Université de Montpellier - Faculté des Sciences
Cursus Master en Ingénierie
Année 2016 / 2017

Jeu « GeoQuizz »

Rapport :

Réalisé par :

Alexis BRETON
Meriem AMERAOUI
Miassa ABBOUTE

Etudiants en L1 _CMI_ Informatique

Encadrés par :

M me . Anne-Elisabeth BAERT

SOMMAIRE

INTRODUCTION

Le module HLSE205 est un module de projet informatique ouvert aux étudiants de première année CMI (Cursus Master en Ingénierie) Informatique, EEA et Mathématiques, de l'Université de Montpellier - Faculté des Sciences.

L'objectif de ce module est d'apprendre et de s'habituer à rédiger des rapports. De ce fait, plusieurs rapports doivent être remis au cours du second semestre (S2) de la première année (L1).

En effet, après avoir déposé, sur l'espace Moodle, un rapport détaillé sur le « Jeu de la vie » et un pré-rapport sur un jeu vidéo à programmer, individuellement, un rapport final est à remettre sur ce jeu vidéo même, en groupe, en plus de son code complet, avant le 18 avril 2017. Ainsi, une soutenance orale est organisée pour le 28 avril 2017.

La formation des groupes, constitués de 3 ou 4 étudiants, a été faite le vendredi 24 février. Aidés de cinq séances communes de TP, donc quinze heures de travail au total à l'université, le jeu a été entamé . Le reste du travail sera effectué hors présentiel.

Comme déclaré dans le pré-rapport remis le 01 mars, nombreux sont les jeux vidéo à développer. Après le Jeu de la Vie, décidés d'un commun accord des responsables des filières Informatique, Mathématiques et EEA, le second jeu doit être le choix de chacun des groupes d'étudiants de ces domaines mêmes.

Pour cela, une liste est mise à notre disposition :

- Jeu de la vie
- Bubble shooters
- Arkanoid
- Isola
- Morpion
- Hex
- Tic Tac Toe
- Bomberman

Cependant, pour notre groupe le choix était tout autre, nous avons choisi de programmer un jeu éducatif et culturel : le « *GeoQuizz* ». Mais qu'est-ce que le « GeoQuizz » ?

CAHIER DES CHARGES

Avant de se lancer dans le développement du jeu, s'intéresser sur ce qu'est le « GeoQuizz », son but et ses fonctionnalités est très important.

Définition du « GeoQuizz »

Le mot « GeoQuizz » est un mot valise, formé par la fusion de deux mots. « Geo » pour géographie : [Science de la connaissance de l'aspect actuel, naturel et humain de la surface terrestre][1] et « Quizz » : [Questionnaire permettant de tester des connaissances générales ou spécifiques ou des compétences][2].

Le « GéoQuizz » est un jeu à but éducatif, regroupant trois (03) types de jeux : localisation, zone et, comme son nom l'indique, quizz.

Toutes les tranches d'âge peuvent y jouer, que se soit pour : apprendre en s'amusant, tester ses connaissances, préparer un examen (brevet, baccalauréats ou autres), ou simplement réviser ses acquis et se les remettre en tête (en effet, il peut être considéré comme un jeu de mémoire).

Ce jeu est trouvé sous plusieurs formes. Pour les adeptes de technologie, il est présenté gratuitement en ligne (Tout le monde y a accès). Et pour les plus traditionnels, des livres le proposant existent :

- Geography Quiz Book de Sachin Singhal (Indien)
- Rupa Book Of Geography Quiz de Subbiah Muthiah (Indien)
- Geography Quiz Book de Arnab Goswami (Indien)
- Canadian Geographic Quiz Book de Doug MacLean (Canadien)
- The little Geography Quiz Book de David Deal

Cependant, tous les sites et livres ne présentent pas le jeu de la même manière.

En effet, certains le propose sous forme de QCM ou questions/réponses, d'autres en utilisant des cartes géographiques : du monde, d'un continent, d'un pays ... .

Comme jeu, nous sommes partis sur cette seconde idée.

But du jeu

Le but du « GeoQuizz » : trouver un maximum de pays en un minimum de temps. Ça a l'air simple, mais pas tout à fait.

Des questions seront posées sur le thème de la géographie (Où se trouve tel pays ?) le joueur devra y répondre en choisissant sur la carte le pays qui lui semble être le bon.

Il y a des niveaux de difficulté, bien sur : faible, moyen et difficile (le choix des ces niveaux est expliqué dans la page suivante), en plus, du chronomètre. A chaque niveau, le gamer devra respecter un temps de jeu limité à dix secondes.

Le choix des niveaux (Comment le choix des niveaux a-t-il été fait ?)

Le nombre exact de pays à travers le monde n'est pas facile à déterminer. L'ONU par exemple reconnaît en 2012, 197 pays. Certains présentent un nombre beaucoup plus élevé de pays à travers le monde ... .

Il y a des pays plus connus que d'autres, ne serait-ce que par leur position géographique, leur histoire, leur politique etc. Ceux-là seront dans le niveau facile. Les moins connus, seront dans le niveau moyen. Et les pays peu connus seront dans le niveau difficile.

Fonctionnalités du jeu

L'idée générale et le but du jeu étant énoncés un peu plus haut, ses fonctionnalités sont les suivantes :

- Affiche une carte du monde vierge et d'un menu interactif (instructions) ;
- Demande au joueur de choisir un continent sur la carte ;
- Zoom sur le continent choisi ;
- Demande le niveau de difficulté ;
- Affiche un nom de pays à trouver ;
- Demande au joueur de choisir un pays sur la carte ;
  - Affiche le pays choisi en vert si c'est le bon, et demande un autre pays
  - Affiche un message d’erreur sinon
- Compte et affiche le score ;
- A la fin de la partie, demande le nom du joueur et sauvegarde le score ;
- Affiche le tableau des scores du joueur ;
- Propose de repartir à l’accueil (et rejouer donc choisir un autre continent et/ou un autre niveau de difficulté) ;

Les détails comme :

- Les sons joués, lors des sélections ;
- La présence ou pas d'une musique de fond ;
- Le changement de couleur du pays lors du passage de la souris dessus ;
- Le minuteur/chronomètre ;

et bien d'autres encore ne sont pas énumérés mais seront présents.

