Programmation d'un jeu de *hornuss* en Javascrip

Travail de maturité réalisé au Lycée Denis-de-Rougemont de Neuchâtel
sous la direction de M. Manuel Fragnière

Audrey Loeffel

Avant-propos

Le Comité de rédaction est heureux de publier ce travail de maturité. D'une part, il inaugure un nouveau domaine et d'autre part, il correspond aux normes de qualité et d'intérêt demandé pour le Bulletin. Toutefois, se posait le problème de rendre accessible au plus grand nombre l'aspect relativement technique de la programmation. L'organisation judicieuse du rapport choisie par son auteure a permis de trouver une solution. En effet, le rapport présente tout d'abord une description globale du jeu et des éléments de programmation mis en oeuvre, les problèmes rencontrés et les différentes solutions envisagées. Une deuxième partie décrit de de façon plus précise chaque fonction programmée. Finalement, l'annexe donne le code Javascript précédé des structogrammes des fonctions et d'une description des variables.

Il semblait que la première partie permet à la fois à tout un chacun de se rendre compte du travail réalisé tout en permettant aux connaisseurs de Javascript de deviner les aspects plus techniques qu'ils pourront aussi obtenir en s'adressant auprès de l'auteure ou en consultant les archives du Lycée Denis-de-Rougemont. (Rédaction)

Introduction

J'ai choisi ce thème car l'informatique m'a toujours intéressée et j'envisage de continuer mes études dans ce domaine. J'avais envie de relever un défi: créer un jeu, mais un jeu original et inédit, et non reproduire une version de plus d'un jeu déjà existant. J'ai découvert le *hornuss* par hasard et l'aspect "local" m'a plu. Mon choix s'est donc porté sur le *hornuss*.

Le *hornuss* est un sport traditionnel suisse principalement pratiqué en Suisse allemande. Il existe depuis plus de 400 ans cependant il est très peu connu et peu de personne le pratique. Il ressemble à un mélange entre le golf et le baseball. Pendant le jeu, 2 équipes s'affrontent. L'une doit projeter le *hornuss* (projectile) le plus loin possible. Le lanceur le frappe avec un fouet métallique sur une rampe de lancement. L'autre équipe, positionnée en face du lanceur, dans le terrain, doit essayer d'intercepter le *hornuss* en vol avec des palettes (panneaux en bois). Après plusieurs lancers les équipes permutent. Le score est comptabilisé en fonction des pénalités attribuées à l'équipe des lanceurs lorsque le *hornuss* touche le sol, et de la distance parcourue par le *hornuss*.

Mes objectifs étaient de programmer, en JavaScript, un jeu de *hornuss* en deux dimensions avec un lanceur et plusieurs intercepteurs, avec un graphisme élémentaire. L'utilisateur devait pouvoir jouer contre l'ordinateur, il fallait donc faire "réfléchir" l'ordinateur. Je voulais également que mon jeu soit le plus ressemblant possible, dans le déroulement du jeu, au sport réel.

Je devais trouver une méthode pour afficher le jeu. Ensuite je devais calculer les mouvements du *hornuss* et des palettes, trouver un moyen de faire lancer et intercepter l'utilisateur et, puisque l'ordinateur doit jouer contre l'utilisateur, le faire réfléchir afin qu'il puisse aussi lancer et intercepter.

L'affichage

L'affichage du jeu se fait avec l'élément *canvas*, une zone de dessin insérée dans le code HTML de la page web. Cependant, avec Microsoft Internet Explorer (IE), le *canvas* ne fonctionne pas. C'est pourquoi, lors du chargement de la page, la fonction 'navigateur()' teste le navigateur et avise l'utilisateur du problème rencontré avec IE.

Dans la section HTML, la largeur et la longueur du *canvas* sont définies. Alors que dans la section JavaScript, les dessins, les textes et les images sont programmés.

Trajectoire du *hornuss*

Calcul de la trajectoire

La trajectoire décrite par le *hornuss* est, en négligeant les frottements, une parabole donnée par une combinaison d'un mouvement rectiligne uniforme (MRU) horizontalement et un mouvement rectiligne uniformément accéléré (MRUA) d'accélération verticale g (figure 1).

