PROJET ISN – JEU DU TYPE TOWER DEFENSE
(Robots vs Virus)

En ce début d'année, lorsque nous avons cherché les objectifs que nous voulions atteindre, nous avons posé les domaines que nous souhaitions découvrir ou approfondir. Tout d'abord, il nous fallait un projet aussi valorisant côté professeurs que côté amis. Deuxièmement, nous désirions quelque chose de visuel à présenter. Certes, un algorithme de 1000 lignes permettant des résolutions de systèmes à n inconnues au x-ième degré aurait été passionnant à réaliser, mais ce n'était pas assez « beau » à notre goût. Nous avons donc opté pour un jeu, mais qui devait rester intéressant d'un point de vue algorithmique et mathématique, il nous fallait donc trouver le genre de jeu permettant de remplir ces conditions. Nous avons hésité entre un jeu de quête en 2D et un tower defense, nous avons choisi le second type.

Le choix du langage a été vu assez rapidement, car j'avais dans l'idée de faire le jeu en JavaScript, et que tout ce choix de langages était bien obscur pour les autres membres. Après quelques heures didactiques ils ont rapidement maîtrisé les bases du langage et certains ont atteint ce que j'appellerai le stade avancé, ce travail a donc été extrêmement bénéfique pour nous tous.

La découverte de ce qu'est vraiment la conception d'un jeu a été intéressante, on se retrouve face à des problèmes visibles et solvables (la fenêtre plante et nous donne la ligne d'erreur), ou d'autres bien plus frustrants et durs à fixer (le jeu se lance et, sans retourner d'erreur, ne fait pas ce qui est attendu). L'aspect prise de tête a vraiment été constructif, car il a fallu optimiser le code au maximum pour que tous les ordinateurs puissent faire tourner le jeu sans trop de difficultés, faire des calculs de trajectoire, réaliser une pseudo-IA, créer des classes, faire un jeu qui s'écoule en fonction du temps, gérer les collisions... Bref, le programme était copieux.

Entrons sans attendre dans le vif du sujet, c'est à dire le jeu lui même.

I – Présentation générale

A/ Principe général du jeu

C'est un jeu de stratégie dans lequel des « ennemis » avancent sur une route en suivant un chemin prédéfini. Le joueur doit les empêcher d'aller au bout du parcours en plaçant des tourelles sur les bords de la route, ces tourelles tirent sur les ennemis pour les détruire.

Lorsqu'un monstre est détruit il rapporte de l'argent au joueur que ce dernier pourra réinvestir dans l'achat de nouvelles tours.

Lorsqu'un monstre atteint le bout du chemin, le joueur perd une vie sur les 30 qu'il possède au départ. Lorsque toutes ses vies sont perdues, c'est la fin de la partie !

Les monstres arrivent un par un mais de façon de plus en plus rapprochée. Lorsqu'un certain nombre de monstres a été envoyé, on parle de « vague envoyée », et l'envoi se stoppe jusqu'à ce que plus aucun monstre ne soit sur le terrain. A ce moment, le joueur voit son argent multiplié par 110% et bénéficie de quelques secondes de répit avant l'envoi de la vague suivante, plus grande et plus difficile.

B/ Cahier des charges

Le jeu devait répondre à certains critères que je vais ici lister afin d'être pleinement fonctionnel.

Tout d'abord, il nous fallait un moyen de décrire le terrain, qui soit assez simple, compacte, efficace.

Dans un deuxième temps, nous avons dû trouver un moyen pour que les monstres suivent le chemin, et ce fluidement (si le case-par-case est assez facile à gérer avec un petit peu d'ingéniosité, Le pixel par pixel est plus complexe).

Ensuite, il fallait que l'on puisse poser les sprites de tourelle et conserver leur position en

mémoire, et dans un second temps leur permettre de détecter les monstres à portée de tir, faire feu sur eux, et changer de cible lorsque le monstre ciblé ne peuvent plus être attaqué. Le mieux aurait été de faire tourner la tête rotative de la tourelle avec un peu de trigonométrie mais après quelques essais j'ai remarqué que ce n'était pas beau, donc j'ai laissé tomber l'idée.

Après, il a fallu gérer les projectiles. C'est à dire contrôler à tout instant t une trajectoire rectiligne uniforme depuis la tourelle jusqu'à la position du monstre à t=0. Et le projectile doit s'arrêter dès qu'il est assez près d'un monstre pour pouvoir considérer qu'on l'a touché (on parle de HitBox), et infliger des dégâts au monstre correspondant, et ce même si ce n'est pas le monstre visé au départ.

