Rapport de projet

14 juin 2006 (revu et corrigé le 5 août 2006)

Table des matières

CHAPITRE 1

Introduction

Un an... déjà...

Un an après notre admission dans cette étrange école qu'est Epita, après avoir découvert un endroit où l'on peut croiser des étudiants dans les locaux à toute heure du jour ou de la nuit, après s'être imprégné de l'ambiance agréable qui règne entre les élèves et les profs, après s'être lancé dans une aventure implacable mais néanmoins terriblement excitante, voici venue... **la fin du projet**.

C'est en général dans ces moments là que l'on trouve que le temps est passé trop vite ! Les meilleurs enseignements étant ceux que l'on se forge par soi-même, revenons maintenant sur le sujet de ce rapport : le projet informatique 2005-2006 de l'**E**PiCHAT **T**eam à Epita, **H**overboard **XT**rem .

Afin d'avoir une vision globale du périple que fut la réalisation de ce projet, il y aura en premier lieu une présentation rapide de celui-ci et de l'équipe, puis un récapitulatif de notre répartition des taches et de l'évolution du projet tels que nous les avions prévus et tels qu'ils l'ont été en réalité. S'ensuit la description précise de notre travail divisée en catégories qui représentent des différentes composants du jeu.

Voilà en somme toutes les informations que vous pourrez trouver dans ce rapport, dont nous espérons qu'il sera aussi clair qu'exhaustif.

Toute l'**E**PiCHAT **T**eam vous souhaite une bonne lecture et vous donne rendez-vous sur http ://www.hxt.fr pour pouvoir télécharger la dernière version de notre travail !

CHAPITRE 2

Présentation

2.1 Hoverboard XTrem > Le projet

Au cas où le lecteur aurait raté les trois premiers épisodes, rappelons les points clés de notre projet.

Tout d'abord, il faut savoir que l'année de Sup à Epita demande à ses étudiants de réaliser un projet libre tout au long de celle-ci, les seules contraintes étant les langages qui sont imposés (Delphi ou Caml) et notre imagination. Comme cette année étant la seule où nous avons le droit de réaliser un jeu, nous nous sommes jetés sur l'occasion.

Dans quel genre de jeu nous sommes nous lancés ? Le titre de celui-ci comporte un indice non négligeable : hoverboard. Les amateurs de cinéma sauront qu'un hoverboard est une planche de skate-board qui a la particularité de flotter à quelques centimètres du sol.
Nous avons ensuite ajouté au titre la particule "Xtrem" (extrême) afin de bien symboliser le fait que notre jeu était bien parti pour décoiffer le joueur ! **H**overboard **XT**rem est donc un jeu de course en 3D orienté arcade pour Windows.

2.2 EPiCHAT Team > Le groupe

L'EPiCHAT Team est composé de deux Sup B2 et de deux Sup C1.

De gauche à droite et de haut en bas nous avons :
- FAHMI Marc aka elfahme - SupB2
- BRIX Jean Rémi aka JRMOLO - SupC1
- BALLABENI Stefano aka Le MiDU - SupC1
- RAUD Cédric aka spycam - SupB2 (chef de projet)

CHAPITRE 3

Organisation

3.1 Répartition des taches

Lors de l'établissement du cahier des charges, en faisant un rapide tour de table nous avions établi une première répartition des taches globales suivant les affinités de chacun. Voici ce que cela avait donné :

	Stefano	Cedric	Marc	Jean-Remi
Moteur graphique			×	×
Moteur physique	×	×		
Moteur du jeu	×		×	
Sons			×	×
Réseau	×	×		
Intelligence Artificielle		×		×
Graphismes			×	×
Site	×	×		

Seulement, au fur et à mesure que l'on découvrait ce en quoi consistait réellement les différentes catégories susnommées, il y eu de légers changements à cette répartition de sorte que l'on se retrouve aujourd'hui avec un tableau ressemblant plus à celui-ci :

	Stefano	Cedric	Marc	Jean-Remi
Moteur graphique			×	
Moteur physique		×		
Moteur du jeu		×	×	
Sons				×
Réseau	×			
Intelligence Artificielle		×		×
Graphismes		×	×	
Site		×		

Ces changements ne doivent pas être considérés comme ayant un effet négatif sur la productivité du groupe mais plutôt comme une restructuration positive dont le but est justement que chacun puisse se consacrer à la tache qui l'intéresse le plus car il ne faut pas perdre à l'esprit que nous faisons le projet plus par passion que par obligation !

3.2 Évolution temporelle

L'évolution temporelle est beaucoup plus importante que la répartition des taches dans le sens où la réglementer un minimum permet de ne pas se laisser dépasser par les évènements. Rien de tel qu'un planning à suivre et des objectifs à respecter pour nous motiver à consacrer du temps au projet. L'avancement des différentes catégories est représenté par un dégradé de gris, le moins avancé étant le plus clair et le terminé le plus foncé.

	Nov. + Déc.	Jan.	Fev.	Mars	Avril	Mai	Juin
Moteur graphique							
Moteur physique							
Moteur du jeu							
Sons							
Réseau							
Intelligence Artificielle							
Graphismes							
Site							

Pour rappel, les soutenances ont eu lieu début janvier, fin février, fin avril et mi-juin.

Il est très difficile de représenter l'avancement actuel du projet par un tableau similaire car le degré d'avancement dépend beaucoup de l'échelle selon laquelle on se repère. Un travail peut à la fois être très avancé ou

insuffisant suivant le point de vue de la personne qui émet le jugement. Mais pour simplifier, nous pouvons tout de même dire que le moteur physique, le moteur de jeu ainsi que le site et l'éditeur de map sont bien plus avancés que nos prévisions, contrairement à l'intelligence artificielle et au réseau. Tout ceci sera expliqué plus en détail dans les parties suivantes.

CHAPITRE 4

Contenu

4.1 Moteur graphique

Voici donc venue la dernière soutenance de l'année de sup et la fin du voyage (ou pas) de notre jeu HoverBoard X-Trem (espérons que sa vie ne vienne seulement que de commencer).

Le Moteur graphique, élément essentiel dans tout jeu vidéo et permettant l'immersion de l'utilisateur dans notre jeu, il est aussi nécessaire qu'il y ait un confort visuel. Les graphismes du jeu, que cela soit dans le menu, dans sa présentation ou dans l'environnement lui même donne une dimension particulière à celui-ci qui le rend unique. Il faut donc attacher beaucoup d'importance à cette partie. Ce n'est peut être pas la partie la plus importante cependant elle donne une identité au jeu qui permet de le différencier aux autres. Le moteur à beaucoup évoluer au cours du temps depuis la toute première soutenance de Janvier. Nous allons donc faire un récapitulatif de tout ce qui à été fait depuis le début de l'année après la formation du groupe.

Un nouveau départ Tout d'abord, l'une des tâches les plus ardues fut le démarrage, l'amorçage : par où commencer, quoi faire et ensuite, que viser. Ainsi notre première tache fut de s'atteler à la compréhension de ce qui allait nous permettre de donner un rendu du jeu lui-même, il s'agit bien de la libraire OpenGL. Les didacticiels fusent sur le net, certains plus ou moins explicites et avec le recul on peut remarquer que cette bibliothèque est très complète et plus ou moins simple d'accès, qu'elle a énormément évolué au cours du temps pour s'adapter aux technologies actuelles, de nouvelle cartes graphiques plus performantes sortent tous les mois. Cependant il est intéres-

sant de noter aussi qu'elle accumule du retard du point de vue des nouvelles extensions et nouvelles possibilités qui sont plus nombreuses, marquantes et simple d'accès sur DirectX. Mais cette librairie reste tout de même accessible facilement, du fait de sa simplicité pour débuter, de ses ressources extrêmement nombreuses notamment sur internet et parce qu'elle est libre.

Petit à petit Ainsi l'histoire du jeu, de notre groupe peu commencé , doucement mais surement. Ainsi on a commencé par savoir ouvrir une fenêtre OpenGL, puis afficher quelque chose dessus, un carré par exemple puis l'animer grâce aux translations, rotations et autres mouvements. Puis il s'agit de donner forme à ces nouvelles connaissances pour les appliquer au jeu. Ainsi un simple HoverBoard qui n'était réellement qu'un cube fut créé et animé par un mouvement d'oscillation et que l'on pouvait déplacer sur une surface plane grâce au touche du clavier. Le jeu commençait doucement à prendre forme. Il était temps de charger des objet modélisés en 3D sur l'écran. Et c'est ainsi que fut sélectionné le formant 3ds, un format binaire et performant. Il n'est cependant pas l'unique format disponible mais reste très répandu. On pouvait maintenant charger un objet 3d que l'on déplaçait dans l'espace.

Gros changement Il ne restait plus donc qu'à donner une certaine cohérence visuelle à tout ceci pour que notre jeu prenne vie. Mais cela ne fut pas des plus simple car beaucoup d'obstacles vinrent entraver notre chemin. Et les bugs ont énormément ralenti le développement du jeu sachant qu'on passait beaucoup de temps a tenter de faire marcher quelque chose correctement qu'à apporter de nouvelles fonctionnalités. D'ailleurs actuellement tout le jeu étant codé en objet il plus simple de séparer les parties, de les travailler séparément et de les réunir plus tard mais de manière beaucoup plus intuitive. Ainsi d'abord la partie graphique et une partie du noyau était en objet or actuellement toutes les parties du jeu son bâties selon différentes classes pour bien différencier chacune des parties composant le jeu. Le moteur physique se développant peu a peu, nous avions besoin d'un réel environnement de jeu et non seulement une surface plane. Et c'est là qu'intervient l'éditeur de map. Un éditeur ma foi très complet et opérationnel (mais pas totalement désolé, mais ne vous inquiétez pas, il y aura peut-être une mise à jour de celui-ci sur le site du jeu).

4.1.1 Bilan

On peut remarquer que le moteur graphique dans sa version actuelle fonctionne très bien, sans trop gros ralentissements sur la plupart des PC testés mais il reste tout de même basique. Nous avions montré lors d'une soutenance précédente certains effet de lumière, brouillards et autres qui ont finalement été abandonné, on a préféré s'attarder plus sur le moteur lui même et un fonctionnement optimale ce qui est actuellement le cas. Le

jeu est plutôt fluide pour l'instant et c'est satisfaisant. On pourrait rajouter que même si le moteur graphique ne gère pas les lumières, celles-ci ont été implémentées et donc prêtes à être utilisé. Cependant comme la gestion des lumière n'est pas des plus simple, elle fut mise de côté. Pour conclure, le moteur graphique est fonctionnel et fluide donc les objectifs sont plus ou moins atteints. Avec simplement quelques effets graphiques supplémentaires, l'objectif aurait été atteint.

4.2 Moteur physique - *Cédric*

4.2.1 Prévisions

Etant donné que le moteur physique est la partie du jeu qui se charge de déplacer tous les objets à l'écran, concevoir un jeu de course sans lui relève de l'absurderie la plus totale. Voilà pourquoi les objectifs prioritaires lors du développement de ce moteur étaient d'avoir un comportement crédible - notament lors des virage-, des collisions avec la map, une sensation de vitesse et une souplesse des contrôles. Cependant, pour arriver à un résultat pareil, il ne fait aucun doute que le chemin à parcourir sera semé d'embûches de toute sortes. Voici donc les différentes étapes par lequel le moteur est passé avant de nous arriver dans sa version finale.