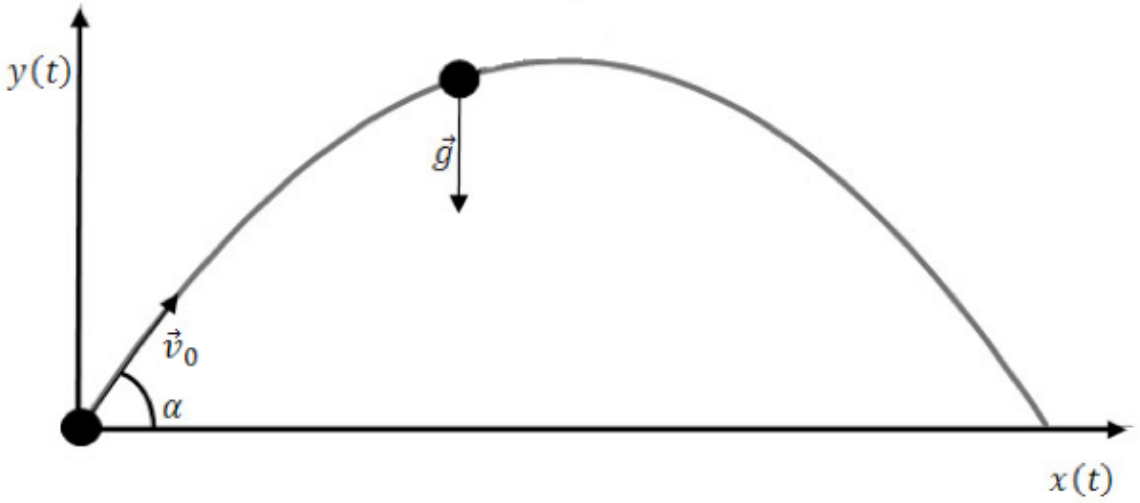

fig 1. La trajectoire du *hornuss*

avec

$$\vec{g}=\begin{pmatrix}0\\-g\end{pmatrix} \quad ; \quad \vec{v}_0=\begin{pmatrix}v_0\cos(\alpha)\\v_0\sin(\alpha)\end{pmatrix}$$

L'équation du mouvement est $\vec{x}(t)=\frac{1}{2}\vec{g}\,t^2+\vec{v}_0 t$. Cette équation se décompose en l'équation du mouvement horizontal $(I)\,x(t)=v_0\cos(\alpha)t$ et vertical $(II)\,y(t)=\frac{-1}{2}g\,t^2+v_0\sin(\alpha)t$.

Les palettes décrivent le même mouvement mais avec un angle de 90°.

Programmation de la trajectoire

Pour faire avancer le *hornuss* et la palette j'utilise deux objets. L'un pour calculer les coordonnées en fonction du temps et l'autre pour afficher. Ce dernier possède deux méthodes, une pour afficher le *hornuss*, l'autre pour afficher la palette.

Dans l'objet 'ParametreLanceur'[1], sont introduits l'angle, la vitesse et le temps. Ensuite les coordonnées sont calculées avec la méthode 'coordonnees' grâce aux équations (I) et (II).

[1]Petite aide à la compréhension : cet objet contient de fait les paramètres du *hornuss qui évoluent tout au long du jeu.* (ndlr)

L'objet 'Affichage' est ensuite appelé avec x et y comme attributs. Ces paramètres sont utilisés par la méthode 'dessin' et la méthode 'dessinPalette' pour afficher, respectivement, le *hornuss* et la palette.

Le temps est incrémenté et les deux objets sont crées chaque fois que la fonction 'execute()' est réalisée. Cette fonction est appelée par un *timer* 24 fois par seconde. Cet intervalle de temps rend le mouvement continu.

Les deux objets avec leurs paramètres et leurs méthodes sont symbolisés dans la figure 2.