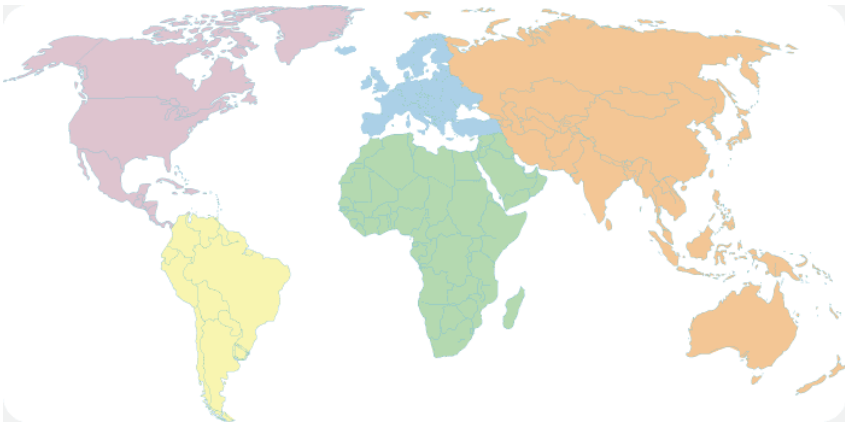
D’autres fonctionnalités, toujours dans le thème de la géographie, comme :

- Affiche une capitale et demande au joueur de sélectionner le pays correspondant ;
- Affiche un drapeau et demande au joueur de sélectionner le pays correspondant ;
- Propose un pays et demande au joueur de trouver des régions, des départements, des villes … ;
- Affiche une des sept (07) merveilles du monde et demande au joueur de trouver le pays ;

étaient proposées, mais faute de temps, n’ont pas été ajoutées.

Exemples sur certaines fonctionnalités du jeu

1. Affichage d'une carte du monde vierge



Carte du monde vierge

2. Zoom sur le continent choisi

Continent Sud-Américain

3. Affichage du tableau des scores du joueur

Joueur	Score	Niveau
Lucie		
Anne		
Julien		

ORGANISATION DU PROJET

La réussite d'un projet passe par une organisation rigoureuse et efficace des membres du groupe. En effet, certaines méthodes sont préférées à d'autres.

Organisation du travail

Le côté relationnel est un point très important , car le moins évident à gérer est bien l'humain. Pas évident de jongler avec les caractères et les compétences de chacun.

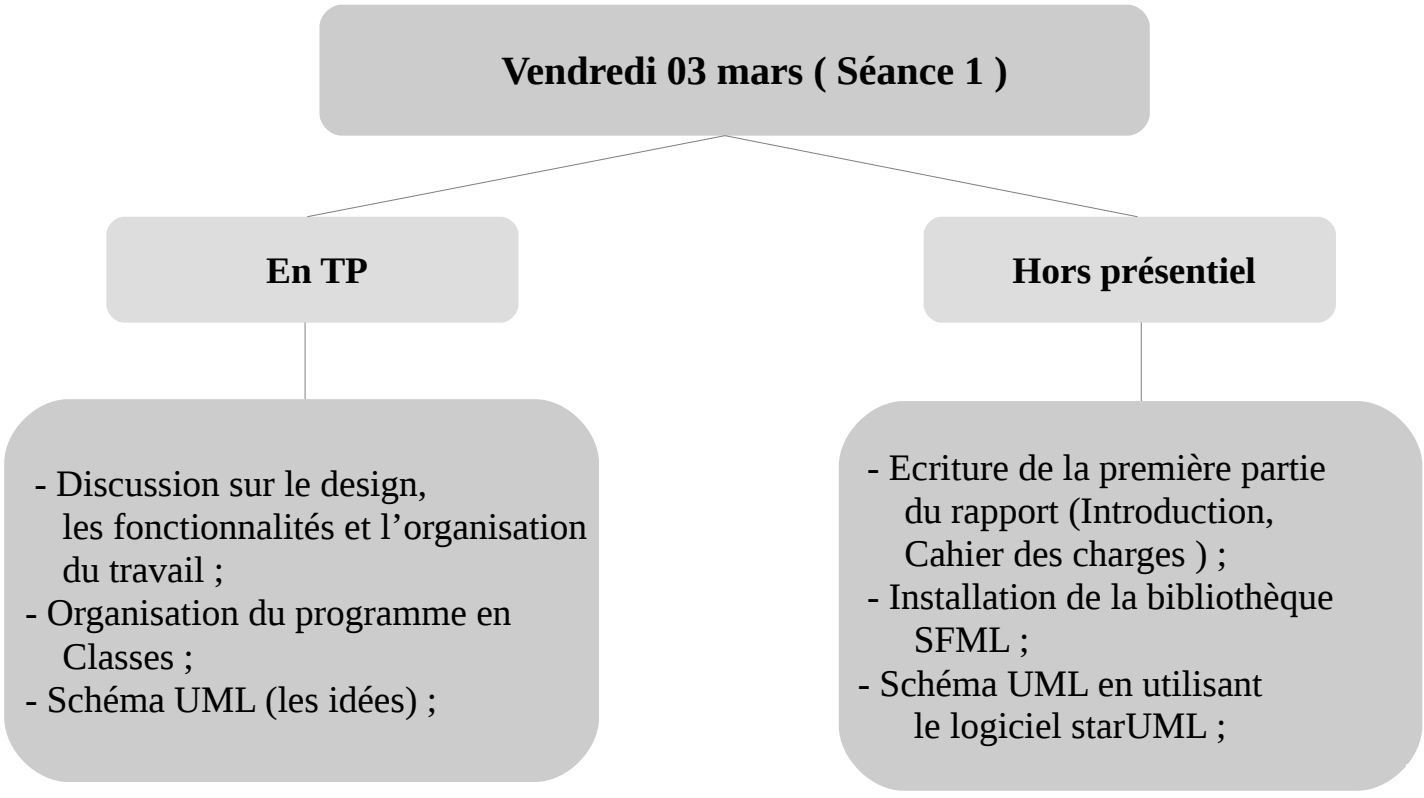
Effectivement, cela s'est vite confirmé. Etant un groupe de trois (03) étudiants, se mettre d'accord sur le jeu à développer n'a pas été chose facile. Chacun de nous avait sa propre idée et des arguments assez convaincants. Cependant, ayant tous le même objectif : réussir !! et sachant que la cohésion de l'équipe est essentielle, le désaccord n'a pas duré longtemps. Chacun a pu s'exprimer et apporter aux autres ses idées.

Tous voulions une idée originale de jeu mais faisable en un intervalle de temps limité et, donc, trouvâmes un compromis : le « GeoQuizz ». En effet, ce jeu n'a été proposé par aucun autre groupe.