4.2.2 Détection des collisions

Déplacements classiques

La première tache est l'initialisation d'une fenêtre graphique de base avec un objet déplaçable au clavier. Histoire de me familiariser un peu avec GLFW (notre librairie graphique, pour ceux qui lisent le rapport en partant de la fin) je commence par représenter un rectangle en 3D, fier représentant de la nouvelle génération d'Hoverboards. On assiste ainsi à la naissance de ceci :

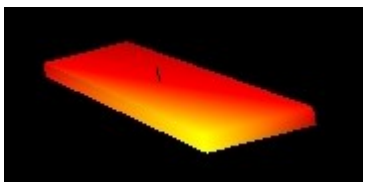


FIG. 4.1 – Hoverboard 1er

Une fois ceci fait, le faire se déplacer ne fut pas trop difficile :

```
if (glfwGetKey( GLFW_KEY_UP )= GLFW_PRESS) then // Déplacement en avant
    begin
     posz:=posz+vitesse;
    end;
```

La condition `glfwGetKey( GLFW_KEY_UP )= GLFW_PRESS` permet de tester si la touche 'Flèche haut' est enfoncée. Si oui, la position en z (n) est augmentée. Note : dans mon programme, l'axe z est l'axe vertical et pointe vers le haut de l'écran. C'est ensuite la fonction '`glTranslatef( posx, 30.0, posz );`' qui se charge de déplacer véritablement l'objet à chaque tour dans la boucle du jeu. Seulement il est vite apparu qu'une simple translation ne pouvait pas suffire pour déplacer la planche. Le schéma suivant explique pourquoi :

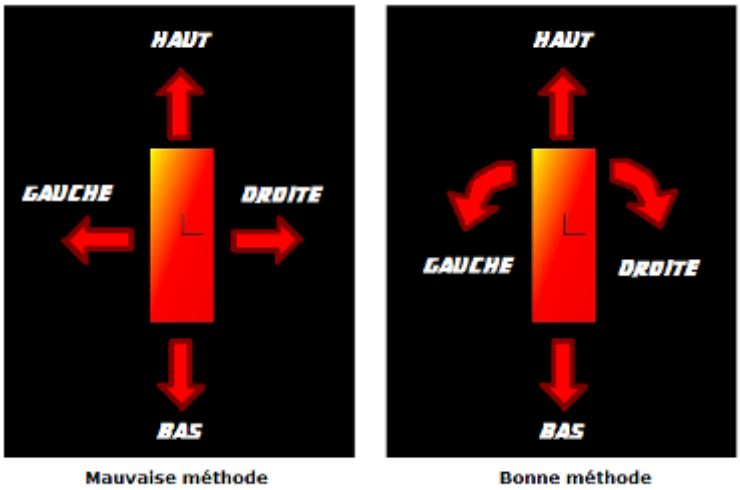


FIG. 4.2 – Schéma des déplacements

Les touches horizontales doivent donc servir à faire tourner la planche et les touches verticales à la faire de déplacer. Pour cela, nos amis cosinus et sinus viennent nous donner un coup de main :

```
posz := posz + speed/fps*cos(DegToRad(angley));
```

Comme vous pouvez le voir, nous multiplions le cosinus à speed/fps. Comme leurs noms l'indiquent, ces variables sont en relation avec la vitesse de la planche et le nombre d'images par secondes (frame per second) de l'application. Le coefficient speed est une constante pour le moment car je ne m'occupe que des collisions. Faire dépendre la vitesse du jeu au nombre de FPS est primordial : cela permet que le jeu tourne à la même vitesse sur différentes machines où le nombre de fps peut varier du simple au dixuple ! A titre d'exemple, cette petite application avec le simple rectangle tournait à 500 fps sur mon portable pour 50 fps sur les PCs de l'école...

La dernière modification nécessaire pour les déplacements est l'inversion de la direction lors de la marche arrière.

4.2.3 Collisions avec les bords de la map

Maintenant que nous avons une planche qui se déplace, l'heure est venue de détecter les bords de notre map virtuelle qui n'est pour l'instant qu'un simple rectangle recouvrant une grande partie de l'écran. Ces fameux bords sont simplement définis pour le moment par quatre variables : minx, maxx, minz et maxz. Il suffirait de comparer les valeurs de posx et posz avec ces valeurs minimales et maximales me diriez vous. Que nenni ! En effet, posx et posz correspondent au centre de ma planche et mon but est de tester si les bords de la planche ne dépassent pas les bords de la map. Analysons la situation... Les points du rectangle de la planche sont stockés dans le tableau suivant :

```
{Points[1][1]
         *---------- Numéro du coin [1...4]
                   | 1 -> Point en bas  à gauche
                   | 2 -> Point en bas  à droite
                   | 3 -> Point en haut à droite
                   | 4 -> Point en haut à gauche
            *------- Coordonnées [1;2]
                   | 1 -> x
                   | 2 -> z}
  Points: array [1..4] of array [1..2] of Double;
```

Il faudrait donc mettre à jour ces coordonnées à chaque translation ou rotation de la planche. Pour cela, nous créons un second tableau
(`Points,MPoints: array [1..4] of array [1..2] of Double;`) que nous

mettrons à jour à l'aide d'une fonction qui se charge de calculer toutes les coordonnées à partir de posx, posz et angley, l'angle de la planche.

Il n'y a plus ensuite qu'à tester si la planche reste bien dans la map à chaque nouvelle position. Dans le cas contraire, elle ne bouge pas sur l'axe où le point est en dehors.

4.2.4 Collisions avec des objets sur la map

Nous attaquons maintenant la grosse partie du moteur physique. En effet, si vérifier si la la planche ne sort pas était simple, vérifier qu'elle n'entre pas en collision avec un objet quelconque l'est déjà moins ! Pour commencer, l'idéal est de mettre en place la détection avec des objets rectangulaires. Ceux-ci sont stockés dans le tableau suivant :

```
{Tableau dynamique listant tous les quadrilatères de la map :
  obj_quad[0][1][1]
           *---------- Numéro de l'objet [0...(Nombre d'objets-1)]
              *------- Numéro du coin [1...4]
                     | 1 -> Point en bas  à gauche
                     | 2 -> Point en bas  à droite
                     | 3 -> Point en haut à droite
                     | 4 -> Point en haut à gauche
                 *---- Coordonnées [1;2]
                     | 1 -> x
                     | 2 -> z }
  obj_quad: array of array[1..4] of array[1..2] of Double;
```

Le tableau `obj_quad` est dynamique ce qui signifie qu'il suffit d'agrandir sa taille pour ajouter un autre élément. Notez que le tableau en lui même n'empêche pas d'enregistrer des objets non rectangulaires mais nous parlerons d'eux un peu plus loin... Qu'est ce que qui définit si un point de la planche est à l'intérieur ou à l'extérieur d'un rectangle ? Il doit respecter les conditions suivantes :

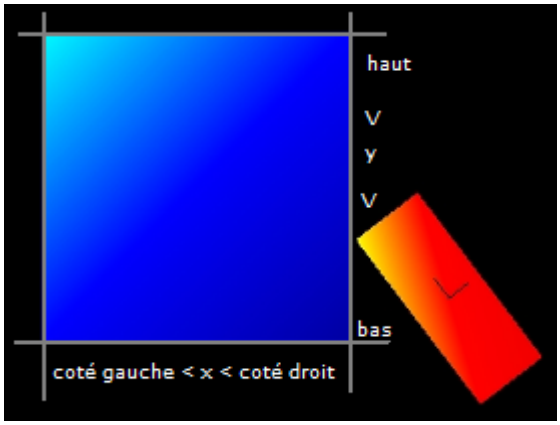


FIG. 4.3 – Collision avec un rectangle

Il ne reste ensuite qu'à effectuer ce test sur tous les points de la planche et sur tous les rectangles.

Mais que se passerait-il si notre quadrilatère n'était pas un rectangle ? Notre méthode n'est alors plus valable car la valeur à ne pas dépasser en x dépend de z et inversement !

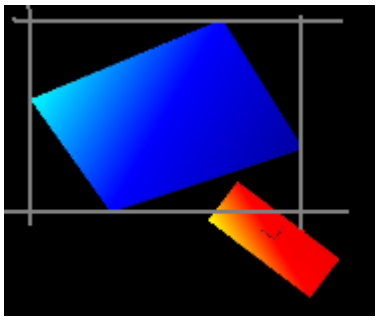


FIG. 4.4 – Collision avec un quadrilatère

Néanmoins, nous pouvons quand même affirmer que si le point n'est pas dans le rectangle qui est formé par les extrémités, il n'y a pas de collisions. Ainsi, inutile de continuer à tester si le point ne remplie pas cette condition.

Pour vérifier si le point est vraiment dans la forme, j'ai opté pour la méthode suivante : je prends les quatre extrémités du quadrilatère deux à deux en tournant dans le sens trigonométrique, et j'étudie la position du point à tester par rapport à la droite formée par mes deux extrémités. Dans notre cas, le point doit ainsi répondre aux conditions suivantes :

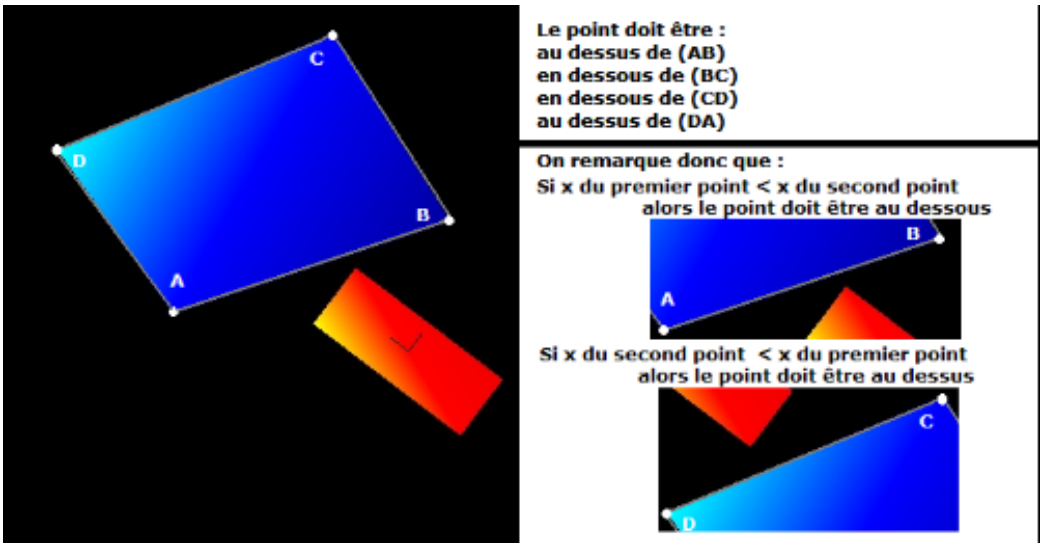


FIG. 4.5 – Collision avec un quadrilatère, explication

Du coté de l'implémentation, la fonction qui s'occupe du test des collisions va ainsi tester tous les points de l'Hoverboard avec toutes les droites formées par les extrémités de tous les quadrilatères. Pour tester si le point est en dessous ou en dessus de la droite, la formule suivante convient à merveille (nous prendrons le A comme premier point, B comme second et X et Y les coordonnées du point à tester) :

$(\frac{PointB_y - PointA_y}{PointB_x - PointA_x} * (X - PointB_x) + PointB_y) >= Y) = (PointA_x < Point_x)$

Une fois cette formule trouvé le reste n'était de l'écriture vu qu'adapter le code pour qu'il puisse aussi gérer des triangle était singulièrement identique à ce que nous avons fait jusqu'à présent. De la même manière, il est nécessaire d'ajouter un second test à celui-ci : tester que toutes les extrémités des objets n'entrent pas en collisions avec l'intérieur de la planche. En somme, le test inverse que le précédent. Sans celui-ci, nous arriverions à des situations comme celle-ci :

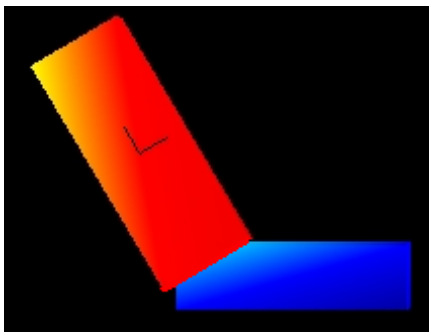
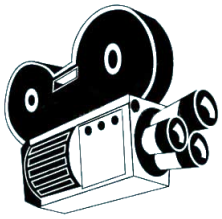


FIG. 4.6 – Collision avec un quadrilatère, le second cas

Après avoir développé un moteur physique capable de détecter une collision avec un quadrilatère ou un triangle quelconque on pouvait se demander quelles seraient les évolutions les plus logique de se moteur. Fallait-il enchaîner directement sur une gestion basique de la 3D ou au contraire améliorer le système existant ? L'arrivé de la programmation orientée objet (POO) au sein de l'équipe a réglé la question : la plupart de ce que nous avions fait pour la soutenance précédente était à refaire ou du moins à adapter grandement. C'est pourquoi la plupart des changements apportés à cette soutenance ne sont pas visible directement : le moteur a été partiellement réécrit et le résultat à l'écran est presque identique, à quelques détails près.

