Présentation du projet d'informatique : flyingTurtles

M. Boutserin, R. Deyres, M. Léger

6 décembre 2015

Résumé

Lors de ce projet nous allons développer un jeu en C++ basé sur le jeu Angry Birds, il sera nommé flyingTurtles. Ce jeu se base sur la mécanique car assimilé à un lancer de balle sans frottements dans un premier temps puis avec des effets tels que les intempéries et un sol changeant aléatoirement entre chaque partie. Le but sera donc en un nombre de lancers limités de détruire les ennemis qui se trouveront face aux tortues.

Table des matières

I Présentation du jeu 2

II Simulation du lancé d'une tortue 3

III Création du score 5

IV Architecture du programme 5

V GitHub 7

VI Conclusion 7

I Présentation du jeu

FlyingTurtles est un jeu basé sur le même principe qu'Angry Birds avec les mêmes bases physique.

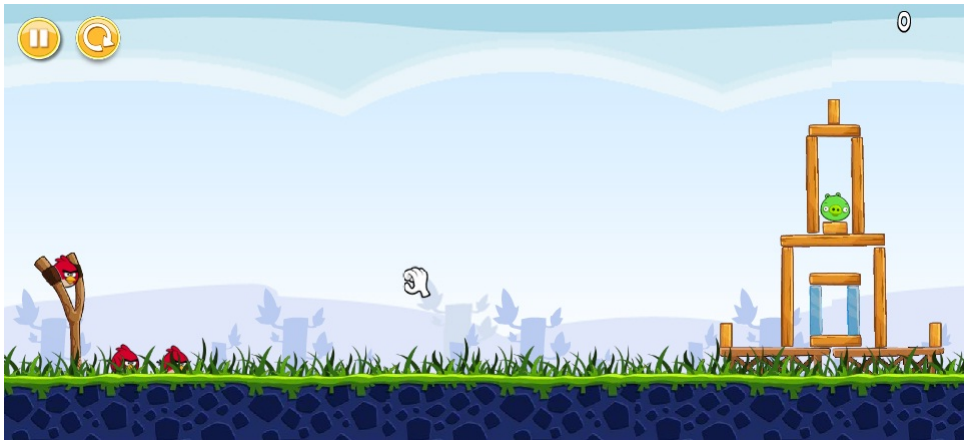

FIGURE 1 – Exemple d'un niveau d'Angry Birds

On se base sur le lancé d'une balle pour détruire un ennemi génèré aléatoirement avec du vent et un sol différent pour chaque partie afin de durcir le jeu.

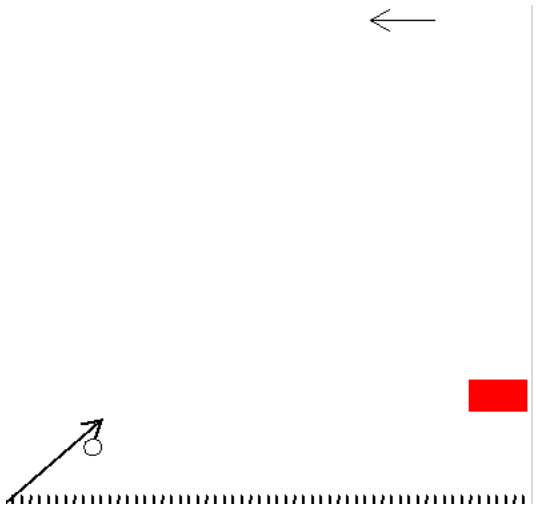

FIGURE 2 – Exemple d'une génération aléatoire de flyingTurtles (herbe au sol)

II Simulation du lancé d'une tortue

2.1) Lancé d'une tortue

— *Cas d'un lancé sans frottements :* on considère les constantes m (la masse de la tortue) et la constante de gravité g=9,81m/s. Les conditions initiales sont entrées par l'utilisateur en cliquant avec la souris : v_0 la vitesse initiale en m/s et α l'angle de lancer initial.

On a donc la situation initiale suivante :

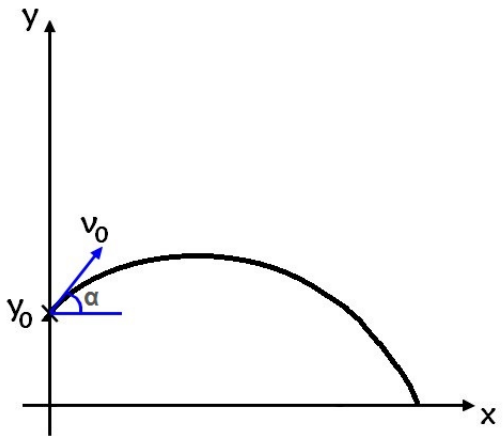

FIGURE 3 – Conditions initiales d'un lancé de balle

Pour évaluer la trajectoire de la tortue, nous simulons sa course en utilisant la méthode d'Euler et le principe fondamental de la dynamique :

1) On connaît la position et la vitesse de la tortue à l'instant présent (les conditions initiales sont déterminées par le jeu et l'utilisateur à l'instant t_0 et par le simulateur ensuite).

