de la situation.

Dans un premier temps, le joueur choisit son personnage (tous les âges et sexes doivent être représentés) afin que le joueur puisse s'identifier.

Une fois son avatar sélectionné, le joueur part sur un chemin sinueux caractéristique du plateau de « Jeu de l'oie ». Les cases représentent les réserves d'eau et les services à la population de la région. Notre héros va à la rencontre des personnes en charge du secteur de l'approvisionnement pour connaitre la source du problème. Mais le parcours est semé d'embuches.

Nous avons pensé à un système de règles simples, comprenant quatre types de cases :

- Case standard : un quiz.
- Case « Eau » : spécifique sur le système des eaux -> un quizz, puzzle, jeu de construction thématique.
- Case « Village » : quiz en lien avec des problématiques liées aux différentes communes ou un petit jeu qui pousse à la collaboration entre les joueurs.
- Case « Spéciale » : raccourci ou retour en arrière sur le plateau.

Les déplacements s'effectuent avec un dé. Après chaque lancer et avant l'arrivée sur la case s'ouvre une épreuve didactique qui permet de valider le déplacement. Ces épreuves intermédiaires peuvent être en lien avec la case.

*Exemple : pour accéder à la case « Réservoir d'eau potable de Miège et région » nous est présenté un quiz sur la chaine de traitement de l'eau, présentant des questions sur les techniques de filtration.*

Ces épreuves intermédiaires permettent d'introduire des connaissances, de sensibiliser et d'entraîner la mémoire des joueurs qui pourront accéder au fur et à mesure de l'augmentation de leurs connaissances plus rapidement aux cases importantes. En cas d'échec, le joueur reste sur place et finit son tour. Nous pouvons également imaginer que l'échec d'entrée sur une case entraîne des conséquences et des désagréments.

*Exemple : en cas d'échec du jeu intermédiaire pour entrer sur la case « Commune de Miège », les pourparlers avec les politiques pour créer un nouveau réservoir d'accumulation n'ont pas lieu et la sécheresse perdure.*

Nous pouvons également prévoir des cases spéciales à l'image du jeu original comme le toboggan, remplacé ici par le funiculaire du SMC, des pistes de ski ou un téléphérique qui permettent de monter et descendre selon le résultat du challenge.

Enfin, nous trouverons des étapes clés sur les cases « Village» où la coopération des joueurs sera de mise afin de les sensibiliser à l'importance de la collaboration entre les communes.

Ce type de jeu permet une certaine évolution. Nous pouvons facilement ajouter de nouveaux challenges pour les questions aléatoires, rallonger le parcours afin d'approfondir la thématique et varier le niveau des questions.

Le jeu se joue au moyen d'un clavier et d'une souris afin d'avoir un *gameplay* le plus simple d'utilisation.

Ceci est notre scénario de base qui a été proposé et accepté par notre client. Il comprend l'intégralité de jeu tel que nous l'avons imaginé. Le délai étant trop court et des problèmes techniques étant apparus, nous n'avons malheureusement pu développer que la partie comprenant les cases « Standard ».

### 5.1.2. Mockup

Afin d'illustrer nos *users stories* (US) développées plus bas, nous avons réalisé des *mockups* pour dessiner notre conception du jeu. Il existe différents outils afin de concevoir un *mockup.* N'ayant pas trouvé celui qui convenait à notre projet, nous avons choisi de les dessiner à la main.

Afin de faciliter la numérisation, nous avons utilisé Rocketbook. Cette application propose la numérisation de texte, image, schéma, etc. depuis un cahier effaçable, de manière simple et efficace.


Figure 23: Logo Rocketbook
Source : Tiré de https://www.virginmegastore.ae/brands/ROCKETBOOK

### 5.1.3. Quiz

Dans notre état de l'art, au point 3.2, nous avons fait ressortir les différents types de jeu qui embrassent l'idée de *serious game*. Nous voulions amener plusieurs types de jeu afin de rendre l'application la plus ludique possible. Ayant rencontré des problèmes lors de la conception de notre jeu sérieux, nous avons dû revoir et limiter notre scénario. À ce stade de développement, nous nous sommes concentrés sur un type de case seulement : les cases « Standard ». Lorsque le joueur arrive sur une case, une question s'affiche.

Les quiz sont un élément de base dans la pédagogie. Ils n'ont pas comme unique but d'être utilisés à des fins d'évaluations. Ils ont un réel pouvoir de formation. Selon Julien Theler, les questionnaires sont « stimulant[s], adapté[s] à différents niveaux et facile[s] à maintenir à jour » (Theler, 2015). Lors de la rédaction de ce rapport, nous avons estimé que pour parfaire l'enseignement et la sensibilisation par les quiz, en cas de bonne réponse, nous pourrions afficher une petite note explicative.

*Exemple : À la question 24 en Annexe III Quizz fourni par le CREALP : Comment est-il possible de savoir où s'écoule l'eau de fonte d'un glacier ? (Réponse correcte en vert)*

a. *À l'aide d'essais de traçage*
b. *À l'aide d'une cartographie détaillée*


Figure 24: Note explicative du quiz
Source : Données personnelles, texte central tiré de Wikipédia (« Traçage (hydrogéologie) », 2017)

Notre jeu de plateau dispose de 63 cases. Afin d'avoir suffisamment de matière, nous pensons qu'il faudrait deux à trois fois plus de quiz que de cases. Ceci dans le but de ne pas lasser le joueur après sa deuxième partie car il connait déjà certaines questions et réponses. Pour notre prototype, nous ne disposons que de 24 questions, c'est pourquoi elles se répètent aléatoirement. Les différents quiz nous ont été fournis par le client avec, à chaque fois, les bonnes et les mauvaises réponses.

Pour le côté technique, nous avons pu nous appuyer sur la documentation et les tutoriels d'Unity qui sont, comme mentionné auparavant, très bien fournis. Notre quiz se base sur un *live session* d'Unity, disponible à cette adresse :

https://unity3d.com/fr/learn/tutorials/topics/scripting/intro-and-setup (« Intro and Setup », s. d.).

Dans notre projet, les quiz sont composés de deux scènes : une appelée « 08A.Persistent », qui contient un *GameObject* où nous trouvons l'ensemble des questions écrites en dur avec la bonne et les mauvaises réponses, et une autre nommée « 08B.Quizz », où les questions sont affichées de manière aléatoire. La prochaine étape serait de charger l'ensemble des questions depuis un fichier Json. Cette méthode rend l'ajout, la modification et la suppression des questions plus aisés. Il faut également que les quiz soient multilingues.

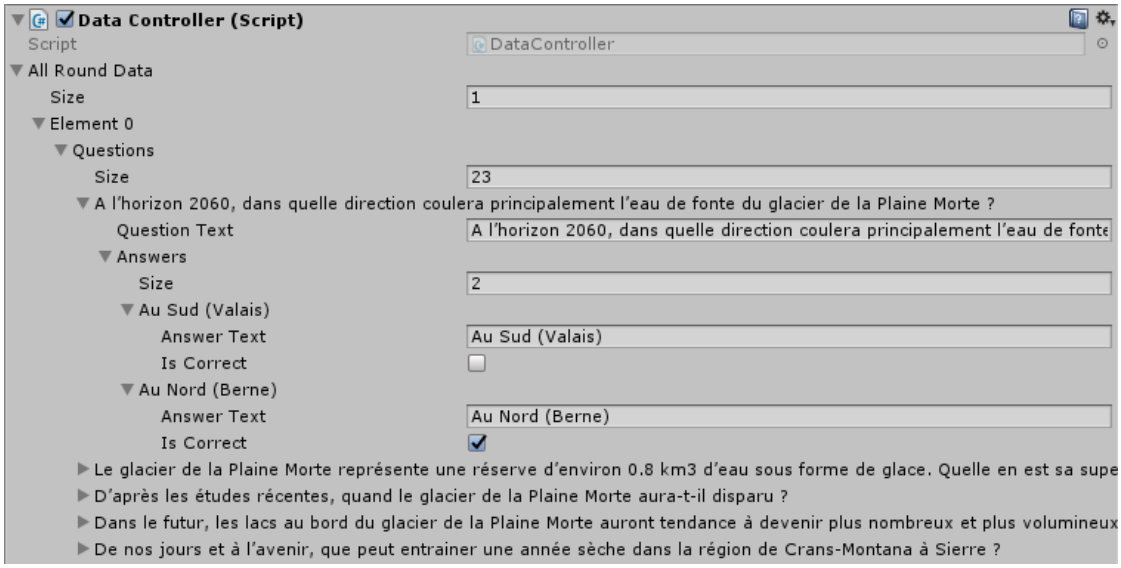


Figure 25 : Scène persistante contenant les quiz
Source : Donnée personnelle

Lors d'un changement de scène, par exemple, lors du passage de l'introduction au menu, Unity détruit les différentes variables et objets relatifs à l'ancienne scène. Pour « 08A.Persistent », le *gameObject* « DataController », objet qui contient nos questions, est chargé et stocké en mémoire lors du premier appel à travers la méthode « DontDestroyOnLoad() » afin de permettre l'affichage de la question sélectionnée et des réponses relatives à la scène « 08B.Quizz ».

Le tableau « RoundData » peut contenir plusieurs « QuestionData », un ensemble de questions. Dans la suite de notre scénario, nous pouvons imaginer avoir plusieurs « RoundData » qui contiendraient chacun des types spécifiques de questions relatives à un type de case : un « RoundData » qui englobe des questions standard sur l'eau, un autre qui contiendrait des quiz sur le système des eaux pour les cases « Eau ».

Notre scène « 08B.QUIZZ » est composée d'un script « GameController ». Il a la tâche de sélectionner aléatoirement une question dans la liste et de l'afficher. Les différents boutons de réponse sont créés automatiquement dans le panel « AnswersPanel » prévu à cet effet. À cette fin, nous avons créé un prefab, composant unique, qui peut être réutilisé plusieurs fois. Cela est possible grâce à la classe « SimpleObjectPool » reprise sur le tutoriel d'Unity (« Answer Button », s. d.).

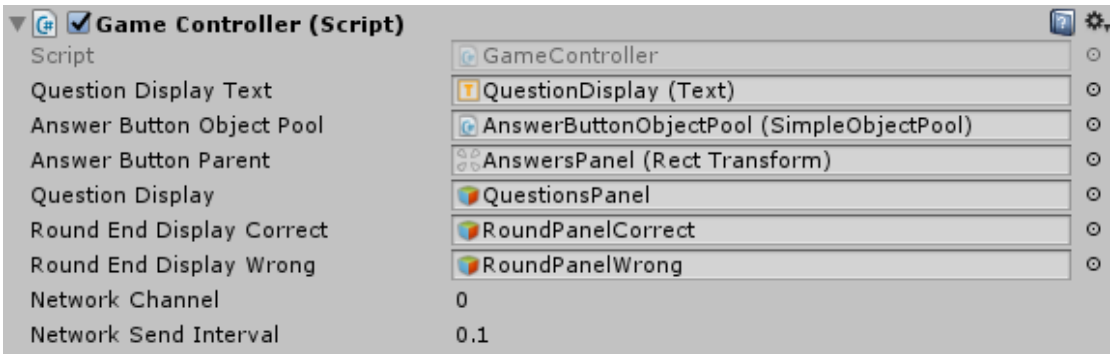



Figure 26: Contrôleur scène quizz
Source : Donnée personnelle

Notre scène « 08B.Quizz » possède trois panels : « QuestionsPanel », « RoundPanelWrong », « RoundPanelCorrect ». Ils sont activés et désactivés selon un ordre strict dans la classe « GameController ». Au début de la scène, du *round* (du tour), la question est affichée sur « QuestionsPanel », les deux autres sont désactivés. Lorsque l'utilisateur clique sur une réponse, le *round* prend fin. Lorsque le joueur sélectionne une réponse, « QuestionsPanel » est désactivé. En cas de bonne réponse, « RoundPanelCorrect » est activé, en cas de mauvaise réponse, c'est « RoundPanelWrong ». Dans le même temps, nous mettons à jour le score du joueur, s'il répond correctement, sinon, son score reste le même.

```
/*
 *End round manager, if ansewer is correct: show green display and set oldscore to current score
 * else show red display and set score to oldscore to the GameBoardScript public stativ int score and oldScore
 */
public void EndRound(bool answer)
{
    isRoundActive = false;
    questionDisplay.SetActive(false);

    if (answer)
    {
        roundEndDisplayWrong.SetActive(false);
        roundEndDisplayCorrect.SetActive(true);

        SoloBoardScript.oldScore = SoloBoardScript.score;
        DiceGB.oldScore = DiceGB.scorePlayer;

    }
    else
    {
        roundEndDisplayWrong.SetActive(true);
        roundEndDisplayCorrect.SetActive(false);

        SoloBoardScript.score = SoloBoardScript.oldScore;
        DiceGB.scorePlayer = DiceGB.oldScore;
    }

}
```

Figure 27: Contrôle des réponses du quizz
Source : Donnée personnelle

## 5.2. Graphisme

La partie graphique est un élément prépondérant pour la réalisation d'un jeu attrayant. Nous avons imaginé un graphisme dans les standards d'aujourd'hui, proche de ce que nous retrouvons dans l'application Android « Compost Challenge », avec un effet cartoon.

Nous avons choisi de n'avoir qu'une seule caméra, offrant une vue 3D aérienne, et une seule lumière avec la possibilité de zoomer sur le jeu.

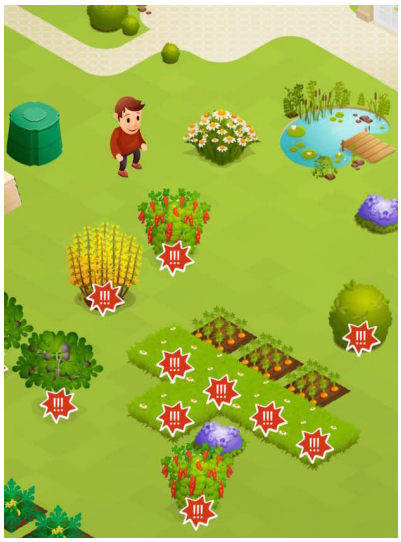


Figure 28: Graphisme CompostChallenge
Source : Tiré de https://compostchallenge.com/app/uploads/2015/09/IMG_3177.jpg

Au début de notre projet, nous nous sommes essayés à la création de texture, modélisation et animation 3D avec le logiciel Blender. Mais comme nous ne disposions pas de connaissances graphiques ni de temps, nous avons opté pour l'utilisation d'éléments graphiques gratuits sur l'*Asset store* d'Unity. Cette option nous a contraints à ne pas pouvoir offrir une identité visuelle propre à notre prototype.

Figure 29: Logo Blender
Source : Tiré de https://commons.wikimedia.org/wiki/File:Logo_Blender.svg

Les graphismes de notre *serious game* sont composés d'*assets* gratuits disponibles sur le *store* d'Unity et en *low poly*. Les graphismes étant réalisés en polygone, moins il y en a, moins le jeu demande de ressources et plus il est performant (Tshabangu, 2016). Développant pour une plateforme web, nous avons privilégié la rapidité. Au départ, nous voulions trouver des personnages âgés, des jeunes, des hommes et des femmes, afin que le joueur puisse s'identifier. La limitation du graphisme cartoon en 3D et du gratuit nous a fait opter pour des personnages également non humains et un petit bonhomme (voir Figure 35 et Figure 36).

## 5.3. Architecture Unity

### 5.3.1. Architecture des dossiers

Nous trouvons ci-dessous la structure du répertoire de notre projet qui contient tous nos *assets*[1]. Les guides de bonne pratique conseillent ce mode de structure afin de maintenir une

[1] Les assets représentent tous les fichiers qui peuvent être utilisés dans un projet. Les scripts, les textures, les matériaux, source audio, etc., sont tous des assets. (« Unity - Manual », s. d.-a)

bonne organisation. Nous avons préféré regrouper l'ensemble des éléments téléchargés sur l'*Asset store* dans un répertoire nommé « AssetStores ». Mis à part les éléments du terrain, nous n'avons aucun élément à la racine. (Korzuszek, 2016).

Dans les autres dossiers, nous trouvons principalement notre développement. Le dossier « CartoonStylizedWater » est un élément acheté que nous n'avons pas pu déplacer dans le dossier « AssetStores ». Le déplacement cause des soucis de fonctionnement. Dans les répertoires « Texture », « Materials », « Audios » et « Prefabs », nous avons certains éléments de l'*Asset store*, que nous avons déplacés ou copiés afin de les retrouver et de les utiliser plus facilement.

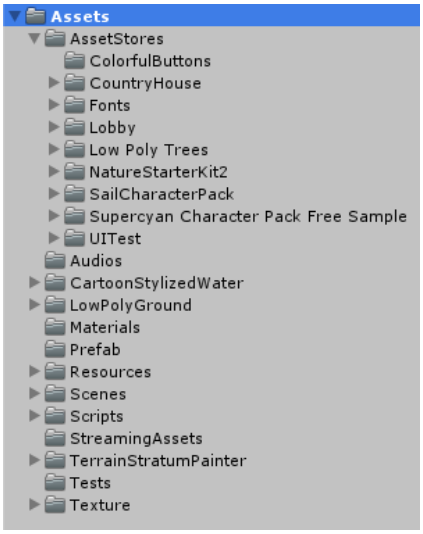



Figure 30: Architecture projet Unity
Source : Donnée personnelle

### 5.3.2. Asset store

Dans ce dossier, nous avons les différents éléments téléchargés sur le *store* d'Unity. Ils sont tous gratuits, hormis Water Creator CartoonStylizedWater que nous avons acquis pour la somme de quinze dollars.

- ColorfulButtons :

Cet ensemble regroupe un pack de boutons colorés. Nous l'avons utilisé pour les boutons « Play » et « Pause » dans notre scène d'introduction. Cette fonction permet de mettre sur « Pause » le texte parlé de l'introduction qui est, dans notre prototype, représenté par une musique d'ambiance.

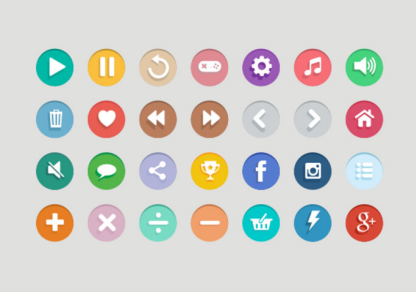


Figure 31: ColorfulButtons
Source : Tiré de https://assetstore.unity.com/packages/2d/gui/icons/colorful-buttons-44118

- CountryHouse

Dans notre scénario, nous avons imaginé que les personnages sortent de leur maison, pour comprendre les raisons de la coupure d'eau. Nous avons utilisé le décor ci-dessous (Figure 32) afin d'illustrer le scénario : les protagonistes partent de leur maison. Nous avons été séduits par son aspect cartoon et il est *low poly*.

Figure 32: Low Poly Country House
Source : Tiré de https://assetstore.unity.com/packages/3d/low-poly-country-house-66203

- Lobby

Unity propose un lobby, un hall de jeux, un salon permettant de se connecter à un jeu en ligne, complet, afin d'intégrer rapidement un mode multijoueur. Cet élément sera détaillé au point 5.4.2.

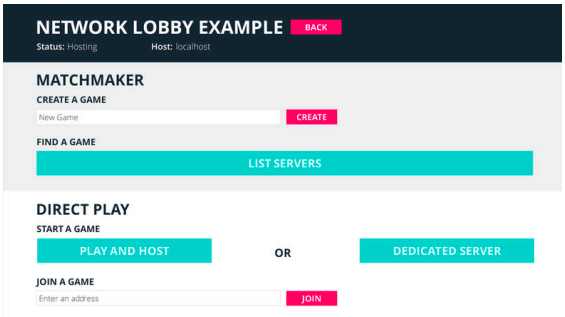



Figure 33: Network Lobby
Source : Tiré de https://assetstore.unity.com/packages/essentials/network-lobby-41836

- Low Poly Trees

Cet *asset* propose un ensemble d'arbres utilisé pour le décor.


Figure 34: Low Poly Trees Seasons
Source: Tiré de https://assetstore.unity.com/packages/3d/vegetation/trees/low-poly-trees-seasons-67486

- Supercyan Character Pack Free Sample

Ce personnage est le premier que nous avons utilisé. Gratuit, il contient également des animations que nous n'avons pas incorporées dans le jeu. Il contient deux versions : une haute qualité, et une autre pour les versions mobiles. Afin de garantir un meilleur fonctionnement, nous avons choisi d'utiliser la deuxième.

Ce personnage est le seul que nous avons durant les *sprints* un et deux. La conception du jeu étant multijoueur, sur un seul écran, nous avons voulu ajouter d'autres couleurs de t-shirts, afin de distinguer les personnages des joueurs. Ne sachant pas si la licence nous le permettait, nous avons pris contact avec le créateur afin de lui expliquer notre démarche de *serious game* à but non lucratif et dans un cadre académique. Comme nous n'avons pas eu de réponse, nous nous sommes permis de modifier le t-shirt du personnage et d'ajouter trois couleurs à l'aide du logiciel Adobe Photoshop.

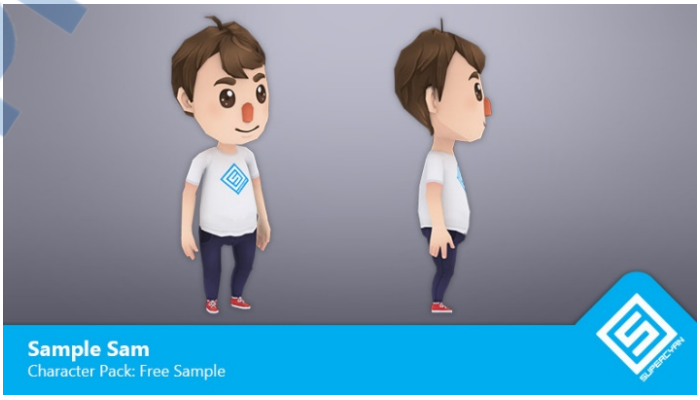



Figure 35: Supercyan Character Pack Free Sample
Source: Tiré de https://assetstore.unity.com/packages/3d/characters/humanoids/character-pack-free-sample-79870

- Sail Character Pack

À partir du *sprint* trois, nous avons utilisé ce pack afin d'offrir un choix plus varié aux joueurs. Ces personnages fantastiques comprennent une version féminine, masculine et une version pirate. Bien que nous ayons aimé proposer des personnages auxquels les joueurs puissent s'identifier, nous avons également pensé à intégrer des personnages fantastiques à notre prototype afin de pouvoir relever le ressenti des premiers joueurs.

L'utilisation de ces personnages est également due au fait que nous n'avons pas trouvé d'autres individus humanoïdes qui correspondaient à nos attentes. Un seul personnage, même avec des couleurs différentes, offre un faible choix.

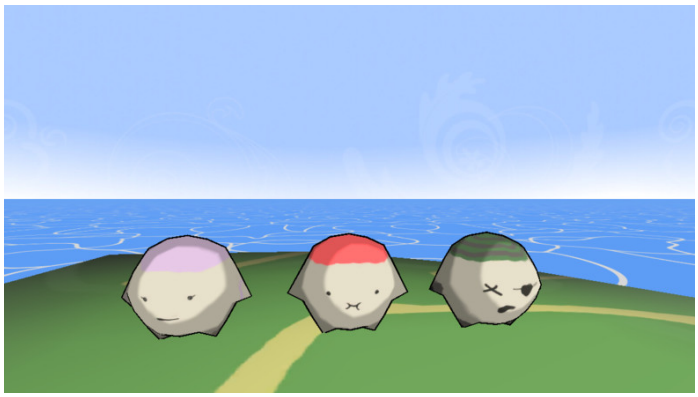


Figure 36: Sail Character Pack
Source : Tiré de https://assetstore.unity.com/packages/3d/characters/sail-character-pack-18880

- UITests

Dans un souci de pouvoir, au moins une fois, mettre à l'épreuve les différentes scènes de notre *serious game*, nous avons cherché un système qui permet d'accélérer le processus de test et d'éviter au développeur de devoir essayer manuellement les différentes interactions possibles avec l'écran. UITests nous permet de créer un script qui automatise ce type de test. Ce point sera développé au point 5.3.9.


Figure 21: Unity UI Test Automation
Source: Tiré de https://assetstore.unity.com/packages/tools/unity-ui-test-automation-72693

- Water Creator CartoonStylizedWater

Un *serious game* sans animation d'eau est étrange. N'ayant pas de connaissances dans l'animation 3D, et dans l'optique d'avoir un graphisme uni, nous avons été obligés d'acheter Water Creator CartoonStylizedWater pour une somme modique de 15 dollars. De type cartoon, cet outil nous permet simplement et rapidement de concevoir une animation d'eau.


Figure 37: Water Creator CartoonStylizedWater
Source : Tiré de https://assetstore.unity.com/packages/2d/textures-materials/water-creator-71677

Cet élément s'ajoute aux diverses fonctionnalités d'Unity. L'*asset* ouvre une nouvelle boite d'outil, l'onglet « CartoonStylizedWater », comme le montre la figure ci-dessous. En cliquant sur « 0.CreateBezierObject », cela crée un ensemble de points que nous pouvons modéliser comme nous le souhaitons : ajouter des points pour allonger la rivière, faire descendre l'eau et orienter le sens d'écoulement. Une fois le cheminement conçu, il nous suffit de cliquer sur « 8 : Run All » et le plug-in crée automatiquement toute l'animation et même des chutes d'eau.

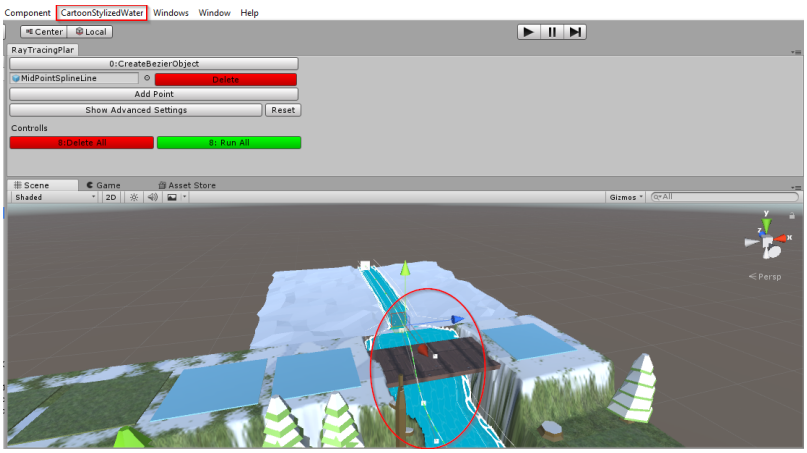



Figure 38: Utilisation water creator
Source : Données personnelles

Afin d'endiguer la rivière créée, il est impératif de concevoir des bordures. Nous avons pu le faire en « creusant » dans notre objet terrain jusqu'au niveau 0, avec l'outil de conception disponible sur Unity. Il permet d'ajouter du relief, d'estomper les pentes et peindre des textures comme de la neige sur notre plateau de jeu. Nous avons utilisé ce même outil pour concevoir notre plateau de jeu en damier. Nous avons ajouté un terrain et surélevé chaque niveau de deux points (paramètre « Height », Figure 40). Nous avons également créé une zone représentant les futurs cases « Village » de notre scénario (voir Figure 42).