Après s'être mis d'accord, vint le moment de la planification. L'outil requis est le planning. Le but de ce dernier est :

- déterminer si les objectifs sont réalisés ou dépassés ;
- suivre l'avancement du projet ;
- affecter les ressources aux tâches ;

Le travail à faire étant trop important, à peu près 25 % a été exécuté pendants les heures de TP et 75 % hors présentiel :



Vendredi 10 mars (Séance 2)

En TP

- Correction de la première partie du rapport ;
- Finalisation du schéma UML ;
- Explication du schéma UML ;
- Recherches sur internet ;
- Répartition des tâches pour les séances qui ont suivies ;

Hors présentiel

- Ecriture de quelques algorithmes/codes ;
- Quelques modifications du schéma UML ;
- Installation de certains logiciels (Code::Blocks, Visual Studio)

Vendredi 17 mars (Séance 3)

En TP

- Finalisation de la première partie du rapport ;
- Ré-explication du schéma UML ;
- Explication du programme ;
- Début programmation des classes ;

Hors présentiel

- Discussion et préparation de la seconde partie du rapport ;
- Discussion graphisme du jeu ;
- Programmation des classes Restantes ;

Vendredi 24 mars (Séance 4)

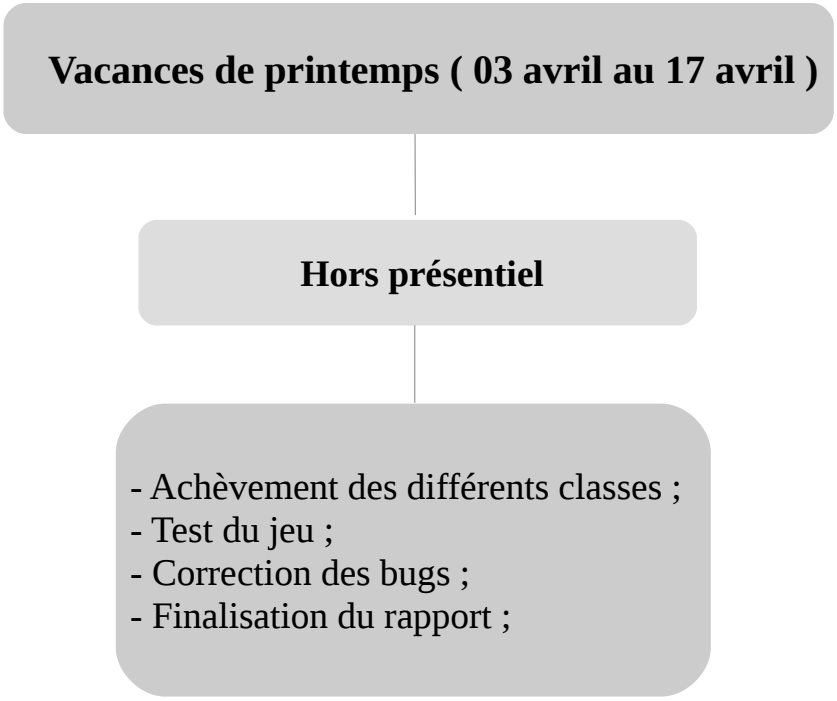
En TD et Hors présentiel

- Ré-installation de Code::Blocks ;
- Installation d'autres éditeurs ;
- Amélioration et progression sur le code du jeu ;
- Recherche idées et solutions pour le remplissage de différentes parties d'une image (demande aide sur forums et envoi mails à des ingénieurs spécialisés) ;
- Répartition des tâches pour la Semaine et le week-end ;

Vendredi 24 mars (Séance 4)

En TD et Hors présentiel

- Création des fenêtres et boutons ;
- Démonstration du jeu ;
- Recherche/modification/création algorithme remplissage ;
- Discussion sur les parties restantes du rapport final ;



Choix des outils de développement

Le développement d'un jeu vidéo décent nécessite des outils plus que décents. En effet, d'un ordinateur frais, d'un système d'exploitation récent, de logiciels puissants résultera forcément un jeu vidéo digne.

Le développement de ce jeu étant en groupe, être d'accord sur les choix des outils est important. Ils doivent être communs à tous. Les logiciels, le langage de programmation, les bibliothèques de ressources … ne peuvent différer.

Cependant, trois étudiants de la même filière ayant trois ordinateurs différents peuvent ne pas avoir le même OS (Operating System). Ce qui a été le cas, trois PC distincts et trois OS distincts : Windows, Linux et Mac, ce qui a engendré quelques problèmes : la disponibilité de certains logiciels sur certains OS mais pas sur d'autres, la non gratuité de certains logiciels et la non compatibilité d'autres …

Des compromis ont, donc, été faits, plusieurs logiciels ont été installés/téléchargés puis testés pour être utilisés ou mis de côté/supprimés.

La non concordance des OS a été une des entraves rencontrées pour le développement du « GeoQuizz ». Cela a causé une perte de temps assez importante pour cause d'installation et de ré-installation …

Ensuite, vint le temps de choisir le langage de programmation le plus adapté au développement de jeux. Il en existe plusieurs : C#/C/C++, Python, Perl, Java, Turbo Pascal, Lua … Néanmoins, certains correspondent plus que d'autres (performance, compatibilité, besoins). Il est, donc, nécessaire de choisir le langage maîtrisé en prenant compte des qualités de chaque.

Après avoir effectué certaines recherches, le C++ est le plus convenu pour des jeux vidéos performants. Effectivement, c'est un langage de programmation compilé : il est portable, permet la manipulation des objets en plus d'excellents moteurs graphiques. C'est également le langage étudié en L1 (HLIN202) et donc le seul en commun des membres du groupe.

Comme énoncé, le C++ utilise les concepts de la programmation orientée objet, ce qui est, évidemment, un plus :

- Une meilleure organisation des fonctions : chaque fonction est intégrée à une classe ;
- La modularité : des éléments peuvent facilement être rajoutés sans avoir à modifier tout le code grâce à l'héritage ;
- La clarté du code : facilité à le maintenir ;

Quelques exemples de classes

- Une classe Pays qui rend un booléen (*true* ou *false*) ;
- Une classe Continent qui est composé de pays ;
- Quelques classes filles qui héritent de la classe mère ;

les détails de plusieurs classes sont dans la partie « Analyse du projet ».

