Projet Conquête Spatiale

Licence 3 - Langages Web

Thomas Lebrun
Bilal Makhlouf

Sommaire :

1 - Travail réalisé

Notre jeu se déroule sur deux pages web : la page de configuration et la page de jeu. En effet, en arrivant en jeu le joueur se retrouve devant un formulaire lui permettant de choisir le nombre de planètes, la population de départ des joueurs, ainsi que le nombre d'ennemis (intelligences artificielles) qu'il souhaite affronter. Il suffit alors ensuite de valider le formulaire, et le joueur se retrouve envoyé vers la page de jeu.

En arrivant sur la page de jeu, le joueur peut apercevoir en haut de son écran un menu contenant une case pour chaque joueur, c'est l'afficheur de tour. Il permet de savoir à tout moment qui doit jouer, et permet aussi de savoir qui est encore en partie (une case grisée et en italique représente un joueur éliminé). Le reste de l'écran est composé de la carte, et d'un bouton permettant de terminer son tour (désactivé quand c'est pas au tour du joueur humain de jouer).

On constate alors que sur la carte sont disposées toutes les planètes, dont celles des différents joueurs. Chaque joueur possède sa couleur, donc les labels de population des planètes sont aux couleurs de leur allégeance. (Une planète neutre aura simplement un label "?" blanc). Chaque joueur possède au départ une planète et cette planète possède la population de départ remplie lors de la configuration. Les autres planètes (neutres) ont une population située entre 1 et 50, mais cachée. Ensuite, toutes les planètes qui appartiennent au joueur duquel c'est le tour tournent sur elles-mêmes.

Pour jouer, il suffit de faire un glisser déposer depuis une planète alliée vers n'importe quelle autre planète pour démarrer un transfert de population (ou une attaque si la planète de destination n'est pas alliée !). En faisant ainsi, le jeu affiche ensuite un menu avec un formulaire, permettant de voir et configurer les informations de l'attaque/transfert en cours, et de le confirmer ou non. On peut ainsi régler la population à envoyer. Une fois le transfert ou l'attaque confirmé(e), un vaisseau s'affiche à l'écran, avec son trajet représenté par un trait de la couleur du joueur à qui le vaisseau appartient. Au dessus du vaisseau s'affiche la population à bord si jamais il appartient au joueur humain (joueur 1 / bleu).

Les tours des joueurs s'enchaînent jusqu'à ce qu'il ne reste qu'un joueur en vie. Un joueur est considéré vivant si il lui reste de la population dans un/plusieurs vaisseaux et/ou une/plusieurs planètes.

2 - Présentation des frameworks et librairies utilisées

Pour réaliser ce projet, nous avons décidés de faire appel à deux frameworks et quelques librairies pour nous aider à le réaliser en temps et en heure, et avec la vision que l'on en avait. On a donc utilisés deux frameworks :

Quintus.js : http://www.html5quintus.com/

Quintus.js est un très léger moteur de jeux vidéos basé sur les canvas HTML5. Il embarque avec lui la plupart des fonctionnalités de bases utiles à la création de divers jeux web en 2D. Il permet notamment de gérer la gravité, le placement d'objets, leur déplacement, certaines animations, la gestion de feuilles de sprites, ou encore les collisions. C'est un moteur avant tout léger et facile à appréhender. Il n'est pas encombré de trop nombreuses fonctionnalités, et permet de tout réaliser soi-même avec une bonne base de code pour certaines parties complexes (gestion du canva, collision, etc…). Nous nous en servons pour représenter le jeu en lui même. Il nous aide donc à placer les planètes, les afficher proprement, les faire tourner, afficher les vaisseaux, tracer les vecteurs de trajets, afficher les populations, etc. Nous avons quand même eu à tout coder nous mêmes pour le jeu, mais le framework nous a permis d'abstraire quasiment entièrement la gestion du canva.

Pure CSS : https://purecss.io/

Pure CSS est un framework CSS permettant de fournir quelques bases très légères pour certains éléments clés, comme les formulaires ou encore les boutons. Ce framework est très facile à utiliser et est extrêmement léger (c'est d'ailleurs sa force). Il nous permet d'obtenir de beaux formulaires et boutons, et d'avoir un bon placement des éléments au sein des formulaires. Tout le reste, les couleurs, le placement des éléments hors formulaire et autres est géré par nous mêmes.