4.2.5 L'objet TCamera

Contrairement à ce que l'on pourrait croire, la caméra d'une scène OpenGL a plus de lien avec la partie physique qu'avec le moteur graphique. En effet, si l'on y regarde de plus près, qu'est-ce qui la caractérise ? La position



de l'oeil, la position du point que l'on regarde et la direction qui indique où est le haut. Après, les valeurs des paramètres précédents ne dépendent que de calculs mathématique qui s'appuient sur les données du moteur physique. Auparavant, la fonction `gluLookAt` qui se charge de positionner la caméra prenait comme paramètres des variables globales où étaient stockées les coordonnées de l'objet à regarder (en l'occurrence la planche). Depuis la POO, il a fallut entièrement revoir la manière dont étaient stockées les informations et les coordonnées en question sont désormais propres à un objet TBoard qui représente la planche que l'on suit. Histoire de pouvoir utiliser plusieurs caméras dans le jeu, j'ai créé un objet TCamera qui contient toutes les informations utiles pour configurer le positionnement de la vue.

L'objet se décompose de la manière suivante :

```
TCamera = Class
  Public
   FCZ, FCRZ, Fps, FCRSpeed, FCSpeed, FProx, FLength: Single;
   procedure Init;
   procedure Reset;
   procedure setFps(nfps: Single);
   procedure RotateLeft;
   procedure RotateRight;
   procedure MoveUp;
   procedure MoveDown;
   procedure Show(Board: TBoard);
  End;
```

Comme on peut le voir, les fonctions possibles avec cet objet sont la remise à zéro de la position de la caméra, la rotation autour de l'objet, le déplacement en hauteur et l'assignement d'un objet à suivre.

4.2.6 Les Objets sur la map

La dernière version du moteur gérait les quadrilatères et les triangle quelconques. Il a donc été nécessaire de stocker ces éléments dans des objets. La structure objet est particulièrement adaptée pour les obstacles de la map. En effet, cela permet de leur donner un par un des paramètres personnalisés

et de rendre la map encore plus variée. Par exemple, nous avons un objet TQuad dans lequel sont stockées ses coordonnées ainsi que les caractéristiques de chacune de ses faces (couleur ou texture). Chaque objet peut donc avoir son propre style : un bâtiment, un arbre, etc... L'objet TMap contient juste la taille de la map ainsi qu'une liste où sont stockés tous les objets de la map.

4.2.7 L'objet TBoard

Le plus important dans le moteur physique c'est l'objet que l'on déplace et qui est la cause de tous les tests de collisions. Etant donné que la partie réseau nécessitera la présence de plus d'un Hoverboard sur le terrain, l'objet TBoard s'est imposé de lui même.
Ses composantes principales sont ses coordonnées en 3D mais également les indices de rotation sur les trois axes.
A partir de là, il a falloir faire un choix sur la forme physique que pourrait prendre l'hoverboard. Vu qu'un hoverboard est plutôt rectangulaire, un tableau où sont stockées les coordonnées des extrémités de la planche fera l'affaire. A cela on ajoute une variable qui indique la hauteur de la planche et une classe permettant de savoir de quelle manière dessiner l'objet (relatif à la partie graphique).
Néanmoins si l'on observe la partie graphique, on découvre que les mouvements de la planche seront provoqués par une translation ou une rotation de la matrice. Cela signifie que le tableau où sont stockées les coordonnées de ses extrémités de la planche sera fixe et ce sera au niveau graphique que les mouvements seront appliqués. Or, il est indispensable pour les calculs physiques de savoir précisément à quel endroit par rapport au repère de la map sont situés chacune des extrémités de la planche. Nous avons donc besoin d'un deuxième tableau qui sera mis à jour à chaque mouvement.

Intéressons nous maintenant à la manière donc les mouvements de la planche seront calculés. A la dernière soutenance, un simple appui sur une touche provoquait un déplacement direct de la planche en modifiant ses coordonnées. Or, dans l'optique d'avoir un moteur physique qui se rapproche le plus proche possible de la réalité, il a fallu revoir entièrement la manière dont cela fonctionnait.
Dans la réalité, tous les objets obéissent à un ensemble de forces dont la résultante détermine s'ils restent au repos ou non. Il restait donc à implémenter une gestion des forces pour la planche. Première question, où ces forces seront elles appliquées ? Dans la mesure où nous faisons un jeu de course arcade et non pas une simulation, il n'était pas nécessaire de pouvoir appliquer une forces sur un point quelconque de la planche. Ici, les extrémités suffisent. Et comme chaque extrémité sera amenée à recevoir plusieurs forces en même temps il nous faut rajouter un tableau dynamique de vecteur à notre objet.

Les fonctions pour ajouter une force à une extrémité de la planche sont toutes simples et ne font qu'ajouter un vecteur de plus dans le tableau. Là, où le challenge est un peu plus relevé, c'est lorsqu'il s'agit de calculer la résultante des forces pour pouvoir faire avancer l'objet.

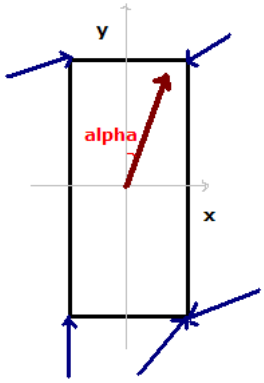


FIG. 4.7 – Exemple de forces appliquées à un objet

La résultante (en rouge sur le schéma) s'obtient en sommant tous les vecteurs entre eux grâce à des calculs mathématiques. Mais comme les mouvements de la planche sont limités à une rotation et une translation il faut extraire de cette résultante les informations nécessaires pour pouvoir les traduire en ces deux mouvements. Ici, c'est les fonctions cos et sin qui viennent -comme toujours- nous porter secours.

Et ça marche !

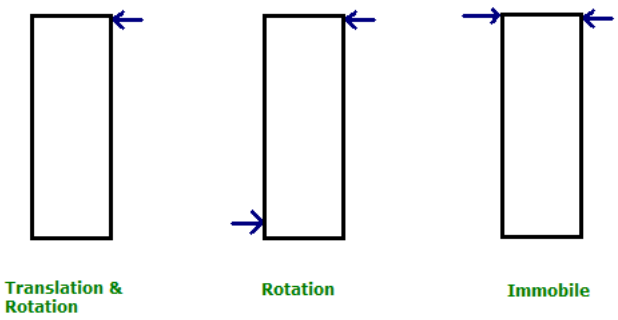


FIG. 4.8 – Représentation des mouvements suivant les forces appliquées

J'en ai profité pour intégrer le système d'accélération de Stéfano de la première soutenance directement dans l'objet ce qui rend les déplacement plus fluides et agréables.

Nous venons de voir que notre moteur physique utilise bien des forces pour le déplacement de la planche mais reste encore incapable de les utiliser pour les collisions. De plus tous les déplacements étaient effectués sur un seul plan, ce qui ne pouvait rester tel quel dans la nouvelle version. Voici donc les deux taches à accomplir pour cette nouvelle mouture du moteur.

4.2.8 Fonctionnement du moteur gérant les forces

Avec les forces, le fonctionnement est simple : plusieurs forces sont ajoutées aux extrémités de la planche au fur et à mesure de l'avancement dans la boucle de jeu, puis, lorsque l'on approche delà fin de la partie physique, une fonction se charge de calculer la nouvelle position de la planche suivant les forces précédemment ajoutées. Jusqu'à maintenant, les forces étaient ajoutées uniquement lorsque l'on appuyait sur une touche. Par conséquent, la planche ne se souciait pas vraiment de la présence d'un objet sur son chemin, et fonçait dessus sans aucune pitié avant de le traverser entièrement de manière théâtrale.

Le but du jeu est donc de déterminer une force qui opposera une résistance à la planche lorsqu'elle essayera de traverser un élément. Cette force est colinéaire à la normale du plan :

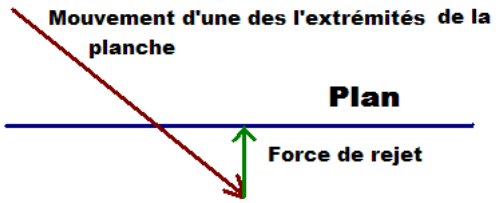


FIG. 4.9 – Schéma représentant la force de rejet par rapport à une droite

Auparavant, je détectais les collisions de la manière suivante : à l'aide des coordonnées des extrémités supérieures d'un objet et des équations de droite, je détectais si un point était à l'intérieur ou à l'extérieur d'un quadrilatère quelconque. S'il était à l'intérieur, j'empêchais la planche de bouger. Cette méthode permet uniquement de détecter si une collision a lieu ou non et ne permet en aucun cas de déterminer la force de rejet que nous avons besoin. Cependant, lorsque j'ai commencé à envisager le passage à la 3D du moteur physique, je pensais tout d'abord porter cette méthode en 3D pour pouvoir tester si un point est à l'intérieur ou non qu'un objet semi-quelconque à 6 faces puis calculer la force de rejet.
Seulement, cette idée avant un gros défaut : comment calculer la force de rejet si l'on ne connaît pas la face qui vient d'être traversée ?

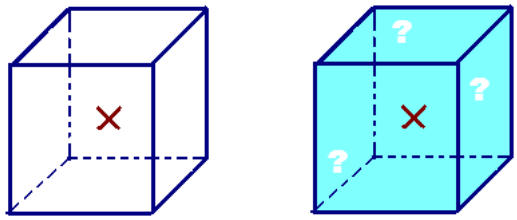


FIG. 4.10 – Quelle face a été traversée ?

Utiliser une autre méthode s'imposait. La méthode pour connaître quelle face était traversée nécessitait de prendre le problème dans l'autre sens : on ne cherche plus si un point est à l'intérieur d'un quadrilatère mais si un segment entre en collision avec un plan. Ainsi, on teste les faces une par une et l'on connaît laquelle est concernée par la collision.
Il est même possible pousser le principe encore plus loin : au lieu de ne faire que détecter la collision, la fonction pourra servir pour renvoyer directement le vecteur de rejet si elle a lieu, le vecteur nul sinon.