Après ça, comme interface de programmation pour jeux vidéo, la bibliothèque SFML a été choisie. Elle est multimédia (pour le graphisme, l'audio...), multi-plateforme (s ur la plupart des systèmes d'exploitation : Linux, Windows... très pratique ! Tous les membres du groupes ont un OS différent) et multi-langage (pour les langages C (C,C++,C#) , Java, Ruby, Python...). De plus, le nombre de sites et de vidéos expliquant son utilisation est énorme, ce qui a aidé pour l'apprentissage.

Enfin, comme déclaré précédemment, utilisant trois systèmes d'exploitation différents, des bugs ont été rencontrés.

Plusieurs éditeurs de texte/code ont été installés :

Sur Windows : Visual Code ;
Sur Mac : Code::Blocks, Xcode ;
Sur Linux : Visual Code, Code::Blocks, Emacs et Eclipse .

Au début, Code::Blocks a été choisi comme environnement de développement car il est développé pour Linux, Windows et Mac OS X, mais aussi très extensible, entièrement configurable et destiné au langage C++. Néanmoins, ayant rencontré quelques anomalies et après qu'un des membres de l'équipe ait choisi Visual Studio pour ses nombreux avantages : [il s'ouvre à toutes les tendances du moment, et devient un véritable " Couteau suisse " dans le monde du développement, qui pourra satisfaire divers profils de développeurs et permettre de développer tout type d'application … il offre une prise en charge des langages de programmation C#, VB.NET, F#, C++, Python, JavaScript, HTML et bien plus][3], d'un commun accord, le jeu a été codé sur Visual Studio (OS : Windows).

Le logiciel StarUML (logiciel de modélisation UML) a aussi été utilisé par un membre du groupe pour fournir une méthode normalisée pour visualiser la conception : il est très simple d'utilisation, nécessitant peu de ressources système. Il constitue une excellente option pour une familiarisation à la modélisation.

Et pour le côté graphique, le logiciel GIMP (logiciel de retouche d'image) a été utilisé pour ses caractéristiques : il est gratuit, il est très rapide, il possède une interface graphique simple et facile à utiliser, c'est un modèle de développement libre et ouvert qui propose diverses options (retouche et édition d'images, dessin à main levée, rognage, réajustement …). Des versions de GIMP existent pour le MAC OS X.

MÉTHODOLOGIE

Contrairement aux idées reçues, un bon programmeur ne se caractérise pas par sa non-utilisation d'une feuille et d'un stylo. Ces outils " traditionnels " peuvent s'avérer bénéfiques pour le développement d'un quelconque jeu, comme pour écrire la méthodologie.

Une fois l'équipe intégrée, les outils de développement choisis, se lancer directement dans le développement du jeu n'est pas une très bonne idée. Une méthodologie doit être déterminée, dans le but d'organiser le travail et de faciliter l'apprentissage !!

Etant trois (03) étudiants à travailler sur le projet, la méthodologie à adopter est, évidemment, plus simple que pour un travail individuel :

- Communiquer/Echanger avec les membres de l'équipe sur chaque tâche effectuée/ajoutée/supprimée/modifiée ;
- Noter les caractéristiques (fonctions) principales du jeu ;
- Approfondir les détails ;
- Etablir " un manuel d'utilisation du jeu " (pour n'omettre aucun détail) ;
- Communiquer/Echanger ;
- Ecrire quelques algorithmes en langage naturel des fonctions de base ;
- Apprendre à manipuler tous les logiciels/outils utilisés (le développement sera moins complexe) ;
- Traduire les algorithmes en langage choisi ;
- Communiquer/Echanger ;
- Se lancer dans la programmation ;
- Tester/Debugger ;
- Concevoir la structure du jeu ;
- Améliorer autant que possible ;
- Jouer, jouer et jouer ;
- Faire jouer la famille et les amis ;

Comme vu ci-dessus, les étapes Communiquer/Echanger et Jouer sont essentielles ! La première a eu pour objectif d'informer les membres de l'équipe sur chaque tâche effectuée/ajoutée/supprimée/modifiée ou autre. Pour la seconde, son but a été de tester encore et encore pour détecter tous les bugs (un jeu qui bugge n'est pas amusant). La dernière phase a aussi eu son importance. En effet, observer comment les personnes réagissent et interagissent avec le jeu s'est avéré bénéfique pour son amélioration.

ANALYSE DU PROJET

Programme orienté objet

Les étudiants du groupe ayant déjà des connaissances de base en programmation orientée objet grâce à une expérience avec le langage Java, il a été décidé de programmer le « GeoQuizz » en orienté objet.

En effet, cette méthode de développement apporte une meilleure organisation du code et une plus grande clarté qui a facilité le codage en équipe puisque chaque module peut être réduit en une classe comportant une ou plusieurs fonctionnalités. Les relations entre les classes permettent aussi de mieux comprendre comment chaque module fonctionne, et de ce fait, éviter d'avoir un gros code unique dont il est difficile d'en comprendre l'organisation. La structure du programme en classes permet également une meilleure modularité, dans le cas où une classe doit être ajoutée ou améliorée, les classes utilisatrices n'ont pas besoin d'être modifiées. Ce qui peut être utile dans la perspective d'ajouter des fonctionnalités en temps libre.

Comme l'organisation du programme est en classe, la réalisation d'un schéma UML, dans le but de représenter leur hiérarchie, est donc possible. Ceci a facilité, d'autant plus, l'organisation de l'équipe et la compréhension du fonctionnement des différents éléments.

State pattern

Le jeu a été constitué de plusieurs menus : il y a donc, le menu principal, suivi du menu du choix du continent, puis un menu du choix de la difficulté, suivi d'un menu où se déroule la partie du jeu, et enfin le menu des scores où le joueur enregistrera son score et aura accès au tableau des scores.