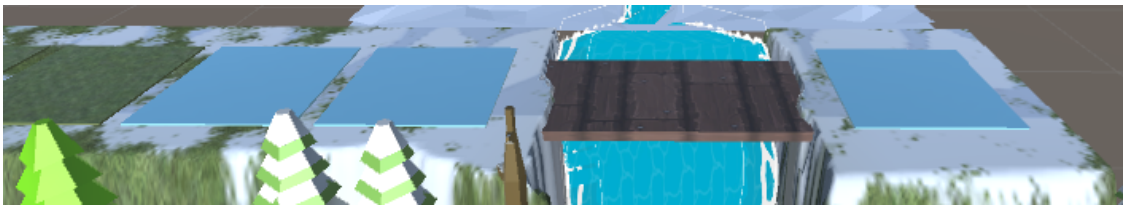


Figure 39: Peinture de neige sur le terrain
Source : Donnée personnelle

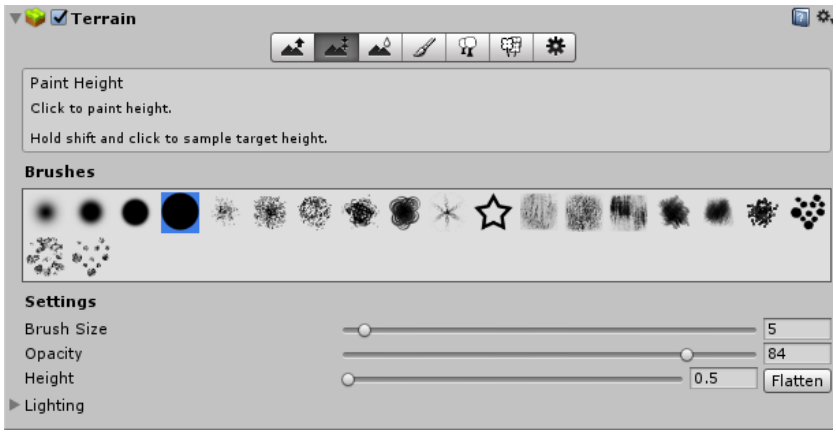



Figure 40: Outil Unity modélisation du terrain
Source : Donnée personnelle

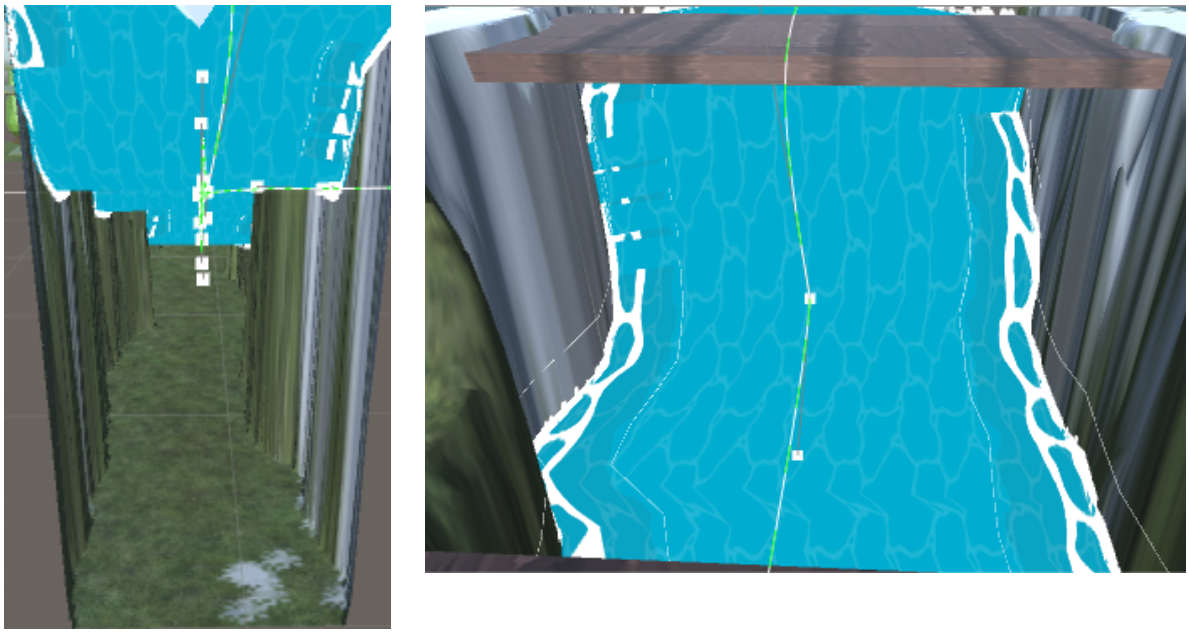


Figure 41: Animation eau
Source : Données personnelles

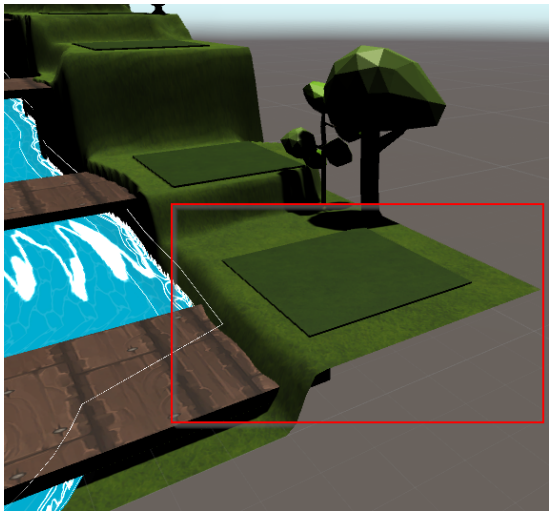


Figure 42: Espace entre deux niveaux
Source : Donnée personnelle

- Wood Texture

Nous avons utilisé cette texture pour les bordures de notre plateau de dé, ainsi que pour les ponts du plateau de jeu.

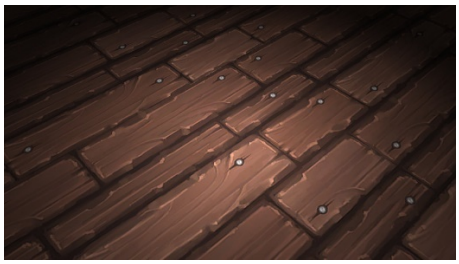


Figure 43: Wood Texture
Source: Tiré de https://assetstore.unity.com/packages/2d/textures-materials/wood/wood-texture-floor-24970

- NatureStarterKit2

Nous avons repris de cet *asset* la texture pour illustrer l'herbe et les différentes cases du jeu.


Figure 44: Nature Starter Kit 2 asset
Source : Tiré de https://assetstore.unity.com/packages/3d/environments/nature-starter-kit-2-52977

### 5.3.3. Audio

Notre *serious game* est également à destination des plus jeunes, qui ne savent pas encore forcément lire. C'est pourquoi il est important d'avoir cette phase d'introduction qui est également lue. Nous pouvons également contenter certaines personnes « lassées » par la lecture.

Nous avons jugé important d'avoir un texte parlé lors de l'introduction qui explique le contexte du jeu. Bien que ce texte ne soit pas rédigé et que nous n'ayons pas encore fait appel à un acteur pour le lire, nous avons simulé son existence à travers une musique libre de droits, *Spring In My Step*, réalisé par Orbital Music et disponible sur sa chaine YouTube à cette adresse : https://www.youtube.com/watch?v=fxnn9FR-4Vk.

### 5.3.4. Materials

Dans ce dossier, nous retrouvons l'ensemble des matériaux que nous avons utilisés ou créés.


Figure 45: Dossier Materials
Source : Donnée personnelle

Dans la conception de jeux vidéo, les matériaux sont les éléments qui vont habiller les différents objets. Ils sont composés de textures qui sont de simples images bitmap 2D (« Unity - Manual », s. d.-c).

Le dossier « Materials » contient les points du dé, les bordures du plateau pour la scène du dé qui n'est pas directement accessible dans notre prototype dû à des changements de priorités. Nous avons également les différents matériaux pour concevoir notre personnage en quatre couleurs distinctes.

### 5.3.5. Prefab

Ce dossier contient tous les modèles d'objets que nous avons créés afin de faciliter leur réutilisation. Par exemple, nous avons deux plateaux de jeu, un pour chaque mode de jeu, solo et multijoueur. Les éléments des décors peuvent être modifiés directement dans le prefab « House » et « Decor » que nous avons créés, afin que ces modifications prennent effet dans les deux plateaux instantanément.


Figure 46: Dossier Prefab
Source : Données personnelles

La création d'un prefab est aisée. Il suffit soit de clic-droit sur le projet, créer un *prefab* et glisser l'objet que nous voulons directement à l'intérieur, soit de glisser et déposer le *gameObject* dans notre dossier « Prefabs ». En cas de modification du modèle sur la scène, il faut cliquer sur « Apply »

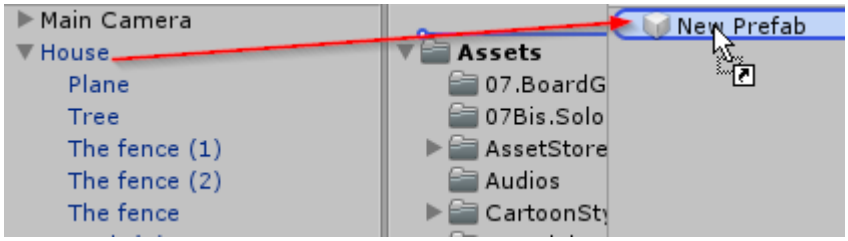



Figure 47: Création d'un prefab
Source : Donnée personnelle

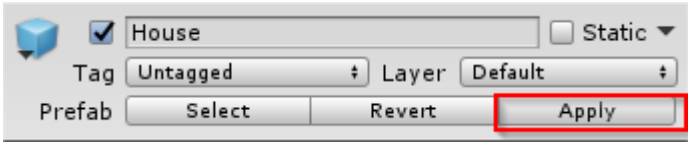



Figure 48: Application modification d'un prefab
Source : Donnée personnelle

### 5.3.6. Ressources

Ce dossier contient les différentes images que nous avons utilisées lors du développement de notre *serious game* : les drapeaux qui représentent chacune des langues, et les faces de dé en 2D utilisées sur le plateau de jeu. Pour pouvoir utiliser une image dans un bouton, il faut au préalable modifier son type en « Sprite (2D and UI) » (voir Figure 49) (Graham, 2014).

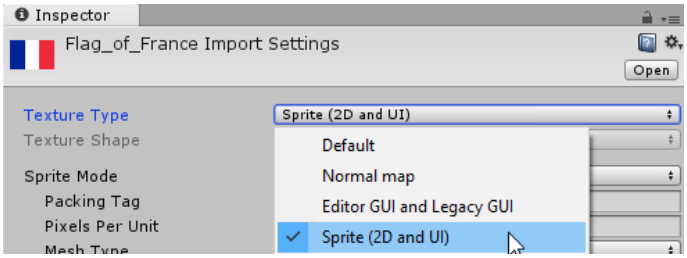



Figure 49: Modification du type de texture d'une image
Source : Donnée personnelle

### 5.3.7. Scènes

Ici, nous avons l'ensemble de nos scènes que nous avons numérotées afin d'en faciliter l'usage. Chaque scène représente un état du jeu : la scène du choix des langues, celle d'introduction, etc. Notre *serious game* comporte deux scènes principales : « 07.BoardGame », mise en place pour le jeu multijoueur et « 07Bis.SoloBoard », pour permettre de lancer une partie en « Solo ».

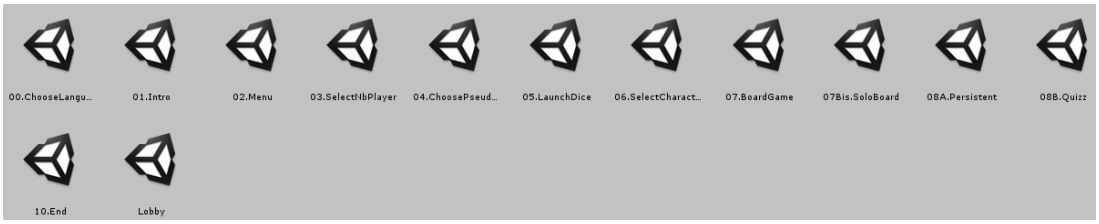



Figure 50: Dossier Scènes
Source : Donnée personnelle

Le passage d'une scène à une autre est effectué à travers la classe « SceneManager » d'Unity. Pour les interactions simples de passage entre les scènes, nous avons créé un script « _SceneManager » avec des méthodes « public », car les boutons ne peuvent faire appel à des méthodes qu'à travers ce type. Nous avons déposé ce script dans les différentes scènes soit dans un *gameObject* créé à cet effet, soit dans *MainCamera.* Dans la figure ci-dessous, nous trouvons un exemple de fonctionnement : le script est mis dans le *gameObject* « Main Camera », dans la propriété « OnClick » du bouton, nous déposons notre gameObject qui

contient le script de gestion des scènes. Enfin, nous sélectionnons la méthode souhaitée. S'il existe un paramètre pour l'exécution de la méthode, un champ apparait à cet effet.

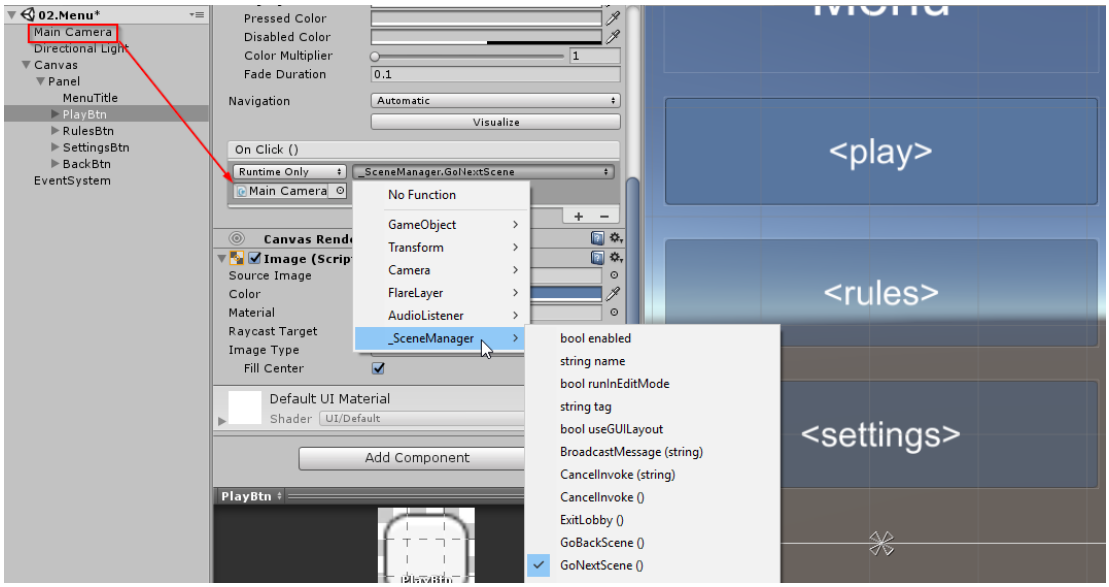



Figure 51: Scenemanager gestion bouton
Source : Donnée personnelle

Ce script contient quatre méthodes :

- GoSceneName : permet d'aller à la scène selon son nom mis en paramètre.
- GoNextScene : va à la scène suivante selon l'index configuré dans les paramètres de compilation.
- GoBackScene : pareil que GoNextScene, mais va à la scène précédente.
- ExitLobby et ExitDelay : permettent de quitter la scène « Lobby » et de revenir à la scène précédente.

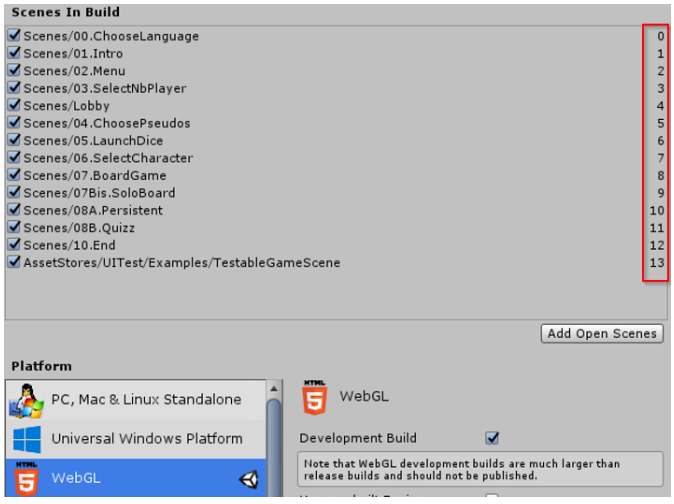



Figure 52: Index build
Source : Donnée personnelle

```
7   public class _SceneManager : MonoBehaviour {
8
9       public void GoSceneName(string name)
10      {
11          SceneManager.LoadScene(name);
12      }
13
14      public void GoNextScene()
15      {
16          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
17      }
18
19      public void GoBackScene()
20      {
21          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
22      }
23
24      public void ExitLobby()
25      {
26          NetworkManager.singleton.StopClient();
27          NetworkManager.singleton.StopHost();
28
29          NetworkLobbyManager.singleton.StopClient();
30          NetworkLobbyManager.singleton.StopServer();
31
32          NetworkServer.DisconnectAll();
33          StartCoroutine(ExitDelay());
34      }
35
36      IEnumerator ExitDelay()
37      {
38          yield return new WaitForSeconds(0.1f);//attends un peu
39          Destroy(NetworkLobbyManager.singleton.gameObject);
40
41          yield return new WaitForSeconds(0.1f);//attends un peu
42
43          SceneManager.LoadScene("03.SelectNbPlayer");
44      }
```

Figure 53: _SceneManager classe
Source : Donnée personnelle

### 5.3.8. Scripts

Dans ce répertoire, nous avons regroupé l'ensemble des scripts que nous avons créés. Nous passerons en revue les principaux. Les dossiers « Solo » et « Multijoueur » vont être traités au point 5.4. « ChoosePseudo » et « Player » vont être explicités au chapitre 5.5.1. Enfin, le contenu du dossier « Quizz » a été expliqué plus haut au point 5.1.3.

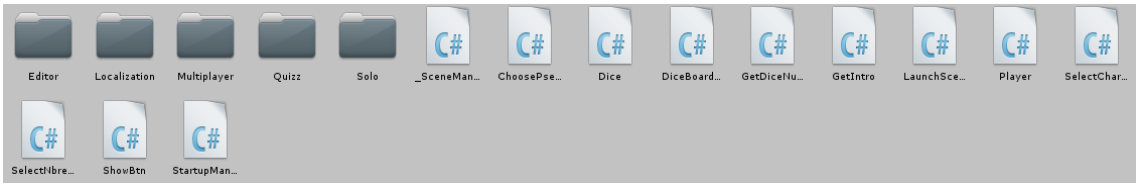



Figure 54: Dossier Scipts
Source : Donnée personnelle

Dans le dossier « Editor », nous avons l'outil de gestion des fichiers de traduction. Cet élément a été développé dans le but de permettre de rajouter de nouveaux champs textes rapidement. Il va créer un outil directement dans l'interface d'Unity qui permet de charger, sauvegarder, modifier les différents fichiers de traduction au format Json.

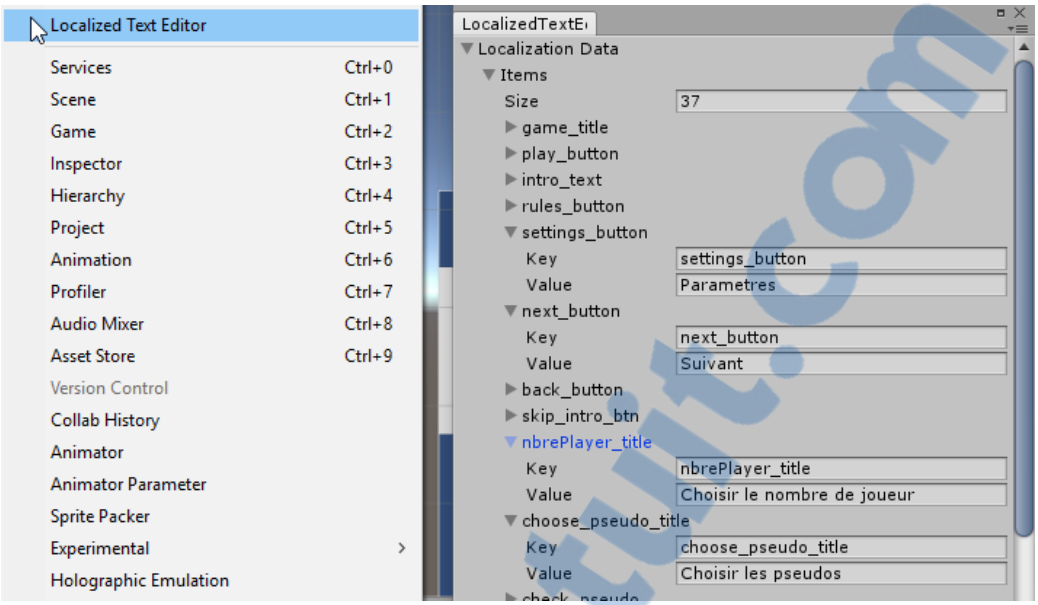



Figure 55: Unity Localized Text Editor
Source: Donnée personnelle

Dans le dossier « Localization », nous avons nos scripts qui permettent un jeu multilingue. À ce stade, nous avons désactivé les autres langues avec l'accord du professeur accompagnant, car les quiz ne sont pas encore compatibles avec le multi-langage. Nous avons trois fichiers : « LocalizationData », « LocalizationManager », « LocalizedText » qui vont permettre la lecture de notre fichier Json. Nous avons quatre fichiers Json : français, anglais, allemand et italien. Ces fichiers doivent être dans un dossier « StreamingAssets ». Le contenu de ce dossier est accessible et chargé dans la scène du choix des langues. Ces quatre fichiers sont construits de manière identique : un nœud items, qui contient l'ensemble des *keys* et des *value*. Chaque *value* est liée à une *key* qui est le paramètre à fournir à chaque champ texte que nous voulons traduire.


```
{
   "key":"settings_button",
   "value":"Parametres"
},
{
   "key":"next_button",
   "value":"Suivant"
},
{
   "key":"back_button",
   "value":"Retour"
},
{
   "key":"skip_intro_btn",
   "value":"Passer l'introduction"
},
```


Figure 56: Fichier traduction Json
Source : Donnée personnelle

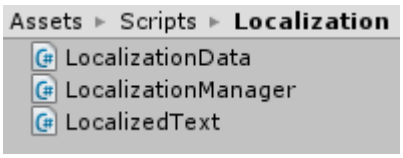



Figure 57: Contenu dossier Localization
Source : Donnée personnelle

- « LocalizationData » : va permettre la lecture du fichier Json à LocalizationManager.
- « LocalizationManager » : Élément central, cette classe lie le contenu du fichier Json sélectionné par l'utilisateur selon la langue de son choix. Il va le charger dans un *Dictionary*, et fournir la traduction à tous les gameObject tout au long des différentes scènes. Nous avons utilisé la classe UnityWebRequest afin de permettre au navigateur de télécharger le fichier dans le dossier « StreamingAssets ». Sur les autres plateformes, telles qu'Android, il suffit de renseigner le chemin d'accès du dossier « StreamingAssets ».
- « LocalizedText » : élément à fournir à nos différents champs de texte qui sont multilingues. Il suffit ensuite de créer la clé et la valeur dans les fichiers Json, ajouter un composant « LocalizedText » aux champs voulus et notifier la clé comme l'exemple ci-dessous :

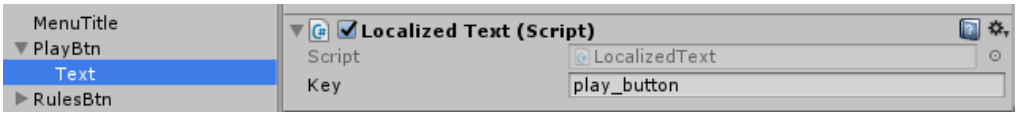



Figure 58: Exemple traduction LocalizedText
Source : Donnée personnelle

Nous avons également le script « StartupManager » qui est utilisé dans la scène de la sélection de la langue. Mis dans un *gameObject* du même nom, il va patienter et attendre que la classe « LocalizationManager » ait chargé tous les éléments de traduction et passer ensuite à l'introduction.

```
5   public class StartupManager : MonoBehaviour {
6
7       // Use this for initialization
8       private IEnumerator Start()
9       {
10          //Attend que Localizationmanger, fichier Json des langues, est chargé -> isReady
11          while (!LocalizationManager.instance.GetIsReady())
12          {
13              yield return null;
14          }
15
16          SceneManager.LoadScene("01.Intro");
17      }
18  }
```

Figure 59: Script StartupManager
Source : Donnée personnelle

Nous avons également divers scripts qui ont chacun un rôle spécifique. La gestion de l'introduction est gérée par trois éléments « PlayPauseScript », « LaunchNextSceneTimer » et « ShowBtn ».

« PlayPauseScript » permet de manipuler les boutons « Play » et « Pause ». Elle offre à l'utilisateur la possibilité de stopper le texte parlé, imagé ici par une musique, et reprendre la lecture. Au début de notre scène, nous avons activé le bouton *Pause* et désactivé le bouton *Play*. Lors du clic de l'utilisateur, le processus s'inverse.

```
7   public class PlayPauseScript : MonoBehaviour {
8
9       public Button btnPlay;
10      public Button btnPause;
11
12      void Awake()
13      {
14          btnPlay.gameObject.SetActive(false);
15      }
16      void Start()
17      {
18
19          btnPlay = btnPlay.GetComponent<Button>();
20          btnPause = btnPause.GetComponent<Button>();
21          //Show bouton Play, disable Pause
22          btnPause.onClick.AddListener(() => PlayPause(0));
23          //Show bouton Pause, disable Play
24          btnPlay.onClick.AddListener(() => PlayPause(1));
25      }
26
27      void PlayPause(int mode)
28      {
29          switch (mode)
30          {
31              case 0:
32                  btnPlay.gameObject.SetActive(true);
33                  btnPause.gameObject.SetActive(false);
34                  break;
35              case 1:
36                  btnPlay.gameObject.SetActive(false);
37                  btnPause.gameObject.SetActive(true);
38                  break;
39          }
40
41      }
42  }
```

Figure 60: Script boutons Play et Pause de l'introduction
Source : Donnée personnelle

Nous avons également ajouté à nos deux boutons une fonction « OnClick() » qui va leur permettre de jouer leur rôle. Pour cela, il suffit d'ajouter une fonction et d'y glisser notre *gameObject* qui contient la source audio. Enfin, nous affectons la fonction « AudioSource.Play » et « AudioSource.Pause » à chacun des éléments. Ces deux méthodes sont fournies par Unity3D lorsqu'il reconnait un *gameObject* contenant de l'audio.

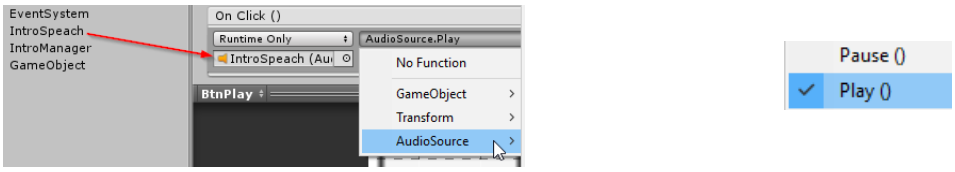



Figure 61: Introduction configuration du bouton Play
Source : Donnée personnelle

« LaunchNextSceneTimer » a comme fonction de passer automatiquement au menu lorsque la musique arrive à son terme. Nous avons une minuterie qui est décrémentée dans la méthode « Update() ». Une fois le compteur à zéro, nous chargeons la scène suivante rentrée en paramètre.

```
6   public class LaunchNextSceneTimer : MonoBehaviour {
7
8       public float timer;
9       public string sceneName;
10
11      public void GoToScene(string sceneName)
12      {
13          SceneManager.LoadScene(sceneName);
14          Debug.Log("Play Ckicked");
15      }
16
17      public void Update()
18      {
19          timer -= Time.deltaTime;
20
21          if(timer <= 0.0f)
22          {
23              GoToScene(sceneName);
24          }
25      }
26  }
```

Figure 62: Script LaunchNextSceneTimer 1
Source : Donnée personnelle

Nous pouvons directement insérer nos paramètres depuis l'onglet *Inspector* du *gameObject* auquel le script est attaché. Nous avons choisi un *timer* de 120 secondes qui correspond à la durée de la musique.

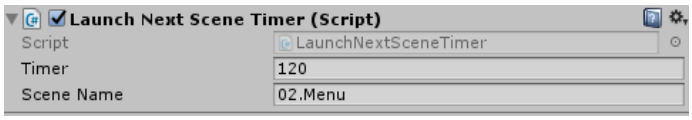



Figure 63: Script LaunchNextSceneTimer 2
Source : Donnée personnelle

« ShowBtn » permet d'afficher un bouton dans le but de passer la scène d'introduction rapidement. Cela a été pensé pour les personnes qui connaissent déjà le jeu et son contexte. Son fonctionnement est semblable à « LaunchNextSceneTimer ». Au lieu de charger la scène suivante, ce script active le bouton qui est au préalable désactivé dans la méthode « Start() ».

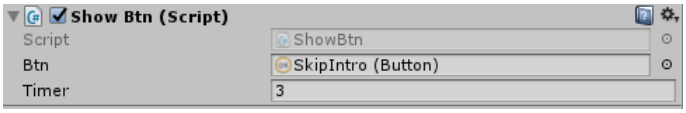



Figure 64: Script ShowBtn
Source : Donnée personnelle

Ces trois scripts sont incorporés à un *gameObject* « IntroManager ». Nous pouvons gérer et modifier les différentes variables, *public* seulement, de nos scripts directement depuis l'interface graphique. Cette méthode nous permet, par exemple, de rallonger ou raccourcir le *timer* sans devoir ouvrir Visual Studio.