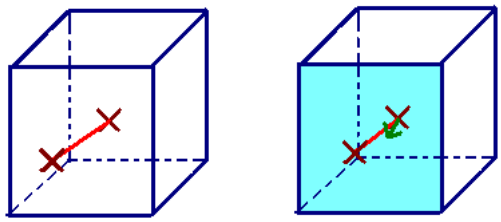


FIG. 4.11 – Nous connaissons directement la face traversée par le segment.

Le plus compliqué reste donc d'écrire cette fameuse fonction.
Sans vouloir rentrer trop dans les détails, celle-ci effectue les opérations suivantes :

- On passe en paramètre de la fonction un polygone à 4 cotés ainsi qu'une ligne
- Elle détermine le vecteur normal au plan du polygone
- Elle calcule les distances respectives des extrémités de la ligne avec le plan (si ces distances sont de même signe, elle renvoie le vecteur nul)
- Elle détermine ensuite les coordonnées de l'intersection de la ligne avec le plan
- Il reste à vérifier que l'intersection a bien lieu dans le cadre du polygone, pour cela, on vérifie que la somme des angle des vecteurs formés par l'intersection et les extrémités est égale à 360°
- Si oui, cela signifie que notre ligne traverse bien la face, reste donc à calculer le vecteur de rejet
- Celui ci correspond au déplacement de l'extrémité la plus proche du plan de la ligne vers le point le plus proche dans le plan de l'extrémité précédente (lire la phrase lentement !)

Il ne reste plus ensuite qu'à ajouter la force renvoyée à l'extrémité de la planche concernée puis de recalculer les coordonnées.

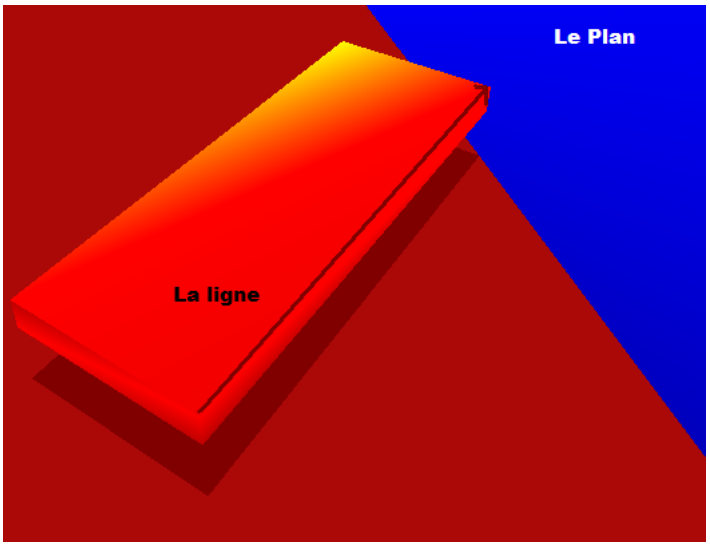


FIG. 4.12 – Exemple concrêt de collision

Concernant l'aspect 3D, cette méthode a un énorme avantage : peu importe la position du plan ou de la ligne, le vecteur renvoyé est en 3D. Il suffit donc d'incliner le plan pour avoir une force de rejet avec une composante en Z non nulle. Cependant, si notre fonction est capable de gérer la 3D, le reste du moteur n'arrivait pas encore à suivre. Il fallu donc transformer certains points en 3D, modifier la fonction qui se chargeait de calculer la force résultante, etc... Afin de simplifier les calculs, la rotation en X (inclinaison de la planche d'avant en arrière) a été écarté pour cette soutenance. Mais la planche se déplace bien en hauteur en respectant les coordonnées renvoyée par notre fonction bien aimée.

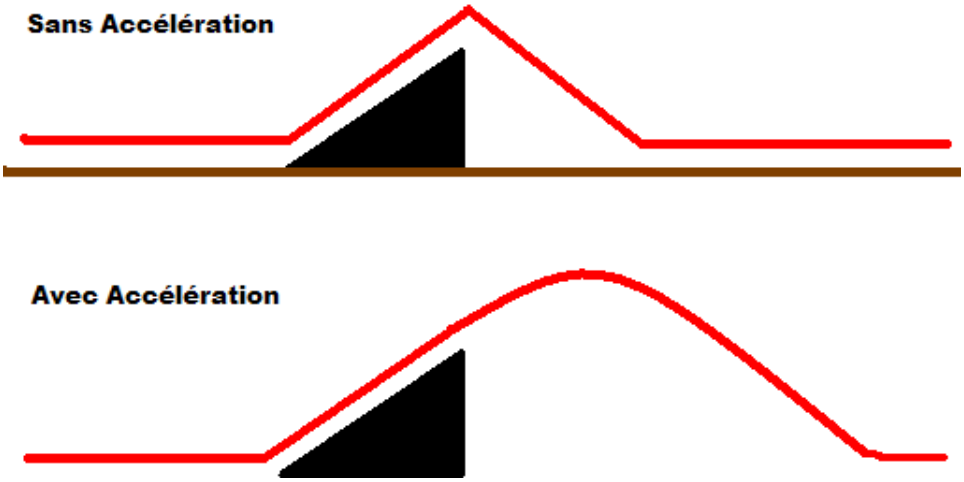
Si les collisions peuvent amener la planche à s'élever, il serait judicieux qu'elle puisse redescendre !
Il a donc fallu implémenter une gestion de la gravité, qui ajoute une force sur la planche dès que celle-ci ne touche plus le sol.

4.2.9 Version finale du moteur

Accélération en 3D

Voici enfin venue la véritable nouveauté physique de cette soutenance finale, le moteur qui gère l'accélération en 3D. En quoi cela est-il utile ? C'est bien simple : avec la version précédente, une collision ou une pression des touches du clavier agissait directement sur les vecteurs vitesses de la planche. Le mouvement était donc immédiat. Hors, dans le monde réel, lors que l'on souhaite faire avancer une voiture, nous n'appuyons pas sur la pédale de vitesse mais sur la pédale d'accélération ! En effet, si l'on enfonce l'accélérateur un instant, que l'on laisse la voiture prendre de la vitesse et que l'on lache brusquement l'accélérateur, la voiture ne va pas s'arrêter net mais plutôt ralentir progressivement. Voilà le genre d'effets réalistes que peut nous apporter l'accélération.

Mais il y a encore plus important : imaginons que nous sautons sur un tremplin situé au milieu d'une route. Le schéma suivant illustre ce qui se passe suivant si l'on gère ou non l'accélération dans le moteur de déplacement :



Comme nous pouvons le voir, la version sans accélération perd beaucoup de son intérêt puisque prend un tremplin n'a pas plus d'effet avec une forte vitesse initiale ou non.

Cette nouvelle version du moteur n'est qu'une sorte de mise à jour de la version précédente. En effet, le système de détection reste identique puisque qu'une collision intervient bien directement sur notre vitesse et non sur notre accélération. Le seul changement à effectuer est de faire en sorte que la pesanteur ainsi que les touches du clavier ajoutent un vecteur d'accélération à la planche au lieu d'un vecteur vitesse. A chaque tour de la boucle de jeu, le nouveau vecteur vitesse est la somme de lui même avec ce vecteur accélération. Pour augmenter la vitesse, il suffit d'une accélération positive et pour la décélérer, d'une accélération négative. La subtilité de l'implémentation réside dans le fait que la situation n'est pas aussi simple que ça étant donné que le vecteur accélération est dans l'espace et qu'il faut définir une vitesse maximale sous peine de voir son hoverboard voler vers l'infini et au delà !

Inclinaison de la planche

Même si ça ne se remarque pas du premier coup d'oeil, notre planche gardait un défaut qui rendait ses déplacements un peu raides : elle ne pivotait pas d'avant vers l'arrière ou l'inverse lorsqu'elle montait une pense ou descendait une descente. Pour cela, la méthode que j'ai adoptée est de récupérer l'angle de l'inclinaison du vecteur vitesse de la planche par rapport au sol et de faire varier progressivement la valeur de l'inclinaison de la planche suivant celui-ci. Ainsi, passer d'une ligne droite à une pente abrupte se déroulera sans accrocs : si l'inclinaison du vecteur vitesse est supérieure à la mienne alors je me penche progressivement vers lui, je reste immobile sinon.

Zones spéciales

Le seul problème qui reste est que l'on peut se déplacer librement sur la map mais que la course ne se termine jamais ! ! Voilà pourquoi il est nécessaire de définir des blocs spéciaux dont la moindre collision avec eux déclenche une action spéciale. Ainsi toutes les faces affichées à l'écran disposent d'un booléen qui m'indique que si je fonce sur cette plaque la course va s'arrêter et m'afficher un message. Parmi les autres types de faces, il existe aussi les faces accélératrices : lors de la collision avec l'une d'entres elles, la planche va recevoir une poussée d'accélération dans une direction prédéfinie. Cette fonction est utile pour les tremplins.

4.2.10 Bilan

Il n'est pas abusif de dire que l'objectif initial est atteint : la planche est très maniable et malgré quelques bugs mineurs, son déplacement est très agréable. De plus, le système de zones spéciales est très modulable et il serait très facile d'ajouter de nouvelles fonctions comme un ralentissement lorsque l'on est sur l'eau ou une mort subite sur les cases de feu. Mais c'est surtout la capacité d'adaptation du moteur qui est appréciables puisque la planche est souvent mise dans des situations inconfortables à cause de l'éditeur de map qui permet de laisser libre cours à son imagination.

4.3 Moteur du jeu - *Cédric*

4.3.1 Le Menu

Par commodité, nous avons décidé d'inclure le menu et l'affichage à l'écran durant la course (HUD) dans la partie moteur du jeu. Bien que complètement absent du cahier des charges, le menu a néanmoins un rôle très important puisqu'il fait part entière de l'exécutable du jeu et que c'est lui que l'on voit en premier lors du lancement. Il se doit d'être le plus explicite possible afin de ne pas frustrer le joueur avant même qu'il n'ait commencé à jouer ! Pour la décoration de celui-ci nous avons choisi des tons très proches de ceux du site,jaunes-orangées. L'astuce de la navigation repose sur un cube tournant dont les faces sont en fait les différents menus du jeu. En transparence apparaît un arrière plan qui se déplace de manière régulière. Cet ensemble créé un effet sympathique et efficace dans la mesure où il donne envie de jouer au jeu.

4.3.2 La structure du jeu

Le moteur du jeu est en quelque sorte la colonne vertébrale de celui-ci : c'est lui qui lie toutes les autres parties. Grâce à la découverte de la POO en milieu d'année, nous nous sommes tous penchés à nouveau sur notre moteur pour revoir le code et nous avons entrepris la tache difficile de redéfinir sur papier les différents objets du jeu, en inscrivant clairement les champs et méthodes de chacun. Ce travail posa les fondations d'une nouvelle version du moteur du jeu, entièrement réécrite à partir de notre schéma. Le but de ceci était de pouvoir unifier les différentes parties tout en permettant de se servir de l'une sans être obligé de modifier l'autre.