Tous ces menus sont, en fait, des *états* du jeu. Ce programme a donc pris ces différents états :

- L'état menu principal ;
- L'état continent ;
- L'état difficulté ;
- L'état jeu ;
- L'état scores ;

Avec cette division du « GeoQuizz » en différents états, l'utilisation du modèle de conception (ou Design Pattern en anglais), appelé *State Pattern*, est devenue possible. Ce dernier fonctionne avec une classe « Contexte » qui contient des instances d'une super-classe « Etats ». Chaque état hérite de cette super-classe. Grâce au polymorphisme, le « Contexte » peut changer l'état qui est actif et exécuter ses fonctionnalités sans se préoccuper de quel état il s'agit. Ceci a facilité l'écriture des différents états et le passage d'un menu à un autre. Dans le cas de ce jeu, le « Contexte » est une unique instance d'une classe qui a été nommée « Engine » (le moteur du jeu). La super-classe « Etats » a été appelée « GameState » et est une interface. Les différentes sous-classes ont été étiquetés comme suit :

- « MenuState » pour le menu du jeu ;
- « ContinentState » pour le menu du choix du continent ;
- « DifficultyState » pour le menu du choix de la difficulté ;
- « PlayingState » pour le menu où le jeu se déroule ;
- « ScoreState » pour le menu des scores ;

C'est l'instance de la classe « Engine » qui s'occupe de changer et d'afficher létat courant. Chaque état appelle une méthode de la classe « Engine » pour changer l'état actif. Et chacun des états possède également trois méthodes principales de type *void* :

- Une méthode input() qui gère les interactions avec l'utilisateur (entrées clavier et souris) ;
- Une méthode update() qui met à jour l'environnement (position des affichages, gestion du temps etc.) ;
- Une méthode draw() qui s'occupe d'afficher tous les éléments à l'écran ;

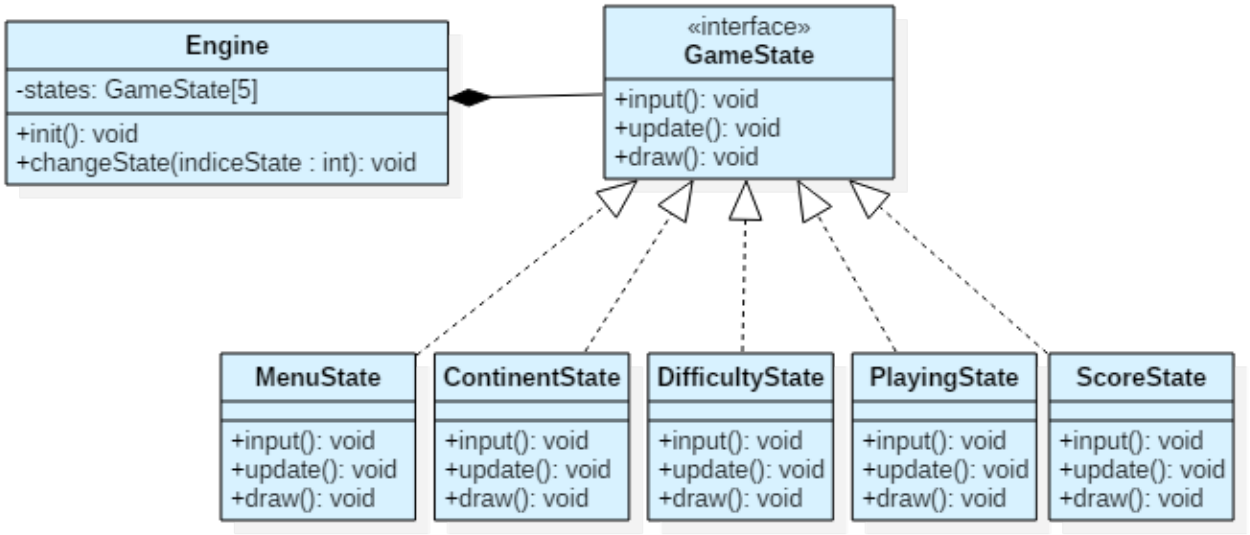


Figure 1 – Diagramme des classes « Engine » et « GameState » avec ses sous-classes

Structure des classes

Les principales classes de ce jeu vont avoir besoin d'autres classes pour pouvoir être fonctionnelles. Ainsi chaque état va hériter par l'interface « GameState » d'une référence sur un objet de type « Player » et d'une référence sur un objet de type « Map ».

La classe « Player » est une classe qui comporte les paramètres du joueur : son nom, le continent et la difficulté choisis, et son score. Tous ces paramètres seront enregistrés dans un fichier grâce à une méthode *saveScore* de la classe « ScoreState ». De plus, chaque score avec ses paramètres de jeu qui seront chargées à partir de ce fichier, sera une instance de la classe « Player ». La classe « Map », quant à elle, contient la carte du monde.

Chaque état possède de même des instances d'une classe « Bouton ». Celle-ci possède des méthodes « testSelected » et « testClick » qui prennent en paramètres les coordonnées de la souris et qui renvoient des booléens en fonction du passage de la souris ou du clic sur le bouton.

Le programme est composé de plus d'une interface « Surface » qui possède un nom et deux méthodes abstraites qui sont, de manière analogue aux boutons, « testSelected » et « testClick». Une classe « Continent » hérite de cette interface et possède une méthode «loadFromFile » pour charger ses données depuis un fichier. Elle possède également un tableau d'objets de type « Pays ». Cette classe « Pays » hérite également de l'interface «Surface » en y ajoutant, comme les continents, une méthode « loadFromFile ». Les pays possèdent aussi un booléen pour savoir s'ils ont été trouvés par le joueur et d'un Sprite avec une texture pour les afficher à l'écran et changer leur couleur. Un pays est en effet une texture qu'on ajoute par-dessus la texture de la carte.

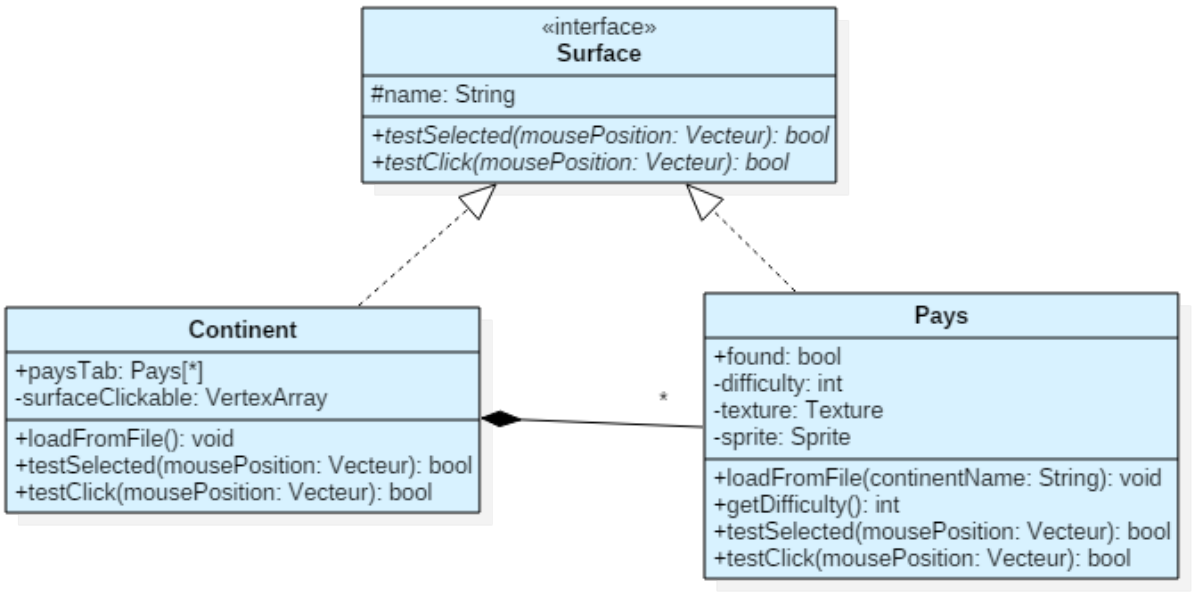


Figure 2 – Diagramme des classes « Surface », « Continent » et « Pays »