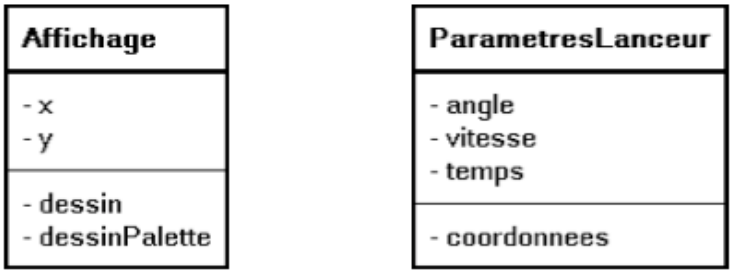

fig 2. Les objets de base

Le lancer du *hornuss*

La variable globale 'jeu1' permet de différencier le joueur. La variable 'etat' permet de savoir si le *hornuss* n'a pas encore été lancé ou si, au contraire, il est déjà en l'air.

Par l'utilisateur

Le choix de la vitesse et de l'angle avec lesquels le *hornuss* sera lancé, s'effectue avec la souris. A chaque mouvement de celle-ci, la fonction 'parametres()' est appelée. Elle affiche une ligne depuis le *hornuss* qui se trouve sur la rampe de lancement jusqu'au curseur de la souris. La longueur de ce trait ainsi que l'angle formé avec le sol renseignent les paramètres, vitesse et angle. Lors du clic de la souris ces paramètres sont conservés puis la fonction 'lancer()' est exécutée. Cette dernière appellera la fonction 'execute()' qui lancera le *hornuss* si les conditions 'etat' et 'jeu' sont réunis, c'est-à-dire si les deux variables sont à 0.

Par l'ordinateur

L'angle et la vitesse sont définis aléatoirement dans la fonction 'initialiser()'. Lorsque l'utilisateur presse sur la touche **D**, la fonction 'start()' est appelée. Celle-ci appellera ensuite la fonction 'execute()' qui lancera le *hornuss*.

L'animation du lancer

J'avais d'abord essayé de faire une image animée en format "gif" pour l'animation du lanceur, cependant l'élément *canvas* ne permet pas d'afficher une image dans ce format. J'ai crée alors la fonction 'animation()'. Celle-ci permet d'afficher les images de l'animation à la suite avec un intervalle de 0.3 seconde.

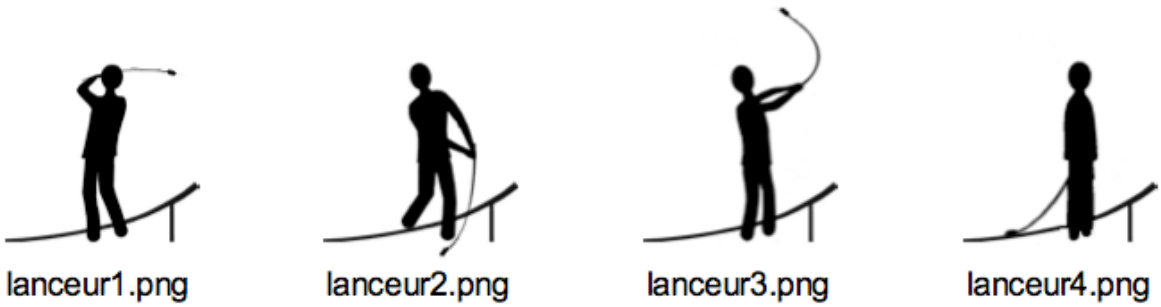

fig 3. La décomposition des mouvements du lanceur

Les images crées à cette occasion sont données dans la figue 3.

En attendant que le *hornuss* soit lancé, l'image "lanceur1.png" est affichée par la fonction 'initialiserCanvas().' Lorsqu'il est lancé, cette même fonction affiche alors l'image "lanceur4.png".

Le changement des équipes

Dès dix lancers effectués les équipes permutent; les lanceurs deviennent intercepteurs et inversement.

Ce changement est analysé à chaque fois que le *hornuss* est intercepté ou s'il est à terre par la fonction 'changement()', c'est-à-dire à chaque fois que les fonctions 'palette()' ou 'execute()' sont réalisées. La fonction 'changement()' compte le nombre de lancers effectués par l'ordinateur. Une fois le nombre de lancers atteints, 10 fois, la variable 'jeu' est mis à 0 et c'est au tour de l'utilisateur de lancer. Lorsque le nombre de lancers de l'utilisateur est atteint le jeu est terminé.