L'étape suivante a été de créer un moteur de jeu extrêmement simplifié. Le rôle d'un moteur de jeu (game engine) est de gérer... Tout ! C'est une classe à partie de laquelle on instancie un objet en début de code et qui est un peu le chef d'orchestre du déroulement. Néanmoins, comme nous avons implémenté assez tard cette classe elle ne gère pas tout comme elle aurait été censée le faire, mais seulement les niveaux, et les envois de monstre, ainsi que les fins de vagues. Dans un prochain projet plus ambitieux dont nous avons l'idée, nous souhaitons faire un moteur de jeu bien plus complet qui sera capable de gérer l'image, la physique (gravité, frottements...), les IA, les ombres.

Il nous a également été nécessaire de développer le système d'argent et de points de vie du joueur, ainsi que les interfaces de jeu (affichage des barres de vie, des attributs des tourelles, de la monnaie et des vies restantes, du numéro de la vague...)

La dernière chose à apporter a été des animations pour les monstres, car des monstres figés ne sont pas très agréables au regard, ainsi qu'un mode avance rapide pour ne pas passer trop de temps devant les vagues trop longues.

C/ Monstres et tourelles

1/ Monstres

Les monstres disposent de points de vie, d'une valeur de dégâts, d'une vitesse et d'une valeur de gain.
Points de vie : Lorsqu'ils sont à 0, le monstre est mort
Valeur de dégâts : Nombre de points de vie que le monstre inflige au joueur lorsqu'il passe
Vitesse : Elle définit la vitesse du monstre en case/seconde
Valeur de gain : Nombre d'argent rapporté au joueur lorsque le monstre est détruit.

Différents types de monstre :

Le PinkPathogen :

Vie : 300
Valeur de dégâts : 1
Vitesse : 1,5
Valeur de gain : 1

Le RedShield :

Vie : 1500
Valeur de dégâts : 2
Vitesse : 0,75
Valeur de gain : 5

Le GreenHealer :

Vie : 250
Valeur de dégâts : 3
Vitesse : 1,5
Valeur de gain : 10
Se régénère naturellement

Les monstres deviennent plus résistants au fur et à mesure que les vagues avancent.

2/ Tourelles

Les tourelles sont les structures défensives que peut placer le joueur pour empêcher les monstres d'atteindre le bout du chemin. Elles possèdent des points de dégâts, une cadence de tir, une portée, un prix, et parfois une capacité spéciale.
Dégâts : nombre de points de vie que chaque projectile lancé par la tourelle inflige à un ennemi
Cadence de tir : Temps de latence entre deux projectiles. Plus elle est basse et plus la tourelle tire vite.
Portée : Distance maximale, en pixels, à laquelle une tourelle peut atteindre un monstre (La distance tourelle-monstre est mesurée entre le centre de la tourelle et le centre du monstre)
Prix : Quantité d'argent que le joueur doit verser pour acheter une tourelle
Capacité spéciale : certaines tourelles disposent de spécificités

Différents types de tourelle :

La BeamTurret :

Dégâts : 60
Cadence de tir : 1
Portée : 128
Prix : 100
Capacité spéciale : aucune

Le BeamDragonizer :

Dégâts : 1
Cadence de tir : 0,01
Porte : 256
Prix : 400
Capacité spéciale : Aucune

L'Electrificator : (en cours d'implémentation)

Dégâts : 30
Cadence de tir : 1
Portée : 32
Prix : 200
Capacité spéciale : L'électricité se propage de monstre en monstre proches de moins d'une case

II – Du côté du code

A/ Justification du langage

Pour développer ce jeu j'ai fait le choix du langage Javascript. En effet, tout d'abord j'avais déjà un peu d'expérience et le langage est extrêmement souple (pas d'encapsulation notamment). Il est relativement léger, ne nécessite pas de compilation ou de logiciel à installer, et il utilise l'élément HTML <canvas> qui permet de faire de la 2D avec relativement d'aisance. De plus, à présent nous pouvons incorporer ce jeu à des pages web, ce qui aurait été compliqué en Java's cool. Il répondait donc à tous les critères pour être le langage dont nous avions besoin.

B/ Les « maps » de jeu

Une map (=carte) est ici un agencement de tuiles de chemin qui permettent au joueur de visualiser le parcours des monstres et aux monstres de se déplacer automatiquement.
Nous avons opté pour un système de « calques » : On a plusieurs maps pour décrire une map composée de toutes les autres.