Globalement, tout le jeu est contenu dans un seul et unique objet, l'objet 'Game'. Toutes les parties qui seront nécessaires pour avoir quelque chose de jouable sont situées à l'intérieur.
Voici la liste officielle de tous les objets :

- `TGame`
- `TWindow`
- `TGraphic`
- `TPhysics`
- `TInput`
- `TNetwork`

```
-- TClient
-- TServer
- TAi
- TData
-- TCamera
-- TMap
--- TFog
--- TLight
--- TObjet
-- TPlayer
--- TBoard
--- THuman
-- THud
- TSound
```

Le petit nouveau est le TData. Cet objet a, comme son nom l'indique, une vocation de stockage de données. C'est à l'intérieur qu'est situé tout ce qui constitue la map et les éléments du jeu. Le but du TData est d'être une partie mobile et d'être amené à être lu ou modifié par les autres parties. Celles-ci ont donc un accès permanent au contenu du TData par le biais d'un pointeur vers l'instance de l'objet.
Nous retrouvons les objets classiques déjà présents à la soutenance précédente comme le TWindows, le TGraphic, le TPhysics ou le TSound. Le boucle de jeu n'a rien perdu de sa légèreté, et ne contient qu'un appel à la fonction Run() de toutes les parties "actives" du jeu :

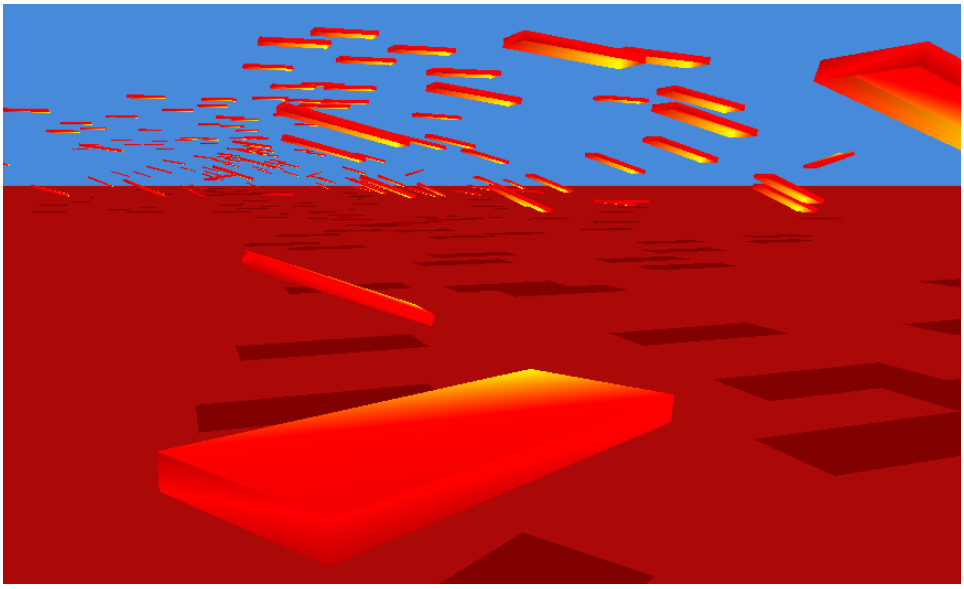
```
// Gestion des entrées
Input.Run;
 // Partie physique
Physics.Run;
// Partie graphique
Graphic.Run;
```

4.3.3 Les avantages de l'objet

L'énorme avantage d'utiliser des objets est de pouvoir manipuler certaines parties du code très facilement. Par exemple, lorsque l'on appuie sur pause, il suffit de ne pas exécuter Physics.Run et le jeu devient soudainement immobile.

Mais les avantages de l'objet sont encore plus flagrants dans avec l'exemple suivant : il suffit de faire une boucle lors de la création des objets "TBoard" pour afficher autant de planches que nous le souhaitons à l'écran. Et le plus fort est que chaque planche s'animera de manière autonome : si on la lache

sur une pente, elle va glisser le long de celle-ci jusqu'à ce qu'ellle arrive tout en bas. Comme la capture d'écran suivante le prouve, afficher 200 planches en simultané à l'écran est un jeu d'enfant !

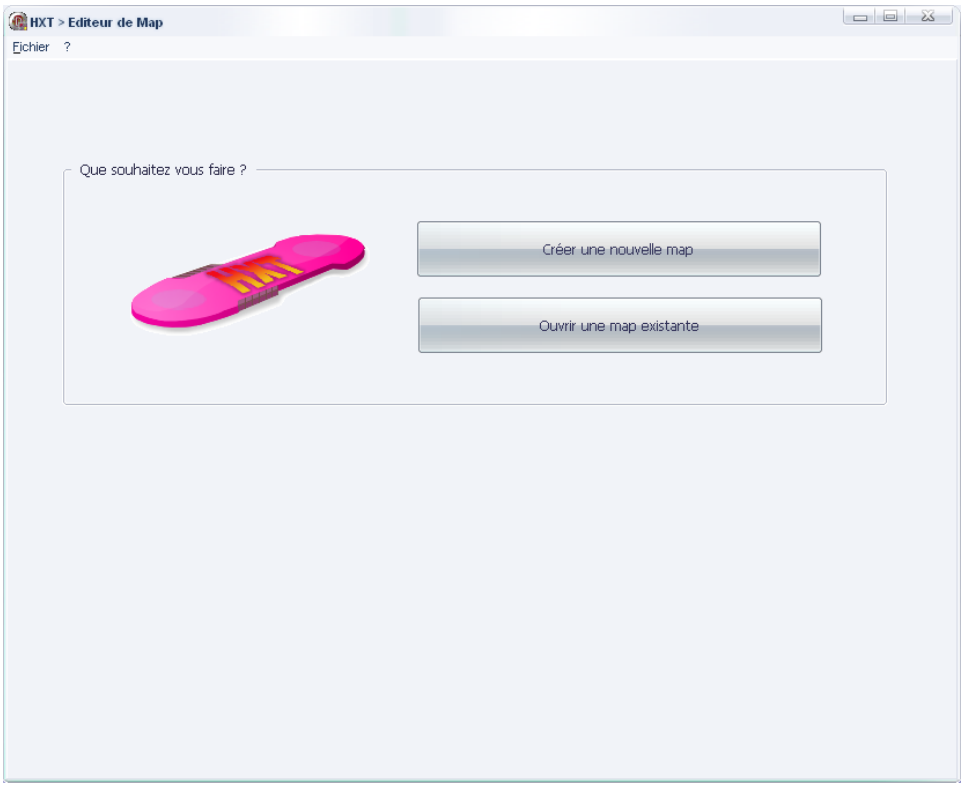


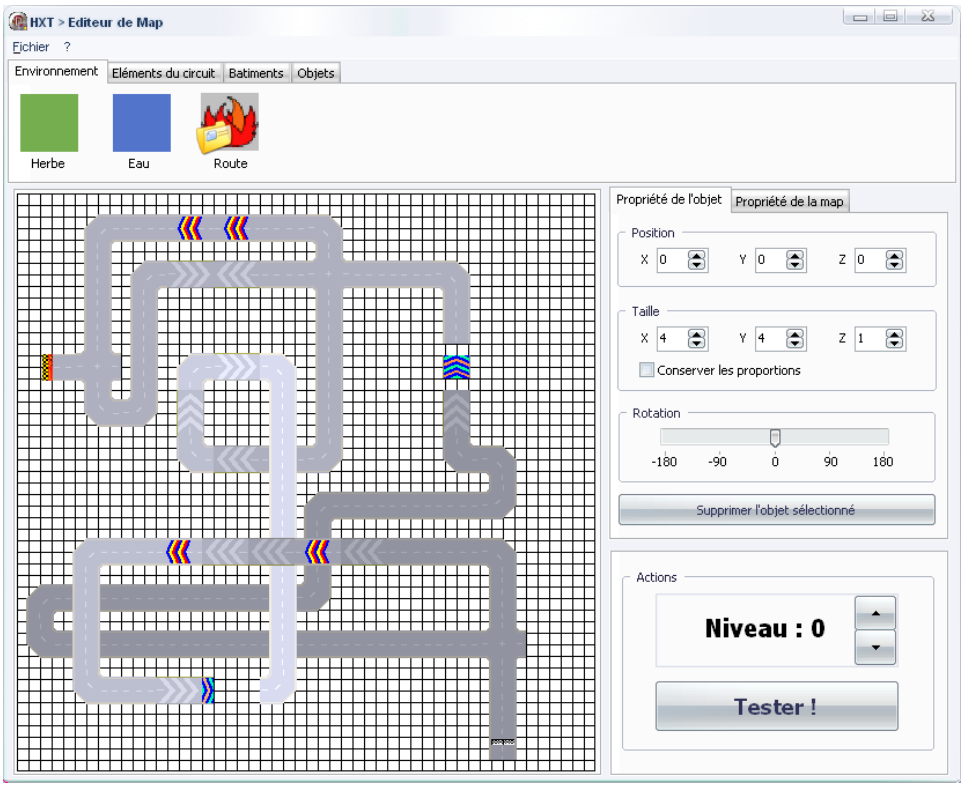
4.4 Éditeur de map - *Marc*

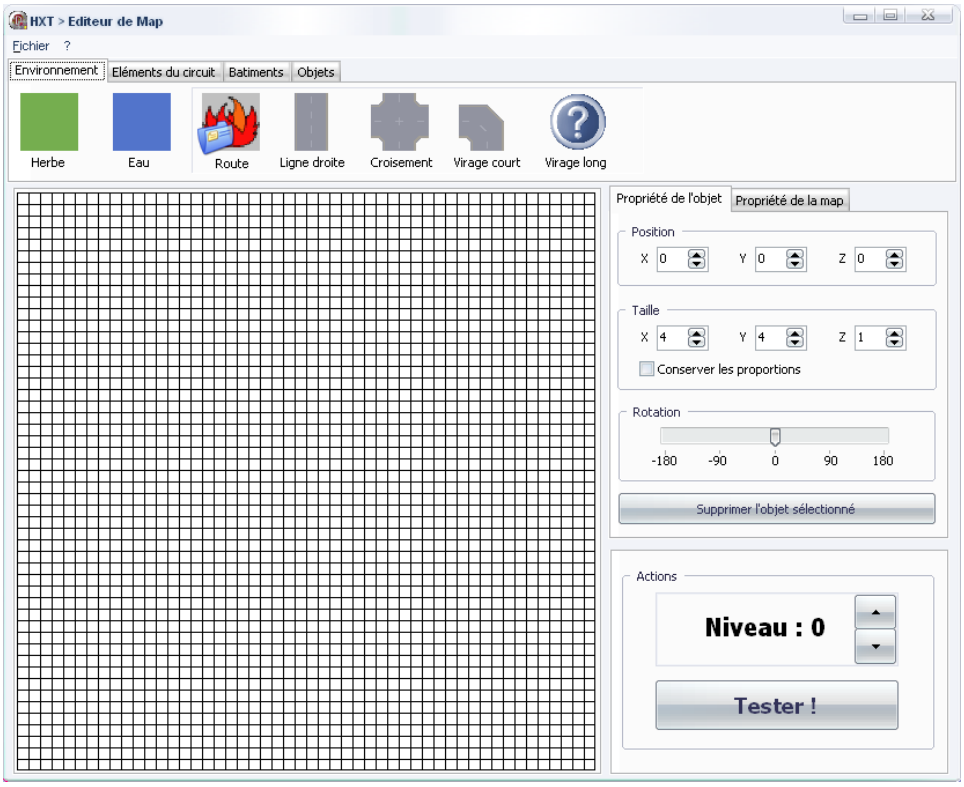
L'éditeur de map devenu très important à partir de la troisième soutenance, celui-ci, nous permet de créer des maps à partir de l'éditeur(je vous crois pas) puis de les charger dans le jeu pour y jouer. Nous avons tenté de faire un éditeur muni d'une interface conviviale, simple d'accès et utilisable immédiatement. Mais il était tout de même primordial qu'il contienne tous les éléments dont nous avons besoin pour le jeu en lui même. Lors de l'ouverture nous avons la possibilité soit d'ouvrir un fichier map '.hxt' existant soit de créer une nouvelle map à partir de l'éditeur. Jusque là rien de bien féroce.