L'interception du *hornuss*

Les intercepteurs sont répartis dans la zone de jeu et doivent intercepter le hornuss en vol.

J'avais d'abord pensé mettre un *timer* différent pour le *hornuss* et pour chaque palette. Pour cela j'avais besoin de quatre fonctions différentes; l'une pour le *hornuss*, les trois autres pour les palettes. Dans chacune de ces fonctions j'appelais la fonction 'initialiserCanvas()'. Ceci provoquait des saccades dans les mouvements car les *timers* n'étaient pas synchronisés.

J'ai ensuite essayé d'appeler la fonction 'initialiserCanvas()' seulement depuis la fonction qui lance le *hornuss*, là encore il y avait des problèmes de fluidité. Les quatre fonctions n'étant pas liées et synchronisées, la palette pouvait ne pas être visible car elle était affichée et immédiatement effacée.

Finalement j'ai choisi d'utiliser un seul *timer* et une seule fonction qui coordonne toutes les autres. Ainsi le *hornuss* et les palettes sont affichés en même temps, ce qui évite les saccades.

Par l'utilisateur

J'ai commencé par programmer l'interception de l'utilisateur car il fallait trouver un système avec lequel il était possible de déclencher manuellement le *hornuss* et les palettes dans les temps. J'ai donc fait plusieurs essais d'interception en variant le nombre d'intercepteurs et le mode de lancer.

Les premiers essais que j'ai effectués ne comportaient qu'un seul intercepteur. J'avais mis un bouton sous l'élément *canvas* qui permettait de lancer la palette, cependant ce n'était pas pratique et très peu jouable si j'augmentais le nombre d'intercepteurs.

J'ai ensuite essayé avec un intercepteur que l'utilisateur pouvait déplacer à gauche ou à droite avec les touches **A** et **D** et lancer la palette avec la touche **W**. Mais cette technique n'aurait pas été réalisable avec trois intercepteurs; l'utilisateur n'aurait pas eu assez de temps pour bouger les trois intercepteurs puis de lancer les palettes.

Puis j'ai créé une fonction qui se déclenchait au clic de la souris sur l'élément *canvas*. Si les coordonnées de la souris étaient comprises dans une zone autour l'un des intercepteurs, la palette était lancée. Mais là encore il y avait un problème de jouabilité; l'utilisateur n'avait pas le temps de cliquer à la suite sur les trois intercepteurs. Si le premier lancer n'interceptait pas le *hornuss*, l'utilisateur n'avait pas le temps de lancer les palettes suivantes.

Finalement, je suis revenue à ma deuxième idée, c'est-à-dire de lancer avec le clavier, mais avec trois intercepteurs fixes. L'utilisateur doit appuyer sur **J**, **K** ou **L** pour lancer les palettes.

Avec ce système l'utilisateur dispose de suffisamment de temps pour lancer les trois palettes et intercepter le *hornuss*.

L'utilisation de ces touches lancera verticalement les palettes des trois intercepteurs, seulement si l'utilisateur est l'intercepteur.

Pour cela, la fonction 'start()' mettra à 1 les variables d'état des palettes, ce qui permettra ensuite à la fonction 'palette()' de lancer les palettes.

Par l'ordinateur

Le mécanisme d'interception de l'ordinateur est similaire à celui de l'utilisateur excepté que la fonction 'execute()' appelle également la fonction 'intercepter()'.

Cette fonction calcule le temps t_1 du parcours du *hornuss* pour atteindre le point (x_1, y_1), c'est-à-dire l'intersection entre la trajectoire du *hornuss* et celle de la palette. A ce temps t_1 est soustrait le temps du lancer vertical t_{la} nécessaire à la palette pour atteindre ce même point. Cette différence de temps, t_{dep}, permettra de définir l'instant de lancer de la palette (figure 4).