-La map dessinée, chaque tile (=tuile, c'est une image qui représente une case) est associée à un nombre et est dessinée en fonction de ce nombre et des coordonnées matricielles de ce dernier
-La map de déplacements, chaque nombre correspond à une direction que prendra le monstre lorsqu'il passera dessus
-La map de tourelles, elle contient les ID de la tourelle sur chaque case, 0 pour une case vide ou de chemin

Ces maps se superposent et permettent au jeu de tourner.

C/ Présentation des classes

1/Les ennemis

Cette classe gère tous les types d'ennemis

Constructeur : (vie, url, degat, x, y, vitesse, dollar, statut)

Attributs principaux (hormis ceux détaillés précédemment) :

x, y → coordonnées matricielles
posX, posY → coordonnées du pixel haut-gauche
vivant → booléen
statut → Tableau gérant des statuts du monstre (électrifié, empoisonné...) pour l'instant seul électrifié est utilisé

Méthodes :

dessinerMonstre() : méthode permettant le dessin du monstre en (posX ; posY)
perdreVie(degats) : lorsqu'elle est appelée on retranche de la vie du monstre la valeur de dégâts
deplacerMonstre() : le monstre « lit » la case sous ses pieds et change de direction en conséquence. On incrémente sa position

2/ Les tourelles

Cette classe gère tous les types de tourelles

Constructeur : (url, degats, x, y, radius, rate, type)

Attributs principaux (hormis ceux détaillés précédemment) :

targetted → booléen permettant de savoir si un monstre est ciblé
ID → entier naturel définissant le numéro de la tourelle pour y accéder
compteur → entier naturel gérant le temps de latence entre deux tirs
projectiles → matrice ligne contenant les objets projectiles
type → type de la tourelle (voir les variables globales Type)
tbElectricite → Matrice d'objets Électricité propre aux Electrificators

Méthodes :

dessiner() : dessine la tourelle à sa place
distance(monstre) : fonction permettant de connaître la distance monstre-tourelle
target() : permet de choisir un monstre assez proche pour l'attaquer
untarget() : décible le monstre visé
attack() : lance un projectile sur le monstre ciblé si le temps de latence est dépassé

3/ Le joueur

Cette classe gère l'état du joueur au cours de la partie

Pas de constructeur spécifique

Attributs :

vie : initialement à 50, la partie est finit lorsqu'elle atteint 0
money : argent dont le joueur dispose
vivant : booléen permettant de savoir si le joueur est mort ou pas

Méthodes :

perdreVie(degats) : Retranche la valeur de dégâts aux points de vie du joueur et gère l'attribut vivant
budget(argent) : gère toutes les transactions, si la transaction donne un montant positif alors elle retourne true, sinon elle retourne false

4/ Les projectiles

Cette classe gère les projectiles classiques

Constructeur : (x, y, url, vitesse, cible, degats)

Attributs :

cible : ID du monstre ciblé à t=0
auContact : booléen permettant de savoir si un monstre a été touché
x, y : coordonnées au pixel du projectiles
depX, depY : composantes du vecteur déplacement
degats : dégâts infligés par le projectile quand il frappe un monstre

Méthodes :

deplacer() : si elle est appelée à t=0, elle calcule le vecteur déplacement. Sinon elle déplace en fonction du vecteur. Teste les collisions monstre-projectile
dessiner() : dessin du projectile à sa place (x ; y)

5/ L'électricité

Cette classe gère les éclairs projetés par l'Electrificator, qui se propagent de monstre en monstre

Constructeur : (x, y, cible, parent)

Attributs :

ref : élément parent de l'éclair actuel (éclair précédent ou tour)
x, y : coordonnées matricielles de l'éclair
cible : cible de la portion d'éclair actuelle

Méthodes :

genererChemin() : fonction permettant à l'éclair de se propager et de blesser la cible, il capte un monstre à moins d'une case de l'éclair actuel et génère une portion d'éclair dessus, qui utilisera à son tour la méthode genererChemin().

D/ Fonctions et algorithmes utiles

1/ Découpage d'une case d'un tableau

fonction cut(tableau, position), source : Fonction personnelle

Cette fonction permet d'ôter la case position à la matrice ligne tableau, et de réduire le tableau à sa nouvelle dimension.

2/ Saut de ligne

fonction setText(), source : Fonction personnelle

Comme on ne peut ajouter du texte à l'écran que par le biais de ses coordonnées et que les sauts de ligne ne sont pas gérés on les réalise manuellement lorsque le caractère '\n' est détecté dans une chaîne de caractère.