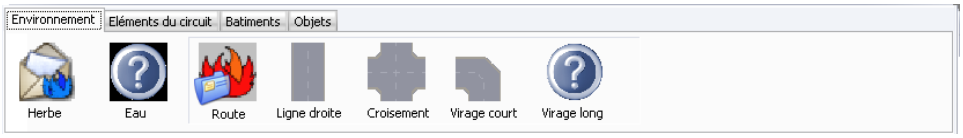
Si l'on clique sur Ouvrir une map existante une nouvelle fenêtre s'ouvre nous permettant de sélectionner le fichier à ouvrir. Seuls les fichier maps '.hxt' appararaîtront dans cette fenêtre. On parcourt donc les dossiers jusqu'à trouver le fichier que l'on souhaite ouvrir et on le sélectionne (ou double clique). L'éditeur se charge automatiquement de lire le fichier et après quelque instant la map apparaîtra dans l'éditeur, il est donc possible maintenant de la modifier et cela à tout instant.

Si l'on clique sur créer une nouvelle map, un cadrillage apparaîtra sur la zone de contrôle de l'éditeur. Ceci est la zone de travail, c'est ici que tout se


HXT > Editeur de Map
Fichier ?
Que souhaitez vous faire ?
Créer une nouvelle map
Ouvrir une map existante


HXT > Editeur de Map
Fichier ?
Environnement
Eléments du circuit
Batiments
Objets
Herbe
Eau
Route
Propriété de l'objet
Propriété de la map
Position
X 0
Y 0
Z 0
Taille
X 4
Y 4
Z 1
Conserver les proportions
Rotation
-180 -90 0 90 180
Supprimer l'objet sélectionné
Actions
Niveau : 0
Tester !

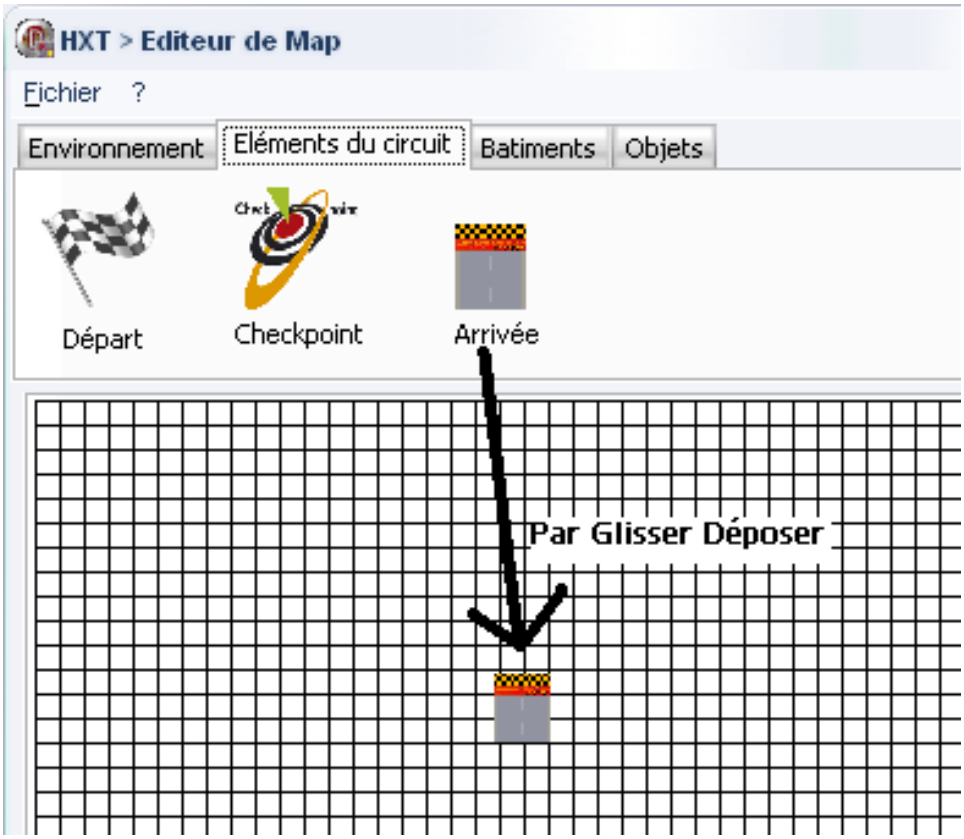




passe. Toute modification apparaîtra sur ce cadrillage.

Si l'on regarde plus haut, on peut remarquer différents onglets. Ces onglets contiennent les différents éléments que l'on peut placer sur la map, il suffit de naviguer entre ces différents onglets et l'on peut remarquer que cette éditeur contient les éléments de base pour un jeu de course et certains éléments supplémentaires propres à celui-ci (peut-être d'autres éléments plus tard). Il y a les éléments environnement permettant de placer le circuit sur la map, l'onglet "<éléments du circuit"> permettant de placer des points important sur la map puis des modules additionnels pour rendre la map plus interactive.

Pour ajouter un élément sur la map, cela reste très simple, nous avions dit que nous souhaitions un éditeur simple d'accès mais cependant complet.



Il suffit donc de placer la souris sur l'élément que l'on souhaite ajouter sur la map, de rester cliqué puis de le déposer la où on le souhaite (un cliquer/déposer pour les plus intimes).

L'élément que vous venez de placer se trouve automatiquement sélectionné (il est entouré d'un rectangle rouge).

Grâce au panneau de droite , on peut accéder aux paramètres de l'élément sélectionné. Ces paramètres sont quel que soit l'axe (cad X, Y ou Z) sa taille, sa position et un angle de rotation(inutile pour les bâtiments). Cet élément peut aussi être supprimé grâce au bouton supprimer visible en dessous des paramètres. Sachant qu'une map peut s'étaler sur plusieurs niveaux, un bouton niveau est présent et permet de se placer directement au niveau souhaité, facilitant la navigation dans l'éditeur.

L'éditeur de map fonctionne grâce à la manipulation de canvas, copie d'un canvas dans un autre en partie ou dans sa totalité. La sauvegarde des données se fait dans un type record qui contient les informations dont on a besoin pour

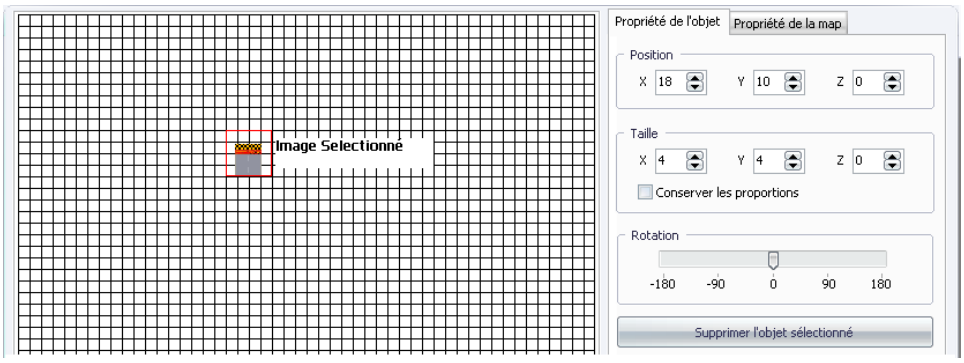


Image Selectionné
Propriété de l'objet
Propriété de la map
Position
X 18
Y 10
Z 0
Taille
X 4
Y 4
Z 0
Conserver les proportions
Rotation
-180
-90
0
90
180
Supprimer l'objet sélectionné


Actions
Niveau : 0
Tester !

un élément. Donc notre "<form"> contient un tableau de ce type record. Ce tableau n'a pas une taille fixe (donc dynamique) donc cette taille varie fonction des besoin et de la place nécessaire. La manipulation de canvas est tout de même gourmande en mémoire et nécessite un certain nombre de calculs qui lorsque le nombre d'éléments d'image devient trop important, le programme se voit ralentit. Il faudrait donc optimiser cela.

4.4.1 Prévisions

l'éditeur de map s'avère être plutôt une réussite car n'étant pas du tout prévu dès le départ, il nous facilite grandement la vie pour la création de circuit. Il reste effectivement des bugs qui traînent par-ci par-la mais existe-il vraiment des programmes qui n'ont aucun bug ?

4.5 Intelligence Artificielle - *Cédric & Jean Remi*

4.5.1 Prévisions

Il était prévu que l'on puisse jouer totalement seul contre l'ordinateur mais cette partie ne sera pas achevée. La raison à cela vient du fait que notre format de map est bien trop ouvert pour laisser à l'ordinateur la possibilité de calculer le meilleur chemin. Si l'on avait choisi une map sous forme de matrice, il nous aurait suffit d'implémenter un petit A* et le problème aurait été réglé. Or, étant donné que la solution de l'A* est écartée, il ne nous restait rien pour que l'ordinateur puisse se déplacer sur la map hormis la possibilité que nous définissions une liste de points dans l'espace que l'ordinateur n'aurait qu'à suivre pour aller du début à la fin de la map. Comme ceci aurait été très ennuyeux à la longue, nous avons préféré nous concentrer sur d'autres parties plus utiles du projet.

4.6 Son

4.6.1 FMOD, librairie sonore - *Jean Remi*

La partie sonore m'a beaucoup intéressé, elle est plus complexe que prévu, c'est en fait sur cette partie que j'ai passé le plus de temps. Sans grande surprise j'ai choisi d'utiliser la librairie FMOD, une API gratuite et qui gère de manière assez complète tout ce qui touche l'environnement sonore d'un jeu. La musique et les bruitages du jeu sont donc implémentés grâce à FMOD. Cette librairie nous offre une palette d'outils simples et performants à la gestion des effets sonores. Elle supporte entre autre les formats mp3, wav et midi.

Après une longue recherche, j'ai trouvé comment déclarer les types Tsample nécessaire au lancement d'un son.

```
FSOUND_Init(44100, 32, 0) ;
fond_sonore:=FSOUND_Stream_Open('roky.mp3',FSOUND_NORMAL,0,0);
FSOUND_Stream_Play(1,fond_sonore);
```

J'ai maintenant pour objectif de charger plusieurs bruitages, avec plusieurs musiques selon les niveaux du jeu, dès que notre personnage touchera un objet il y aura un bruitage qui se déclenchera.

4.6.2 Musique et bruitages

Le jeu devait contenir un environnement sonore afin que l'on puisse accompagner la glisse de notre hoverboard d'effets audio et mettre en fond une musique.

De l'initialisation à la lecture

L'initialisation se fait tout simplement à l'aide de la procédure FSOUND_Init. Celle ci prend compte le nombre de plages existant dans le projet.

La musique et les bruitages sont deux effets sonores bien distincts. La structure FSOUND_STREAM est utilisée pour la musique et la structure FSOUN0D_SAMPLE pour les bruitages. Chacune de ces structures possède ses propres routines. Il suffit d'une seule procédure pour charger une musique, que l'on peut lancer en boucle.
Toute musique et tous bruitages chargés sont libérés de la mémoire à la fermeture de jeu.

Une musique est lue avec FSOUND_Stream_Play. Un bruitage est lu avec FSOUND_PlaySound. Lorsque une musique se termine on en relance une autre.