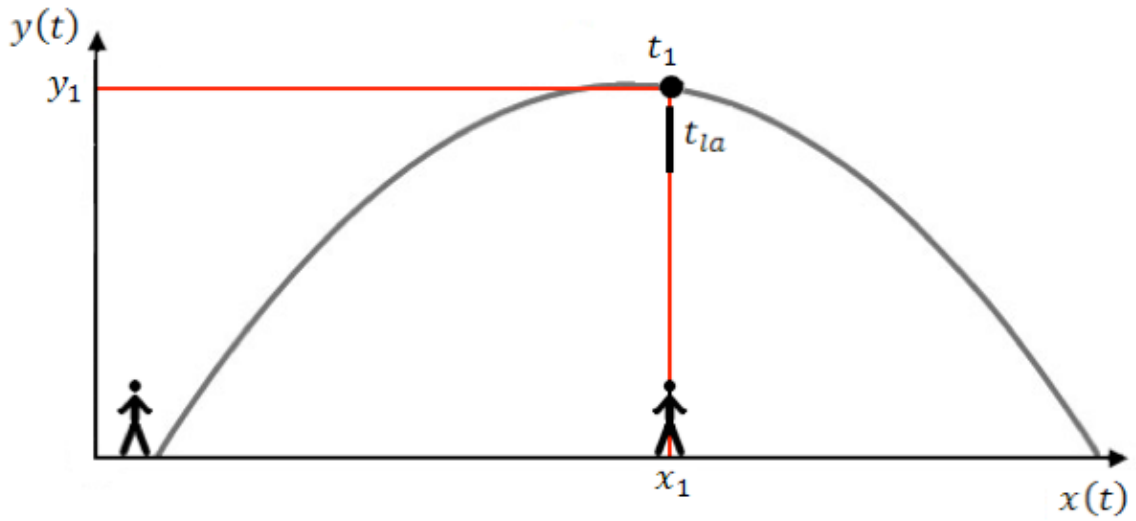

fig 4. Trajectoire du *hornuss* et d'une palette

Le temps t_1 en x_1 se calcule avec l'équation (I) : $t_1 = \frac{x_1}{v_0 \cos(\alpha)}$.

La hauteur y_1 du *hornuss* au temps t_1 est donnée par : $y_1 = \frac{-1}{2} g\, t_1^2 + v_0 \sin(\alpha)\, t_1$

Le temps t_{la} qu'il faut à la palette pour atteindre y_1 s'obtient avec l'équation correspondante pour la palette $y_1 = \frac{-1}{2} g\, t_{la}^2 + v_{palette}\, t_{la}$ en isolant t_{la} : $t_{la} = \frac{v_{palette} - \sqrt{v_{palette}^2 - 2gy_1}}{g}$

A ce temps j'ajoute une valeur aléatoire entre -1/4 et 1/4, une marge d'erreur afin que l'ordinateur puisse manquer le *hornuss*.

Le temps t_{dep} auquel la palette devra être lancée pour intercepter le *hornuss* est : $t_{dep} = t_1 - t_{la}$

L'animation des intercepteurs

L'idée était d'utiliser la même fonction que celle qui affiche le lanceur, c'est-à-dire la fonction 'animation()'. Cependant il y avait deux problèmes principaux :

- Comme la fonction était appelée qu'une seule fois, elle était relancée par le *timer*. L'affichage n'était donc pas synchronisé avec la fonction 'execute()'. Les images ne

s'affichaient pas ou étaient rapidement effacées. Dans le cas du lanceur, cette méthode fonctionne car l'animation a lieu avant de lancer la fonction 'execute()'.

- Le lanceur s'effaçait lorsque l'animation de l'un des intercepteurs était lancé.

J'ai alors partagé cette fonction en deux autres fonctions, l'une pour le lanceur et la seconde pour les intercepteurs.

La seconde fonction, 'animation_intercepteur()', était appelée par la fonction 'execute()' et devait afficher les différentes images. Les images étant toujours saccadées, j'ai revu complètement l'approche en lui attribuant le rôle d'indiquer l'image qui sera affichée.