Puis, nous avons deux scripts qui ne font pas partie directement de notre jeu dans son état actuel. Lors du *sprint* un, nous avions créé une scène où les joueurs peuvent lancer le dé afin de déterminer l'ordre de la partie. Nous avions également pour objectif de l'utiliser pour les lancers de dé sur le jeu de plateau. À la suite de divers problèmes de gestion du score des joueurs notamment, nous avons décidé de mettre de côté cette scène.

Nous avons eu de la difficulté à réaliser la gestion d'un jeu multijoueur sur un écran. Par exemple, nous voulions faire relancer les joueurs qui avaient obtenu le même score, mais nous avons eu de la peine à maitriser la méthode « Update() » d'Unity qui se lance autant de fois que de *frame per second* (FPS)[2]. Cela fait que même avec l'utilisation de conditions dans cette méthode, le compteur que nous avions utilisé s'incrémente plus lentement que la méthode « Update() » ne s'exécute. Malgré ces problèmes, nous avons opté pour une solution vraiment multijoueur, explicitée au point 5.4.2.

Nous avons repris et adapté le tutoriel de PushyPixels disponible sur sa chaine YouTube[3]. Nous avons pensé à différentes méthodes et consulté d'autres tutoriels, mais nous n'avons pas trouvé de technique plus simple. Sa manière de faire nous a évité, dans un premier temps, de devoir utiliser la classe « Random » afin de générer un score aléatoire. Les points du dé proviennent de son projet (PushyPixels, 2015).

Cette scène est construite avec un plateau et des cubes sur les côtés pour le décor. Afin de garder le dé dans une zone précise, ce dernier est dans une boite construite avec des cubes invisibles. Notre dé comporte un *rigidybody* et plusieurs *sphere collider*, un pour chaque côté. Sous le plateau, nous avons un cube invisible avec un *box collider*. Les *colliders* sont des composants qui permettent de gérer la collision de différents objets. Les *rigidbody* sont des

[2] *Frame per second* (FPS), nombre d'image par seconde. Sur Unity, la méthode « Update() » s'exécute autant de fois que de FPS. Si nous avons 30 FPS sur un jeu, par exemple, cette méthode va s'exécuter 30 fois par seconde.
[3] Tutoriel Breakfast With Unity: 3D Dice Roller Part 1 : https://www.youtube.com/watch?v=LVKqTzK9-AI et la partie 2: https://www.youtube.com/watch?v=4Ht2sHoxY1E&t=961s

éléments que nous ajoutons à un objet afin de lui ajouter de la physique : force, torsion, vélocité, etc. (« Le composant Rigidbody avec unity dans physique », 2017). Lorsque le dé est lancé et qu'il touche le plateau, le cube invisible détecte la face du dé à son contact et donne son opposé.

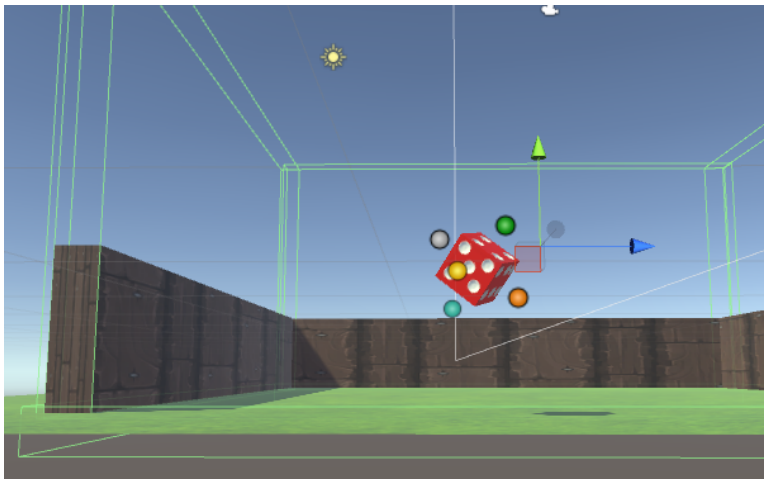


Figure 65: Lancer de dé 3D
Source : Donnée personnelle

Lorsque nous cliquons sur le plateau, la méthode « Launch() » de notre script « Dice » s'exécute. Nous allons ajouter différents éléments de physique au *rigidbody* de notre dé, tel que la « Force » - force appliquée sur une direction donnée - et le « Torque » - force de rotation d'un objet.

```
94      void Launch()
95      {
96          countSpace++;
97          //_player.played = true;
98          diceNumber = 0;
99          float dirX = Random.Range(0, 500);
100         float dirY = Random.Range(0, 500);
101         float dirZ = Random.Range(0, 500);
102         transform.position = new Vector3(2, 5, 0);
103         transform.rotation = Quaternion.identity;
104         rb.AddForce(Random.onUnitSphere * forceAmount, forceMode);
105         rb.AddTorque(Random.onUnitSphere * torqueAmount, forceMode);
106         rb.AddForce(transform.up * 100);
107         rb.AddTorque(dirX, dirY, dirZ);
108     }
109 }
```

Figure 66: Méthode Launch(), script Dice
Source : Donnée personnelle

Une fois le dé à terre, le score est affiché grâce à la classe « GetDiceNumber() », qui retourne la valeur opposée de la surface en contact avec le sol. Le score est renvoyé à la variable *public static* « diceNumber » et est affiché à l'écran.

```
public class GetDiceNumber : MonoBehaviour {

    Vector3 diceVelocity;
    public static int diceNumber;

    private void FixedUpdate()
    {
        diceVelocity = Dice.diceVelocity;
    }

    private void OnTriggerStay(Collider col)
    {
        if (Dice.countSpace > 0) {
            if (diceVelocity.x == 0f && diceVelocity.y == 0f && diceVelocity.z == 0f)
            {
                switch (col.gameObject.name)
                {
                    case "Side1":
                        Dice.diceNumber = 6;
                        break;
                    case "Side2":
                        Dice.diceNumber = 5;
                        break;
                    case "Side3":
                        Dice.diceNumber = 4;
                        break;
                    case "Side4":
                        Dice.diceNumber = 3;
                        break;
                    case "Side5":
                        Dice.diceNumber = 2;
                        break;
                    case "Side6":
                        Dice.diceNumber = 1;
                        break;
                }

            }
        }
        else {
            Dice.diceNumber = 0;
        }

    }
}
```

Figure 67: Script GetDiceNumber
Source : Donnée personnelle

Lors des sprints un et deux, le script « SelectNbrePlayer » devait servir à initialiser le nombre de joueurs à créer. Ayant opté par la suite pour un mode « Solo » et un autre « Multijoueurs », ce script nous sert maintenant à orienter le joueur, soit vers la scène du choix du pseudo pour le mode « Solo », soit vers le « Lobby » pour le mode multijoueur.

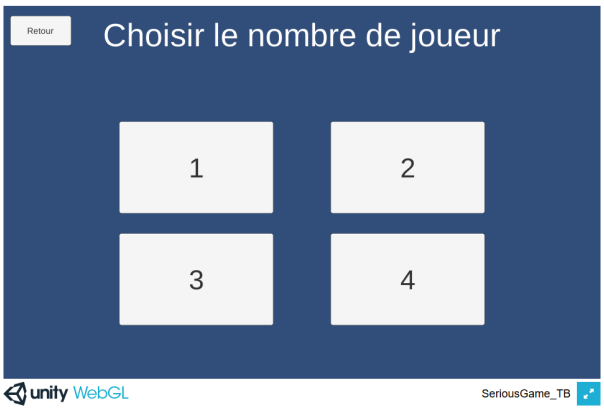



Figure 68: Scène de sélection du nombre de joueurs
Source : Donnée personnelle

```
8   public class SelectNbrePlayer : MonoBehaviour {
9
10      public Button btn1;
11      public Button btn2;
12      public Button btn3;
13      public Button btn4;
14
15      public static int nbrePlayer;
16
17      public void EnableInputField(int nbPlayer)
18      {
19          nbrePlayer = nbPlayer;
20      }
21
22  }
```

Figure 69: Script SelectNbrePlayer
Source : Donnée personnelle

Le script « SelectNbrePlayer » n'est utilisé plus que par le bouton « Solo » qui envoie la valeur 1, afin de créer un joueur. Le bouton « Multiplayers » mène au « Lobby », expliqué au point 5.4.2.

### 5.3.9. Tests

Dans ce dossier, nous retrouvons notre fichier de test, créé à partir de l'*asset* UITests (voir 5.3.2). Les tests sont lancés via l'outil « Test Runner » d'Unity. UITests permet de tester les interactions graphiques de notre jeu. Nous n'avons découvert cet outil que tard dans notre développement et ne l'avons utilisé que pour un test : nous cliquons sur la langue française, dans la première scène, et nous voulons que la bonne traduction se charge. Afin de valider le test, nous avons ajouté un champ « lang » invisible dans notre « Panel » de la scène du choix des langues. Si la bonne langue est chargée, « lang » aura comme valeur « fr », si le fichier de langue n'est pas chargé, trouvé, contient des erreurs ou si le mauvais fichier est chargé, le test échoue.

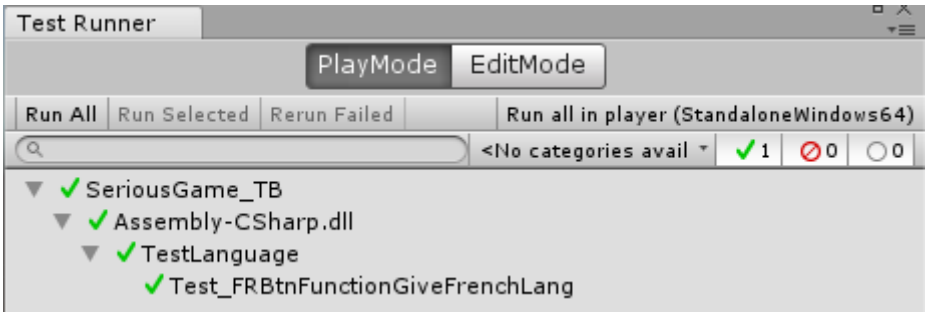



Figure 70: Test chargement de la langue française
Source : Donnée personnelle

Unity offre également la possibilité d'exécuter des tests de performances. Ce service étant limité aux utilisateurs disposant d'une licence payante, nous n' avons pas ce type de test.

### 5.3.10. Texture

Ce dossier comprend les différentes textures que nous avons reprises et parfois modifiées (voir 5.3.2). Nous avons des fichiers images au format PNG, ainsi que PSD (format Adobe Photoshop). Si nous voulons appliquer une texture à un modèle 3D, il faudra créer un *material* à partir de la texture. Si nous déposons une texture sur un élément 3D qui n'a pas de *material*, Unity le conçoit automatiquement dans un dossier « Materials » créé à la racine du dossier « Texture ».

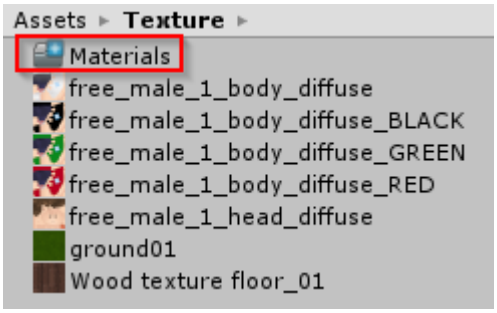



Figure 71: Dossier Textures
Source : Donnée personnelle

## 5.4. Mode de jeu

Dans ce chapitre, nous expliquerons les deux modes de jeu que nous avons développés dans notre jeu sérieux. Durant les deux premiers *sprints*, nous avons voulu implémenter un seul mode : le participant joue derrière son écran et sélectionne le nombre d'adversaires et le jeu se déroule sur un seul PC. À la suite d'une réflexion sur la pertinence d'avoir un jeu de ce type, et ayant eu des problèmes d'implémentation : gestion du lancer de dé tour par tour, sélection des personnages par les différents joueurs, nous avons décidé de nous orienter au *sprint* trois vers un jeu vraiment multijoueur. Nous avons perdu deux à trois semaines en essayant d'implémenter le jeu sur un seul écran. Ce changement nous a permis d'avancer dans notre développement et de proposer un prototype fonctionnel bien qu'il ait encore quelques bugs pour la partie multijoueur. Enfin, pour permettre au joueur d'effectuer une partie seule, nous avons également un mode « Solo ».

Les deux modes ayant une dynamique de jeu et une implémentation du code différentes, nous avons choisi de les séparer. Ils ont chacun leur script et leur scène propre, sauf pour la partie quiz et certains scripts du lancer du dé. Nous avions également la possibilité de proposer le jeu uniquement via un accès par le lobby d'Unity, décrit au point 5.4.2. Dans les paramètres du « LobbyManager », nous pouvons choisir le nombre maximum et minimum de joueurs, ainsi que le nombre de joueurs par appareil. Nous avons choisi un maximum de quatre, un minimum de deux et un joueur par *device*.

Nous aurions pu mettre la valeur du nombre de joueurs minimum à un, afin de nous passer du mode « Solo » et de permettre au joueur de jouer en ligne, mais seul. Nous n'avons pas choisi cette méthode, car une des étapes suivantes du projet est d'intégrer les WebSocket, afin de permettre au joueur de se connecter directement au serveur de jeu proposé par

Unity et de créer des tables de jeu où d'autres personnes peuvent se connecter et débuter une partie à plusieurs. Si nous mettons à un le nombre minimum de joueurs, nous pouvons imaginer des personnes qui créent une table et lancent une partie seule. Cela a un coût. Avec la licence *Personnal*, Unity offre 20 *concurrents users* (CCU), qui correspondent au nombre de joueurs qui peuvent jouer en ligne simultanément, indépendamment du nombre de tables de jeu. Ainsi afin d'éviter de surcharger inutilement les serveurs, nous pensons qu'il est préférable d'avoir un mode « Solo » hors ligne et un mode multijoueur en ligne avec un, deux ou trois adversaires.

Mise à part la création de scripts différents, l'utilisation de deux modes de jeux n'a pas demandé énormément d'efforts supplémentaires. En utilisant les prefabs (voir 5.3.5), nous n'avons dû construire qu'un seul plateau. Les modifications apportées sur le terrain, les cases ou les décors s'appliquent automatiquement sur les scènes. Le seul élément qui diffère est que sur le mode « Solo », le joueur peut sélectionner son personnage et sur le mode « Multijoueur », il est défini à l'avance.

### 5.4.1. Solo

Ce mode permet de jouer seul. Dans le futur, nous l'imaginons sans l'aspect collaboratif de notre scénario, car cela ne fait pas de sens. Il se distingue du mode « Multijoueur » par la sauvegarde du personnage créé dans une base de données via API PHP (voir 5.5.1). Lorsque le joueur entre son pseudo et clique sur le bouton « Suivant », le joueur est créé. Le script qui se rapporte à cette fonctionnalité, « ChoosePseudo » et la classe « Player » sont expliqués au point 5.5.1. Ensuite, le joueur a la possibilité de choisir parmi différents personnages que nous avons pris sur le store.


Figure 72: Dossier Solo
Source : Donnée personnelle

La sélection du personnage est faite via le script : « SelectCharacter ». Nous avons découvert et adapté cette méthode sur la chaine YouTube de « N3K EN » disponible à cette adresse : https://www.youtube.com/watch?v=IFTjcPvCZaM (N3K EN, 2016). Dans cette classe, nous avons ajouté les différents personnages trouvés sur l'*asset store* d'Unity que nous avons insérés dans un *gameObject* « CharacterListSolo ».

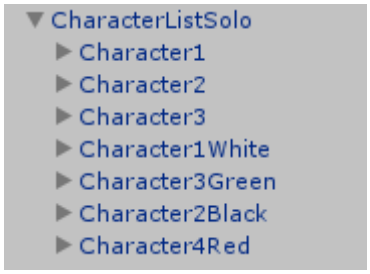



Figure 73: Liste des choix des personnages CharachterListSolo
Source : Donnée personnelle

Nous désactivons l'ensemble des composants enfants de ce *gameObject* et nous n'activons que celui qui correspond à la variable « index », sauvegardée via les « PlayerPrefs », expliqués au point 5.5.2. Si le joueur a déjà joué une fois à notre *serious* game, la ligne 16 de la figure ci-dessous va récupérer le choix de l'utilisateur sauvé lors d'une précédente partie. Ce choix est enregistré lors de la confirmation de la sélection.

```
7   public class SelectCharacter : MonoBehaviour
8   {
9
10      private GameObject[] characterList;
11      private int index = 0;
12
13
14      private void Start()
15      {
16          index = PlayerPrefs.GetInt("CharacterSelected", 0);
17
18          characterList = new GameObject[transform.childCount];
19
20          //Fill array with character
21          for (int i = 0; i < transform.childCount; i++)
22          {
23              characterList[i] = transform.GetChild(i).gameObject;
24          }
25          //Visible off of all character
26          foreach (GameObject go in characterList)
27          {
28              go.SetActive(false);
29          }
30          //Visible true selected character
31          if (characterList[index])
32          {
33              characterList[index].SetActive(true);
34          }
35      }
36
37      public void CharacterLeftRight(bool left)
38      {
39          //Visible false old model
40          characterList[index].SetActive(false);
41          if (left == true)
42          {
43              index--;
44              if (index < 0)
45              {
46                  index = characterList.Length - 1;
47              }
48          }
49          else
50          {
51              index++;
52              if (index == characterList.Length)
53              {
54                  index = 0;
55              }
56          }
57          //Visible true new model
58          characterList[index].SetActive(true);
59      }
60
61      public void ConfirmBtn()
62      {
63          PlayerPrefs.SetInt("CharacterSelected", index);
64          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
65      }
```

Figure 74: Script sélection des personnages mode Solo
Source : Donnée personnelle

Le script « SoloBoardScript » contient la logique de notre jeu solo. Elle est incluse dans un *gameObject* dans la scène « 07Bis.SoloBoard ». Dans la méthode « Awake() », nous trouvons l'initialisation et la mise en ordre alphabétique de toutes nos cases, qui sont en fait des cubes 3D aplatis. Cette méthode est appelée avant le début de la scène et lorsque tous les *gameObjects* sont chargés. Comme le chargement s'effectue de manière aléatoire (« Unity -

Scripting API », s. d.-a), pour retrouver nos cases, nous leur avons donné un tag et nous avons été obligés de nommer convenablement chacune de nos cases « Cube001 », « Cube002 », « Cube003 », etc. Le tag permet de charger tous les *gameObjects* avec la référence « Case » dans un tableau. Puis, en utilisant « System.Linq », nous avons réordonné ce tableau par ordre alphabétique.

Dans la méthode « Start() », nous initialisons la couleur des différentes cases. « Start() » est appelé une seule fois, avant la méthode « Update() » et après la méthode « Awake() ». Ainsi après chaque quiz, lorsque nous rechargeons la scène du plateau, la case où doit se trouver le joueur est en noir. La gestion du score est gérée par deux variables *int*, « oldScore » et « score ». Lors du lancer du dé, la variable « score » s'incrémente de la valeur du dé. En cas de bonne réponse : *oldScore = score*, la nouvelle position est gardée en mémoire, sinon, *score = oldScore*, le joueur doit retourner à sa position initiale.

La méthode « GetScore() » va être lancée par le script « SoloDiceGB ». Il va récupérer et surligner les cases qui correspondent au score. Lors du lancer du dé, le score va s'incrémenter et la case correspondante est affichée en magenta. Nous avons en outre inséré un contrôleur qui lance la scène de fin lorsque le joueur arrive ou dépasse la dernière case. Enfin, la méthode « Update() » met à jour le score total du joueur.

```
public class SoloBoardScript : MonoBehaviour
{
    public Text scoreNumber;
    public static GameObject[] allCase;

    public static int score = 0;
    public static int oldScore = 0;
    public static int totalScore = 0;

    private void Awake()
    {
        //Initialize and order case
        allCase = GameObject.FindGameObjectsWithTag("Case");
        allCase = allCase.OrderBy(x => x.name).ToArray();
    }
    // Use this for initialization
    void Start()
    {
        Debug.Log("CASE " + allCase.Length);
        allCase[oldScore].GetComponent<Renderer>().material.color = Color.black;

    }

    //2D dice score
    public static void GetScore()
    {
        allCase[score].GetComponent<Renderer>().material.color = Color.white;
        allCase[oldScore].GetComponent<Renderer>().material.color = Color.white;
        oldScore = score;
        score += SoloDiceGB.finalSide;
        totalScore += SoloDiceGB.finalSide;

        if (score > allCase.Length - 1)
        {
            score = allCase.Length - 1;
            SceneManager.LoadScene("10.End");

        }
        allCase[score].GetComponent<Renderer>().material.color = Color.magenta;
        allCase[oldScore].GetComponent<Renderer>().material.color = Color.black;

    }

    // Update is called once per frame
    void Update()
    {
        //Update totalScore to UI
        scoreNumber.text = totalScore.ToString();
    }
}
```

Figure 75: Script SoloBoardScript
Source : Donnée personnelle

Le scripte « SoloDiceGB » gère le dé 2D. Il va charger les différentes images contenues dans le dossier « Resources > DiceSides » dans l'ordre alphabétique. Le *collider* permet de détecter le clic de l'utilisateur. Une fois lancé, il n'est plus possible de relancer le dé. Lorsque nous jouons le dé, la méthode « RollTheDice() » choisit aléatoirement un nombre entre zéro et cinq, et affiche l'image qui correspond à l'ordre dans le dossier « DiceSides ». Le score est ensuite incrémenté de un Si nous avons fait un trois, le dé va recevoir le rendu de l'image « dice3 », qui correspond à la face trois du dé. Nous ajoutons un au score, qui est récupéré par la méthode « SoloBoardScript.GetScore() ».

Nous avons repris le concept développé par Alexandre Zotov dans son tutoriel vidéo YouTube (Alexander Zotov, 2018). Même si nous aurions aimé utiliser notre jeu de plateau de dé en 3D, cette méthode permet de lancer rapidement le dé, directement dans le jeu. Notre dé 3D étant dans une autre scène, lors du passage de l'une à l'autre, il y avait quelques secondes de délai, le temps de charger la scène principale. De plus, nous n'avons pas réussi à lui donner l'aspect d'un vrai dé. Lors du lancement, même si nous utilisons la gravité, nous avons plus l'impression de lancer un gros cube en mousse qu'un véritable dé. Dans une prochaine étape, nous avons pour idée d'augmenter sa vitesse de rotation et de l'intégrer directement dans la scène principale, jouable avec un jeu de caméra.

```
6   public class SoloDiceGB : MonoBehaviour
7   {
8       private Sprite[] diceSides;
9       public static int finalSide = 0;
10      public Collider dieCollider;
11
12      private SpriteRenderer rend;
13
14      // Use this for initialization
15      void Start()
16      {
17          rend = GetComponent<SpriteRenderer>();
18          dieCollider = this.GetComponent<Collider>();
19          diceSides = Resources.LoadAll<Sprite>("DiceSides/");
20      }
21
22      private void OnMouseDown()
23      {
24          dieCollider.enabled = !dieCollider.enabled;
25          StartCoroutine("RollTheDice");
26      }
27
28      private IEnumerator RollTheDice()
29      {
30          int randomDiceSide = 0;
31
32          for (int i = 0; i <= 20; i++)
33          {
34              randomDiceSide = Random.Range(0, 6);
35              rend.sprite = diceSides[randomDiceSide];
36
37              yield return new WaitForSeconds(0.05f);
38          }
39          finalSide = randomDiceSide + 1;
40          SoloBoardScript.GetScore();
41
42          Debug.Log(randomDiceSide + " -> " + finalSide);
43      }
44  }
```

Figure 76: Script SoloDiceGB
Source : Donnée personnelle

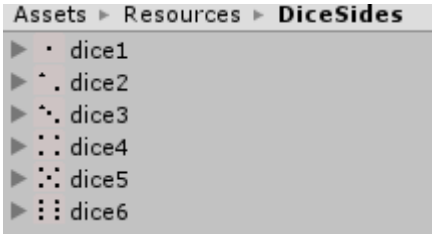



Figure 77: Dossier Resources > DiceSides
Source : Donnée personnelle

Nous avons enfin le script « SoloPlayerMove » qui va gérer les déplacements de l'utilisateur. Le joueur ne peut déplacer son joueur que sur les éléments ayant le tag « Case » et de la couleur blanc ou magenta. Nous avons inséré ce script dans la liste de personnage « CharacterListSolo ». Notre jeu reprend un déplacement de *click to* move. Le personnage se déplace là où nous avons pointé notre souris (« Creating a Click to Move Game », s. d.). Dans un premier temps, nous avions utilisé les « NavMesh » et « NavMeshAgent ». Le premier permet de construire un terrain et de délimiter des zones comme « Walkable », où le personnage peut marcher et à qui nous ajoutons un élément « NavMeshAgent » qui lui permet de se mouvoir et éviter les obstacles de manière automatique (« Unity - Scripting API », s. d.-b). À cela nous avions ajouté une caméra à la troisième personne qui suit le déplacement du personnage. Cette méthode rend une très bonne dynamique de jeu.

Les déplacements se calculent selon un vecteur à trois dimensions dans un jeu 3D, selon l'axe x, y et z. Dans notre cas, la position du personnage était bien gardée en mémoire. Mais quand nous répondions au quiz et que nous revenions sur le plateau, le vecteur correspondait à la position du joueur là où il devait se trouver, à la position x et z, mais il se trouvait toujours au premier niveau de notre plateau, zéro, selon l'axe y, même si la valeur de y était de par exemple 3.

À la suite d'un problème de positionnement, nous avons opté pour la « téléportation », le déplacement d'un pion comme nous le faisons avec un vrai jeu de l'oie. Le jeu va envoyer une ligne imaginaire et infinie, le point de départ de cette ligne est la caméra, en direction de l'endroit où nous cliquons avec notre souris, class « Ray ». Nous utilisons un « RaycastHit » avec la variable « hit » afin de retourner les informations de l'objet sur lequel la *ray*, la ligne imaginaire, a touché. Les éléments que nous voulons rendre accessibles, ici les cases, devront obligatoirement être composées d'un *collider* afin que la collision de l'objet et du *ray* soit détectés. Enfin, « Physics.Raycast » va être utilisé pour contrôler que le « hit » contienne des éléments (kristiel, 2016). Si c'est le cas, nous contrôlons que l'objet possède le tag « Case » et sa couleur.

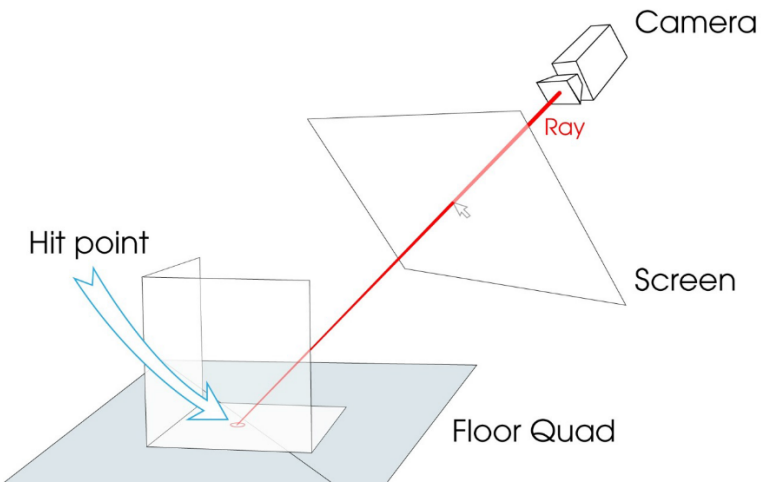



Figure 78: Fonctionnement ray et hit point
Source : Tiré de http://randomguydev.com/survival-shooter-in-unity/

Dans notre jeu, nous avons choisi d'illustrer par la couleur magenta la case où le joueur doit aller après le lancer du dé, et noir, pour l'endroit où il doit être après la réponse au quiz. Lorsque nous cliquons dessus, le « hit.point » (le point de collision) est le vecteur qui comprend les coordonnées où le joueur doit se déplacer à travers la méthode « MoveToLocation(Vector3 targetPosition) ». Si la case a une surcouche magenta, nous lançons le quiz. Si elle est noir, il doit relancer le dé.

```
7   public class SoloPlayerMove : MonoBehaviour
8   {
9       public GameObject playerObject;
10      public static Vector3 playerPos = new Vector3(3,0,3);
11
12      void Update()
13      {
14          playerObject.transform.position = playerPos;
15
16          if (Input.GetMouseButtonDown(0))
17          {
18              Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
19              RaycastHit hit;
20              if (Physics.Raycast(ray, out hit, 1000))
21              {
22                  //Player move to the score case
23                  if (hit.collider.gameObject.tag == "Case" && hit.collider.gameObject.GetComponent<Renderer>().material.color == Color.magenta)
24                  {
25                      StartCoroutine(SetTargetPosition(hit.point));
26                  }
27
28                  if (hit.collider.gameObject.tag == "Case" && hit.collider.gameObject.GetComponent<Renderer>().material.color == Color.black)
29                  {
30                      StartCoroutine(SetTargetOldPosition(hit.point));
31                  }
32              }
33          }
34      }
35
36      IEnumerator SetTargetPosition(Vector3 position)
37      {
38          MoveToLocation(position);
39          yield return new WaitForSeconds(2f);
40          SceneManager.LoadScene("08A.Persistent", LoadSceneMode.Additive);
41      }
42
43      IEnumerator SetTargetOldPosition(Vector3 position)
44      {
45          MoveToLocation(position);
46          yield return new WaitForSeconds(2f);
47      }
48
49      public void MoveToLocation(Vector3 targetPoint)
50      {
51          playerPos = targetPoint;
52      }
53  }
54
```

Figure 79: Script SoloPlayerMove
Source : Donnée personnelle

### 5.4.2. Multijoueur

Le mode multijoueur est la partie centrale de notre application. C'est dans ce mode de jeu que nous allons trouver toutes les fonctionnalités émises par le CREALP, bien qu'elles ne soient pas encore implémentées, dans une prochaine étape, en plus des cases « Village », nous allons implémenter diverses subtilités afin d'obliger les joueurs à collaborer. Nous avons pensé, par exemple, que si des joueurs se retrouvent sur une même case, afin de pouvoir se déplacer, ils doivent se concerter pour répondre à un quiz ou résoudre ensemble un puzzle. À cela, nous pouvons ajouter des bonus et malus selon la réponse donnée et la vision à court/moyen/long terme qui est attachée à la question et sa réponse.