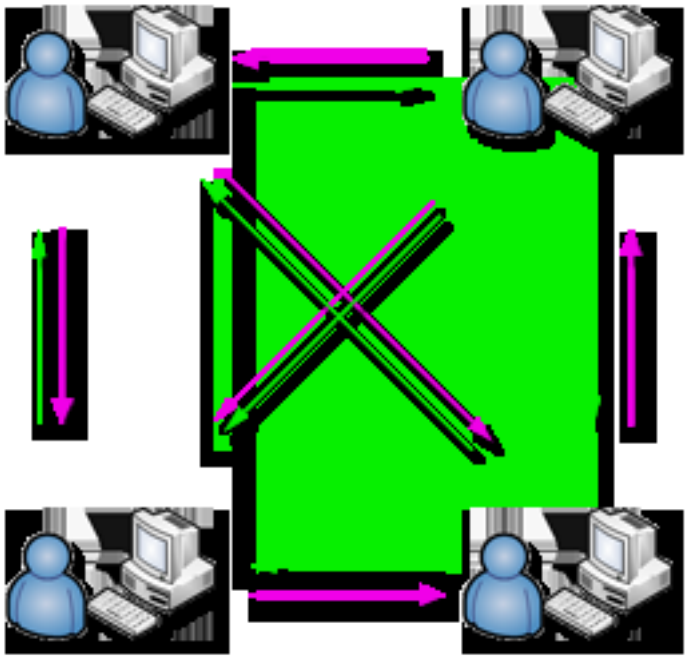
Chaque musique a pour but de créer une ambiance associée au jeu. C'est pourquoi nous avons cherché à savoir quelles sensations nous voulions transmettre au joueur durant une partie de manière à ce qu'elle corresponde le mieux à l'esprit HOVERBOARD Xtrem.

4.7 Réseau et multijoueur

Les parties physiques et graphiques d'un jeu sont importantes pour le rendu à l'écran, mais sans gameplay un jeu n'est rien, et une des manières

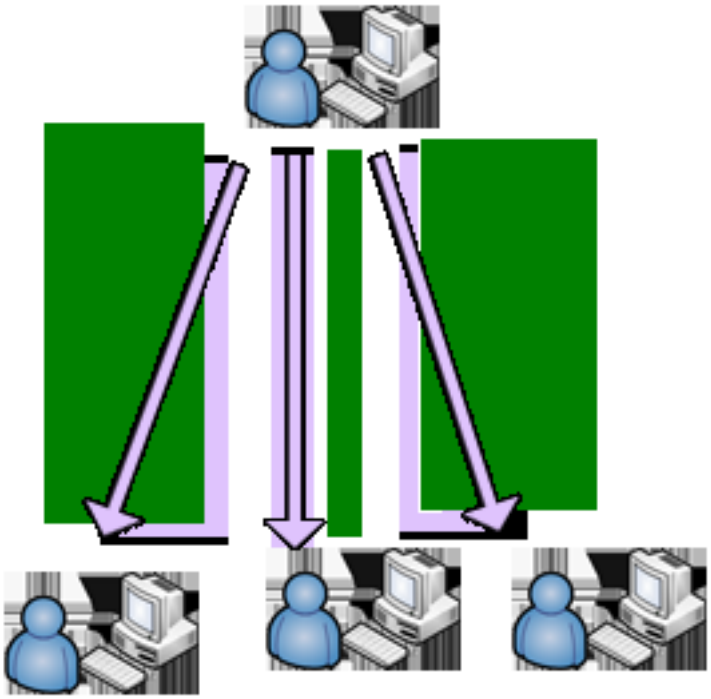
d'ajouter un plaisir de jeu considérable et non négligeable, est l'implémentation de la possibilité de jouer à plusieurs sur la même partie, et de préférence chacun sur son écran.

Dans l'optique d'une implémentation de mode multijoueur sur **H**overboard **XT**rem , je me suis tourné vers le composant ICS qui est reconnu pour être au moins aussi simple à utiliser que mal documenté.
Après avoir étudié les différentes possibilités de gestion des positions des joueurs sur la carte, j'en suis venu à n'avoir plus que deux possibilités.
Ma première possibilité était de faire un réseau dans lequel chaque joueur communique avec tous les autres joueurs de la partie, leur transmettant ainsi ses coordonnées, et eux faisant la même, il était simple d'afficher sur chaque ordinateur la position exacte des autres joueurs.



Cette structure nous permet en effet d'avoir quelque chose de simple, mais le rôle du "serveur" devient nettement plus faible (vu que toutes les machines ont le même 'rang' dans cette configuration) et contrôler la partie est difficile vu qu'on n'a aucun moyen de savoir, de modifier ou d'empêcher les transmissions entre deux machines.

C'est donc pour cela que s'est imposé une autre façon de voir les choses, il s'agit de "centraliser" le pouvoir à une machine qui serait cette fois-ci un vrai serveur qui aurait le contrôle sur la possibilité des machines à envoyer des informations à une autre.



Le principe est cette fois, que chaque client (en bas du schéma) envoie au serveur sa position dans la carte, et que le serveur de son côté envoie à tous les autres clients les positions de toutes les planches.
L'inconvénient est que s'il y a beaucoup de joueurs sur le réseau, le paquet envoyé par le serveur est excessivement gros et peut ralentir l'ensemble des applications.
Ce sera malgré tout ce type de réseau que nous allons utiliser, et nous limiterons à 8 le nombre possible de joueurs par serveur.

4.7.1 Les threads

Pour écouteur les émissions de données depuis un serveur, ainsi que pour en envoyer, on utilise (en tout cas avec ICS), des sockets. Au moment de l'écoute, les sockets entrent dans une procédure appelée 'MessageLoop', et qui n'est autre qu'une boucle while, qui tournera pendant une période imprécise.
Laisser cela tel-quel serait un gros problème pour le jeu, vu qu'une fois que l'écoute serait lancée, personne ne pourrait effectuer ses calculs (le programme serait bloqué à cause de la boucle while). C'est donc pourquoi il a fallut utiliser la programmation en Threads.

Les threads sont en quelque sorte des processus d'exécution de tâches. Mais l'avantage est que plusieurs threads peuvent tourner pendant l'exécution d'un même programme.

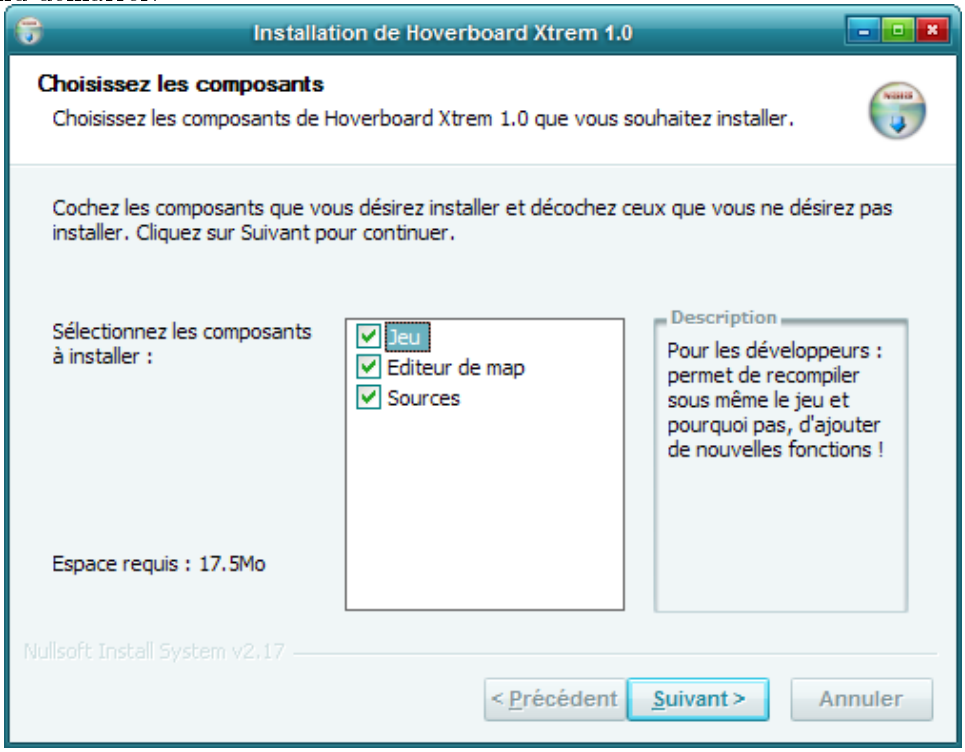
Ainsi, en faisant tourner notre boucle d'écoute dans un thread, elle ne pose aucun problème à l'exécution du reste du jeu.
La programmation en thread aura été indispensable à la mise en place du réseau.

4.7.2 Prévisions

L'objectif était donc de mettre en place un système de jeu en réseau qui permettrait de prolonger l'expérience de jeu sur le net, ainsi que d'y découvrir les maps faites par les fans du monde entier.
Le principal problème rencontré par rapport au réseau, c'est le retard avec lequel on s'y est pris. Et même si l'envoi de données d'un serveur à plusieurs clients et inversement fonctionnait, il a été impossible de l'implémenter dans le jeux avant la soutenance finale.
Le réseau aura donc été un de points du cahier des charges auquel on n'aura pas pu se tenir.

4.8 Installation et désinstallation

Parmi les divers créateurs d'installateurs disponible sur internet, celui qui a retenu notre attention est NSIS, de par son coté open-source mais aussi par son haut niveau de personnalisation.
Comme indiqué dans le manuel fourni avec le boîtier du jeu, il est possible de choisir d'installer ou non des éléments comme l'éditeur de map ou le code source. Comme prévu, il sera possible de désinstaller le jeu par le biais du Lien vers la procédure de désinstallation accessible directement depuis le menu démarrer.



4.9 Site web

4.9.1 Introduction

Dans la mesure où nous devons conduire un projet informatique du début à la fin, notre démarche est similaire à celles que nous serons amenés à poursuivre dans la vie professionnelle. La principale différence est qu'au lieu de suivre celle-ci à des simples fins éducatives, le principal objectif des développements futurs sera d'avoir un produit fini qui tienne la route commercialement parlant. Or, même si l'on sait qu'on a le meilleur produit du monde, il est de renommée publique que la bonne foi du vendeur ne suffit pas pour faire vendre ! C'est pour cela qu'il ne faut surtout pas négliger l'aspect marketing du projet afin d'éviter que le produit fini soit mis sur le marché dans l'ignorance générale des acheteurs potentiels.

Voilà pourquoi j'ai tenu à mettre l'accent sur ce qui pourrait être considéré comme négligeable et secondaire face au reste du projet : le site web. En effet, avec la forte importance d'internet dans la société actuelle, avoir un site web pour présenter son produit est devenu un passage obligatoire dans n'importe quel plan commercial. Dans un processus de "fabrication" traditionnel, l'aspect développement et marketing sont pris en charge par deux équipes différentes pour la simple et bonne raison que les compétences requises pour programmer un jeu et trouver un slogan accrocheur ne sont pas les mêmes.
Cependant, même si le travail d'un développeur n'est pas de faire du marketing et que la vocation initiale de notre site web est juste de présenter le projet tout en offrant la possibilité de le télécharger, j'ai tenu à avoir une approche plus générale du site et à lui donner toute l'importance qu'il mérite vis à vis du jeu. Il a donc été conçu dans une optique d'attirer l'oeil du visiteur, de le pousser à en savoir plus sur HOVERBOARD Xtrem, le jeu.