Nous avons aussi utilisé une librairie :

Toastr.js : https://codeseven.github.io/toastr/

Toastr.js est une librairie très simple d'utilisation, et qui permet de réaliser des sortes de notifications "toast". Ces notifications s'affichent où on le souhaite sur l'écran, et sont semi transparentes. C'est une librairie facilement configurable, et les messages peuvent s'afficher les uns à la suite des autres. Nous nous servons principalement de cette librairie afin de garantir une affichage clair et facilement compréhensible des notifications en jeu :

lorsqu'un joueur est éliminé, qu'une attaque ou un transfert se termine, ou encore que la partie est terminée.

3 - Fonctionnement du jeu / Algorithmes principaux

Fonctionnement général :

Lorsque le formulaire de configuration est rempli, les données sont envoyées vers la page de jeu via POST. En PHP, sur la page de jeu, on se sert de ces données pour générer le menu du haut pour les tours des joueurs. Ensuite, on transmet les différentes données de la carte à Javascript, qui s'en servira pour générer et afficher la carte suivant un algorithme spécifique. Puis, on a différentes classes enregistrées dans Quintus.js : les vaisseaux et les planètes. Les planètes sont instanciées et placées au début de la partie sur la carte, et les vaisseaux sont instanciés, et placés au fur et à mesure des actions des joueurs. Le jeu a été codé de telle sorte que l'on possède un coeur interactif quasiment autonome avec les vaisseaux et planètes, pouvant facilement interagir entre-eux et interagir avec le joueur / éventuel reste du code via des actions / accesseurs. On rajoute donc là dessus le gestionnaire de tours et l'intelligence artificielle, et tout fonctionne.

Nous allons donc maintenant donner des détails sur le fonctionnement de certaines parties plus précises du jeu ainsi que certains algorithmes.

Algorithme de placement des planètes :

L'algorithme de placement des planètes est assez simple, sans pour autant être trop basique. On récupère les différentes variables de la configuration que le joueur souhaite, et ensuite, on entre dans une boucle for, pour chaque planète. On génère une position aléatoire. Si il y a une planète trop proche (on définit dans le code des constantes permettant de régler cette distance), on régénère une position aléatoire, et on incrémente le compteur. On effectue ceci jusqu'à une certaine limite d'essais définie aussi dans une constante. Si on dépasse cette limite, on place la planète aléatoirement, et on utilise le système de collisions de Quintus.js pour déplacer la planète jusqu'à une position valide. Une fois la planète placée, on l'instancie comme il faut. Lorsque toutes les planètes sont placées, on en sélectionne autant de différentes qu'il y a de joueurs dans la partie, et on en attribue donc une à chaque joueur, et on configure leur population de départ suivant la configuration donnée par le joueur.

Fonctionnement des vaisseaux "autonomes" :

Les vaisseaux sont des instances de la classe Ship que l'on incorpore à l'objet principal que représente Quintus (l'objet "Q"). De ce fait, on pourra facilement placer des vaisseaux sur la carte. On peut quasiment considérer les vaisseaux comme autonomes. Une fois qu'un vaisseau est placé et configuré, il s'auto-gérera. En effet, cette classe possède un booléen nommé "my_turn" permettant au vaisseau de savoir si il doit jouer une

action. Si ce booléen est à faux, le vaisseau reste immobile et ne fait rien. Lorsque ce booléen passe à vrai, le vaisseau se déplace sur son vecteur, d'une unité de déplacement, et décremente son compteur de tours. La vitesse du vaisseau est parfaitement calculée de sorte à ce qu'à chaque tour il avance suffisement pour atteindre sa destination à temps (par rapport à l'estimation donnée lors de l'envoi du vaisseau). Une fois son déplacement terminé, le vaisseau vérifie donc si il lui reste des tours à jouer, si oui, il décremente le nombre de tour qu'il lui reste à jouer et enfin, il passe "my_turn" à faux, et donc termine son tour. Si il ne lui reste plus de tours à jouer, c'est qu'il a atteint sa destination. Dans ce cas, il résout son action en effectuant le transfert ou l'attaque, et gère lui même l'issue de la bataille, affiche le message qui correspond, et se détruit tout seul.

Fonctionnement des planètes :

Les planètes possèdent un fonctionnement très similaire à celui des vaisseaux. Chaque planète gère elle-même son label de population suivant le joueur qui la possède, ainsi que sa population. Une planète est aussi une classe incorporée à l'objet principal que représente Quintus. Une planète possède également un booléen "my_turn". Ce booléen permet de rendre interactive ou non la planète. Ainsi, lorsqu'il est à faux, ou alors que le joueur la possède est une IA, il est impossible pour le joueur principal d'interagir avec la souris dessus. On ne peut pas lancer de vaisseaux depuis une telle planète, en tant que joueur (soit elle ne nous appartient pas, soit ce n'est pas notre tour). De plus, lorsque ce booléen est à faux, la planète ne tourne pas. Lorsque le booléen est à vrai, la planète tourne. Une IA peut donc utiliser ses planètes pour attaquer. Si le booléen est à vrai et que la planète appartient au joueur, elle se met à écouter les actions de glisser-déposer afin de permettre au joueur de démarrer des transferts / attaques. Enfin, une planète bloque sa croissance si elle est neutre, et autorise la croissance sinon. On ordonne à une planète de faire grandir sa population avec une méthode qu'il suffit alors d'appeler.