L'implémentation du mode multijoueur nous a passablement retardés. L'intégration d'un *Lobby,* une salle d'attente avant de pouvoir jouer en ligne, s'est avéré facile par l'utilisation de l'*asset* d'Unity gratuit disponible sur son store. Mais la logique de jeu, l'accès et la configuration aux *gameObjects* sont différents. C'est l'une des raisons pour laquelle, nous avons séparé totalement le mode solo de celui-ci.

Nous avons trouvé trois moyens de concevoir le jeu multijoueur avec Unity : la solution d'Unity, Photon et PlayFab. Pour des raisons de temps et de simplicité, nous avons opté pour la solution d'Unity. Son *asset* est complet, et la configuration de base est bien documentée sur leur site. Nous avons également utilisé et adapté le tutoriel YouTube d'Holistic3D (Holistic3d, s. d.).

#### 5.4.2.1 Installation

Dans un premier temps, nous avons pris l'*asset* sur le store d'Unity. Le *package* est importé avec différents prefabs et des scènes d'exemple qui nous ont servi de base pour récupérer le nom et la couleur du joueur par exemple. Nous avons créé une scène « Lobby » à l'intérieur de laquelle, nous avons ajouté le prefab « LobbyManager ».

Le *gameObject* « LobbyManager » possède un script de configuration du même nom. Nous avons modifié et/ou ajouté les éléments entourés en rouge à la Figure 80 :

- Lobby Scene : notre scène qui fait office de lobby.
- Play Scene : la scène où les joueurs vont débuter la partie, ici, « 07.BoardGame ».
- Max Players : le nombre maximum de joueurs, quatre, qui peuvent jouer ensemble à une partie.
- Max Players Per Con[nection] : le nombre d'utilisateurs par ordinateur, ici un.
- Minimum Players : deux, le nombre de joueurs minimum pour débuter une partie.
- Game Player Prefab : prefab de notre personnage pour le jeu.
- Use WebSocket : « oui », permet à la version WebGL de se connecter à la partie hébergée avec la version Windows.
- Network Port : 80, le port de connexion. Si nous redirigeons le port 80 vers l'hôte, les utilisateurs WebGL peuvent s'y connecter depuis internet. Sinon, il ne fonctionne que dans un réseau local
- Player Spawn Metho[d] : Round Robin, permet de placer les joueurs à des endroits fixes au début de la partie.
- Prematch Countdown : Compteur avant le lancement de la partie, ici 10. Si une partie débute avec trois joueurs, le temps d'attente permet à un dernier joueur de s'y connecter.
- Script Event System Checker : Ajouter un objet « EventSystem » au prefab « LobbyManager » afin de récupérer les clics des joueurs
- Script Network Lobby Hook : permet de récupérer des informations du joueur comme la couleur.

Le *lobby* est un moyen coutumier du monde du jeu vidéo qui permet aux utilisateurs de se connecter à une partie. Dans notre cas, la scène qui le compose permettra de créer une table de jeu et d'attendre que des adversaires s'y connectent. Actuellement, cela n'est pas possible. L'unique moyen de jouer en ligne est d'utiliser une version du jeu compilée pour Windows et de créer une connexion.

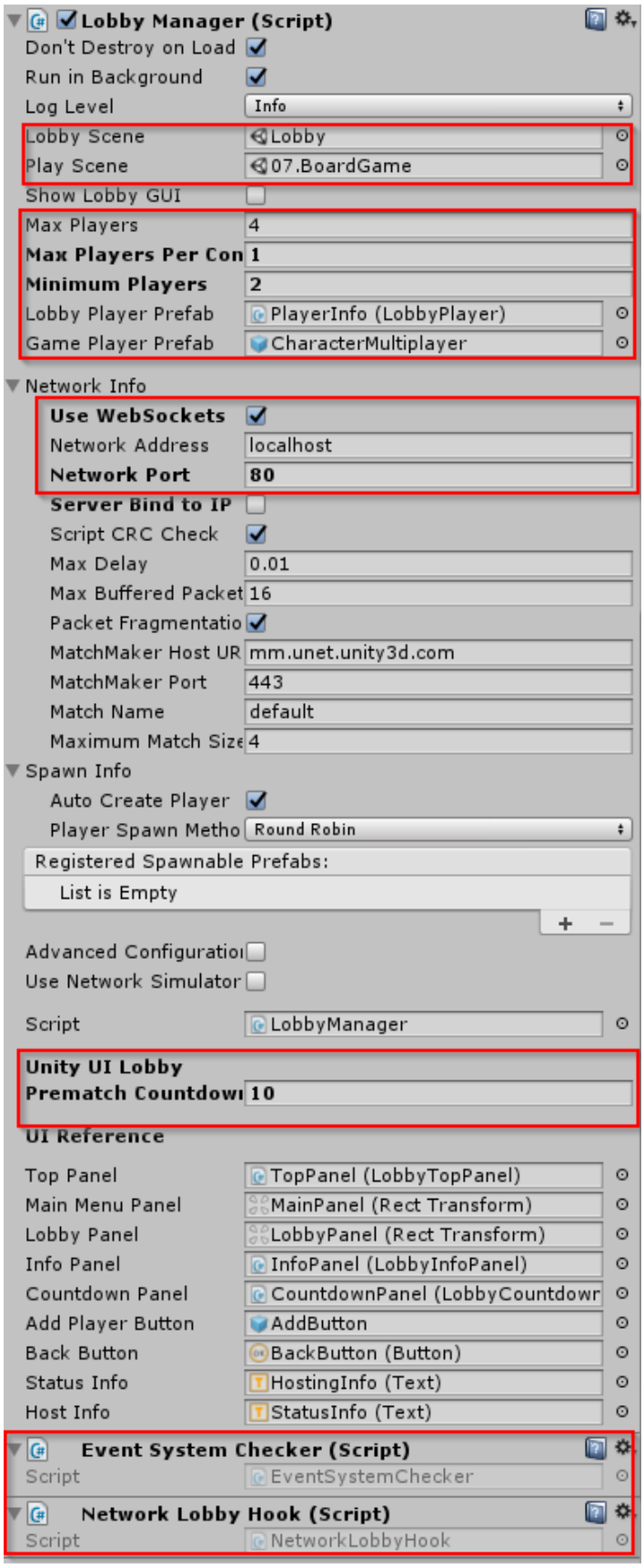



Figure 80: LobbyManager configuration
Source : Donnée personnelle

#### 5.4.2.2 Configuration du personnage

Dans ce mode, les joueurs ne disposent pas encore de la possibilité de choisir leur personnage. Afin que le « LobbyManager » puisse créer chacun des joueurs, nous devons lui fournir le prefab de l'avatar.

Nous avons choisi d'utiliser et modifier le prefab, nommé « CharacterMultiplayer », de l'un des personnages du Sail Character Pack (voir Figure 36, personnage du milieu). Pour pouvoir l'utiliser dans le mode multijoueur, nous avons dû ajouter divers composants :

- Network Identity : Ce composant d'Unity permet de donner une visibilité sur le réseau à l'objet auquel il est lié, dans le but qu'il soit visible parmi les autres utilisateurs (Cardinale, 2018). Nous avons utilisé ce composant afin de connaitre le joueur à qui appartient l'objet, ici le personnage.
- Network Transform : Cet élément permet de synchroniser la position des objets. Notre jeu étant en 3D, nous avons synchronisé les axes XYZ dans « Rotation Axis ».
- Setup Local Player : Ce script va construire notre personnage selon les informations du joueur : pseudo et couleur choisies dans le *lobby.*
- Player Move : Ce script permet de gérer le déplacement du personnage.

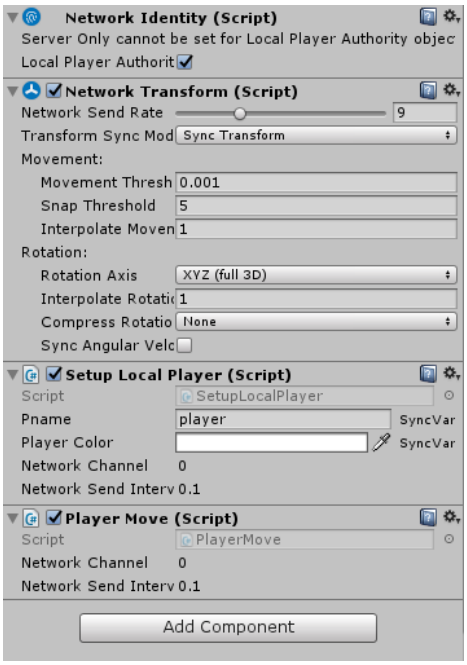



Figure 81: Configuration personnage du mode multijoueur
Source : Donnée personnelle

Une fois les modifications effectuées, nous avons dû glisser-déposer notre prefab dans le champ « Game Player Prefab » de la Figure 80. Notre joueur possède également un composant « Text Mesh » où nous affichons son nom.

Dans un jeu, nous avons la possibilité de faire apparaitre les joueurs soit à des positions fixes, soit aléatoires. Nous avons choisi le positionnement fixe, puisque nous voulons que tous les joueurs débutent sur la case de départ. Comme énoncé plus haut, le positionnement

du joueur s'effectue par l'argument « Player Spawn Method : Round Robin » dans le « LobbyManager ». Ensuite, il nous suffit de créer des *gameObjects* vides et attacher le composant « Network Start Position » à chacun des objets dans la scène de jeu. Lors du lancement de la partie, les joueurs sont placés automatiquement dans l'ordre de création de nos différents « SpawnPoint ».

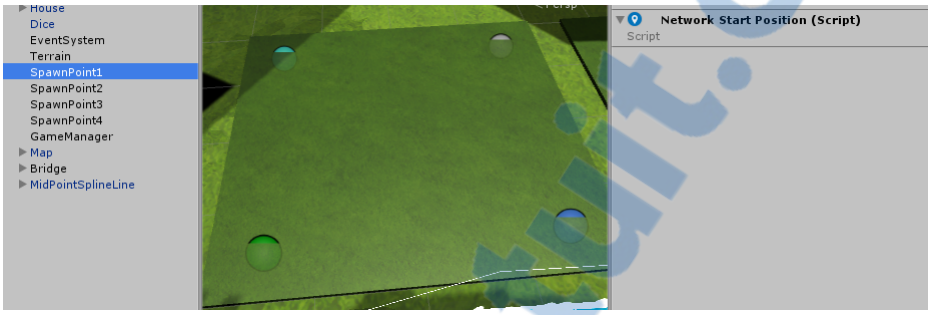



Figure 82: Configuration point d'apparition sur le plateau et configuration du LobbyManager
Source : Donnée personnelle

#### 5.4.2.3 Script multijoueur

Dans notre dossier « Multiplayer », nous avons cinq scripts :

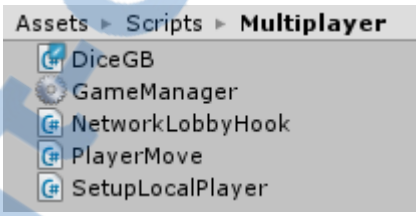





Figure 83: Dossier Multiplayer
Source : Donnée personnelle

Le script « DiceGB » est pareil que la classe « SoloDiceGB » du mode « Solo ». La seule différence et qu'il ne fait pas appel à la méthode « public static void GetScore() » (voir Figure 76). Ce script n'a comme unique but que de lancer le dé 2D et d'émettre un score. L'affichage de la case en magenta se fait via le « GameManager ».

Le « GameManager » ressemble à « SoloBoardScript ». Il n'a plus qu'une fonction, afficher la case magenta et/ou noir selon la valeur de « DiceGB.scorePlayer » et « DiceGB.oldScore ».

```
8   public class GameManager : NetworkBehaviour
9   {
10      public static GameObject[] allCase;
11
12      private void Awake()
13      {
14          allCase = GameObject.FindGameObjectsWithTag("Case");
15          allCase = allCase.OrderBy(x => x.name).ToArray();
16      }
17
18      void Update()
19      {
20          if (DiceGB.scorePlayer >= allCase.Length - 1)
21          {
22              DiceGB.scorePlayer = allCase.Length - 1;
23              SceneManager.LoadScene("10.End");
24          }
25          allCase[DiceGB.scorePlayer].GetComponent<Renderer>().material.color = Color.magenta;
26          allCase[DiceGB.oldScore].GetComponent<Renderer>().material.color = Color.black;
27      }
28  }
```

Figure 84: Script GameManager
Source : Donnée personnelle

« NetworkLobbyHook » est repris et adapté d'un des scripts d'exemple de l'*asset* « Lobby ». Il permet de charger différentes valeurs pour le personnage du joueur. Nous avons choisi de récupérer la couleur sélectionnée et d'afficher le pseudo. Ces éléments sont repris de la classe « SetupLocalPlayer »

```
7   public class NetworkLobbyHook : LobbyHook {
8
9       public override void OnLobbyServerSceneLoadedForPlayer(NetworkManager manager, GameObject lobbyPlayer, GameObject gamePlayer)
10      {
11          LobbyPlayer lobby = lobbyPlayer.GetComponent<LobbyPlayer>();
12          SetupLocalPlayer localPlayer = gamePlayer.GetComponent<SetupLocalPlayer>();
13
14          localPlayer.pname = lobby.playerName;
15          localPlayer.playerColor = lobby.playerColor;
16      }
17  }
```

Figure 85: Scipt NetworkLobbyhook
Source : Donnée personnelle, adaptée du script d'exemple de l'*asset* « Lobby »

Le script « PlayerMove » ressemble à celui du mode « Solo ». Il possède néanmoins quelques différences. Lorsque nous cliquons sur case magenta pour avancer, le lancement de la nouvelle scène s'effectue en utilisant la classe « NetworkManager » et sa méthode *void ServerChangeScene(string newSceneName)*, au lieu de faire appel à la classe « SceneManager .

Comme énoncé plus haut, ce script est attaché directement à notre prefab du personnage multijoueur. À cet effet, nous avons utilisé *DontDestroyOnLoad(this)* afin de ne pas détruire les personnages lors du passage de la scène du plateau vers les quiz. Nous avons également utilisé les instructions de la ligne 21 à 25 de la Figure 86 afin de n'attacher ce script qu'au joueur de l'utilisateur. La condition « if( !LocalPlayer) » permet d'exécuter la suite de la méthode qui n'aura impacté que le personnage du joueur. L'utilisation de cette condition n'est possible qu'avec l'ajout du composant « Network Identity » au *gameObject* lié au script, ici « CharacterMultiplayer ».

```
9   public class PlayerMove : NetworkBehaviour
10  {
11
12      public static GameObject playerObject;
13
14      private void Awake()
15      {
16          DontDestroyOnLoad(this.gameObject);
17      }
18      // Use this for initialization
19      private void Start()
20      {
21          if (!isLocalPlayer)
22          {
23              return;
24          }
25          playerObject = this.gameObject;
26      }
27
28      void Update()
29      {
30          if (Input.GetMouseButtonDown(0))
31          {
32              Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
33              RaycastHit hit;
34              if (Physics.Raycast(ray, out hit, 1000))
35              {
36                  //Player move to the score case
37                  if (hit.collider.gameObject.tag == "Case" && hit.collider.gameObject.GetComponent<Renderer>().material.color == Color.magenta)
38                  {
39                      StartCoroutine(SetTargetPosition(hit.point));
40                  }
41                  if (hit.collider.gameObject.tag == "Case" && hit.collider.gameObject.GetComponent<Renderer>().material.color == Color.black)
42                  {
43                      StartCoroutine(SetTargetOldPosition(hit.point));
44                  }
45              }
46          }
47      }
48
49      IEnumerator SetTargetPosition(Vector3 position)
50      {
51          MoveToLocation(position);
52          yield return new WaitForSeconds(2f);
53          //Load quizz scene
54          NetworkManager.singleton.ServerChangeScene("08A.Persistent");
55      }
56
57      IEnumerator SetTargetOldPosition(Vector3 position)
58      {
59          MoveToLocation(position);
60          yield return new WaitForSeconds(2f);
61      }
62
63      public void MoveToLocation(Vector3 targetPoint)
64      {
65          playerObject.transform.position = targetPoint;
66      }
67  }
```

Figure 86: Script multijoueur PlayerMove
Source : Donnée personnelle

« SetupLocalPlayer » a été adapté du tutoriel d'Holistic3D (Holistic3d, 2016). Il va configurer le personnage du joueur et ses valeurs comme la couleur et le nom.

```
7   public class SetupLocalPlayer : NetworkBehaviour
8   {
9
10      [SyncVar]
11      public string pname = "player";
12      [SyncVar]
13      public Color playerColor = Color.white;
14
15
16      [Command]
17      private void CmdChangeName(string newName)
18      {
19          pname = newName;
20          this.GetComponentInChildren<TextMesh>().text = pname;
21      }
22
23      private void Start()
24      {
25          //if (isLocalPlayer)
26          //{
27              Renderer[] rends = GetComponentsInChildren<Renderer>();
28              foreach (Renderer r in rends)
29              {
30                  r.material.color = playerColor;
31              }
32          //
33      }
34
35      private void Update()
36      {
37          this.GetComponentInChildren<TextMesh>().text = pname;
38      }
39  }
```

Figure 87:Script SetupLocalPlayer
Source : Donnée personnelle, adaptée de (Holistic3d, 2016)

## 5.5. Sauvegarde des données

L'utilisation d'une base de données dépend de l'application. Les requêtes à une base de données sont plus lourdes par exemple que la simple lecture d'un fichier Json dans lequel nous sauvegardons nos données. Selon la quantité de données et le nombre d'utilisateurs, une base de données demande vite beaucoup de coûts et de maintenance.

Nous avons décidé d'utiliser trois types de sauvegardes :

- Une solution XAMPP : base de données MySQL phpmyadmin, serveur PHP déployé sur un serveur cloud Ubuntu loué chez Scaleway, permis par l'utilisation du *Cross-origin resource sharing* et de la classe WWWForm et UnityWebRequest d'Unity.
- Utilisation de PlayerPref d'Unity, permettant d'effectuer des sauvegardes dans l'IndexedDB API du navigateur
- Lecture/écriture d'un fichier Json

### 5.5.1. Base de données - XAMPP

Nous avons dû utiliser une solution XAMPP installée sur un serveur distant, car le WebGL ne supporte pas les bases de données incluses, comme c'est le cas sur Android par exemple. Sur les jeux mobiles, nous pouvons utiliser les bases de données intégrées des différents mobiles : SQLite (« Mobile databases », 2016). La plateforme web n'ayant pas accès à une base de données directe, nous avons dû monter une solution XAMPP et créer des APIs PHP afin de pouvoir exécuter des requêtes HTTP (GET, POST).

Nous avons choisi d'utiliser la solution XAMPP, Apache, MySQL et PHP, pour sa simplicité d'installation et de configuration. De plus, nous avons déjà des connaissances en PHP. Dans le cadre d'une mise en production, il est important d'installer chacun des outils de manière séparée afin de garantir une meilleure sécurité, l'utilisation de XAMPP étant réservée à la phase de développement seulement. Nous avons également mis de côté l'aspect sécurité relatif à l'utilisation d'un serveur PHP et la notion d'authentification. Il est également possible de concevoir ce même type d'architecture avec d'autres outils comme une combinaison MongoDB et Node.js.

Nous avons installé cette solution sur un serveur *cloud* 1-XS loué chez Scaleway à 1.99 euro par mois. La puissance de cette machine, un cœur, 1Gb de RAM et 25Gb d'espace disque est suffisante pour un développement individuel. Nous avons préféré une installation distante à une locale, afin de ne pas devoir installer et configurer la solution sur les deux postes que nous utilisons et afin de permettre un aperçu durant le sprint 1 et 2 à notre client et d'y déployer notre *potentially shippable product*.

Lors de la phase initiale de conception du *serious game*, nous avions pensé à un jeu multijoueur sur un seul appareil. Ayant peu d'objets, seules les informations des différents joueurs sont stockées dans notre base de données, « gamesg », qui ne comporte qu'une seule table, « users », lors de la scène de sélection du pseudo, du clic sur le bouton « Suivant ».

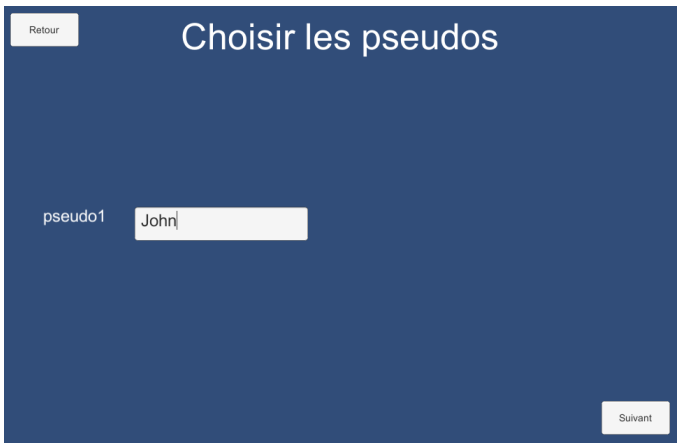



Figure 88: Sélection pseudo mode "Solo"
Source : Donnée personnelle

Le nombre de champs « TextField » est affiché selon le nombre de joueurs prévus à la scène précédente et nous avons développé un contrôleur sur chacun des champs actifs afin d'empêcher des valeurs nulles dans les pseudos :

```
67      //Active les InputFiled
68      private void EnableInputField(int value)
69      {
70          Debug.Log("Value " + value);
71          //Desactive les InputField
72          DisableInputField();
73
74          //Active les InputField - selon value
75          for (int i = 0; i < value; i++)
76          {
77              pseudo[i].gameObject.SetActive(true);
78              label[i].gameObject.SetActive(true);
79          }
80
81
82      }
83      //Desactive les InputFiled
84      private void DisableInputField()
85      {
86          for (int i = 0; i < pseudo.Length; i++)
87          {
88              pseudo[i].gameObject.SetActive(false);
89              label[i].gameObject.SetActive(false);
90          }
91      }
```

Figure 89: Méthode (dés)activation InputField
Source : Donnée personnelle

```
47      void Update()
48      {
49          //Pseudo controlleur
50          for (int i = 0; i < nbrePlayers; i++)
51          {
52              if (string.IsNullOrEmpty(pseudo[i].text) || pseudo[i].text.Equals(" "))
53              {
54
55                  nextBtn.gameObject.SetActive(false);
56                  pseudoTitle.gameObject.SetActive(true);
57              }
58              else
59              {
60                  nextBtn.gameObject.SetActive(true);
61                  pseudoTitle.gameObject.SetActive(false);
62              }
63
64          }
65      }
```

Figure 90: Contrôleur champs pseudo
Source : Donnée personnelle

Dans notre table, nous sauvegardons la position du joueur, la case sur laquelle il se trouve, son pseudo enregistré lors de la scène de sélection d'un pseudo et le score obtenu lors du lancer du dé. L'ID est généré automatiquement lors de l'enregistrement du pseudo. Faute de temps, ces différentes informations ne sont pas mises à jour dans la suite du jeu. Une partie débutée ne peut pas pour l'instant être sauvegardée. Seul, les valeurs de notre objet « Player » sont modifiées.

```
1   public class Player {
2
3       private string pseudo;
4       public int score;
5       public int position;
6       public bool played;
7
8       public Player(string pseudo, int score, int position, bool played)
9       {
10          this.Pseudo = pseudo;
11          this.score = score;
12          this.position = position;
13          this.played = played;
14      }
15      public string Pseudo    { get { return pseudo; } set { pseudo = value; } }
16      public int Score { get { return score; } set { score = value; } }
17      public int Position { get { return position; } set { position = value; } }
18      public bool Played { get { return played; } set { played = value; } }
19
20  }
```

Figure 91: Classe Player
Source : Donnée personnelle

Lors de la création, les valeurs sont initialisées comme suit :

- Id : auto-incrémenté
- Position : 0
- Pseudo : choisi par le joueur dans la scène de sélection du pseudo
- Score : -1

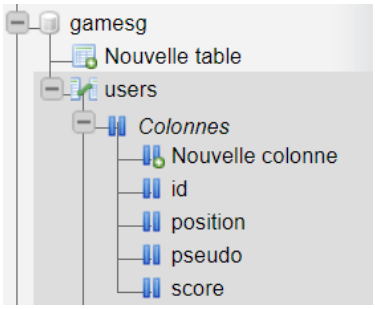



Figure 92: Base de données, table *users*
Source : Donnée personnelle

Le WebGL ayant des restrictions, les accès à la base de données se font via la classe WWWForm et UnityWebRequest d'Unity et notre API PHP. Dans notre script « ChoosePseudo », nous initialisons la variable « url » à l'adresse de notre API et les différentes variables de notre objet « Player ».

```
public class ChoosePseudo : MonoBehaviour
{

    public InputField[] pseudo = new InputField[4];
    //Label pseudo
    public Text[] label = new Text[4];
    public Text pseudoTitle;
    public Button nextBtn;
    public static int nbrePlayers;
    public static string[] pseudoName;
    public static List<Player> players = new List<Player>();
    private static bool wwwResult = false;
    string url = "http://163.172.150.132/SeriousGame/PostUser.php";
    int score = -1;
    int position = 0;
    bool played;
```

Figure 93: Base de données - URL API PHP
Source : Donnée personnelle

Dans notre méthode « SavePseudoName() », nous exécutons la méthode plusieurs fois selon le nombre de joueurs. Nous utilisons une coroutine, méthode de type « IEnumarator », afin d'attendre le retour du téléversement des informations de chaque joueur avec l'instruction « yield return ». Nous avons établi un formulaire « WWWForm » dans laquelle nous insérons les données et envoyons le tout à travers un objet « UnityWebRequest » afin de communiquer avec notre serveur, avec comme argument l'URL et le formulaire. Dans le même temps, nous créons autant d'objets « Player » que de nombre de joueurs, et les ajoutons à une liste « players » de « Player ».

```
98    IEnumerator SavePseudoName()
99    {
100       for (int i = 0; i < nbrePlayers; i++)
101       {
102           Debug.Log(i);
103           // Create a form object for sending data to the server
104           WWWForm form = new WWWForm();
105           // The variable of the player submitting
106           pseudoName[i] = pseudo[i].text.ToString();
107           form.AddField("pseudo", pseudoName[i]);
108           // The score
109           form.AddField("score", score);
110           form.AddField("position", position);
111           played = false;
112           Player player = new Player(pseudoName[i], score, position, played);
113           players.Add(player);
114           Debug.Log(pseudoName[i] + score + position + played);
115
116           // Create a download object
117           var download = UnityWebRequest.Post(url, form);
118
119           // Wait until the download is done
120           yield return download.SendWebRequest();
121
122           if (download.isNetworkError || download.isHttpError)
123           {
124               print("Error downloading: " + download.error);
125               wwwResult = false;
126           }
127           else
128           {
129               // Load next screen
130               wwwResult = true;
131               Debug.Log(download.downloadHandler.text);
132           }
133       }
134       //If post success load next screen
135       if (wwwResult && nbrePlayers > 1)
136       {
137           SceneManager.LoadScene("05A.LaunchDice");
138       }
139       else if (wwwResult && nbrePlayers == 1)
140       {
141           SceneManager.LoadScene("06.SelectCharacter");
142       }
143   }
```

Figure 94: Méthode SavePseudoName()
Source : Donnée personnelle

Les informations sont envoyées à notre API PHP et mappées avec les données ci-dessus à travers $_POST[‘ « variable »’] et nous exécutons une requête INSERT dans notre base de données avec « mysqli_querry ». Afin d’accepter les accents sur les pseudos, nous avons choisi utf8_general_ci comme interclassement à notre base de données et spécifié dans le *header* l’utilisation du charset utf-8 :

```php
<?php
        header('Content-type: text/html; charset=UTF-8');
        $servername = "localhost";
        $username = "root";
        $password = "AdminHevs01";
        $dbName = "gamesg";
        $table    = "users";

        //Make Connection
        $conn = new mysqli ($servername, $username, $password, $dbName);
        mysqli_set_charset($conn,  "utf8" );

        /* Vérification de la connexion */
        if (mysqli_connect_errno()) {
                printf("Echec de la connexion : %s\n", mysqli_connect_error());
                exit();
        }
        $id             = null;
        $pseudo         = mysqli_real_escape_string($conn, $_POST['pseudo']);
        $score          = mysqli_real_escape_string($conn, $_POST['score']);
        $position       = mysqli_real_escape_string($conn, $_POST['position']);

        mysqli_query($conn, "INSERT INTO $table (id, pseudo, score, position) VALUES ('$id', '$pseudo', '$score', '$position')");

        // Close the connection, we're done here.
        mysqli_close($conn);

?>
```

Figure 95: API PHP insertion des joueurs dans la base de données
Source : Donnée personnelle

Travaillant sur une solution distante, nous avons également dû autoriser le partage des ressources cross-origine (CORS). Cette autorisation s'effectue au niveau du serveur web, dans notre cas, Apache. Nous avons ajouté la balise d'acceptation ci-dessous dans le fichier situé dans /opt/lamp/etc./extra/httpd-default.conf de notre solution XAMPP sous Ubuntu serveur :

```
<IfModule mod_headers.c>
    Header set Access-Control-Allow-Origin: *
</IfModule>
```

Figure 96: Activation CORS Apache
Source : Donnée personnelle

Le partage de ressources peut être personnalisé en ciblant des pages spécifiques qui l'acceptent, en utilisant l'URL au lieu du symbole « * » utilisé ici. Il peut également autoriser seulement certaines méthodes HTTP (GET,PUT,DELETE, etc.) avec l'instruction : Header always set Access-Control-Allow-Methods "POST, GET, OPTIONS, DELETE, PUT" (Horn, 2014).