4.9.2 L'aspect technique

Avant de se lancer tête baissée dans le code, il convient de choisir auparavant judicieusement les technologies à employer pour qu'il puisse correspondre à nos attentes.
Les objectifs du site sont les suivants :

- Fournir au visiteur le maximum d'informations sur le projet :
  Présentation du jeu et de l'équipe
  Montrer l'avancement du développement au jour le jour
  Proposer en téléchargement le cahier des charges, les divers rapports, les exécutables du jeu mais aussi son code source
  Afficher un ensemble de liens vers les sites proposant des ressources (images, sons, librairies, etc...) que nous aurions pu utiliser

- Rendre le jeu le plus attrayant possible
- Donner envie au visiteur de revenir en lui offrant une navigation confortable et visuellement riche

Au regard de cette liste, nous pouvons d'ors-et-déjà commencer à choisir les outils les plus adaptés. Comme nous souhaitons pouvoir mettre régulièrement à jour le site afin de pouvoir fournir à nos fans assoiffés les dernières informations concernant l'avancement du projet, la solution la plus répandue (et surtout gratuite) est celle du langage PHP s'appuyant sur une base de données MySQL.

Cela aurait pu suffire mais il aurait été difficile de remplir le dernier objectif avec une simple page statique. Dans ce cas, la technologie Flash pourrait être une solution puisqu'elle permet d'afficher des animations et des éléments graphiques et sonore avec une aisance incomparable. Seulement, pour créer une animation flash, il faut passer par les logiciels propriétaires et la plupart du temps hors de prix.
Tenant à rester dans le domaine **légal**, je me suis donc tourné vers la technologie dont la popularité ne cesse d'augmenter en ce moment : l'AJAX !

Qu'est ce que cet étrange animal ? Tout simplement un raccourci bien pratique pour parler à la fois de "HTML (ou XHTML) et CSS pour la présentation des informations, DOM et JavaScript pour afficher et interagir dynamiquement avec l'information présentée et enfin XML et l'objet XMLHttpRequest pour échanger et manipuler les données avec le serveur web". Tout un programme ! Derrière ces grands mots se cache la possibilité de mettre à jour l'affichage d'une page sans avoir à la recharger depuis le serveur. Les éléments de celle-ci comme les images ou le texte ainsi que les propriétés respectives (position, taille, etc...) peuvent être mis à jour à tout moment via le langage proposé avec tous les navigateurs internet : Javascript. Enfin, avec le joli nom de domaine facile à retenir (**www.hxt.fr**) obtenu grâce à Stefano, nous avons tous les outils en main pour proposer à notre futur fan un agréable voyage...

4.9.3 Visite page par page

Lorsque que le visiteur arrive sur le site il se retrouve face à la page d'accueil. C'est donc elle qui va faire office "d'accroche" -tout comme le slogan de Une des journaux- et qui va devoir être la plus intéressante possible pour lui donner envie de rester surfer plutôt qu'aller faire du parapente dans les Pyrénnées. Son importance est donc capitale !
Pour cela, j'ai opté pour un design "en blocs". C'est à dire que tous les éléments se trouvent dans un cadre afin de bien délimiter les zones d'informations à des fins de lisibilité. La charte graphique se veut chaleureuse au possible, c'est pourquoi uniquement des couleurs chaudes dans des tons rouge/orangé (rappelant le feu, la vitesse, l'adrénaline qui caractérise le jeu tel qu'il est dans nos rêves) ont été sélectionnées.
Au niveau du contenu, cette page fait office de portail pour le reste du site : elle n'est constituée que de gros boutons -s'inspirant du style des menus de jeux vidéos- pointant vers les catégories les plus importantes : les news, la présentation du projet et la page téléchargement.
Voici une capture d'écran de notre petit bijou :

FIG. 4.13 – www.hxt.fr - La page d'accueil

Le premier effet graphique à base de Javascript concerne la planche : celle-ci oscille lentement, comme si elle était véritablement soumis à la force de ses propulseurs. Une simple fonction cosinus et un timer suffisent pour y arriver.

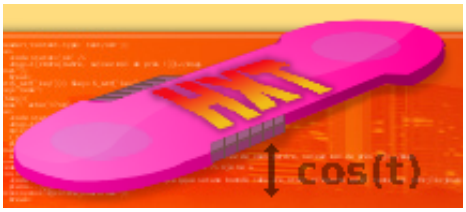


FIG. 4.14 – www.hxt.fr - La planche

Lors du premier clic sur un des boutons du menu ou de la page, le visiteur a alors la surprise de constater que tout s'enchaîne sans entraîner un seul rafraîchissement total de la page : cliquer sur un bouton amène le panneau principal à se fermer avec une animation, l'onglet de la nouvelle page se met alors en surbrillance dans le menu et le panneau s'ouvre de nouveau avec un contenu correspondant à la catégorie sélectionnée. Voici ce que cela peut donner :

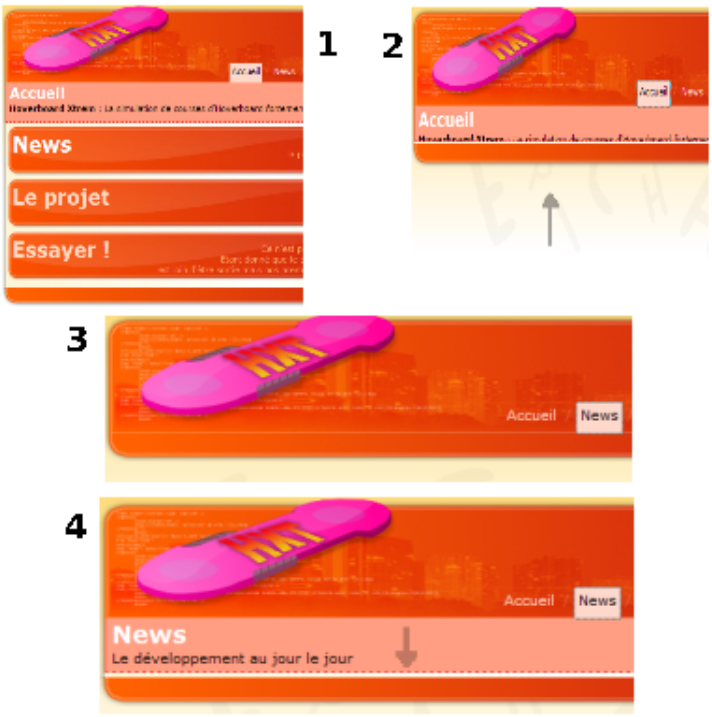


FIG. 4.15 – www.hxt.fr - Animation de transition

Nous arrivons sur la page des news qui est la seule page entièrement dynamique dans la version du site mis en ligne pour la soutenance. C'est ici que seront postées toutes les informations sur l'avancement du projet. Afin de pouvoir interagir avec l'utilisateur, un système de commentaires a été mis en place. La technologie Ajax prend ici tout son sens : alors qu'auparavant il fallait recharcher la page chaque fois que l'on voulait voir les derniers commentaire ou en envoyer un nouveau, notre version permet de réaliser tout ceci de manière la plus fluide possible : un panneau s'affiche par dessus le site, le masquant partiellement avec un effet de transparence et la liste des commentaire est récupérée automatiquement depuis le serveur. Si l'utilisateur souhaite en ajouter un nouveau, il suffit qu'il utilise formulaire prévu à cet effet :

De la même manière, il est possible pour l'administrateur de modifier ou supprimer très facilement un commentaire en cliquant sur les liens indiquées sur la capture suivante :

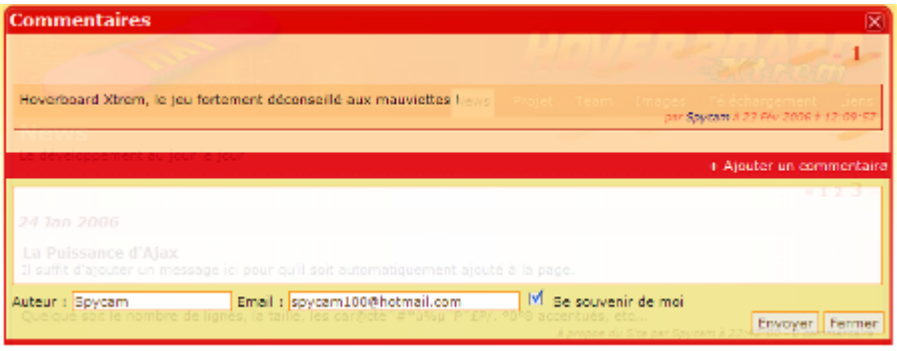


FIG. 4.16 – www.hxt.fr - Boite d'ajout d'un commentaire

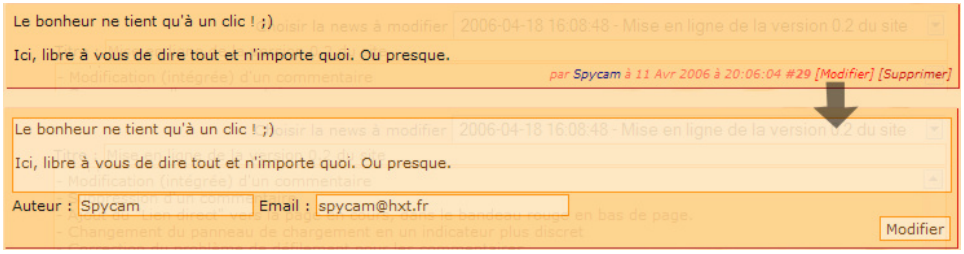


FIG. 4.17 – www.hxt.fr - Modification d'un commentaire

La page suivante est la page de présentation du projet, où se trouve toutes les informations utiles pour celui qui ne connaîtrait pas du tout en consiste le jeu. Ensuite vient la page de présentation de l'équipe, qui explique pourquoi il est politiquement nécessaire à l'équilibre du monde de vouer un culte à l'EPiCHAT. La page Images regroupe quelques captures d'écran de nos travaux précédent, le tout classé par dossiers de la même manière que la page téléchargement.

Enfin vient la catégorie liens où sont regroupées les sites proposant des renseignement utiles, description à l'appui. La dernière version du site permet dorénavant d'en ajouter aussi facilement qu'un commentaire et grâce à l'ajax une sorte d'auto-remplissage des champs lors de la saisie d'un lien est effectuée.

De plus la version 0.2 apporte un compteur de temps avant les soutenances en page d'accueil et la v0.3 dispose d'une partie statique (pour les anciens navigateurs et le référencement par les moteurs de recherche).

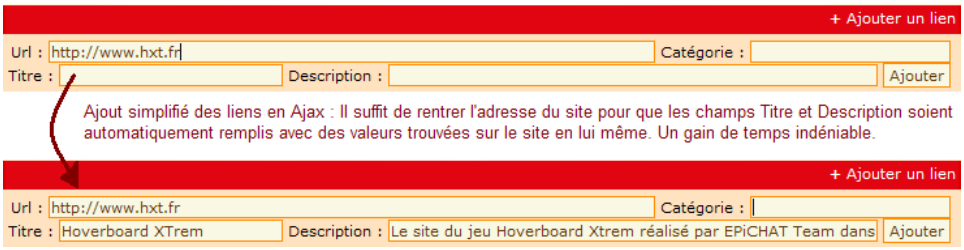


FIG. 4.18 – www.hxt.fr - Auto remplissage des champs

CHAPITRE 5

Bilan

Et voilà, la première aventure épitéenne des quatre compagnons de l'EPiCHAT TEAM touche touche à sa fin, le projet a été tant bien que mal mené à son terme dans la joie et la bonne humeur.
Un début intéressant et riche en expérience pour la suite de notre cursus, et qui ne laisse présager que du bon à l'avenir.
Comme feu Epichat, chacun de ses membres vous salue et vous tire sa révérence, mais uniquement pour revenir meilleurs que jamais dans de nouveaux groupes et projets dès l'année prochaine !