Structure des données

Chaque pays possédant un nom et une difficulté doit avoir des coordonnées sur la carte. La connaissance de ces coordonnées est nécessaire pour placer la texture de chaque pays au bon endroit. Il serait trop lourd et peu pratique d'écrire ces informations directement dans le code. C'est pourquoi, la création d'un fichier texte pour chaque pays, contenant son nom, ses coordonnées et sa difficulté (Figure 3), a été choisie. Cela rendra plus aisé la création et la modification des caractéristiques des pays. Leur texture quant à elles sont chacune stockées au format *.png* dans un dossier.

```
Albanie

2100 866

3
```

Figure 3 – Exemple d'un fichier texte d'un pays

```
Afrique

viewcenter
2050 1310

viewsize
1700 1050

Nbvertex
16
coords
2088 1848
2406 1718
2030 1576
2484 1572
2034 1468
2350 1500
1990 1372
2498 1270
1758 1346
2376 1238
1688 1226
2252 1026
1714 1108
2026 996
1818 976
2004 938

pays
Afrique-du-Sud
Algérie
Egypte
```

Figure 4 – Exemple d'une partie d'un fichier texte d'un continent

Le même travail est effectué pour les continents, en incluant pour chacun, dans leur fichier texte, les coordonnées des points de leur surface cliquable et la liste des pays qu'ils contiennent. Ils comportent également les caractéristiques de leur vue. En effet, la vue sera centrée et zoomée sur le continent qu'aura choisi le joueur, l'entrée des coordonnées de celle-ci est donc, utile, pour chaque continent (Figure 4).

DÉVELOPPEMENT DU PROGRAMME

Algorithme des différents modules

Après avoir analysé et structuré le programme de ce jeu, la partie technique peut être entamée et l'écriture des algorithmes peut être commencée. Les plus importants d'entre eux sont donnés en-dessous, pour une meilleure explication du fonctionnement des classes entre elles.

1. La classe « Engine »

En plus des méthodes *init()* et *changeState()* vues, la classe « Engine » comporte aussi la fenêtre du jeu. Avec la bibliothèque SFML, une instance de la classe *sf::RenderWindow* a été créée. La classe contient aussi un entier indiceActiveState qui servira à déterminer quel état est actuellement activé, ainsi que d'une instance de nos classes « Player » et « Map ». Pour le tableau de pointeurs sur les états, l'outil *vector* de la bibliothèque standard et des *unique_ptr* sont utilisés pour pointer les états. Les *unique_ptr* sont des pointeurs " intelligents " qui détruisent l'objet pointé lorsque le pointeur devient inaccessible. Conformément au principe d'encapsulation, les méthodes sont *public* et les attributs sont *private*. Par ailleurs, une méthode *private*, qui est *void loadgame()*, a été ajoutée. Elle gère le chargement du jeu au démarrage de celui-ci et affiche un écran de chargement (voir ci-dessous l'algorithme de la classe).

```
class Engine
{
public:
      Engine();
      ~Engine();
      void init(); //Lance la boucle principale
      void changeState (int stateNb); //Change le menu actif
      /* Le changement de state se fait en indiquant
      le numero du state que l'on veut
      (0 pour le menu, 1 pour le playing state ...) */
private:
      sf::RenderWindow window;
      std::vector<std::unique_ptr<GameState>p_stateTab;//Table
      au de pointeurs
      int indiceActiveState;
      Player player;
      Map map;
      void loadGame(); //Charge les données et affiche l'écran
      de chargement
} ;
```

Ces algorithmes ci-dessous montre le fonctionnement des méthodes *init()* et *changeState() :*

```
void Engine::init()
{
	loadGame();

	while (window.isOpen())
	{
		p_stateTab[indiceActiveState]->input();
		p_stateTab[indiceActiveState]->update();
		p_stateTab[indiceActiveState]->draw();
	}
}
void Engine::changeState(int stateNb)
{
indiceActiveState = stateNb;
}
```

2. La classe « GameState »

Comme déclaré précédemment, la classe « GameState » est une interface, c'est-à-dire qu'elle n'a pas l'intention d'être instanciée. Ses méthodes doivent donc être virtuelles (*virtual* en C++). Les méthodes virtuelles sont des méthodes qui peuvent être redéfinies dans les classes filles. Dans ce cas, lorsque cette méthode virtuelle d'un object, que l'on considère comme étant de type de la classe mère, est appelée, c'est effectivement la méthode de la classe fille qui est appelée. Dans cet exemple, il y a des objets de type « MenuState », « PlayingState » etc. (classes filles) mais ils sont considérés comme des objets de type « GameState » (classe mère) dans la classe « Engine ». Ainsi, quand dans cette dernière, les méthodes *input()*, *update()* et *draw()* sont appelées, ce n'est pas la méthode de la classe « GameState » qui est appelée mais bien celle de la sous-classe, grâce au mot-clef *virtual*.