Dans sa version actuelle, la base de données n'est utilisée que sur le mode « Solo ». Lors du sprint 3, nous avons convenu de concevoir un jeu vraiment multijoueur, où chaque utilisateur peut accéder au jeu depuis son propre appareil. L'implémentation du mode multijoueur a pris plus de temps que prévu, ce qui nous a empêchés d'utiliser la sauvegarde des variables joueurs dans la base de données.

### 5.5.2. PlayerPref

Nous avons également utilisé une autre méthode d'accès et de sauvegarde des préférences de l'utilisateur. La classe PlayerPref d'Unity va sauvegarder les choix de l'utilisateur et nous pouvons y accéder par la suite. Selon la plateforme de développement choisie dans Unity, cet élément est stocké à différents emplacements. Sur une plateforme macOS, par exemple, elle sera stockée dans le dossier ~/Library/Preferences. Sur la plateforme WebGL, les PlayerPrefs sont sauvegardés dans l'IndexedDB API du navigateur utilisé (« Unity - Scripting API », s. d.-c).

Les PlayerPref ne permettent de sauvegarder que certains types de valeurs : int, string et float (« TUTO UNITY FR : Sauvegarder des données avec les PlayerPrefs (Unity3D) », 2018). Ces éléments sont utilisés pour mémoriser les paramètres du jeu choisis par l'utilisateur comme le volume ou l'axe de la souris par exemple. Nous avons utilisé cette méthode de sauvegarde sur le mode « Solo » lors de la sélection du personnage. L'avatar choisi par l'utilisateur est ainsi gardé en mémoire pour une prochaine partie, du moment que le cache du navigateur n'est pas vidé, à travers la méthode à la ligne 16. Le paramètre « 0 » permet de définir la valeur par défaut si le *PlayerPrefs* est encore inexistant.

```
14      private void Start()
15      {
16          index = PlayerPrefs.GetInt("CharacterSelected", 0);
17
18          characterList = new GameObject[transform.childCount];
19
20          //Fill array with character
21          for (int i = 0; i < transform.childCount; i++)
22          {
23              characterList[i] = transform.GetChild(i).gameObject;
24          }
25          //Visible off of all character
26          foreach (GameObject go in characterList)
27          {
28              go.SetActive(false);
29          }
30          //Visible true selected character
31          if (characterList[index])
32          {
33              characterList[index].SetActive(true);
34          }
35      }
```

Figure 97: PlayersPrefs Get
Source : Donnée personnelle

Lors de la confirmation, nous allons sauvegarder la nouvelle valeur via la méthode « PlayerPrefs.SetInt(« CharacterSelected », index); ».

```
66      public void ConfirmBtn()
67      {
68          PlayerPrefs.SetInt("CharacterSelected", index);
69          SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 2);
70      }
```

Figure 98: PlayersPrefs Set
Source : Donnée personnelle

### 5.5.3.Json

Json est un format de données comme XML. Il permet la sérialisation et déserialisation d'objet. Il sert de fichier de configuration ou pour encoder des documents (« JavaScript Object Notation », 2018). Nous avons utilisé le format Json dans la traduction du jeu et avons prévu de l'utiliser également pour les quiz, qui sont actuellement toujours écrits en dur dans Unity.

Ce *serious game* étant destiné au Valais, il était important d'avoir un jeu multilingue. Pour concevoir une application supportant plusieurs langues, nous avons choisi le format de

données Json. Nous aurions pu également utiliser XML, mais notre fichier Json pouvant être amené à évoluer, nous pensons que l'utilisation de ce format de données et plus *human readable* qu'un fichier XML. Il est également plus léger et rapide qu'un accès à une base de données à chaque changement de scènes ou de données. Les différents fichiers Json qui contiennent les langues sont contenus dans le dossier « StreamingAssets » situé à la racine de notre projet.

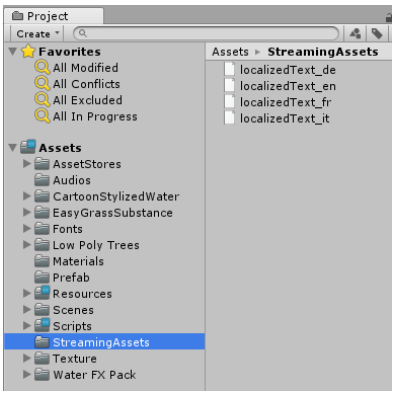



Figure 99: Dossier StreamingAssets
Source : Donnée personnelle

Ce même dossier se retrouve une fois l'application compilée et il est à exporter sur notre serveur web. Si par la suite, nous devons effectuer des changements, il suffit d'accéder au serveur FTP, télécharger le fichier, effectuer les modifications avec le logiciel Bloc-Notes ou Notepad++ et remettre sur le serveur la nouvelle version. Ainsi les données sont mises à jour de manière simple et efficace, sans avoir besoin de connaissances poussées en informatique et dans l'utilisation d'une base de données. C'est pourquoi nous aurions aimé développer cette méthode également pour les quiz, afin de donner la possibilité d'ajouter de nouvelles questions plus facilement.



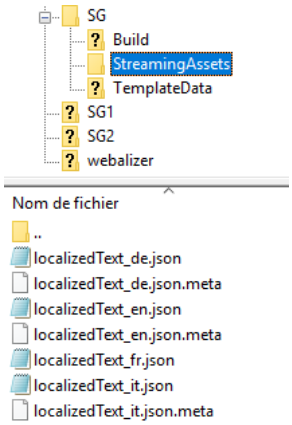



Figure 100 : Fichier Json sur FTP
Source : Capture d'écran réalisée par l'auteur – serveur FTP

### 5.6. Résultat

Nous avons eu beaucoup de mal à gérer l'implémentation du mode multijoueur. Après un long travail de réflexion, nous avons décidé de séparer totalement la partie « Solo » du « Multijoueur », afin d'éviter tous problèmes de fonctionnement dans le modes « Solo » qui, elle, est fonctionnelle. Le mode « Multijoueur » est jouable, mais il possède encore quelques défauts et des contraintes liées en partie à la synchronisation des différents états parmi les utilisateurs.

Actuellement, nous avons également dû compiler une version Windows de notre jeu, car nous n'avons pas eu le temps d'implémenter les WebSocket. Cette version est la seule qui permette d'héberger une partie où les autres joueurs puissent se connecter en utilisant son adresse IP.

L'ordre des joueurs n'est pas implémenté. A ce stade, cela n'a aucune importance, mais il est impératif de respecter le même ordre. Notre jeu contient un souci de connexion que nous n'avons pas encore eu le temps de déboguer. Après avoir joué, les autres joueurs perdent la possibilité de se déplacer, bien que le déplacement des autres joueurs soit synchronisé correctement. Lorsque le premier joueur rejoue, cela réinitialise la connexion et permet aux adversaires de se déplacer à nouveau.

Ensuite, lorsque le premier joueur joue et que la scène des questions s'affiche, tous les joueurs sont redirigés vers cette scène. Une question aléatoire apparait également sur leur écran. Cela n'affecte aucunement le score du premier utilisateur. Lorsque les utilisateurs hôtes doivent répondre à un quiz, la question ne s'affiche que sur l'écran de ce joueur.

## 6. Gestion du projet

Dans cette partie, nous verrons la manière dont nous avons géré notre projet. Ce travail a été élaboré selon la méthodologie Agile. Nous allons d'abord expliquer les différents *sprints* et leur *burn down chart* respectif. Ensuite nous exposerons d'où proviennent les ressources utilisées pour la recherche, ainsi que sa gestion. Enfin, nous expliquerons les différents outils utilisés afin de gérer au mieux son intégration continue.

### 6.1. Projet

La gestion de notre projet s'est déroulée selon la méthodologie Agile. Elle permet une grande flexibilité ainsi que l'intégration de notre client au développement de l'application. Elle correspond parfaitement avec le mode de développement d'un jeu vidéo qui doit être réactif, en proposant, par exemple, de nouvelles mises à jour qui comprennent de nouveaux quiz et mode de jeu afin de ne pas lasser le joueur. Ainsi, les risques de la déception des attentes du client sont faibles.

Nous avons choisi de développer ce projet en cinq *sprints.* Le *sprint* zéro correspond à la phase de recherche. Nous y avons consacré un mois et demi. Par la suite, nous avons effectué des *sprints* de trois semaines. À la fin de chacun des *sprints*, nous avons rencontré notre professeur afin qu'il teste et accepte les *users stories* choisies dans chaque étape. Afin de rendre un prototype fonctionnel, nous avons privilégié certaines fonctionnalités et mis de

côté certaines autres de moindre importance qui peuvent être développées dans un prochain *sprint*.

Lors de la phase de préparation, nous avions prévu de consacrer à ce projet environ 18 heures par semaines, trois journées de six heures par semaines, pour un total d'environ 431 heures.

En Annexe VI Product Backlog, nous trouvons le *product backlog* de notre projet. Il contient toutes les *users stories*, fonctionnalités, que nous avons implémentées ou planifiées pour une prochaine étape. Certaines de ces *users stories*, bien qu'acceptées et intégrées, ne sont plus disponibles dans le prototype. Cela s'explique par le fait que, lors du *sprint* trois, nous avons décidé de changer le mode de jeu. La logique de jeu, qui découle de ce changement, a fait que nous avons dû supprimer une partie de ce qui a été développé.

| Semaine | Du | Au | Formation | Périodes hebdomadaires | TB |
|---|---|---|---|---|---|
| 8 | 19.02.2018 | 24.02.2018 | P1 | 13 | 10 |
| 9 | 26.02.2018 | 03.03.2018 | P2 | 13 | 5 |
| 10 | 05.03.2018 | 10.03.2018 | P3 | 13 | 5 |
| 11 | 12.03.2018 | 17.03.2018 | P4 | 13 | 5 |
| 12 | 19.03.2018 | 24.03.2018 | P5 | 13 | 5 |
| 13 | 26.03.2018 | 31.03.2018 | P6 | 13 | 20 |
| 14 | 02.04.2018 | 07.04.2018 | Vacances de Pâques | | 27 |
| 15 | 09.04.2018 | 14.04.2018 | P7 | 13 | 18 |
| 16 | 16.04.2018 | 21.04.2018 | P8 | 13 | 18 |
| 17 | 23.04.2018 | 28.04.2018 | P9 | 13 | 18 |
| 18 | 30.04.2018 | 05.05.2018 | P10 | 9 | 18 |
| 19 | 07.05.2018 | 12.05.2018 | P11 | 9 | 18 |
| 20 | 14.05.2018 | 19.05.2018 | P12 | 9 | 18 |
| 21 | 21.05.2018 | 26.05.2018 | P13 | 9 | 18 |
| 22 | 28.05.2018 | 02.06.2018 | P14 | 9 | 18 |
| 23 | 04.06.2018 | 09.06.2018 | P15 | 9 | 18 |
| 24 | 11.06.2018 | 16.06.2018 | P16 | 9 | 18 |
| 25 | 18.06.2018 | 23.06.2018 | | | 27 |
| 26 | 25.06.2018 | 30.06.2018 | Examens | | |
| 27 | 02.07.2018 | 07.07.2018 | | | 27 |
| 28 | 09.07.2018 | 14.07.2018 | | | 27 |
| 29 | 16.07.2018 | 21.07.2018 | | | 27 |
| 30 | 23.07.2018 | 28.07.2018 | | | 27 |
| 31 | 30.07.2018 | 04.08.2018 | | | 27 |
| 32 | 06.08.2018 | 11.08.2018 | Retour TB 08.08.2018 à 12h00 | | 12 |
| 33 | 13.08.2018 | 18.08.2018 | | | |
| 34 | 20.08.2018 | 25.08.2018 | | | |
| 35 | 27.08.2018 | 01.09.2018 | Défense orale | | |
| 36 | 03.09.2018 | 08.09.2018 | Défense orale | | |
| | | | | | |
| | | | | Total heures | 431 |

Figure 101: Plan des heures consacrées au projet
Source : Donnée personnelle

#### 6.1.1.1. Sprint 0

Nous avons consacré plus de temps au *sprint* 0. Ce sprint englobe la phase de recherche de l'état de l'art, des technologies, la conception des *users stories*, ainsi que la rencontre avec le client. Elle nous a occupés durant un peu plus d'un mois et demi.

#### 6.1.1.2. Sprint 1

Durant le premier *sprint*, nous avons développé les *users stories* liées à l'architecture du jeu. Nous avons créé les premières scènes. Le joueur pouvait déjà lancer un dé, même si l'US n°15 du Tableau 2 a été rejeté. Bien que les joueurs puissent lancer le dé, nous n'avions pas réussi à programmer sa relance en cas de scores égaux. Nous avons passablement buté sur cette fonctionnalité, qui n'est d'ailleurs pas incluse dans le prototype final. N'arrivant pas à résoudre ce problème, nous l'avons mis de côté. Dans le *sprint* deux, nous l'avons séparé en deux, US 15 et 33 afin de valider la partie fonctionnelle. Durant ce *sprint*, voyant que nous restions bloqués, nous avons décidé d'y ajouter l'US n°2 afin de pouvoir avancer.

| | |
|---|---|
| 1 | En tant qu'utilisateur, je veux avoir un message d'introduction (écrit) afin de comprendre le but et raison du jeu. |
| 3 | En tant qu'utilisateur, je veux avoir un texte parlé également dans le message d'introduction afin que les moins jeunes puissent également comprendre le message ou les paresseux de la lecture. |
| 4 | En tant qu'utilisateur, je veux pouvoir faire pause lors de la lecture orale du texte afin d'aller moins vite et comprendre mieux le message. |
| 6 | En tant qu'utilisateur, je veux pouvoir passer l'introduction afin de ne pas devoir l'écouter une Xième fois. |
| 7 | En tant qu'utilisateur, je veux avoir un menu afin de jouer au jeu, connaitre les règles, ou accéder aux réglages du jeu. |
| 8 | En tant qu'utilisateur, je veux lancer un nouveau jeu. |
| 13 | En tant qu'utilisateur, je veux sélectionner le nombre de joueurs qui vont jouer. |
| 14 | En tant qu'utilisateur, je veux ajouter un pseudo à mon joueur. |
| 15 | En tant qu'utilisateur, je veux lancer un dé. |
| 2 | En tant qu'utilisateur, je veux changer la langue du message d'introduction en FR-DE-EN-IT. |

Tableau 2: User stories sprint 1
Source : Donnée personnelle

Nous avons sous-estimé l'effort à fournir pour la réalisation des *users stories* qui composent ce premier *sprint.* Cela s'explique par le fait que nous étions encore dans une phase de maitrise d'Unity.

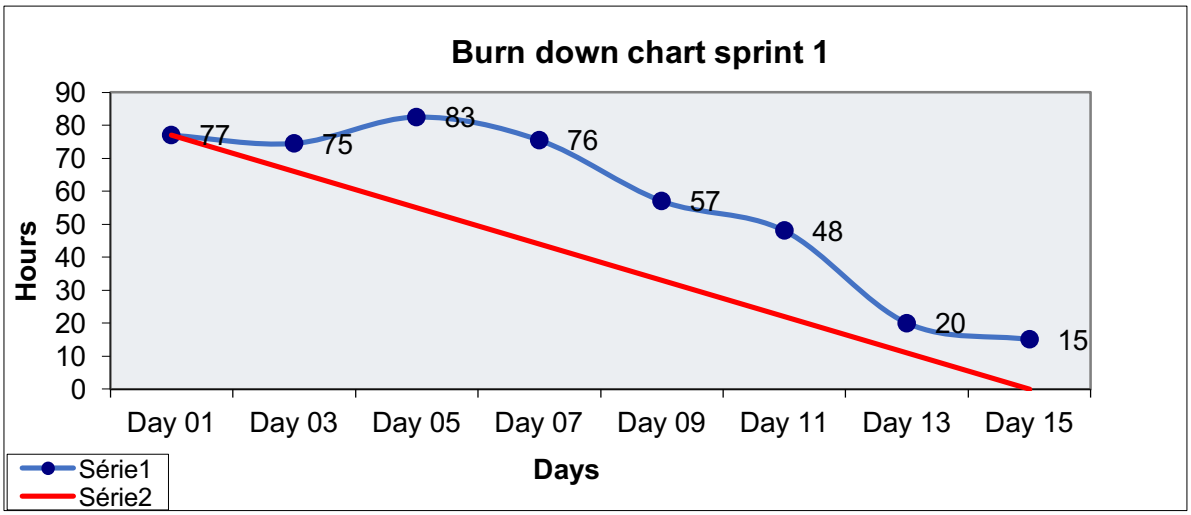



Figure 102: Burn down chart sprint 1
Source: Donnée personnelle

#### 6.1.1.3. Sprint 2

Durant le *sprint* deux, nous avons eu des soucis pour l'implémentation du tour par tout lors de la sélection des personnages disponibles dans le mode « Solo » (US n°16)). Dans le mode multijoueur sur un écran, cette sélection ne fonctionnait que pour le premier joueur. Ce blocage a également eu comme conséquence que nous n'avons pas pu réaliser l'US 17. Ce blocage nous a permis de changer le fonctionnement de notre *serious game* en implémentant un mode multijoueur à travers le réseau.

| | |
|---|---|
| 15 | En tant qu'utilisateur, je veux lancer un dé afin de décider l'ordre des joueurs et de pouvoir jouer. |
| 16 | En tant qu'utilisateur, je veux sélectionner un personnage afin de savoir qui je suis durant la partie. |
| 17 | En tant qu'utilisateur, je veux savoir qui joue à chaque lancer de dé afin de ne pas me tromper. |
| 18 | En tant qu'utilisateur, je veux lancer un dé afin d'avancer selon le score obtenu. |
| 19 | En tant qu'utilisateur, je veux recevoir un quizz aléatoire sur les cases "Standard", afin de pouvoir avancer ou non selon la réponse. |
| 20 | En tant qu'utilisateur, je veux revenir à ma place en cas de mauvaise réponse. |
| 21 | En tant qu'utilisateur, je veux rester sur la case en cas de bonne réponse à la question d'arriver plus vite au sommet. |

Tableau 3: User stories sprint 2
Source : Donnée personnelle

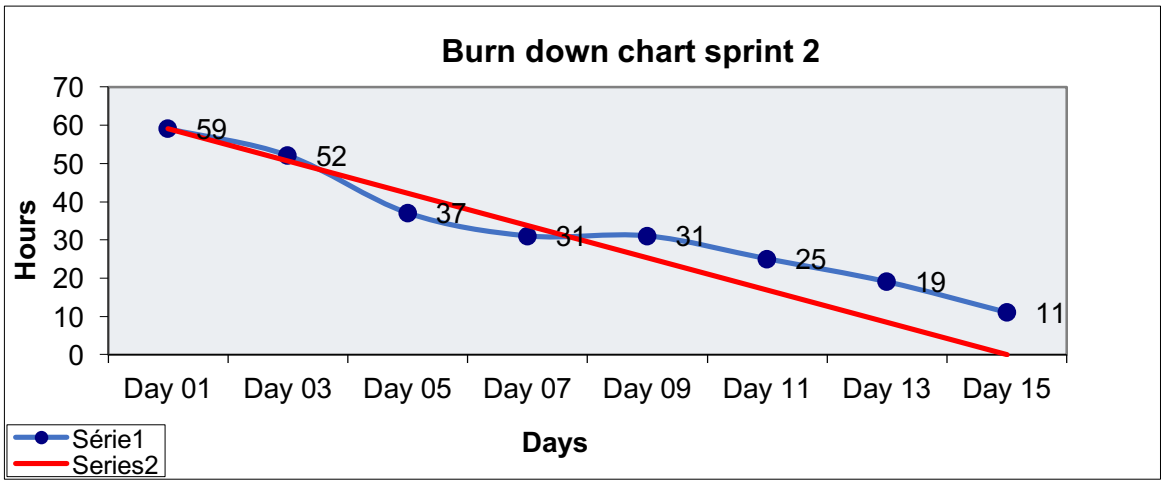



Figure 103: Burn down chart sprint 2
Source: Donnée personnelle

#### 6.1.1.4. Sprint 3

Durant ce troisième *sprint*, nous avons implémenté le changement majeur de notre jeu : créer le mode multijoueur. Nous avions beaucoup d'appréhension quant à cette implémentation. Mais à cette étape du développement, nous n'avons rencontré aucun problème particulier.

| | |
|---|---|
| 34 | En tant qu'utilisateur, je veux créer un jeu afin jouer à plusieurs sur des PCs différents. |
| 35 | En tant qu'utilisateur, je veux rejoindre une partie afin de jouer à plusieurs sur des PCs différents. |
| 36 | En tant qu'utilisateur, je veux me différencier des autres joueurs afin de savoir qui je suis durant la partie. |
| 37 | En tant qu'utilisateur, je veux voir l'évolution des autres joueurs. |

Tableau 4: User stories sprint 3
Source : Donnée personnelle

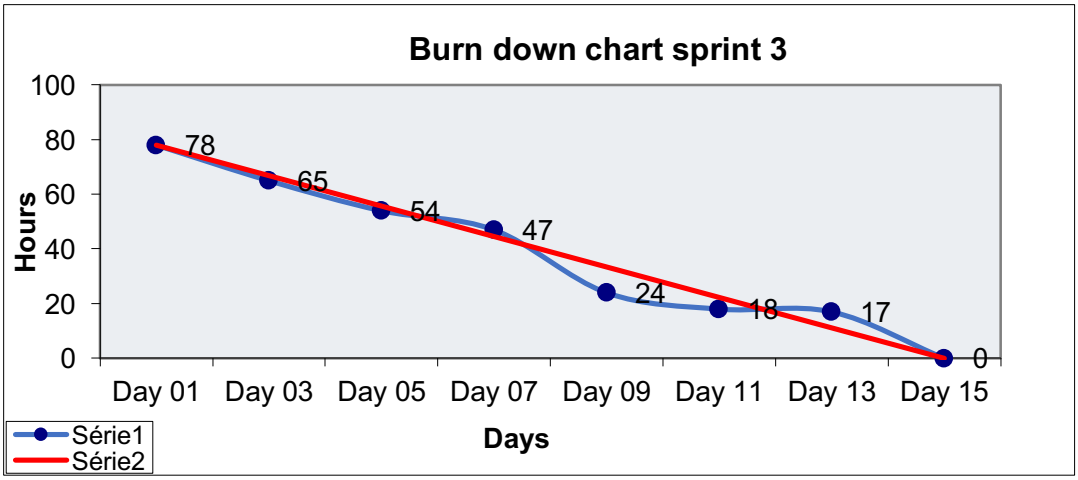



Figure 104: Burn down chart sprint 3
Source: Donnée personnelle

#### 6.1.1.5. Sprint 4

Le *sprint* quatre est notre dernière phase de développement. Elle nous a permis de finir notre prototype. Notre *serious game* est jouable. La *user storie* n°38 a été rejetée, partiellement. Nous n'avons pas été en mesure de l'implémenter totalement (voir 5.6).

| | |
|---|---|
| 38 | En tant qu'utilisateur, je veux voir en évidence la case sur laquelle je dois aller afin de connaitre mon prochain mouvement. |
| 39 | En tant qu'utilisateur, je veux être placé sur la case de départ afin de débuter la partie. |
| 40 | En tant qu'utilisateur, je veux une animation d'eau et des éléments de décor afin d'avoir un jeu plus attrayant visuellement |
| 41 | En tant qu'utilisateur, je veux répondre à des quiz pertinents (futur de l'eau sur le haut-plateau) |
| 42 | En tant qu'administrateur, je veux ajouter, modifier, supprimer un quiz afin de varier le jeu. |

Tableau 5: User stories Sprint 4
Source : Donnée personnelle

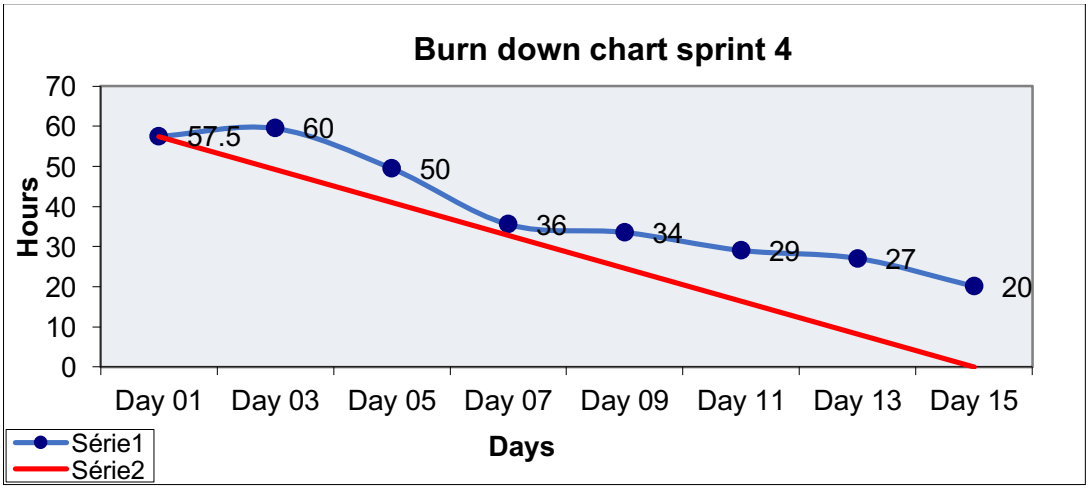



Figure 105: Burn down chart sprint 4
Source: Donnée personnelle

#### 6.1.1.6. Release roadmap – velocity

Afin d'avoir un meilleur aperçu de l'évolution de notre projet, nous présenterons ici la gestion selon la méthodologie Agile, illustrée par différents graphiques.

À chaque itération, nous avons pris en moyenne 25 SP. Sur l'ensemble du travail, nous avons réalisé 20,5 SP par itération. Cela est dû aux difficultés rencontrées lors du *sprint* quatre, où nous n'avons réalisé que 13 SP. Une autre raison est que nous avons favorisé la rédaction de ce rapport au développement.