3/ Récupération des coordonnées de ma souris

fonction getMousePosition(event), source : fonction de Sébastien de la Marck / Johann Pardanaud

Permet de récupérer les coordonnées de la souris à l'écran

4/ Récupération des coordonnées d'un élément HTML

fonction getPosition(element), source : fonction de Sébastien de la Marck / Johann Pardanaud

Permet de récupérer les coordonnées de l'élément HTML element. Cela permettra ensuite d'en

déduire les coordonnées de la souris au sein de cet élément par soustraction.

5/ Obtenir un nombre de Frames Per Second (FPS) stable ou l'accélerer

fonction requestAnimFrame(), source : fonction de Sébastien de la Marck / Johann Pardanaud

Au début nous utilisions une fonction nommée setTimeout(fonction, délai) qui est directement implémentée au langage et qui permet de rappeler une fonction au bout d'un certain intervalle (délai, en ms). Cependant les navigateurs gèrent directement cette option et peuvent allouer de la mémoire vive pour conserver des FPS supérieurs à 30, ce qui est bénéfique et plus simple.
Fonction setTimeout(fonction, délai), implémentée au JS

Pour accélérer le rythme du jeu, on augmente le nombre d'appel par seconde de la fonction jouer, en divisant le délai par un nombre >0. De cette manière le nombre de FPS augmente.

III – Le futur de ce programme et de notre équipe de développeurs...

A/ La fin de ce jeu qui n'est qu'au stade d'Alpha

Pour le moment ce jeu n'est pas très attrayant pour un joueur, on n'y accroche pas ; Tout au long de cette année il est vrai que nous nous sommes plus concentrés sur l'aspect algorithmique et mathématique que sur le côté « jeu » que nous avons un peu laissé de côté.
Cependant, maintenant que les algorithmes les plus gros (projectiles, déplacements, ciblage des monstres, gestion de la map...) sont en place nous pourrons nous pencher plus sur l'aspect « gameplay », c'est à dire implémenter de nouvelles tours, de nouveaux monstres, de nouvelles maps, une meilleure interface, de nouveaux graphismes etc...
De plus, comme nous sommes en JavaScript rien ne sera plus simple que de propager ce jeu sur la toile même si il est évident que son succès sera modeste.
Nous somme assez motivés pour le finir et pour arriver à quelque chose de plus beau, et plus agréable à jouer cependant, nous avons encore d'autres ambitions...

B/ Le futur de l'équipe

Suite à ce projet, nous avons décidé de continuer à développer des jeux pendant notre temps de loisir. Nous souhaitons changer de plate forme, c'est à dire développer des applications sur smartphone, car celles-ci étant plus massivement utilisées et promues nous aurons plus de chance de nous faire connaître. Et, pour peu d'avoir un concept vraiment novateur, nous pourrons espérer gagner en notoriété, voire même bénéficier d'une petite source de revenus.

Comme il a été dit plus haut, nous avons un projet avec plus d'envergure que nous aimerions commencer à développer après les examens. Il s'agira d'un jeu de plate forme dans lequel le joueur dirigera un engrenage, il pourra modifier la taille et le sens de ce dernier et devra trouver un moyen de parcourir différents tableaux en utilisant des interactions avec l'environnement.

Vous avez sans doute remarqué que nous avons tenu à faire le maximum par nous même, que ce soit le code ou les images. Cela nous a certes posé des limites en terme de qualité, de propreté et donc au final de rendu. Néanmoins, nous avons en quelques sortes joué tous les rôles que constitue une équipe de développement : Du game producer au graphiste en passant par les programmeurs et le level designer. Nous avons donc pu un peu « toucher à tout » et nous rendre compte de la quantité de travail qu'il y a derrière un jeu, ainsi que les problèmes qui se posent et leurs solutions.

Pour conclure sur ce projet, je dirai que travailler en équipe nous a tous beaucoup apporté et appris, et que ce coup d'essai nous a permis de prendre conscience de nos capacités, de nos

aptitudes à contourner les problèmes ainsi que de nos lacunes.

Tout cela va nous permettre de nous organiser au mieux lors de la conception de prochains logiciels, afin que la répartition des tâches soit la plus efficace possible, et que nos jeux (car oui on ne va pas se leurrer notre but principal est de développer, au moins pendant notre temps libre, d'autres jeux vidéo) soient les plus attractifs, agréables à jouer, beaux à regarder et addictifs que possible.

L'équipe s'élargira sans doute, cependant la coopération et l'entraide resteront les maîtres mots, aucun d'entre nous ne détient LA solution à tous les problèmes, et il n'y a que ensemble que nous serons capables de les résoudre.