Chaque état doit posséder une référence sur une instance des classes «Engine», « Player », « Map » et aussi sur la fenêtre créée dans la classe «Engine ». Ces références sont passées en paramètre du constructeur de chaque état. Ceux-ci possèdent également une vue qui détermine ce que peut voir le joueur. Tous ces attributs sont mis en *protected* dans « GameState » pour que chaque état puisse en hériter. Le code ci-dessous est ainsi, obtenu :

```
class GameState
{
public:
	GameState(sf::RenderWindow&	window,	Engine&	engine,
	Player& player, Map& map);
	~GameState();
	virtual void input() =0; //Gère les entrées clavier et
	souris
	virtual void update()=0; //Met à jour l'environnement
	virtual void draw()=0; //Affiche ce qu'il faut à l'écran
protected:
	sf::RenderWindow& window;
	Engine& engine;
	Player& player;
	Map& r_map;
	sf::View view; //Vue propre à chaque menu
	sf::Font geoFont; //Police de caractère qui servira aux
	textes des menus
	sf::Color myGrey; //Couleur des textes
};
```

3. La classe « Player »

La classe « Player » possède, comme il a été dit, un nom, un score, une difficulté et un pointeur sur le continent choisi. Elle contient aussi des méthodes *getter* et *setter* pour chaque attribut car il est nécessaire de lire et d'écrire pour sauvegarder et charger les scores. Il y a, d eplus, une méthode *réinitialisation* qui sera appelée à chaque fin de partie et qui réinitialisera tous les paramètres du joueur. Ainsi, le code suivant, dans lequel les *getter* et *setter* ont été omis pour plus de clareté, est obtenu :

```
class Player
{
public:
	Player();
	Player(std::string name, int score, short int difficulty,
	std::string continentName);
	~Player();
	void reinitialisation();
private:
	std::string name;
	int score;
	short int difficulty;
	std::string continentName;
	Continent* p_continentChoosed;
};
```

Voici la définition de la méthode réinitialisation :

```
void Player::reinitialisation()
{
	name = "";
	score = 0;
	difficulty = 0;
	p_continentChoosed = NULL;}
```

4. La classe « Continent »

Pour contenir tous les pays de la classe « Continent », le choix a été d'utiliser, de manière analogue au conteneur de « GameState » dans la classe « Engine », un *vector* de *unique_ptr* sur des « Pays ». Ce *vector* sera déclaré *public* car il sera beaucoup manipulé en lecture et écriture par d'autres classes.

La surface cliquable des continents est un type *VertexArray* de la bibliothèque SFML. C'est un tableau de points qui, en fonction du type de forme primitive choisie (*Triangle Strip*, *Triangle Fan*, *Quad* etc.), permet de créer une forme personnalisée. Dans ce cas, un *Triangle Strip* (chaque point forme un triangle avec les deux derniers points entrés) a été choisi.

Comme vu précédemment, cette classe redéfinie les méthodes *testSelected* et *testClick* de la classe « Surface ». Le mot-clef *override* est donc, mis pour s'assurer qu'elle les redéfinie bien.

```
class Continent : public Surface
{
public:
	std::vector <std::unique_ptr<Pays>> paysTab;

	Continent(std::string name);
	~Continent();
	sf::Vector2f getViewCenter();
	sf::Vector2f getViewSize();
	bool testSelected(sf::Vector2f& mousePos) override;
	bool testClick(sf::Vector2f& mousePos) override;
	void loadFromFile();
private:
	sf::Vector2f viewCenter;
	sf::Vector2f viewSize;
	sf::VertexArray surface;
};
```

Voici ci-dessous le code de la méthode *testClick*. Elle utilise la méthode *testSelected*. Cette dernière fonctionne ainsi : comme la surface du jeu est une forme composée de triangles, une détection du cas où les coordonnées de la souris se trouvent à l'intérieur d'un des triangles de la forme. Le soin de regarder le fonctionnement de cette dernière fonction dans le code complet du jeu a été laissé au lecteur, s'il le souhaite, car elle est assez lourde pour pouvoir être mise ici.

```
bool Continent::testClick(sf::Vector2f & mousePos)
{
     if (testSelected(mousePos) &&
                    sf::Mouse::isButtonPressed(sf::Mouse::Left))
     {
              for (int i = 0; i < paysTab.size(); i++)
              {
              paysTab[i]->setNotSelectedColor();
              }
              return true;
     }
     else
              return false;
}
```

5. La classe « Pays »

Tout comme la classe « Continent », « Pays » hérite de « Surface ». Elle contient un booléen *found* déclaré *public* car il est lu et écrit par la classe « PlayingState ». La méthode testSelected fonctionne différemment que celle de la classe « Continent ». Chaque pays est en fait un rectangle, plus précisément un *Sprite*. Ce *Sprite* contient la texture du pays. D'abord, la détection du cas où la souris est contenue dans le rectangle que forme le Sprite. Ensuite, si c'est le cas, l'appel d'une autre fonction, qui détecte si la souris est sur un des pixels de la texture est effectuée.

```
class Pays : public Surface
{
public:
      bool found;

      Pays(std::string name);
      ~Pays();
      bool testSelected(sf::Vector2f& mousePos) override;
      bool testClick(sf::Vector2f& mousePos) override;
      int getDifficulty();
      void loadFromFile(std::string continentName);
      void setSelectedColor();
      void setNotSelectedColor();
      void setDefaultColor();
      void draw(sf::RenderWindow& window);
private:
      short int difficulty;
      bool selected;
      sf::Texture texture;
      sf::Sprite sprite;
      sf::Color defaultColor;
      sf::Color foundColor;
      sf::Color selectedColor;
      sf::Color notSelectedColor;

      bool contains_pixelPrecision(sf::Vector2f mousePos);
};
```

C’est la méthode contains_PixelPrecision qui détecte si la souris est présente sur un des pixels de la texture. Celle-ci prend en paramètre la position de la souris dans les coordonnées de la carte. Ces coordonnées doivent être converties pour qu’elles soient relatives au *Sprite* du pays. Dans la SFML, pour travailler avec les pixels, un objet de type *Image* est utilisé. Une *Image* est donc créée, en chargeant la texture du pays depuis son fichier *.png* . Cette texture étant à l’origine blanche, il suffit juste de tester si les coordonnées de la souris sont sur un pixel blanc. Voici le code de cette fonction :

```
bool Pays::contains_pixelPrecision(sf::Vector2f mousePos)
{
    sf::Image img = texture.copyToImage();
    sf::Vector2u mousePosRelativeToImg =
         sf::Vector2u(mousePos.x-sprite.getPosition().x,
           mousePos.y - sprite.getPosition().y);

    if(img.getPixel(mousePosRelativeToImg.x,
        mousePosRelativeToImg.y) ==
             sf::Color::White) //Si le pixel à la position de
                               la souris est blanc ...
              return true;
    else
              return false;
}
```

Difficulté rencontrées

La principale difficulté rencontrée a été, lors de la nécessité du coloriage d’un pays d’une certaine couleur au survol de la souris. Ne sachant pas comment faire, une aide a été demandée à l’un des enseignants encadrants du projet. L’utilisation d’un algorithme de remplissage a été proposé par ce dernier. Des recherches ont été effectuées pour l’implémenter dans le code du « GeoQuizz ». Après l’avoir fait, celui-ci fonctionna, mais pas comme souhaité ; l’algorithme remplissait bien effectivement le pays lorsqu’il était sélectionné mais il faisait geler lejeu pendant environ une seconde.