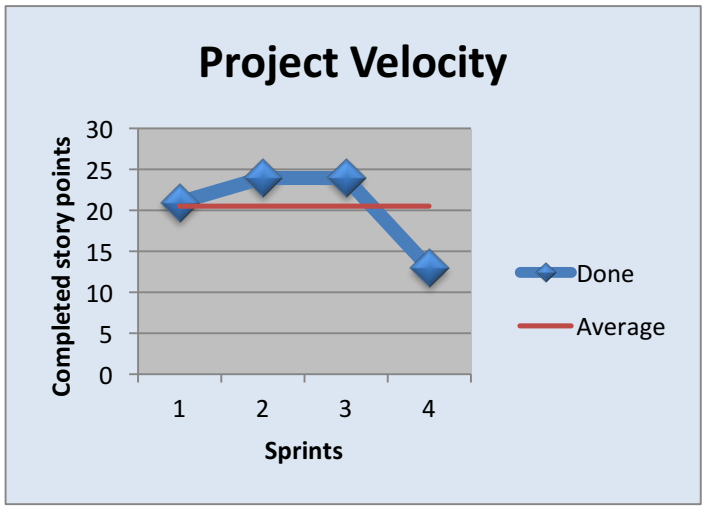



Figure 106: Vélocité du projet
Source : Donnée personnelle

À la Figure 107, nous avons la *release roadmap*. Nous avons eu une bonne marge de progression entre le sprint un et trois : 21, 24 et 24 *story points*. Il faut noter qu'au *sprint* 3 et 4, toutes les US sur lesquelles nous avons travaillé sont nouvelles et qu'elles n'avaient pas été prévues lors de la phase initiale d'écriture des *users stories*. C'est pourquoi, le nombre de SP *Todo* n'a pas diminué.

À la fin de ces quatre *sprints*, selon notre jugement des SP restants, nous sommes à la moitié de notre application. Afin de terminer le développement du *serious game*, nous estimons qu'il nous faudrait encore sept à neuf mois à temps plein. Ce délai s'explique par la difficulté à produire ce type de jeu seul. Les quiz sont intégrés, mais il reste encore à récolter des données auprès de CREALP et à construire les autres types de jeu.

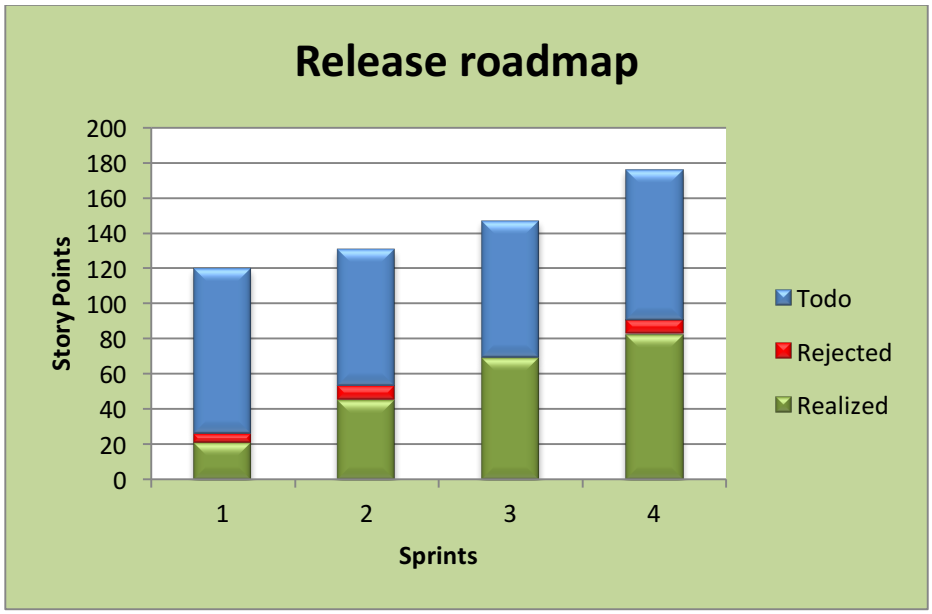



Figure 107: Release roadmap
Source : Donnée personnelle

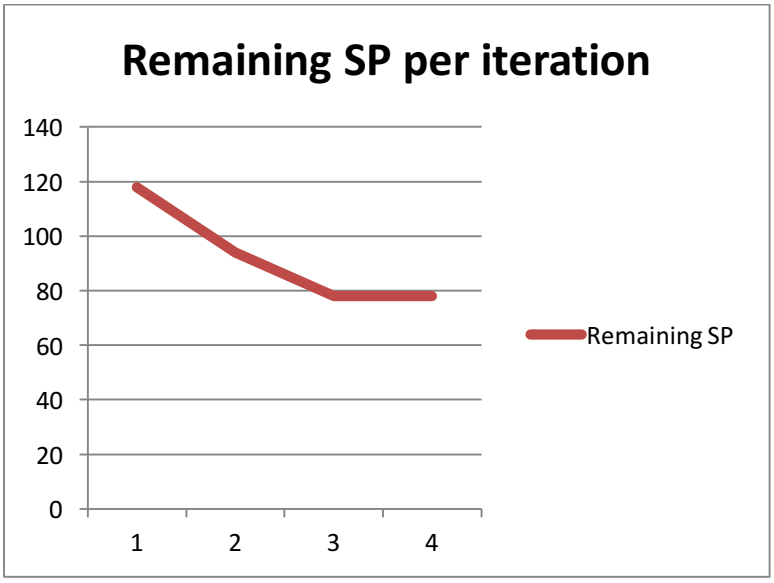



Figure 108: Graphique - SP restant par itération
Source : Donnée personnelle

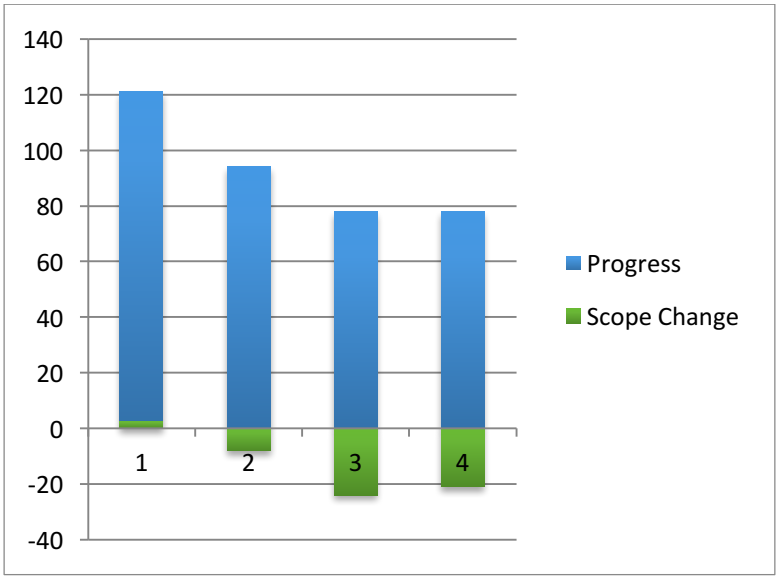



Figure 109: Graphique – progrès et changement du *scope*
Source : Donnée personnelle

| | Sprints | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| Nbr SP Sprint | 26 | 32 | 24 | 21 |
| Done | 21 | 24 | 24 | 13 |
| Todo | 94 | 78 | 78 | 86 |
| | | | | |
| Changed est. | -3 | 0 | 0 | 0 |
| New Stories | 0 | 8 | 24 | 21 |
| Realized | 21 | 45 | 69 | 82 |
| Average | 20,5 | 20,5 | 20,5 | 20,5 |
| Remaining SP | 118 | 94 | 78 | 78 |
| | | | | |
| Rejected | 5 | 8 | 0 | 8 |
| Scope Change | 3 | -8 | -24 | -21 |

Figure 110: Récapitulatif des sprints
Source : Donnée personnelle

## 6.2. Application lifecycle anagement

Il existe de nombreux outils pour suivre un développement Agile. Au début de notre projet, nous avions choisi d'utiliser Tuleap Campus. Mis à disposition par Tuleap, c'est une *application lifecycle management* (ALM). Ces outils fournissent diverses fonctionnalités qui permettent de les regrouper dans un seul endroit : intégration continue, VCS, *user storie,* gestion des bugs, etc.

Open source, Tuleap permet une gestion gratuite et Agile de projets. Nous avons déjà eu l'occasion de travailler avec la version Community Edition, une version autohébergée, dans le cadre d'un module durant notre cursus à l'HES.


Figure 111: Logo Tuleap
Source : Tiré de : https://tuleap-campus.org/my/index.php

Les deux versions diffèrent légèrement l'une de l'autre. N'ayant pas l'habitude de la version Tuleap Campus, nous avons décidé d'opter pour un template Excel que nous avons également utilisé dans plusieurs projets pour la partie gestion des *user stories.*

Notre fichier Excel comprend nos différents US, la notation de leur *story point* (SP), les différentes tâches à effectuer, b*urn down chart sprint*, ainsi que la *release roadmap.* Les *mocukups* réalisés sont stockés dans un document sur Dropbox.

La gestion des versions de notre code source a été faite par l'utilisation d'un dépôt GitHub. Open source et gratuit, GitHub permet simplement et efficacement la mise en place d'un dépôt pour notre code source.

Figure 112: Logo GitHub
Source : https://avatars1.githubusercontent.com/u/9919?s=200&v=4

N'ayant pas eu le temps de réaliser un script afin de permettre l'intégration continue de notre application, nous avons décidé d'opter pour la version payante d'Unity intégrée dans leur logiciel. Unity propose un service Unity Teams, inclus gratuitement dans la version Pro, mais la version « Avancé » qui nous intéresse est payante, 9 dollars par mois, pour la version Personal. Ce service offre un espace de stockage de 25Go, suffisant pour un jeu de notre taille, met à disposition un VCS et permet la compilation dans le nuage. Ainsi, le code source, une fois déployé dans le dépôt d'Unity, est compilé automatiquement. Ceci nous permet de contrôler que la compilation a été effectuée correctement et que nous n'avons pas intégré de régressions dans notre code. Notre application WebGL est donc déployée sur leur serveur. Si la compilation se déroule bien, nous avons également la possibilité de le tester directement dans notre navigateur web, de le télécharger et de le partager avec d'autres collaborateurs.

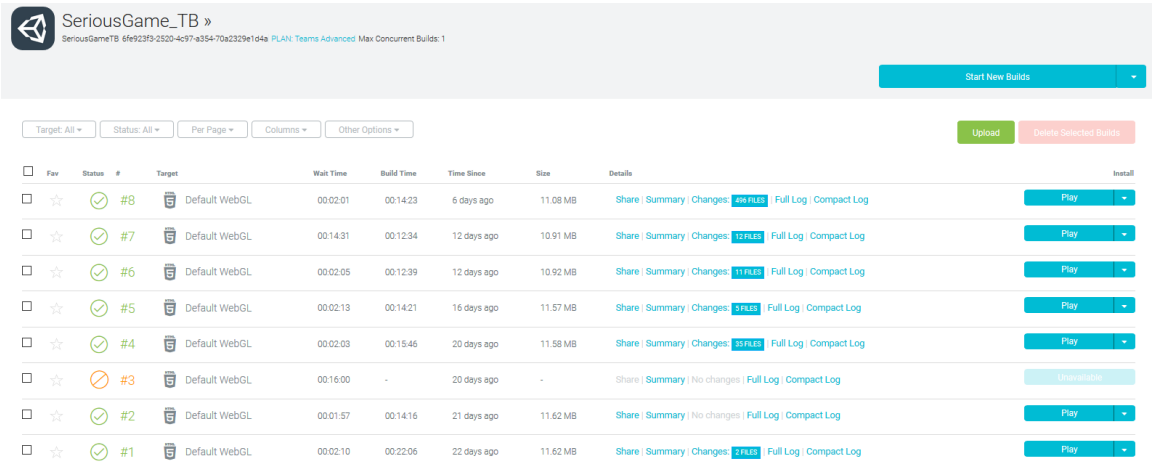



Figure 113: Unity cloud build
Source : Capture d'écran réalisée par l'auteur sur le dashboard d'Unity

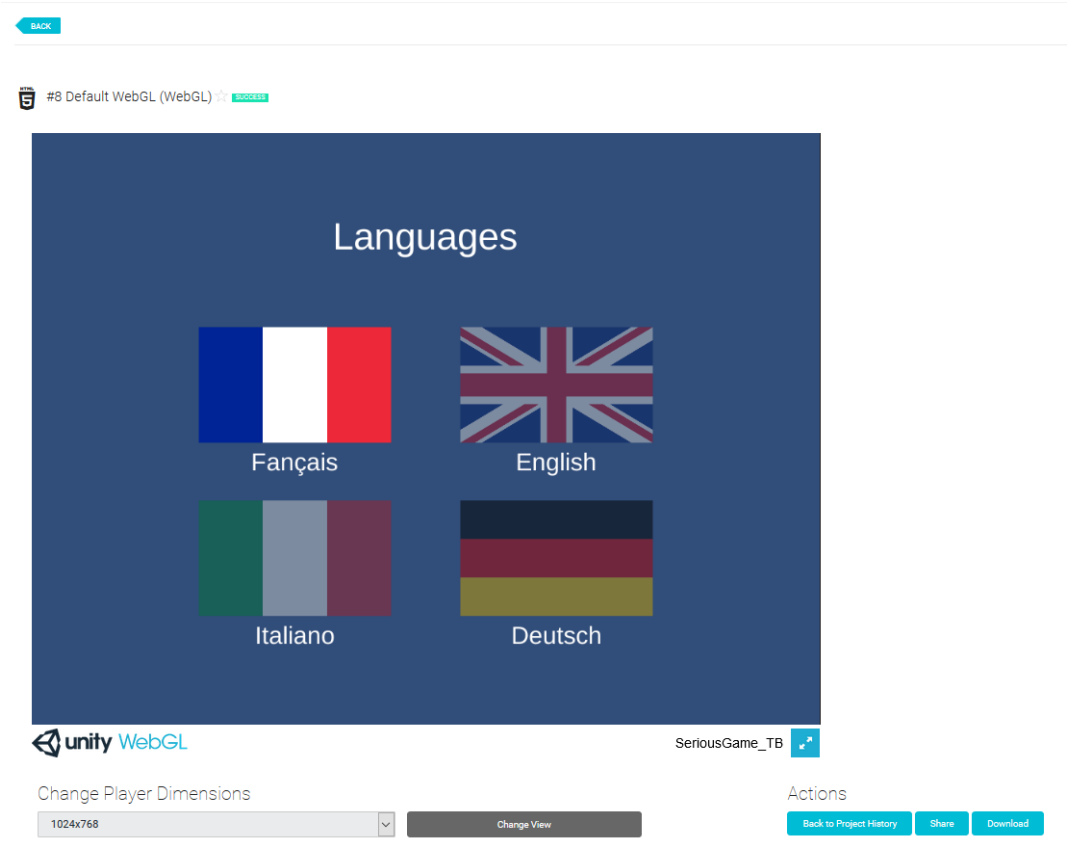



Figure 114: Unity test online
Source : Donnée personnelle - capture d'écran sur le cloud build du dashboard Unity

## 6.3. Documents

Dans le cadre de ce projet, nous avons utilisé des documents de types divers, mais tous sont basés sur des ressources Internet. Pour la partie recherche et analyse, nous nous sommes essentiellement documentés avec des sources Internet : site technique, blog, et site des différents logiciels.

Pour la partie développement, bien que connaissant le langage C#, nous n'avions aucune pratique dans l'utilisation d'Unity, de la conception des scripts ou encore de l'utilisation des *gameObjects*. Afin de nous familiariser avec cet outil, nous avons analysé différents tutoriels : textes et vidéos.

Pour éviter de devoir repartir à zéro, nous utilisons des conceptions qui ont déjà porté leurs fruits. Ainsi, la traduction de jeu est basée sur un *live training* d'Unity : https://unity3d.com/fr/learn/tutorials/topics/scripting/overview-and-goals?playlist=17117. Un expert effectue une démonstration directe avec capture de son écran. Les participants peuvent intervenir en temps réel, le tout est disponible sur leur site internet.

Nous avons également consulté bon nombre de tutoriels vidéo disponibles sur YouTube. Nous avons utilisé par exemple celui d'Alexander Zotov (https://www.youtube.com/watch?v=ouGJJzNPsSk) afin de créer un plateau de lancer de dé qui n'est pas directement disponible sur le prototype final. Nous avons également utilisé un

tutoriel disponible sur la chaine YouTube de Creagines : https://www.youtube.com/watch?v=EByl95OMG2o.

La gestion des sources n'a pas été effectuée avec le *source manager* de Word, mais le logiciel Zottero. Cet outil permet une intégration rapide de nos nombreuses références internet selon la norme de l'*American Psychological Association 6th edition* (APA).

# 7. Conclusion

## 7.1. Conclusion du projet

La conception d'un *serious game* est un travail long et complexe. Il est important de définir précisément le type de jeu sérieux que à développer en fonction du mandant. Chaque genre a un but particulier, dans notre cas le jeu est de type engagé, et il est important de le déterminer avec le client au préalable. Les exigences du mandant, le CREALP, nous ont mené à développer un jeu à but pédagogique et de sensibilisation de la population et des politiques valaisans.

Aujourd'hui, la réalisation d'un jeu sérieux est une réelle plus-value dans le domaine de la sensibilisation. Elle va mener à bien ces objectifs de manière ludique et pédagogique touchant ainsi toutes les classes d'âge.

Il existe une multitude de *serious game* dans le domaine durable et de l'eau. L'état de l'art nous a permis de ressortir les principaux atouts des jeux analysés et de découvrir leurs outils de développement. Il est également important de délimiter la plateforme sur laquelle notre jeu est amené à être joué. Afin de favoriser la meilleure expérience utilisateur, nous avons choisi la plateforme Web qui permet un accès facile et rapide à un plus grand nombre de joueurs. À cause de soucis techniques, nous avons dû compiler une version Windows pour le mode « Multijoueur », mais elle est amenée à disparaitre une fois l'intégration des WebSockets.

L'utilisation d'Unity a permis de combler nos lacunes en matière de modélisation et d'animation 3D par exemple. Il existe d'autres technologies qui permettent la conception de jeu web plus légères, mais aucune n'offre toutes les possibilités offertes par Unity. À notre sens, il s'agit du meilleur outil de développement dans le contexte de notre projet.

Enfin, nous avons développé un jeu vidéo jouable à la fin de nos quatre *sprints* de trois semaines. La méthodologie Agile est à notre sens le plus adéquat pour la conception d'un *serious game*. Le développement par itération nous a permis une plus grande liberté de travail en comparaison avec d'autres méthodes de travail. Chaque jour et chaque semaine, nous savions où nous en étions dans le développement et les tâches qui restaient à accomplir.

## 7.2. Conclusion personnelle

Ce travail a été un véritable défi. Le domaine des jeux vidéo m'a toujours intéressé et j'ai pu le découvrir à travers l'élaboration de ce projet. Le développement de ce *serious game* m'a permis de prendre en main un outil qui m'était encore méconnu : Unity. Hormis le langage C#, J'ai dû prendre les recherches de zéro : comprendre le fonctionnement des scènes, des *gameObjects*, des *PlayerPrefs*, les méthodes « Awake() », « Start ()» et « Update() », etc. Cette étude nous a pris passablement de temps.

Bien que le prototype final soit jouable, il persiste encore quelques bugs qui amènent une petite déception. De plus, je suis également déçu de ne pas avoir eu le temps d'implémenter d'autres types de jeu que les quiz tels que je l'avais imaginé lors de la rédaction du scénario.

Enfin, j'ai aimé travailler autant d'heures dans un projet unique. Ce projet m'a énormément motivé et intéressé. J'y ai consacré plus de 400 heures, mais je n'ai pas vu le temps passer.

# 8. Bibliographie

*3D.CITY*. (s. d.). [Web]. Consulté à l'adresse http://lo-th.github.io/3d.city/

Alexander Zotov. (2018). *Unity Tutorial How To Make Roll The Dice Feature For 2D Board Game With Simple C# Script*. Consulté à l'adresse https://www.youtube.com/watch?v=JgbJZdXDNtg

Animateur - Dossier : Les métiers du jeu vidéo. (2012, décembre 12). Consulté 4 juillet 2018, à l'adresse http://www.jeuxvideo.com/dossiers/00017318/les-metiers-du-jeu-video-animateur-003.htm

Answer Button. (s. d.). Consulté 25 juillet 2018, à l'adresse https://unity3d.com/fr/learn/tutorials/topics/scripting/answer-button

Anthony Lejeune, directeur artistique dans les jeux vidéo. (s. d.). Consulté 4 juillet 2018, à l'adresse http://www.imaginetonfutur.com/anthony-lejeune-directeur-artistique-dans-les-jeux-video.html

Beaucage, N. (26.02). WebGL : Le talent caché du Web. Consulté 5 juillet 2018, à l'adresse https://symetris.ca/fr/blogue/webgl-le-talent-cache-du-web

Brain Age. (s. d.). Consulté 29 juin 2018, à l'adresse http://www.jeuxvideo.com/jeux/nintendo-ds/00029654-brain-age.htm

Cardinale, A. (2018). *Créez des jeux de A à Z avec Unity - IV. Réseau et mode multijoueur (2e édition)*. Éditions D-BookeR.

CiNACity. (s. d.). Consulté 29 juin 2018, à l'adresse https://www.curapy.com/jeux/cinacity/

Cocoon. (s. d.). Consulté 9 juillet 2018, à l'adresse https://developer.playcanvas.com/en/user-manual/publishing/mobile/cocoon/

Coëffé, T. (2017, août 25). État des lieux de l'usage mobile (2017) : temps passé, notifications, propension à payer, applications populaires... Consulté 2 juillet 2018, à l'adresse https://www.blogdumoderateur.com/comscore-mobile-app-report-2017/

Cohard, P. (2015). L'apprentissage dans les serious games : proposition d'une typologie. *@GRH*, (16), 11‑40. https://doi.org/10.3917/grh.153.0011

Collectivités. (s. d.). Consulté 16 juillet 2018, à l'adresse https://compostchallenge.com/collectivites/

Creating a Click to Move Game. (s. d.). Consulté 26 juillet 2018, à l'adresse https://unity3d.com/fr/learn/tutorials/topics/navigation/creating-click-move-game

de Roquefeuil, B. (2017, février 17). Quelle définition donner au serious game (ou jeu sérieux) ? Consulté 13 juillet 2018, à l'adresse https://www.alain-bensoussan.com/avocats/definition-serious-game/2017/02/17/

Degeorges, M. (2016, juin 12). Plus personne ne télécharge d'applications. Consulté 2 juillet 2018, à l'adresse https://www.lesechos.fr/12/06/2016/lesechos.fr/0211017335329_plus-personne-ne-telecharge-d-applications.htm

Demay, G. (s. d.). Blueprint dans les grandes lignes. Consulté 9 juillet 2018, à l'adresse https://openclassrooms.com/fr/courses/2675311-developpez-votre-premier-jeu-video-avec-lunreal-engine-4/3188250-blueprint-dans-les-grandes-lignes

Djaouti, D. (2012, juillet). Définir le « serious gaming ». Consulté 27 juin 2018, à l'adresse http://www.educ-revues.fr/ARGOS/AffichageDocument.aspx?iddoc=41665

Dumouchel, A. (2016, octobre 25). Ice Flows : Sauvons l'Antarctique. Consulté 29 juin 2018, à l'adresse http://www.serious-game.fr/ice-flows-sauvons-lantarctique/

En quoi consiste le métier de producer de jeux vidéo ? (s. d.). Consulté 4 juillet 2018, à l'adresse https://www.letudiant.fr/metiers/les-metiers-qui-recrutent/en-quoi-consiste-le-metier-de-producer-de-jeux-video.html

Février, M. (2015, février 28). Découvrez 8 raisons d'utiliser les serious games. Consulté 1 août 2018, à l'adresse http://weeshiz.com/decouvrez-8-raisons-dutiliser-les-serious-games/

Finding Home – Applications sur Google Play. (s. d.). Consulté 29 juin 2018, à l'adresse https://play.google.com/store/apps/details?id=org.unhcr.findinghome&hl=fr

Gabbud, J.-Y. (s. d.). HES-SO Valais: la classe sans prof et sans note est ouverte. Consulté 1 août 2018, à l'adresse https://www.lenouvelliste.ch/articles/valais/canton/la-classe-sans-prof-et-sans-note-est-ouverte-701582

Game designer. (2017, octobre 31). Consulté 4 juillet 2018, à l'adresse https://www.orientation.ch/dyn/show/1900?lang=fr&id=1589

goss. (2018, avril 18). Le code source d'Unity en accès libre ! Consulté 9 juillet 2018, à l'adresse https://makeyourgame.fun/le-code-source-dunity-en-acces-libre/

Graham, D. (2014, mars 10). How can i use a PNG image file has a button? Consulté 25 juillet 2018, à l'adresse http://answers.unity.com/questions/802106/how-can-i-use-a-png-image-file-has-a-button.html

Graphiste - Dossier : Les métiers du jeu vidéo. (s. d.). Consulté 4 juillet 2018, à l'adresse http://www.jeuxvideo.com/dossiers/00017318/les-metiers-du-jeu-video-graphiste-002.htm

Guillaume. (2014, mars 4). Renault forme ses 15 000 commerciaux à l'aide de Serious Games. Consulté 29 juin 2018, à l'adresse http://myseriousgame.com/renault-forme-15000-commerciaux-serious-games/

Herrenschneider, D. (2016, septembre 11). Comparaison entre Unity 5 et Unreal Engine 4 | SUPINFO, École Supérieure d'Informatique. Consulté 10 juillet 2018, à l'adresse https://www.supinfo.com/articles/single/2139-comparaison-unity-5-unreal-engine-4

Holistic3d. (2016). *Setting up a Network Lobby Part 3* (Vol. 3). Consulté à l'adresse https://www.youtube.com/watch?v=-t9kzrLkF10

Holistic3d. (s. d.). *Setting up a Network Lobby with Unity 5 Part 1* (Vol. 1). Consulté à l'adresse https://www.youtube.com/watch?v=jklWlm5v21k

Horn, B. (2014, juillet 30). Setting CORS (cross-origin resource sharing) on Apache with correct response headers allowing everything through. Consulté 12 juillet 2018, à l'adresse https://benjaminhorn.io/code/setting-cors-cross-origin-resource-sharing-on-apache-with-correct-response-headers-allowing-everything-through/

HTML5. (2018). In *Wikipédia*. Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=HTML5&oldid=149859953

Intro and Setup. (s. d.). Consulté 25 juillet 2018, à l'adresse https://unity3d.com/fr/learn/tutorials/topics/scripting/intro-and-setup

Isatis. (2016, avril 24). Advergames : Quand les marques font leur pub avec des jeux vidéo... Consulté 29 juin 2018, à l'adresse https://level-1.fr/2016/04/advergames-quand-les-marques-font-leur-pub-avec-des-jeux-video/

JavaScript. (2018). In *Wikipédia*. Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=JavaScript&oldid=149923914

JavaScript Object Notation. (2018). In *Wikipédia*. Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=JavaScript_Object_Notation&oldid=149305511

Jeux Engagés. (s. d.). Consulté 29 juin 2018, à l'adresse http://www.serious-game.fr/category/serious-games/political-games/

Jeux sérieux, mondes virtuels - Typologie. (2018, avril 11). [EduSection]. Consulté 11 avril 2018, à l'adresse http://eduscol.education.fr/numerique/dossier/apprendre/jeuxserieux/notion/typologie

Korzuszek, P. (2016, juin 7). 7 Ways to Keep Your Unity Project Organized. Consulté 17 juillet 2018, à l'adresse http://blog.theknightsofunity.com/7-ways-keep-unity-project-organized/

kristiel. (2016, janvier 21). How to Create a Raycast in Unity 3D. Consulté 26 juillet 2018, à l'adresse https://www.studica.com/blog/how-to-create-a-raycast-in-unity-3d

La RTS lance Datak, le jeu qui interroge notre gestion des données. (2016, décembre 13). [RTS Info]. Consulté 13 juillet 2018, à l'adresse https://www.rts.ch/info/suisse/8235789-la-rts-lance-datak-le-jeu-qui-interroge-notre-gestion-des-donnees.html

Le composant Rigidbody avec unity dans physique. (2017, juillet 3). Consulté 25 juillet 2018, à l'adresse https://docs.onouris.com/unity/physique/rigidbody

Le Directeur Artistique - Dossier : Les métiers du jeu vidéo. (2012, décembre 12). Consulté 4 juillet 2018, à l'adresse http://www.jeuxvideo.com/dossiers/00017318/les-metiers-du-jeu-video-le-directeur-artistique-001.htm

Leblal, S. (2017, juillet 26). Clap de fin pour le Flash Player d'Adobe. Consulté 16 juillet 2018, à l'adresse https://www.lemondeinformatique.fr/actualites/lire-clap-de-fin-pour-le-flash-player-d-adobe-68907.html