2) On applique le PFD dans cette situation pour connaître l'accélération de la tortue :
$\sum \vec{F} = m.\vec{a}$
Ici les deux forces que nous considérons sont le vent et la vitesse v_0.

3) On actualise la position de la tortue avec sa vitesse :
$\vec{x_1} = \vec{v_0}.\delta t + \vec{x_0}$
$\vec{x_0}$: vecteur position à l'instant t_0
$\vec{x_1}$: vecteur position à l'instant t_1
$\vec{v_0}$: vecteur vitesse à l'instant t_0

4) On en déduit la vitesse à l'instant t_{n+1} : $v_1 = a_0.\delta t + v_0$
Nous devions ici choisir entre actualiser la position avant ou après avoir actualisé la vitesse. Il nous a semblé plus logique de le faire avant pour que toutes les valeurs à l'instant t_1 dépendent des valeurs à l'instant t_0 et pas un mélange des deux.

2.2) Les rebonds

Pour simplifier la gestion des rebonds, nous avons considéré qu'ils pouvaient être modélisé de la façon suivante, lorsque la balle touche le sol, sa vitesse est modifiée de sorte que $v_1 = -R * v_0$. On considère un coefficient de rebond R positif entre 0 et 1 ne dépendant que du sol et modélisant l'absorption de l'énergie lors du rebond.
Ce revêtement de sol peut être de trois types différents : herbe, béton, boue. Les coefficient de rebond sont repartis comme ceci :

Type de sol	Coefficient de rebond
Boue	R = 0.4
Herbe	R = 0.6
Béton	R = 0.95

TABLE 1 – Coefficient de rebond en fonction du type de sol

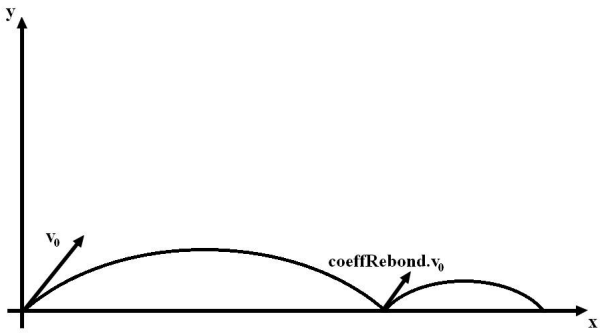

FIGURE 4 – Rebondissement de la tortue

2.3) Modélisation de l'ennemi

La position de l'ennemi est choisie de manière aléatoire par le programme. Sa position est déterminée de la manière suivante :

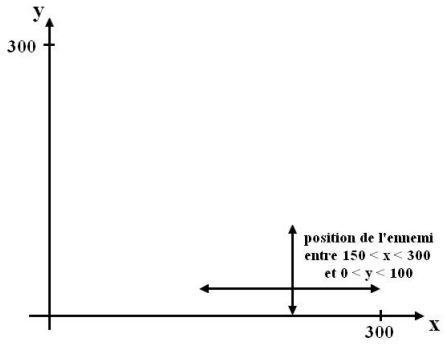

FIGURE 5 – Position de l'ennemi selon x et y

Et sa taille comme cela :

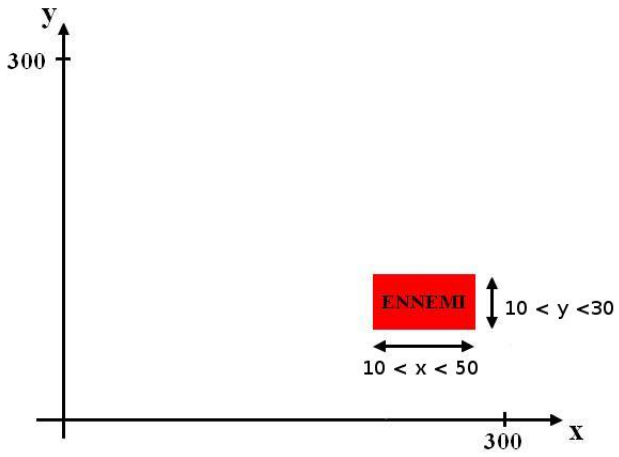

FIGURE 6 – Taille de l'ennemi selon x et y

III Création du score

Le but d'un jeu vidéo est de pouvoir se confronter à d'autres adversaires en comparant leurs scores. C'est donc pour cela que nous avons réaliser un score pour chaque lancer de balle. Le score est inversement proportionnelle à la distance minimal entre la balle et l'ennemi lors d'un lancé. Si la balle touche l'ennemi alors la partie est arrêté et le joueur gagne le maximum en score soit 500 plus la somme des autres scores réaliser aux tours précedents. Le score final est la moyenne de tous les scores précedents. Ainsi chaque joueur peut comparer son score à d'autre sachant que le score max est de 500 et il peut être atteint seulement en touchant l'ennemi au premier lancé.

IV Architecture du programme

2.1) Architecture trois tiers

Afin d'éclaircir le programme, de le rendre plus lisible et plus facile de compréhension, nous avons décidé de créer une architecture pour notre programme. On utilise l'architecture trois tiers : une couche présentation, une autre métier et la dernière données. Ceci permet en effet une meilleure programmation et une plus grande modularité du code.