C'est la fonction 'initialiserCanvas()'[1], appelée par la fonction principale 'execute()', qui affiche les différentes images en fonction de la variable d'état de chaque intercepteur. Les saccades ont alors disparu car l'affichage des intercepteurs est synchronisé avec le reste des éléments affichés dans le *canvas*.

Les images utilisées pour animer l'intercepteur sont données dans la figure 5.

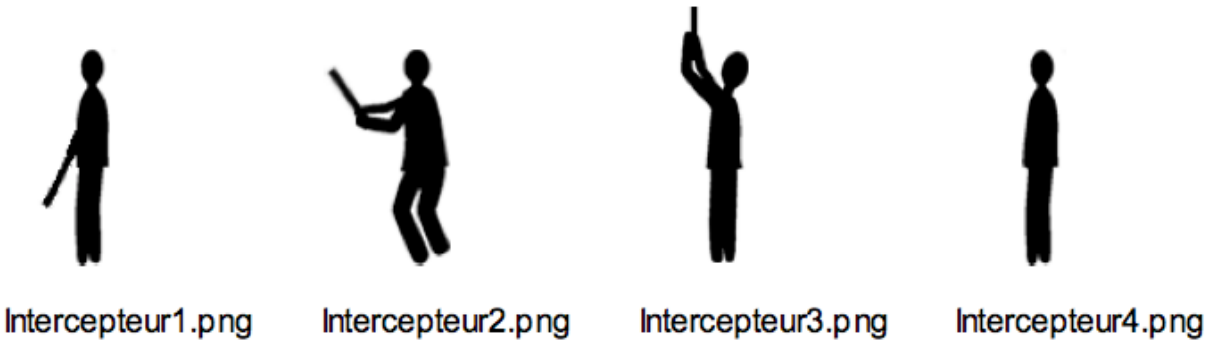

fig 5. Les images de d'un intercepteur

Le score

Le score se calcule dès que le *hornuss* est à terre ou est intercepté.

Lorsque le *hornuss* est à terre, le score se calcule en fonction de la distance qu'il a parcourue. Le lanceur gagnera 1 point s'il a atterri entre le premier intercepteur et le deuxième, 2 points entre le deuxième et le troisième et 3 points s'il tombe après le troisième. Mais s'il tombe avant le premier intercepteur il ne gagne pas de point.

Le système de point lors de l'interception du *hornuss* suit le même principe excepté que la notation sera inversée. Si le *hornuss* est intercepté par le premier intercepteur, l'intercepteur gagne 3 points. 2 points par le deuxième et 1 point par le troisième.

Conclusion

J'ai tenu à garder le plus d'éléments possible du sport réel. Cependant le système de points était trop complexe, j'ai donc préféré calculer le score en fonction de la distance. Le reste me semble assez fidèle à la réalité.

J'ai dû faire face à plusieurs domaines ou difficultés tel que:

- déterminer la balistique pour la trajectoire du *hornuss* et des palettes ainsi que pour leur interception.

- faire communiquer des éléments indépendants entre eux.

- illustrer graphiquement les objets volants le plus réaliste possible.

[1]Que l'on pourrait également nommer 'rafraichirCanvas()' pour bien comprendre son rôle tout au long du déroulement du jeu. (ndlr)

- interagir avec la souris et le clavier sur le déroulement du jeu.

- animer le lanceur et les intercepteurs.

Si je devais améliorer le jeu, je pourrais apporter des niveaux de difficultés, des éléments aléatoires comme du vent ou des oiseaux, ou encore pouvoir jouer en réseau et améliorer le graphisme. Mais la principale modification que j'effectuerais, serait l'utilisation un autre langage de programmation, plus adapté au jeu, que JavaScript.

Bibliographie

HARRIS Andy, *JavaScript & Ajax, Pour les nuls*, Paris, Éditions First, 2010.

HONDERMARCK Oliver, *JavaScript, Le guide complet*, Paris, Micro Application, 2010 (5ème édition).

McFARLANE Nigel, *JavaScript Professionnel*, Paris, Éditions Eyrolles et Wrox Press, 2000.

TITTEL Ed et VANDERVEER Emiliy, *HTML 4 et JavaScript, Pour les nuls*, Paris, Éditions First, 2008.