Lellouche, N. (2017, juillet 26). Adobe annonce la fin de son logiciel emblématique Flash Player. Consulté 16 juillet 2018, à l'adresse http://www.lefigaro.fr/secteur/high-tech/2017/07/26/32001-20170726ARTFIG00211-adobe-annonce-la-fin-de-son-logiciel-emblematique-flash-player.php

Les jeux vidéo au service de la formation. (2017, avril 5). Consulté 29 juin 2018, à l'adresse https://jobtic.ch/fr/magazine/article/1588

Les métiers du jeu vidéo. (s. d.). Consulté 4 juillet 2018, à l'adresse http://www.jeuxvideo.com/dossiers/00017318/les-metiers-du-jeu-video.htm

Lozange Lab. (2015, décembre 8). Consulté 16 juillet 2018, à l'adresse http://www.lozange-lab.com/?og=1

Mallet, V. (s. d.). Familiarisez-vous avec le WebGL et BabylonJS. Consulté 6 juillet 2018, à l'adresse https://openclassrooms.com/courses/3979376-creez-votre-propre-fps-en-webgl/3979383-familiarisez-vous-avec-le-webgl-et-babylonjs

Méli, B. (2018, octobre 24). Un serious game, combien ça coûte ? Consulté 29 juin 2018, à l'adresse https://www.journaldunet.com/ebusiness/crm-marketing/dossier/comprendre-et-utiliser-les-serious-games/un-serious-game-combien-ca-coute.shtml

Mérancourt, W. (2017, mai 16). «Finding Home» ou comment jouer avec le drame des réfugiés. *Le Temps*. Consulté à l'adresse https://www.letemps.ch/monde/finding-home-jouer-drame-refugies

Mobile databases: SQLite and SQLite alternatives for Android and iOS. (2016, juillet 13). Consulté 11 juillet 2018, à l'adresse http://greenrobot.org/news/mobile-databases-sqlite-alternatives-and-nosql-for-android-and-ios/

N3K EN. (2016). *Character Selection (And changing scene) - Unity 3D[Tutorial][C#]*. Consulté à l'adresse https://www.youtube.com/watch?v=IFTjcPvCZaM

PlayCanvas versus Unity WebGLPlayCanvas. (2016, août 4). Consulté 10 juillet 2018, à l'adresse https://blog.playcanvas.com/playcanvas-versus-unity-webgl/

PlayCanvas versus Unreal WebGLPlayCanvas. (2016, août 15). Consulté 10 juillet 2018, à l'adresse https://blog.playcanvas.com/playcanvas-versus-unreal-webgl/

Présentation - CREALP - Centre de recherche sur l'environnement alpin. (s. d.). Consulté 30 mai 2018, à l'adresse https://www.crealp.ch/fr/accueil/le-crealp/presentation.html

PushyPixels. (2015). *Breakfast With Unity: 3D Dice Roller Part 1* (Vol. 1–2). Consulté à l'adresse https://www.youtube.com/watch?v=LVKqTzK9-AI

Ratignier, J. (2017, mai 24). JavaScript, histoire d'un langage longtemps sous-estimé. Consulté 2 juillet 2018, à l'adresse https://medium.com/%C3%A9cosyst%C3%A8me-

des-langages-de-programmation/javascript-histoire-dun-langage-longtemps-sous-estim%C3%A9-c1f2da31759

Serious game « Les Maîtres de l'eau ». (s. d.). Consulté 16 juillet 2018, à l'adresse http://www.eaudeparis.fr/lespace-culture/mediatheque/serious-game-les-maitres-de-leau/

Theler, J. (2015, mars 19). Le QUIZ : et si vous le mettiez au centre de votre design de formation ? Consulté 25 juillet 2018, à l'adresse http://www.e-teach.ch/blog/utiliser-le-quiz-comme-activite-principale-de-la-formation/

Traçage (hydrogéologie). (2017). In *Wikipédia*. Consulté à l'adresse https://fr.wikipedia.org/w/index.php?title=Tra%C3%A7age_(hydrog%C3%A9ologie)&oldid=137883860

Tshabangu, V. (2016, février 18). Optimizing 3D Models for games. Consulté 18 juillet 2018, à l'adresse https://vtsdemozone.wordpress.com/2016/02/18/optimizing-3d-models-for-games/

TUTO UNITY FR : Sauvegarder des données avec les PlayerPrefs (Unity3D). (2018, juin 6). Consulté 12 juillet 2018, à l'adresse https://www.tutounity.fr/article.php?id=78

Unity - Manual: Asset Workflow. (s. d.-a). Consulté 18 juillet 2018, à l'adresse https://docs.unity3d.com/Manual/AssetWorkflow.html

Unity - Manual: Building and running a WebGL project. (s. d.-b). Consulté 14 juillet 2018, à l'adresse https://docs.unity3d.com/Manual/webgl-building.html

Unity - Manual: Materials, Shaders & Textures. (s. d.-c). Consulté 18 juillet 2018, à l'adresse https://docs.unity3d.com/Manual/Shaders.html

Unity - Scripting API: MonoBehaviour.Awake(). (s. d.-a). Consulté 26 juillet 2018, à l'adresse https://docs.unity3d.com/ScriptReference/MonoBehaviour.Awake.html

Unity - Scripting API: NavMeshAgent. (s. d.-b). Consulté 26 juillet 2018, à l'adresse https://docs.unity3d.com/ScriptReference/AI.NavMeshAgent.html

Unity - Scripting API: PlayerPrefs. (s. d.-c). Consulté 12 juillet 2018, à l'adresse https://docs.unity3d.com/ScriptReference/PlayerPrefs.html

Unity Plus. (s. d.). Consulté 9 juillet 2018, à l'adresse https://store.unity.com/fr/products/unity-plus

Unreal Engine Frequently Asked Questions. (s. d.). Consulté 9 juillet 2018, à l'adresse https://www.unrealengine.com/en-US/faq

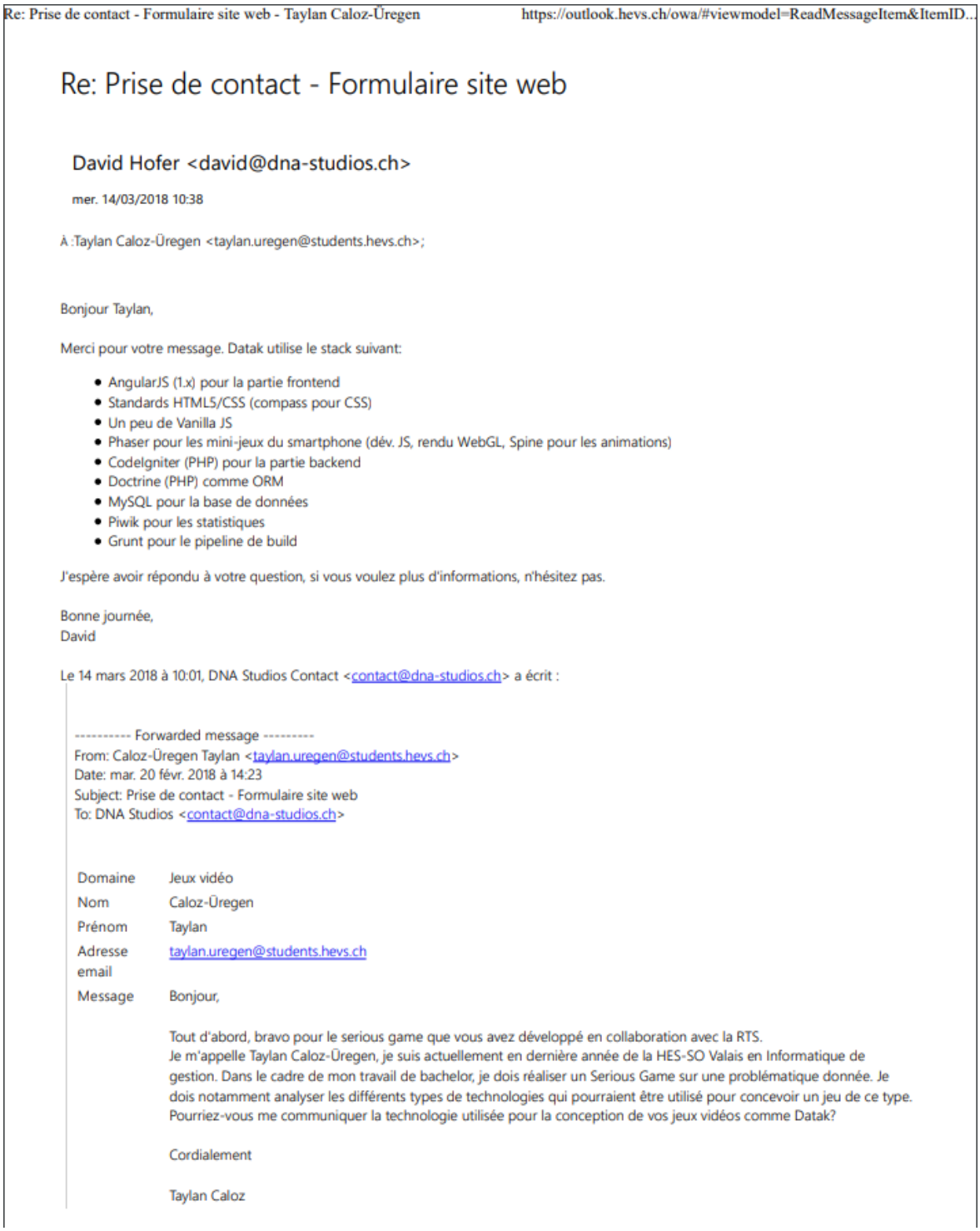
WebGL. (2018, avril 25). Consulté 5 juillet 2018, à l'adresse https://developer.mozilla.org/fr/docs/Apprendre/WebGL

WebGL Html5 Canvas Overview. (s. d.). Consulté 5 juillet 2018, à l'adresse https://www.tutorialspoint.com/webgl/html5_canvas_overview.htm

Weinberg, E. (2016, juillet 12). Finding the best WebGL tool. Consulté 5 juillet 2018, à l'adresse https://medium.com/alistapart/all-the-webgl-tools-and-frameworks-i-have-looked-at-a1c22154591b

# Annexe

## Annexe I Contact DNA Studio



Re: Prise de contact - Formulaire site web - Taylan Caloz-Üregen https://outlook.hevs.ch/owa/#viewmodel=ReadMessageItem&ItemID...

Re: Prise de contact - Formulaire site web

David Hofer <david@dna-studios.ch>

mer. 14/03/2018 10:38

À :Taylan Caloz-Üregen <taylan.uregen@students.hevs.ch>;

Bonjour Taylan,

Merci pour votre message. Datak utilise le stack suivant:

- AngularJS (1.x) pour la partie frontend
- Standards HTML5/CSS (compass pour CSS)
- Un peu de Vanilla JS
- Phaser pour les mini-jeux du smartphone (dév. JS, rendu WebGL, Spine pour les animations)
- CodeIgniter (PHP) pour la partie backend
- Doctrine (PHP) comme ORM
- MySQL pour la base de données
- Piwik pour les statistiques
- Grunt pour le pipeline de build

J'espère avoir répondu à votre question, si vous voulez plus d'informations, n'hésitez pas.

Bonne journée,
David

Le 14 mars 2018 à 10:01, DNA Studios Contact <contact@dna-studios.ch> a écrit :

---------- Forwarded message ---------
From: Caloz-Üregen Taylan <taylan.uregen@students.hevs.ch>
Date: mar. 20 févr. 2018 à 14:23
Subject: Prise de contact - Formulaire site web
To: DNA Studios <contact@dna-studios.ch>

| | |
|---|---|
| Domaine | Jeux vidéo |
| Nom | Caloz-Üregen |
| Prénom | Taylan |
| Adresse email | taylan.uregen@students.hevs.ch |
| Message | Bonjour,<br><br>Tout d'abord, bravo pour le serious game que vous avez développé en collaboration avec la RTS.<br>Je m'appelle Taylan Caloz-Üregen, je suis actuellement en dernière année de la HES-SO Valais en Informatique de gestion. Dans le cadre de mon travail de bachelor, je dois réaliser un Serious Game sur une problématique donnée. Je dois notamment analyser les différents types de technologies qui pourraient être utilisé pour concevoir un jeu de ce type.<br>Pourriez-vous me communiquer la technologie utilisée pour la conception de vos jeux vidéos comme Datak?<br><br>Cordialement<br><br>Taylan Caloz |

Figure 115: Courriel contact DNA Studio- serious game Datak
Source: Donnée personnelle

## Annexe II Contact supercyan

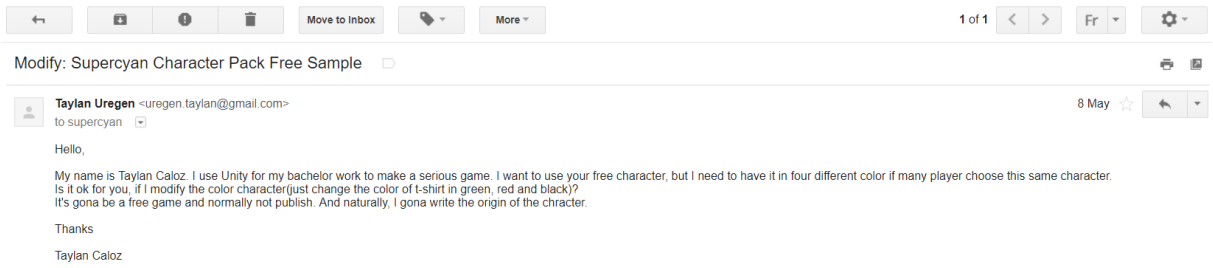



Figure 116: Courriel contact supercyan
Source : Donnée personnelle

## Annexe III Quizz fourni par le CREALP

### Questionnaire Serious Game – premier jet

1. À l'horizon 2060, dans quelle direction coulera principalement l'eau de fonte du glacier de la Plaine Morte ?
    a. Au Nord (Berne)
    b. Au Sud (Valais)
2. Le glacier de la Plaine Morte représente une réserve d'environ 0.8 $km^3$ d'eau sous forme de glace. Quelle en est sa superficie ?
    a. 8.4 $km^2$
    b. 16.8 $km^2$
3. D'après les études récentes, quand le glacier de la Plaine Morte aura-t-il disparu ?
    a. Dans la période 2080 et 2100
    b. Dans la période 2180 et 2200
4. Dans le futur, les lacs au bord du glacier de la Plaine Morte auront tendance à devenir plus nombreux et plus volumineux. Nécessitent-ils une surveillance ?
    a. Oui, à cause des risques de rupture
    b. Non, car ils sont éloignés des activités humaines
5. De nos jours et à l'avenir, que peut entrainer une année sèche dans la région de Crans-Montana à Sierre ?
    a. Un pompage de l'eau du Rhône pour l'irrigation des vignes, des jardins et des cultures
    b. Des risques de pénuries d'eau en fin d'été et en hiver
6. Les changements socio-économiques et climatiques ont un impact sur la gestion durable et intégrée de l'eau à l'horizon 2080 dans la région de Crans-Montana à Sierre. Lequel de ces changements est jugé le plus important d'après les experts ?
    a. Les changements socio-économiques
    b. Les changements climatiques
7. À quel niveau doivent être gérés les problèmes liés à l'eau du Haut-Plateau ?
    a. Au niveau communal et local
    b. Au niveau intercommunal et régional

8. La consommation en eau potable de Crans-Montana région a varié entre 7.7 millions de $m^3$ en 2010 et 8.2 millions de $m^3$ en 2011. Quelle est la quantité approximative d'eau potable utilisée pour l'arrosage des pelouses et des jardins ?
    a. 1 million de $m^3$ par année
    b. 3 millions de $m^3$ par année
9. Comment évolueront les quantités d'eau disponibles pour la région de Crans-Montana à Sierre, à l'horizon 2050 ?
    a. Les quantités diminueront
    b. Les quantités stagneront
    c. Les quantités augmenteront
10. Dans le futur, la fréquence des années sèches devrait-elle augmenter ou diminuer en Valais ?
    a. Augmenter
    b. Diminuer
11. Quel est le nom du barrage qui se situe entre la commune d'Ayent et d'Icogne ?
    a. Le barrage de Zeuzier
    b. Le barrage de Vatseret
12. Mis à part l'hydroélectricité, quel peut être le rôle des barrages pour une gestion durable l'eau ?
    a. Stocker l'eau et la répartir en cas de pénurie
    b. Accélérer l'évaporation et la chute de précipitations
13. Quand est-ce que le volume d'eau disponible pour la région de Crans-Montana est-il le plus important ?
    a. En été
    b. Au printemps
14. Au début d'été, d'où provient principalement l'eau des sources du Haut-Plateau ?
    a. De la fonte du glacier
    b. De la fonte de la neige
15. Les usages représentaient entre 10.5 et 13.6 millions de $m^3$ d'eau par année au début des années 2010 (sans l'hydroélectricité). Quelle est la quantité d'eau disponible par année sur le Haut-Plateau ?
    a. 40 millions de $m^3$
    b. 140 millions de $m^3$
16. Quel est le nom du projet soutenu par les communes de Crans-Montana, Icogne, Lens, Sierre, St-Léonard, Varone, Venthône, Miège, Veyras et Salquenen, ainsi que par Sierre-Energie SA, Energies Sion Région et Electricité de la Lienne SA, pour la gestion durable de l'eau dans la région ?
    a. Lienne-Raspille
    b. Collab'Eau'ration
17. Par le passé, qu'ont construit les hommes pour irriguer les cultures ?
    a. Des conduites forcées
    b. Des bisses
18. La plaine du Rhône est connue pour être une des régions les plus sèches de Suisse avec des précipitations moyennes annuelles inférieures à 700 mm à Sierre (période

1961-1990, MétéoSuisse). Est-ce que les précipitations moyennes annuelles varient avec l'altitude ?

a. Oui, elles diminuent
b. Non
c. Oui, elles augmentent

19. En Valais, qui est chargé d'assurer l'approvisionnement, l'évacuation et le traitement des eaux ?
    a. Le canton
    b. Les communes
20. Le programme national de recherche « Gestion durable de l'eau » (PNR 61) réunit 16 projets, dont l'un d'entre eux s'est intéressé à la région de Crans-Montana à Sierre. Comment ce projet s'appelle-t-il ?
    a. MontanAqua
    b. Eau Plateau
21. L'eau des sources subit-elle des traitements avant d'entrer dans les réseaux de distribution d'eau potable ?
    a. Oui
    b. Non
22. Les eaux souterraines, dont l'eau est exploitée - ou est exploitable - pour l'approvisionnement en eau potable des communes, sont-elles protégées ?
    a. Oui, pour restreindre l'utilisation du sol
    b. Non, car elles sont naturellement protégées de la surface
23. Quel type de gestion faut-il privilégier pour favoriser une gestion intégrée de l'eau ?
    a. Une gestion par limites administratives
    b. Une gestion par bassin versant
24. Comment est-il possible de savoir où s'écoule l'eau de fonte d'un glacier ?
    a. À l'aide d'essais de traçage
    b. À l'aide d'une cartographie détaillée

## Annexe IV Technical guide

Afin de développer notre jeu sérieux, nous avons dû utiliser différents logiciels. Dans le cas d'Unity, il est important d'utiliser la même version. Bien que rétrocompatible, à la suite d'une mise à jour automatique d'Unity, nous n'avons plus été en mesure de compiler le jeu vidéo.

### Unity

C'est le logiciel de développement principal. Il permet d'ouvrir le dossier qui contient le code source. Nous avons utilisé la version 2017.3.1f1, sous licence *Personal* (version gratuite), avec la version 3.5 de .NET (par défaut). Il est disponible à cette adresse :

https://download.unity3d.com/download_unity/a53ad04f7c7f/Windows64EditorInstaller/UnitySetup64-2018.1.3f1.exe.

Nous avons également utilisé la fonctionnalité payante de Unity Teams : https://unity3d.com/fr/get-teams. Elle a été nécessaire afin d'obtenir la fonctionnalité de *cloud building*.

Afin d'ouvrir notre projet avec Unity, il suffit d'ouvrir le dossier du code source dans l'application. Il n'y a pas de fichier d'exécution, il faut charger tout le répertoire.

### GitHub

Nous avons utilisé ce VCS dans un dépôt public. L'ensemble du projet est disponible à cette adresse : https://github.com/zorgade/SeriousGame_TB .

### Visual Studio

Nous avons installé la version *Community Edition 2017 v.15.7.3* comme éditeur de script. Unity installe cet outil, s'il n'est pas encore sur le PC.

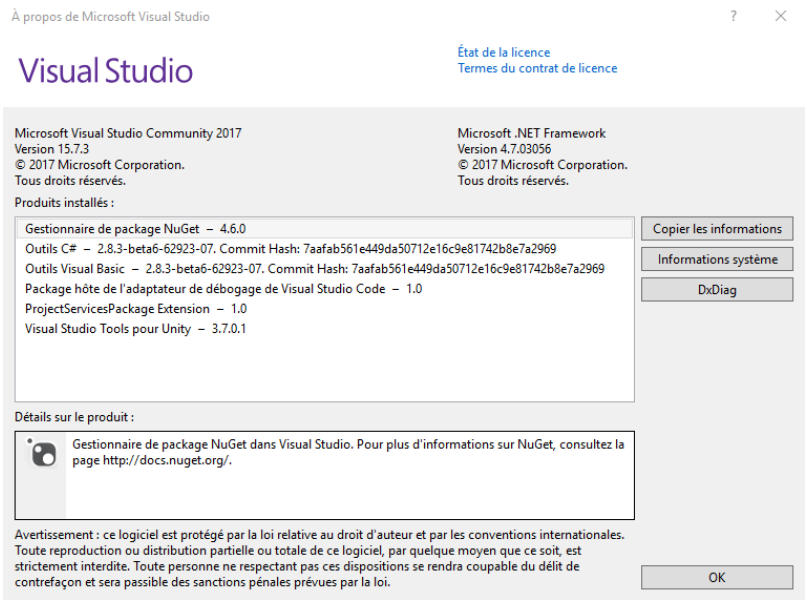



Figure 117: Visual Studio information version
Source : Donnée personnelle

### Adobe Photoshop

Les textures du personnage « Supercyan », le bonhomme humanoïde, ont été modifiées avec Adobe Photoshop CC 2018, v.19.0.


Figure 118: Logo Photoshop
Source : Tiré de https://fr.wikipedia.org/wiki/Adobe_Photoshop

# Annexe V User guide

## Plateforme du jeu

Il existe deux versions du jeu : Web et Windows. La version Web est à utiliser lorsque nous lançons une partie dans le mode « Solo », ainsi que lorsque nous voulons rejoindre une partie en ligne. La version Windows doit être utilisée pour héberger une partie en ligne. Elle peut également être utilisée pour jouer normalement, mais une fois les websocket en place, elle ne sera plus disponible.

## Premiers pas

Dans le premier écran, le joueur a la possibilité de choisir la langue. Actuellement seul le français est disponible :


Figure 119: User guide - scène du choix des langues
Source : Donnée personnelle

Après le chargement de la langue, le joueur arrive sur la scène d'introduction qui doit exposer le but et la problématique du jeu. Dans cet écran, il a la possibilité de stopper et reprendre le texte parlé simulé, dans notre prototype par une musique. Après trois secondes, un bouton permet de passer l'introduction. Sans clic de la part de l'utilisateur, la scène suivante se charge automatiquement après un délai de 120 sec, soit la durée du morceau.

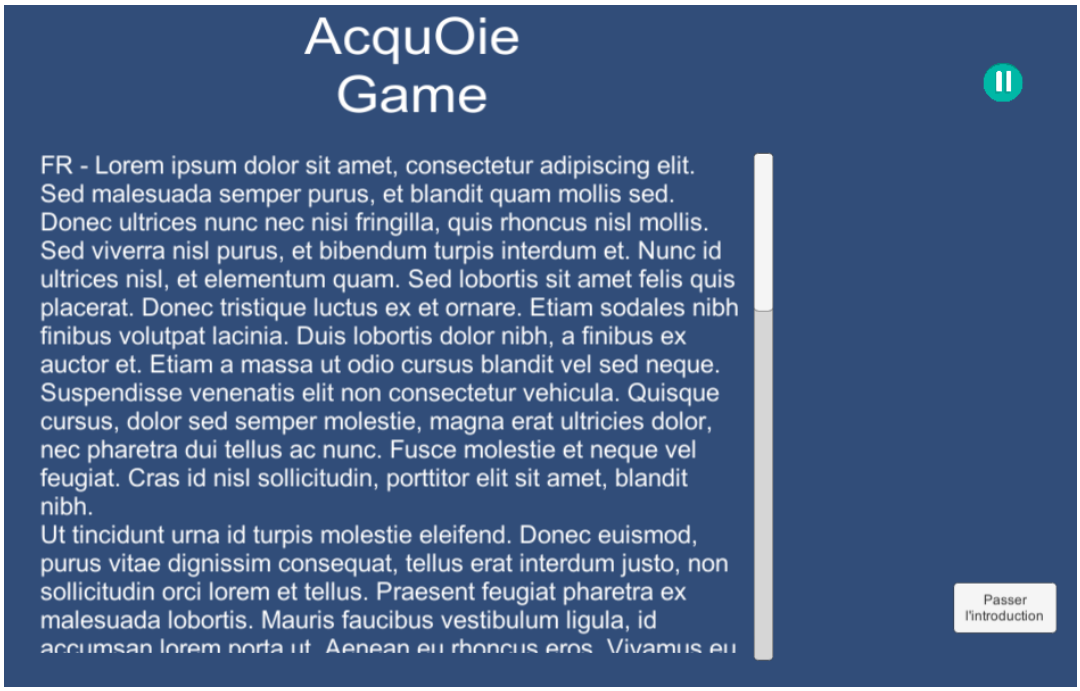



Figure 120: User guide - scène d'introduction
Source : Donnée personnelle

L'écran suivant présente le menu principal. Il permet de commencer une partie en cliquant sur « Débuter le jeu ». Les boutons « Règles » et « Paramètres » ne sont pas encore actifs.


Figure 121: User guide - scène du menu
Source : Donnée personnelle

Une fois cliqué sur « Debuter le jeu », le joueur peut sélectionner le mode de jeu : « Solo » ou « Multijoueur ».

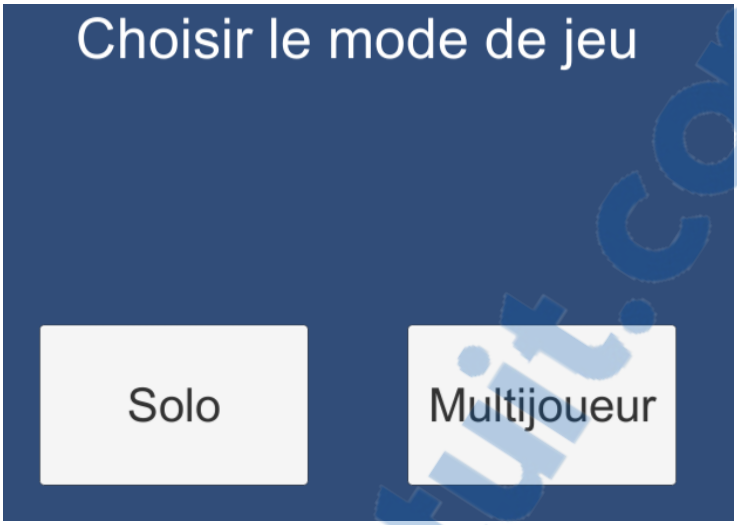



Figure 122: User guide - choix du mode de jeu
Source : Donnée personnelle

### Mode solo

Lors d'une partie en solo, l'utilisateur sélectionne un pseudo. En cliquant sur suivant, il arrive au choix des personnages.

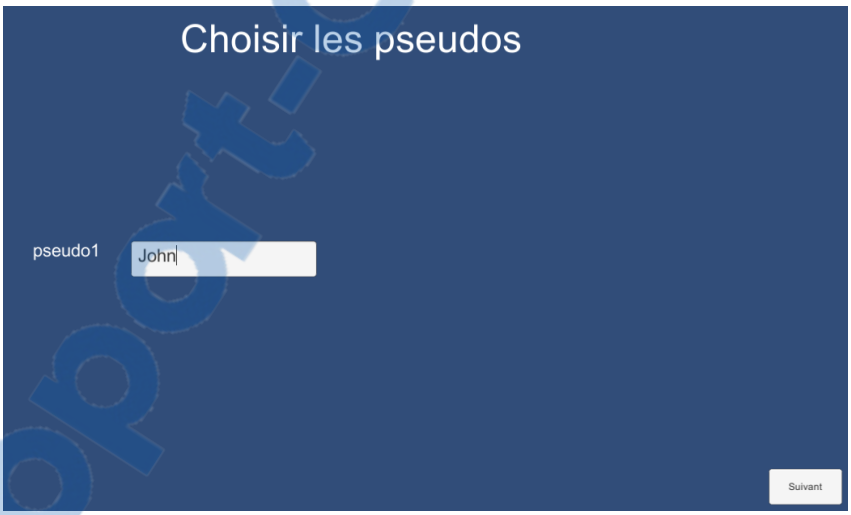



Figure 123: User guide - mode solo, choix du pseudo
Source : Donnée personnelle

À ce stade, le joueur sélectionne son avatar parmi les sept disponibles qui défilent en cliquant sur les flèches de gauche et droite. Il valide ensuite son choix via le bouton « Confirmer ».