La couche présentation est relative à toute la partie graphique du programme, à l'interface et à toutes les données que rentre l'utilisateur.
La couche métier correspond au traitement des données, calculs et logique de l'application (lancer une partie, arrêter le programme ect...). C'est l'intelligence du programme.
La couche données est faite pour l'enregistrement des données.

La couche présentation et la couche données ne communiquent chacune qu'avec la couche métier.

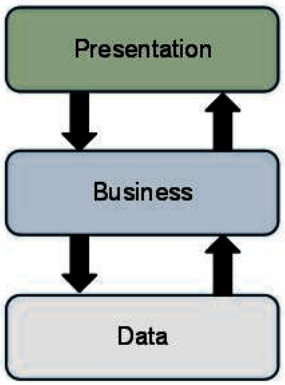

FIGURE 7 – Représentation de l'architecture Trois Tiers

2.2) Diagramme de classe

Le programme n'est qu'avec des objets, c'est de la programmation orientée objet. Nous pouvons donc présenter le programme sous la forme d'un diagramme de classes.

2.2.1) Les classes de la couche métier

Main : entrée du programme
Jeu : Classe responsable de l'orchestration du jeu en lui même. Cette classe centralise les options du jeu (que peut modifier l'utilisateur), ses paramètres (difficulté via la position de l'ennemi, la taille du terrain, les valeurs associées au vent, à la gravité, au sol, etc.), vérifie les conditions de victoire et de défaite, etc.
Options : Classe centralisant l'ensemble des options modifiables du jeu.
SimulationPhy : Classe responsable de la partie simulation physique. Dans les jeux vidéo, on appelle cette partie le moteur physique et il est responsable de la façon dont se déplacent et évoluent les objets dans le monde du jeu.
Objet : Classe représentant un objet du monde du jeu (en l'occurence, l'ennemi et la tortue).

2.2.2) Les classes de la couche présentation

Interface : Classe responsable de la communication avec le joueur via la console (message et options).
Fenetre : Classe permettant l'affichage et centralisant toute la partie graphique liée à la librairie root.

2.2.3) Les classes de la couche données

Trace : Classe permettant d'enregistrer la position et la vitesse d'un objet à chaque instant de la simulation dans un fichier texte.

2.3) Interêt de l'architecture

Les intérêts de ce découpage et de cette architecture sont multiples. Le premier intérêt et la modularité et la réutilisabilité du code, le second est la possibilité de partager le travail.
Dans notre cas, les classes SimulationPhy et Objet peuvent être en théorie réutilisées pour n'importe quelle simulation de cinématique. Et le découpage des fonctions permet également de modifier simplement les méthodes de calcul et de simulation. On a utilisé ici la méthode d'Euler mais on pourrait aisément ajouter la méthode RK4 pour la remplacer. De même on pourrait simplement faire les calculs à l'aide d'une librairie spécialisée comme armadillo ou root. Ces changements ne demanderaient chacun que l'ajout ou la modification d'une fonction.
Pour ce qui est de la répartition du travail, avec notre code une personne pourait travailler sur l'interface pendant qu'une autre travaillerait sur le simulateur et la troisième sur le jeu par exemple.
Enfin, l'architecture et le découpage propre du programme permettent de mettre en évidence les algorithmes du programme et ainsi de plus facilement voir leurs avantages et leurs défauts. Ils permettent également un débogage plus facile.

V GitHub

Afin de partager notre avancement sur le projet ou que nous soyons, nous avons décidé d'utiliser git et le site Github.

Git permet de versionner du code ou des documents, c'est-à-dire que chacun peut ajouter des modifications, et git permet de considérer chacune d'elle comme une nouvelle version du programme ou du document. On peut ainsi fusionner les versions quand les modifications portent sur la même partie du code, voir qui a fait quoi, retirer une modification qui s'avère mauvaise, retrouver un changement pour traquer un bug, etc.

Github est un serveur web permettant aux utilisateurs de git de centraliser et partager leur code. Ce site permet pour de petit projets de se passer de la mise en place d'un serveur disponible en permanence. Il permet aussi de diffuser et partager son projet sur internet. L'adresse de notre projet est la suivante : https://github.com/mboutserin/flyingTurtles.git

VI Conclusion

Lors de ce projet nous voulions réaliser un vrai jeu avec de beau graphisme mais cela nécessite énormément de temps, ce qui nous a manqué. On aurait voulu utliser la bibliothèque Qt qui est une ressource très largement utilisée dans le développement de jeux vidéos ou bien de software car elle permet une grande modularité au niveau graphisme. Notre jeu s'appellant flyingTurtles, nous aurions voulu faire voler des tortues au lieu de lancer des balles.
Cependant, ce projet nous a permis de développer notre connaissance dans le domaine de programmation C++. Nous voulions un projet mélangeant à la fois un domaine physique et esprit de jeu vidéo. Nous aurions voulu plus de temps afin de réaliser un jeu fini.