En fait, à ce moment du développement, l’utilisation d’une texture pour chaque pays n’était pas encore une option. Chacun d’entre eux était constitué, comme les continents, d’un VertexArray pour créer une forme invisible approximative du pays qui était utilisée pour tester la sélection de la souris. La représentation graphique du pays quant à elle était celle de la texture de la grande carte, il n’y avait pas de texture du pays rajoutée par-dessus la carte. Le coloriage du pays directement sur la texture de la carte avec l’algorithme de remplissage au passage de la souris sur le VertexArray, était donc envisageable. Comme vu un peu plus haut, l’utilisation d’une *Image* pour traiter les pixels est nécessaire. Le problème est qu’à chaque coloriage, donc à chaque mouvement de la souris, il faut remettre à jour la texture de la carte avec cette image. Cette texture étant de grande taille (3958 x 2596 pixels), l’ordinateur prenait trop de temps pour mettre à jour cette texture et faisait donc geler notre jeu une seconde à chaque mouvement de souris ce qui rendait le jeu injouable. Ainsi, la conception du pays en utilisant un *Sprite* avec une texture et en modifiant la fonction *testSelected*, a été décidée. Fort heureusement, le choix de la programmation objet a facilité ce changement, car les autres classes n’ont pas été modifiée.

D’autres problèmes techniques ont été rencontrées, notamment dans l’utilisation d’outils assez complexes, comme les unique_ptr et les lectures et écritures dans les fichiers.

MANUEL D’UTILISATION

Comme déclaré dans la partie " Cahier des charges ", le « GeoQuizz » est un quiz géographique. Le principe est, donc, trouver un pays demandé sur une carte géographique vierge du monde.

Comment ça marche ?

En ouvrant le jeu, le gamer accédera directement au menu principal. Ce dernier dispose d’un bouton PLAY qui permet de lancer le jeu, mais aussi des règles du jeu, ainsi que du logo et comme image de fond, une carte géographique pour rappeler le thème.

Pour commencer à jouer, il appuiera donc, sur le bouton Play, il aura à choisir un continent en cliquant sur ce dernier, puis le niveau de difficulté sachant que chaque " level " possède un temps réparti de dix secondes et sept pays à trouver. Tous les joueurs n’auront pas forcément les mêmes connaissances en géographie.

Ci-dessous, un tableau résumant le choix des niveaux brièvement expliqué dans la partie " Cahier des charges" :

Niveau	Explication	Exemples pays
Facile	Pays les plus réputés par leur surface, leur histoire ou leur emplacement géographique	France (Europe), Brésil (Amérique du Sud), Russie (Asie), Canada (Amérique du Nord), Algérie (Afrique)…
Moyen	Pays moins ordinaires que les pays du niveau " Facile ", moins populaires pour différents critères	Chili (Amérique du Sud), Niger (Afrique), Uruguay (Amérique du Sud), Finlande (Europe) …
Difficile	Pays très peu connus : Pays de petite aire, se trouvant aux bouts du globe terrestre, dans le centre des continents, ou des îles éparpillées dans les océans	Tadjikistan (Asie), Ouganda (Afrique), Kiribati (Océan Pacifique), Mindanao (Asie) …

Après le choix du continent et du niveau, vint le temps de jouer en appuyant sur le bouton GO !

Le continent choisi sera zoomé, ce qui améliorera la vision du joueur sur les pays. Il devra ensuite, trouver l'emplacement du pays demandé, en bas de la fenêtre (vers le Sud). En cliquant sur la touche droite de la souris, un autre zoom sera effectué sur le pays pour faciliter sa détection ; si le pays est le bon, il s'affichera en vert, sinon ne changera pas de couleur. Il disposera d'une barre indiquant le score et le temps écoulé ou restant.

Dans le cas de la fin du délai imparti ou l'attribution d'un pays avant dix secondes, une fenêtre apparaîtra. Elle dévoilera le score du gamer, qui sera calculé selon les pays trouvés ou pas et du temps, accompagné d'un message indiquant sa réussite ou pas !

Enfin, en cliquant sur le bouton rouge TABLEAU DES SCORES avec la souris, il pourra sauvegarder son score, en introduisant un pseudo à l'aide du clavier. Ce tableau (comme donné en exemple dans la partie " Cahier des charges ") contiendra donc le pseudo du joueur, son score et le niveau de difficulté de la partie jouée. Cependant, si le joueur le souhaite, il peut ne pas sauvegarder son score en appuyant sur le bouton NE PAS SAUVEGARDER.

CONCLUSION

La réalisation de ce projet " Développement de jeu vidéo " pour les étudiants de CMI a été très bénéfique et enrichissante pour chacun des membres du groupe, en mêlant découvertes et difficultés. Nombreux sont les savoirs approfondis et les compétences acquises.

Concernant le projet " Développement d'un jeu vidéo " : Au cours de cet exercice, ayant un temps limité et beaucoup de travail et étant habitués à travailler individuellement, nous avons appris l'organisation et la répartition des tâches, la gestion du temps et des priorités et, donc, le travail en groupe.

En effet, le projet final étant à rendre avant le 18 avril 2017 et une soutenance orale étant organisée pour le 28 avril de la même année, la constitution d'une équipe, afin de s'entraider, a été importante et intéressante, et donc : écoute, communication, adaptation, responsabilisation, engagement ... sont des qualités acquises.

Quant à la rédaction de ce rapport : Cette pratique a été instructive et productive. Effectivement, le développement de jeux vidéo n'étant pas notre fort, hormis la programmation, son écriture a confirmé certaines idées et abjuré d'autres.

Un réel travail a été fait concernant la conception du « GeoQuizz ». Il ne fallait surtout pas oublier son rôle premier : favoriser l'apprentissage. Cela n'a pas été un simple calque sur les jeux vidéo du marché.

BIBLIOGRAPHIE

[…][1] https://fr.wikipedia.org/wiki/G%C3%A9ographie
[…][2] https://fr.wikipedia.org/wiki/Quiz
[…][3] https://www.developpez.com/actu/87831/Visual-Studio-2015-permet-de-developper-tout-type-d-application-tour-d-horizon-des-nouveautes-de-l-EDI/