Gestion des tours en jeu :

Chaque joueur peut jouer autant d'actions (envoyer des vaisseaux) qu'il veut pendant son tour. Au début du tour du premier joueur, les populations de toutes les planètes non neutres augmentent. Au début de chaque tour, on change le tour, pour passer de joueurs en joueurs. On place dans une liste de taille égale au nombre de joueurs, le nombre de planètes ou vaisseaux possédés par chacun de ces joueurs. Grâce à cela, on voit quel(s) joueur(s) sera ou seront éliminé(s) pendant le tour : celui ou ceux qui n'ont plus de planètes ou de vaisseaux en leur possession. Quand un joueur est éliminé, son tour est passé automatiquement.
On calcule également le nombre de joueurs restants, ce qui nous permet de savoir si la partie est terminée, avec une liste de taille égale au nombre de joueurs qui contient des 1 (joueur en vie) et des 0 (joueur éliminé). Si le nombre de joueurs restants est égal à 1, la partie se termine. Dans ce cas, si le nombre en première position dans la liste est égal à 1, le joueur humain gagne, sinon c'est l'IA qui a gagné.
Si la partie ne se termine pas, alors on va calculer le tour du prochain joueur, et mettre toutes ses planètes ainsi que tous ses vaisseaux à "my_turn = true", ce qui veut dire qu'il pourra les utiliser.

Enfin, on appellera la fonction “wait_ships” qui va attendre la fin d’animation de tous les vaisseaux en déplacement, pour éviter de les couper si un joueur joue trop vite. Une fois les vaisseaux immobiles, on regarde qui doit jouer : un joueur humain ou un joueur ordinateur.

4 - L’intelligence artificielle

Dans ce jeu, les ordinateurs ont leur propre intelligence. Celle-ci est basique, mais assez coriace pour tenir une dizaine de minutes. Voici comment elle fonctionne :

- l’IA commence par parcourir toutes ses planètes possédées.
- parmi elles, elle va prendre celle qui possède le plus de population.
- elle va ensuite sélectionner une planète au hasard parmi toutes les planètes en jeu (y compris les siennes).
- elle va y envoyer un montant de troupes égal à un nombre aléatoire entre 1 et la population de sa planète choisie à l’étape 2.

Ainsi, si la planète choisie pour envoyer ses troupes est une planète ennemie ou neutre, elle sera attaquée.
Si c’est une planète alliée, elle sera renforcée.
Une fois toutes ses actions effectuées, l’IA va passer son tour.

5 - Conclusion

Ce projet de création d'un jeu de conquête spatiale fut très intéressant pour nous deux. Il nous a permis de renforcer et appliquer toutes les connaissances vues en cours, et de voir jusqu'où nous pourrions aller. L'utilisation des frameworks nous a permis de réaliser un projet avec un rendu qui nous satisfait bien. Ces frameworks nous ont aussi donné la possibilité d'implémenter des fonctionnalités plus poussées.

Nous avons aussi eu affaire à de nombreux bugs plus ou moins bloquants tout au long du développement, mais nous en sommes venus à bout grâce à quelques réflexions, et quelques recherches sur le net.

Si vous rencontrez un quelconque problème à l'installation ou à l'utilisation du projet, vous trouverez des informations dans le fichier "README.txt" fourni dans le code source. Vous pouvez également nous contacter via nos mails universitaires :

- thomas.lebrun1@etu.univ-rouen.fr
- bilal.makhlouf@etu.univ-rouen.fr

C'est un projet que nous avons beaucoup apprécié, et travailler dessus était pour nous plaisant. Le framework Quintus.js nous a aussi permis d'essayer plein de choses par rapport aux différents algorithmes et à la façon de gérer le jeu.

Pour travailler ensemble, nous avons mis en place un git, hébergé sur Github. Git étant une technologie que l'on maîtrisait tous les deux depuis notre DUT Informatique, cela nous a paru un choix idéal pour faciliter la collaboration. Si vous souhaitez le voir, voici le lien : https://github.com/Raivk/projet_web_space