Figure 124: User guide – mode solo, choix des personnages
Source : Donnée personnelle

La partie peut enfin commencer. Le joueur atterrit sur le plateau de jeu (voir Figure 125). Sa case s'affiche en noir [1]. Le joueur clique ensuite sur le dé [2] pour le lancer. La case correspondant au chiffre affiché devient magenta [3]. Le résultat est incrémenté [4]. Le score affiché [4] représente l'ensemble des jets du dé et non le score actuel du dé. Afin de lancer le quiz, le joueur doit cliquer sur la case magenta.

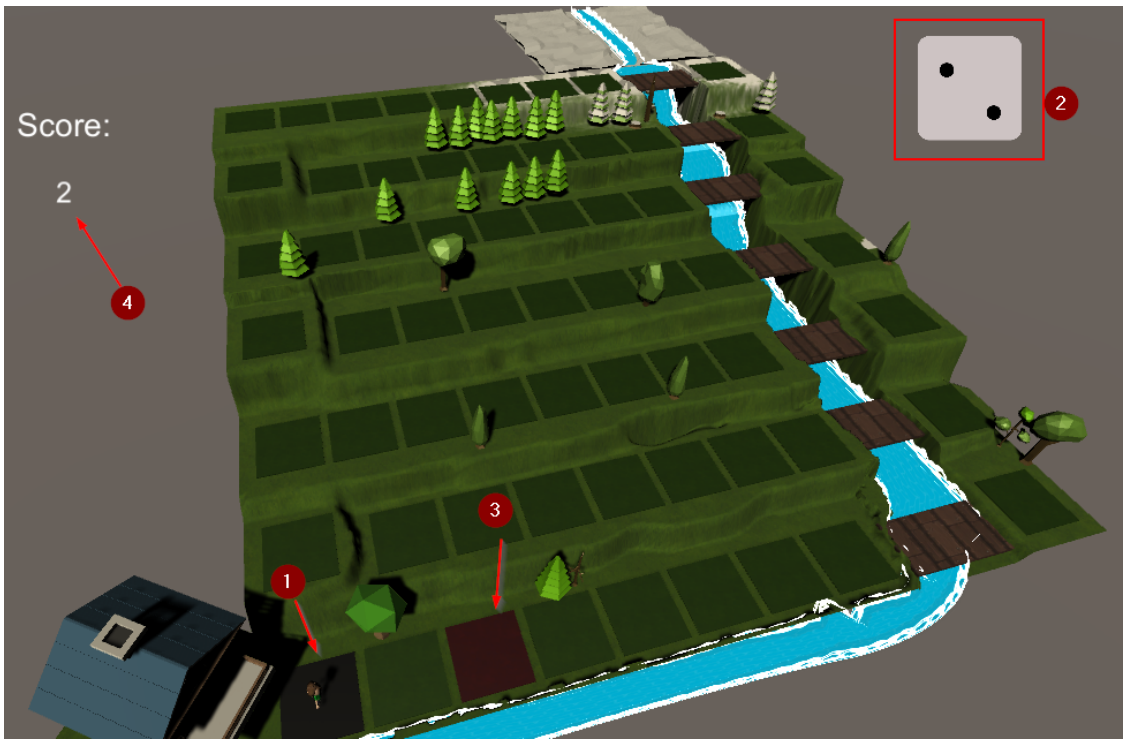



Figure 125: User guide - mode solo, plateau de jeu
Source : Donnée personnelle

Lors du lancement de la scène du quiz, une question à choix multiple s'affiche de manière aléatoire. Elle peut contenir de deux à quatre réponses, une seule étant la bonne.

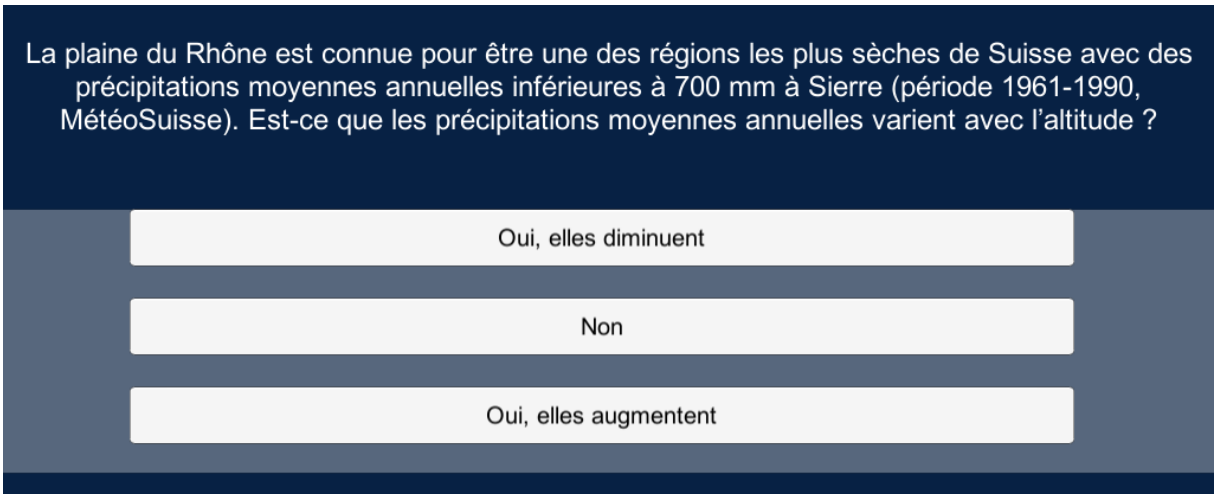



Figure 126: User guide – quizz
Source : Donnée personnelle

En cas de mauvaise réponse, un écran rouge s'affiche. Si le joueur a répondu correctement : écran vert. Il doit ensuite cliquer sur « Retour au jeu » pour revenir sur le plateau de jeu.


Figure 127: User guide - fausse réponse au quiz
Source : Donnée personnelle


Figure 128: User guide - bonne réponse au quiz
Source : Donnée personnelle

En cas de mauvaise réponse, le joueur a la possibilité de revenir à la case précédente, toujours affichée en noir et de relancer le dé ou il peut le relancer directement. Le déplacement suivant se faisant de toute manière à partir de la case initiale en noir.

En cas de bonne réponse, la case magenta devient noire et le joueur peut à nouveau lancer le dé.

Le jeu se termine lorsque le joueur atteint la dernière case. Il n'y a pas de quiz à l'arrivée, un écran de fin s'affiche. Pour recommencer une partie, le joueur doit charger le jeu à nouveau.

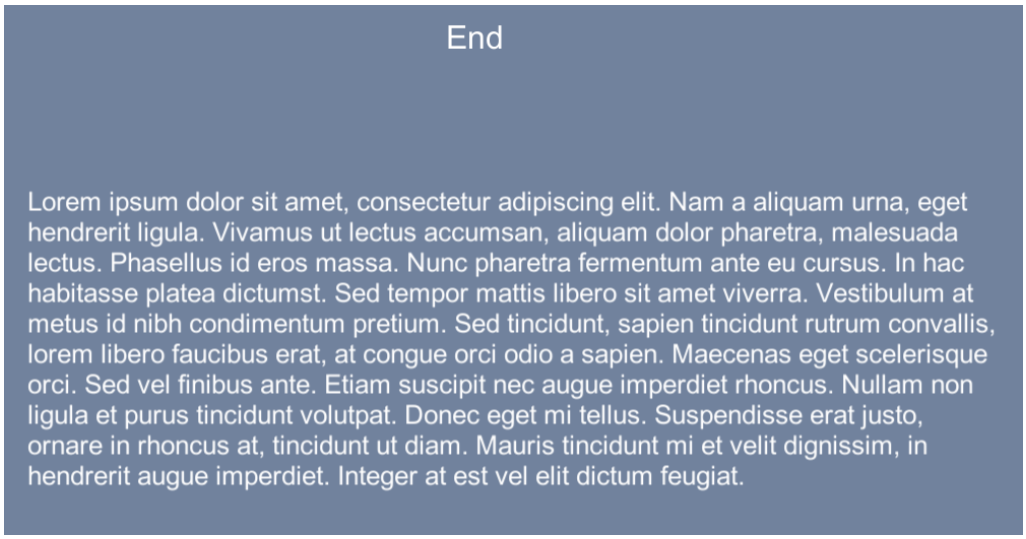



Figure 129: User guide - écran de fin de partie
Source : Donnée personnelle

### Mode Multijoueur

Lorsque le participant sélectionne le mode multijoueur, il arrive sur le lobby. Actuellement, seule une connexion manuelle directe via IP de la machine hôte fonctionne. De plus, elle doit obligatoirement être lancée via la plateforme du jeu Windows.

Il faut au moins deux joueurs pour lancer une partie. Le joueur Windows doit cliquer sur « Play and host » [1]. Les autres joueurs renseignent l'adresse IP de la machine sur le même réseau, ou si la redirection du port 80 est activée, sur l'adresse du modem dans le champ « Join a game » [2] et clique sur « Join » [3].

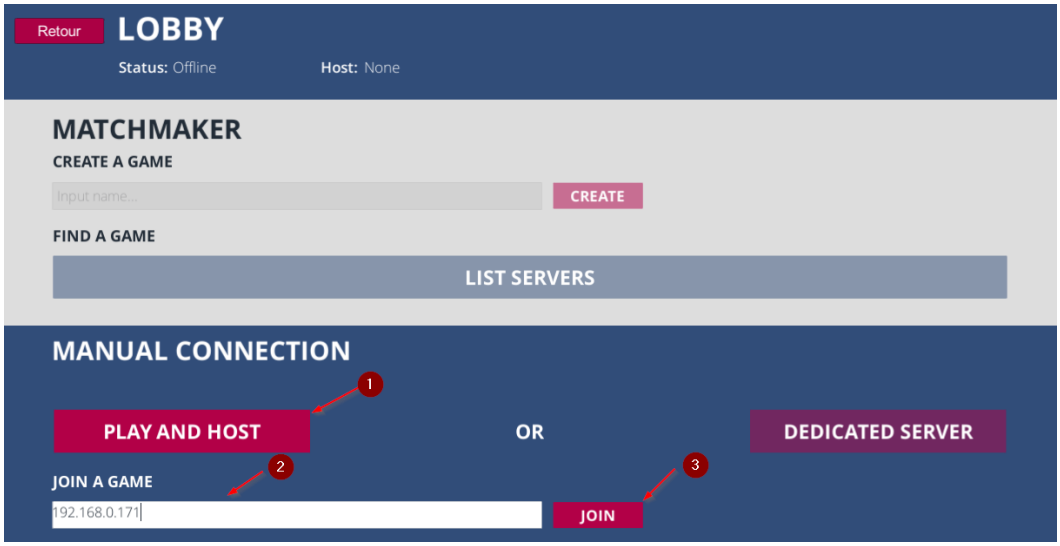



Figure 130: User guide - multijoueur lobby
Source : Donnée personnelle

Lorsque les joueurs sont connectés, ils arrivent dans la « salle d'attente ». Ils peuvent choisir leur couleur [1] et leur nom [2]. Une fois prêts, ils cliquent sur « Join » [3], leur statut passe en « Ready ». Si un joueur se désiste, il peut se déconnecter en cliquant sur la croix [4]. Lorsque tout le monde est prêt, le jeu se charge après 10 secondes.

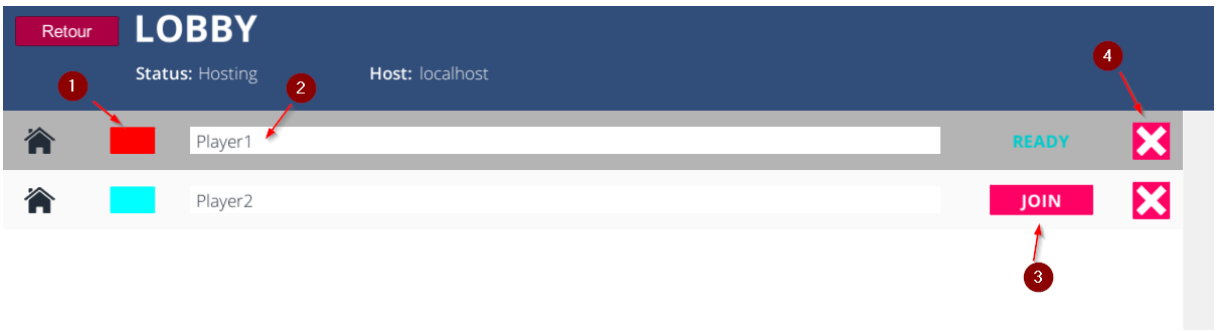



Figure 131: User guide - multijoueur salle d'attente
Source : Donnée personnelle

Une fois le plateau chargé, les joueurs atterrissent automatiquement sur la première case dans un ordre prédéfini (voir Figure 132).

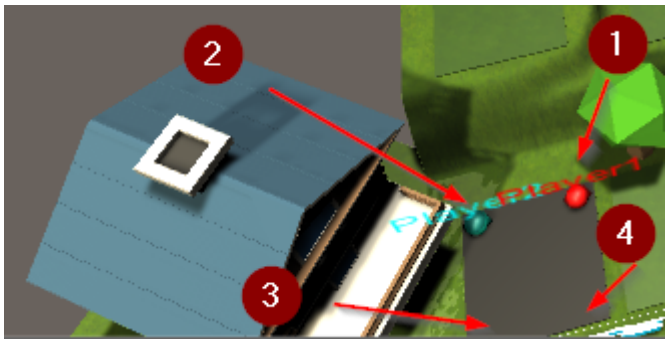



Figure 132: User guide - positionnement initial des joueurs
Source : Donnée personnelle

Le déroulement du jeu est le même que dans le mode « Solo » (voir Figure 125: User guide - mode solo, plateau de jeu). Le jeu contient encore un souci technique (voir chapitre 5.6) : après avoir répondu au quiz, les joueurs deux à quatre ne peuvent pas se déplacer. Ils doivent patienter le tour du joueur 1 et qu'il réponde à la question. La logique de jeu « tour par tour » n'est pas encore intégrée. L'ordre dans lequel les joueurs jouent n'est pas encore défini, mais il faut respecter le même à chaque tour. Lorsqu'un joueur atteint la dernière case, le jeu se termine.

# Annexe VI Product Backlog

| US Nr. | User Stories | | | Acceptance Criteria | Priority | Ne priori | Status | Story Points | Sprint start | Sprint current | Sprint end | US accepte (done done) | MoSCoW |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | En tant que | je veux | afin de | | | | | | | | | | |
| 1 | Utilisateur | avoir un message d'introduction (écrit) | comprendre le but et raison du jeu | Tester que le texte s'affiche correctement | 1000 | | | 1 | 1 | 1 | 1 | | MUST |
| 2 | Utilisateur | changer la langue du message d'introduction en FR-DE-EN-IT | | Tester que les quatres boutons de langue (Fr-En-De-It) change la langue du message d'introduction<br>Tester que la langue séléctionnée soit maintenu pour le reste de l'activité.<br>'Tester que si je choisis le français comme langue dans les paramètres, tous les champs textes soient en français.<br>Tester que si je choisis l'anglais comme langue dans les paramètres, tous les champs textes soient en anglais<br>Tester que si je choisis le allemand comme langue dans les paramètres, tous les champs textes soient en allemand<br>Tester que si je choisis l'italien comme langue dans les paramètres, tous les champs textes soient en italien. | 950 | | | 5 | 1 | 1 | 1 | | MUST |
| 3 | Utilisateur | avoir un texte parlé également dans le message d'introduction | les moins jeunes puissent également comprendre le message ou les paresseux de la lecture | Tester qu'au lancement du jeu, nous entendons le texte parlé. | 900 | | | 2 | 1 | 1 | 1 | | MUST |
| 4 | Utilisateur | pouvoir faire pause lors de la lecture orale du texte | d'aller moins vite et comprendre mieux le message | Tester que le bouton "Pause" arrête le texte parler<br>Tester que le bouton "Pause" se transforme en bouton "Play" lors du clique<br>Tester que le bouton "Play" reprenne la suite du texte parler<br>Tester que le bouton "Play" se change en bouton "Pause" | 600 | 800 | | 3 | 1 | 1 | 1 | | MUST |
| 5 | Utilisateur | pouvoir passer au menu à la fin du texte parlé | | Tester qu'à la fin du texte parlé, la scène change automatiquement vers le "Menu" | 700 | | | 1 | | | | | SHOULD |
| 6 | Utilisateur | pouvoir passer l'introduction | ne pas devoir l'écouter une Xième fois | Tester que le bouton "Passer" s'affiche après quelques secondes<br>Tester que le bouton "Passer" amène à la scène suivante "Menu" | 900 | | | 1 | 1 | 1 | 1 | | MUST |
| 7 | Utilisateur | avoir un menu | jouer au jeu, connaitre les règles, ou accèder aux réglages du jeu | Tester que les trois boutons (Jouer-Règle du jeu-Parametres) apparaissent à l'écran | 950 | | | 1 | 1 | 1 | 1 | | MUST |
| 8 | Utilisateur | lancer un nouveau jeu | | Tester que le bouton "Jouer" mène à la scène de séléction du nombre de participant | 950 | | | 2 | 1 | 1 | 1 | | |
| 9 | Utilisateur | accèder au règle du jeu | | Tester que le bouton "Règle de jeu" mène à la scène qui contient les règles | 800 | | | 1 | | | | | SHOULD |
| 10 | Utilisateur | accèder aux paramètres du jeu | | Tester que le bouton "Paramètres" du "Menu" mène au réglage du jeu | 600 | | | 1 | | | | | SHOULD |
| 11 | Utilisateur | changer la langue du jeu dans les parametres | changer la langue si je ne l'ai pas fait dans l'introduction | Tester que si je choisis le français comme langue dans les paramètres, tous les champs textes soient en français.<br>Tester que si je choisis l'anglais comme langue dans les paramètres, tous les champs textes soient en anglais<br>Tester que si je choisis le allemand comme langue dans les paramètres, tous les champs textes soient en allemand<br>Tester que si je choisis l'italien comme langue dans les paramètres, tous les champs textes soient en italien. | 500 | | | 5 | | | | | SHOULD |
| 12 | Utilisateur | accèder depuis la scène "Régle du jeu" au jeu | lancer une nouvelle partie | Tester que le bouton "Play" mène à la scène de séléction de personnage et du nombre de participant | 600 | | | 1 | | | | | MUST |
| 13 | Utilisateur | séléctionner le nombre de joueur qui vont jouer | | Tester que le nombre de joueur sélectionné soit encadré. | 900 | | | 3 | 1 | 1 | 1 | | MUST |
| 14 | Utilisateur | ajouter un pseudo à mon joueur | | Tester que chaque joueur puisse ajouter un pseudo<br>Tester qu'on ne puisse pas ajouter plus de pseudo que de joueur séléctionné | 900 | | | 3 | 1 | 1 | 1 | | MUST |

Figure 133: Product backlog - part I

Source: Donnée personnelle

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 15 | Utilisateur | lancer un dé | décider l'ordre des joueurs et de pouvoir jouer | Tester que le dé se lance de manière aléatoirement<br>Tester que si il y a un seul joueur, il n'y ait pas de dé à lancer.<br>Tester que si il y a plus d'un joueur, chaque joueur doit lancer le dé afin de déterminer qui commence (le chiffre le plus grand gagne). | 600 | 900 | 🟢 | 5 | 1 | 2 | 2 | | MUST |
| 16 | Utilisateur | séléctionner un personnage | savoir qui je suis durant la partie | Tester que le personnage reste le même durant la partie<br>Tester que chaque joueur puisse choisir un personnage<br>Tester qu'un même personnage ne puisse pas être séléctionner plusieurs fois | 900 | | 🔴 | 8 | 2 | 3 | | | |
| 17 | Utilisateur | savoir qui joue à chaque lancé de dé | ne pas me tromper | Tester que le nom du joueur apparaisse avant le lancer du dé. | 900 | | 🔴 | 5 | 2 | - | | | MUST |
| 18 | Utilisateur | lancer un dé | d'avancer selon le score obtenu | Tester que le joueur ne puisse avancer que selon le score obtenu | 900 | | 🟢 | 5 | 2 | 2 | 2 | | MUST |
| 19 | Utilisateur | recevoir un quizz aléatoir sur les cases "Standard" | pouvoir avancer ou non selon la réponse | Tester qu'un quizz s'affiche sur la case "Standard" de manière aléatoire | 900 | | 🟢 | 5 | 2 | 2 | 2 | | MUST |
| 20 | Utilisateur | rester sur la case en cas de bonne réponse à la question | d'arriver plus vite au sommet | Tester qu'en cas de bonne réponse, le joueur reste sur la case<br>Tester que le joueur commence le tour suivant depuis cette case. | 800 | | 🟢 | 3 | 2 | 2 | 2 | | MUST |
| 21 | Utilisateur | revenir à ma place en cas de mauvaise réponse | | Tester que le joueur retourne à sa case de départ<br>Tester que le joueur commence le tour suivant depuis cette case. | 800 | | 🟢 | 3 | 2 | 2 | 2 | | MUST |
| 22 | Utilisateur | recevoir un challenge "Carte" aléatoire sur les cases "Standard" | pouvoir avancer ou non selon la réponse | Tester que le jeu de carte s'affiche sur la case "Standard" | 800 | | 🔴 | 8 | | | | | MUST |
| 23 | Utilisateur | recevoir un quizz aléatoir sur les cases "Eau" | pouvoir avancer ou non selon la réponse | Tester qu'un quizz s'affiche sur la case "Eau" de manière aléatoire | 750 | | 🔴 | 5 | | | | | SHOULD |
| 24 | Utilisateur | recevoir un challenge "Carte" aléatoire sur les cases "Eau" | pouvoir avancer ou non selon la réponse | Tester que le jeu de carte s'affiche sur la case "Eau" | 750 | | 🔴 | 5 | | | | | SHOULD |
| 25 | Utilisateur | que si plusieurs personnes se trouvent sur la case "Village, le score du prochain joueur qui lance le dé soit le même que pour les autres | promouvoir la collégialité entre les joueurs | Tester que rien ne s'affiche si une seule case "Village" est occupé<br>Tester qu'un message s'affiche si une personne tombe sur une case "Village" alors qu'un joueur se trouve également sur la même case "Village" ou une autre case "Village"<br>Tester que le score du prochain joueur soit le même que pour les autres joueurs qui sont sur une case "Village"<br>Tester que les autres joueurs ne lancent pas leurs dés | 700 | | 🔴 | 8 | | | | | SHOULD |
| 26 | Utilisateur | avancer du nombre de case correspondant à la case "Spéciale" | pour essayer de gagner plus vite | Tester que le joueur avance du nombre de case correspondant de suite | 700 | | 🔴 | 5 | | | | | SHOULD |
| 27 | Utilisateur | reculer du nombre de case correspondant à la case "Spéciale" | | Tester que le joueur recule du nombre de case correspondant de suite | 700 | | 🔴 | 5 | | | | | SHOULD |
| 28 | Utilisateur | aller à la case qui m'est indiquer sur la case "Spéciale" | | Tester que le joueur se déplace sur la case correspondante | 700 | | 🔴 | 5 | | | | | SHOULD |
| 29 | Utilisateur | finir la partie quand un joueur arrive au sommet | | Tester que la partie se termine et que les joueurs soit menés vers la scène de "Fin" | 950 | | 🟢 | 3 | 2 | 2 | 2 | | SHOULD |
| 30 | Utilisateur | recevoir un classement à la fin de la partie | | Tester que le premier arriver soit TOP1, et que les autres suivent selon leur proximité avec la case d'arrivée<br>Tester que si deux joueurs se trouvent à la même place, ils soient à la même place.<br>Tester que le bouton "Menu" mène au menu<br>Tester que le bouton rejouer relance un jeu | 600 | | 🔴 | 5 | | | | | MUST |

Figure 134: Product backlog - part II

Source: Donnée personelle

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 31 | Utilisateur | séléctionner une réponse au Quizz | | Tester qu'on puisse cliquer sur une réponse.<br>Tester que la réponse choisie soit légèrement grisée des autres réponses.<br>Tester que si nous optons pour une autre avis, le marquage change. | 950 | | | 2 | | | | | SHOULD |
| 32 | Utilisateur | valider ma réponse au Quizz | | Tester que le bouton "Valider" apparaisse après avoir cliqué sur une réponse.<br>Tester que le bouton "Valider" colore en vert la réponse, si elle est correcte.<br>Tester que le bouton "Valider" colore en rouge la réponse, si elle est fausse. | 950 | | | 3 | | | | | MUST |
| 33 | Utilisateur | lancer un dé si des joueurs ont le même score | décider l'ordre des joueurs et de pouvoir jouer | Tester que si ils doivent lancer les dés, et que des joueurs ont le même score, ils doivent relancer les dé (seulement ceux qui ont le même score) sans tenir compte des scores précédents.<br>Tester que l'ordre soit bien respecter et afficher durant la scène suivante. | 500 | | | 8 | | | | | MUST |
| 34 | Utilisateur | créer un jeu | jouer à plusieurs sur des PCs différents | Tester que la table se crée<br>Tester que la table est en attente de joueur | 800 | | | 8 | 3 | 3 | 3 | | MUST |
| 35 | Utilisateur | rejoindre une partie | jouer à plusieurs sur des PCs différents | Tester que nous pouvons récupérer la liste des tables<br>Tester que nous puissions nous connecter à une table créer | 800 | | | 5 | 3 | 3 | 3 | | MUST |
| 36 | Utilisateur | me différencier des autres jouerus | savoir qui je suis durant la partie | Tester que le joueur est bien visible<br>Tester qu'un nom apparaisse au dessus du joueur | 700 | 950 | | 3 | 3 | 3 | 3 | | MUST |
| 37 | Utilisateur | voir l'évolution des autres joueurs. | | Tester que la position se mette à jour sur un autre écran | 900 | | | 8 | 3 | 3 | 3 | | MUST |
| 38 | Utilisateur | voir en évidence la case sur laquelle je dois aller | connaitre mon prochain mouvement | Tester qu'après le lancer de dé, la case se mette en magenta pour avancer.<br>Tester qu'après une mauvaise réponse, la case de départ se mette en noir. | 900 | | | 8 | 4 | 4 | | | MUST |
| 39 | Utilisateur | être placer sur la case de départ | débuter la partie | Tester que le joueur 1 soit sur le "spawnposition1"<br>Tester que le joueur 2 soit sur le "spawnposition2"<br>Tester que le joueur 3 soit sur le "spawnposition3"<br>Tester que le joueur 4 soit sur le "spawnposition4" | 900 | | | 3 | 4 | 4 | 4 | | MUST |
| 40 | Utilisateur | une animation d'eau et des éléments de décor | avoir un jeu plus attrayant visuellement | Tester que nous voyons l'eau s'écouler | 900 | | | 5 | 4 | 4 | 4 | | MUST |
| 41 | Utilisateur | répondre à des quizz pertinantes | tester mes compétences dans le domaine de l'eau | Tester que les quizzs s'affichent<br>Tester que nous puissions répondre<br>Tester que la réponse correcte affiche un panneau vert<br>Tester que la réponse rouge affiche un panneau rouge | 900 | | | 2 | 4 | 4 | 4 | | MUST |
| 42 | Administrateur | ajouter, modifier, supprimer un quizz | varier le jeu | Tester que nous puissons changer les quizzs | 850 | | | 3 | 4 | 4 | 4 | | MUST |

Figure 135: Product backlog - part III

Source: Donnée personnelle

## Déclaration de l'auteur

Je déclare, par ce document, que j'ai effectué le travail de Bachelor ci-annexé seul, sans autre aide que celles dûment signalées dans les références, et que je n'ai utilisé que les sources expressément mentionnées. Je ne donnerai aucune copie de ce rapport à un tiers sans l'autorisation conjointe du RF et du professeur chargé du suivi du travail de Bachelor, y compris au partenaire de recherche appliquée avec lequel j'ai collaboré, à l'exception des personnes qui m'ont fourni les principales informations nécessaires à la rédaction de ce travail et que je cite ci-après : M. Pascal Ornstein, et M. Xavier Garcia.