Rapport du Projet 1
Semestre 2

CRÉATION D'UN JEU DANS LE CADRE
D'UN TOURNOI D'INTELLIGENCES ARTIFICIELLES
PROGRAMMATION EN C

Nom du groupe : Super Saiyan God | Nom du projet : SEN PROJECT | 11/03/2018
Sami Heddid
Emile Paumelle
Nathan Fontaine
// PSBN Promo 2022

I – INTRODUCTION
II – PRESENTATION DU JEU
III – ELEMENTS HORS-PROGRAMME UTILISES
IV – DETAILS DE CHAQUE FONCTION
V – EXPERIENCE ACQUISE
VI – LIEN VERS TRAILER

I- INTRODUCTION :

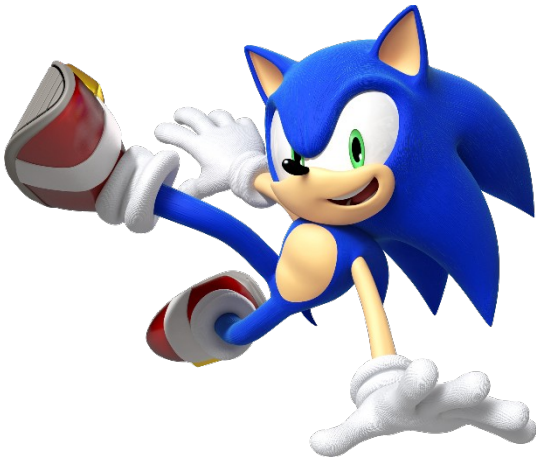
- Dans le cadre du premier projet du Semestre 2, nous avions à créer un jeu afin qu'une centaine d'intelligences artificielles s'affrontent sous forme d'un tournoi. L'arène (l'environnement du jeu lui-même) la plus intéressante (en termes de « combat » entre IA, de beauté du jeu en soi, dynamisme et de simplicité des règles) remporte une petite victoire sur ce qu'il y a à venir : chaque groupe de promo devra alors créer une IA sur le jeu élu, et l'IA la plus forte remporte le tournoi.

- Plusieurs possibilités (vis-à-vis des IA) pour les groupes sont possibles du moment qu'elles respectent les règles du jeu élu ; notamment l'aspect de parier sur certaines IA gagnante, rapportant des points bonus (ou malus) pour les parieurs.

→ Ce projet nous a permis d'aborder certains aspects de la programmation en C jamais vu (SDL, FMOD) et de connaitre quelques aspects de la future vie d'ingénieur TIC : deadline assez courte (bien évidemment de notre point de vue), apprendre certaines notions « dans le tas » qu'on n'avait pas forcément vu avant cela et également de nous conforter dans le choix que nous avons fait en choisissant cette voie.

II- NOTRE JEU, THE SEN PROJECT :

VOUS N'AVEZ JAMAIS PENSE A UN SONIC MANGEANT L'UN DES FAST-FOOD LE PLUS CONNU AU MONDE ? NOUS L'AVONS IMAGINE POUR VOUS !



Le but est en réalité très simple : nous avons plusieurs « skins » de Sonic éparpillés dans la carte (on rappelle qu'il peut y en avoir une centaine), chacun représentant une IA d'un groupe. Les IA s'affrontent entre elles en mangeant le plus de nourriture possibles (mise sous forme de frites et hamburgers) et en évitant les obstacles et les autres IA. L'IA ayant le plus grand score (déterminé par le nombre de nourriture qu'il a mangé) remporte la partie. Pour cela, les groupes devront s'adapter à l'environnement du jeu afin de perfectionner leur IA au maximum dans le but de gagner le tournoi.

Le jeu s'illustre sous forme de SDL accompagné de la musique via FMOD, ce que nous détaillerons plus tard dans le rapport. C'est en fait un tableau 2d où Sonic peut se déplacer en abscisse et en ordonnée (x et y) de 4 manières différentes : HAUT, BAS, DROITE, GAUCHE.

III- ELEMENTS HORS-PROGRAMME UTILISES

a) LA SDL.

Il existe plusieurs types d'affichages en C. Nous avions toujours l'habitude d'utiliser la console (écran noir, cf Color Wars) afin de réaliser des projets. Ce dernier étant relativement libre, nous avons décidé de nous orienter vers un design plus propre afin d'attirer les autres groupes vers notre jeu. Nous avons appris à utiliser la SDL via OpenClassroom qui nous expliquait explicitement quoi faire vis-à-vis de l'affichage.
D'habitude, pour ouvrir un projet dans CodeBlocks, nous utilisions la « console d'application », pour la SDL qui est un projet entièrement à part de la console, nous devions bien évidemment cliquer sur « SDL ». En effet, et ceci n'est qu'un exemple, ***nous pouvons pas « printf » en SDL. (à moins de changer quelques propriétés)***

Détaillons quelques fonctions principales qui nous ont permis de pouvoir afficher la fenêtre.

- SDL INIT VIDEO : constante servant charger le système d'affichage vidéo.
- SDL INIT : charge la SDL grâce à des allocations de mémoire.
- SDL SETVIDEOMODE : Chargement de l'affichage en prenant en compte Hauteur, Largeur, Bits, Flags.
- SDL_FULLSCREEN : mode plein écran.
- SDL HWSURFACE : chargement de la mémoire rapide de l'affichage vidéo.
- SDL_Quit : ferme la SDL.
- SDL_WAITEVENT : attente d'évènement.
- SDL SURFACE : charger la surface de l'écran.
- SDL MAP RGB : Chargement des couleurs de la surface.

De nombreuses autres constantes ont été utilisées vis-à-vis de la SDL, ce que nous détaillerons dans la prochaine partie à venir.

b) LA FMOD.

Comme dit précédemment, la FMOD nous a servi à implémenter de la musique pour dynamiser le gameplay. (on a mis 3 musiques au total.)
Nous l'avons choisi car des bibliothèques comme SDL_audio ou SDL_mixer s'avèrent être compliqué à utiliser d'après OpenClassroom.
Voici quelques fonctions que nous avons utilisées pour manipuler la FMOD.

- FMOD_SOUND : représente le son grâce à un pointeur.
- FMOD SYSTEM CREATE SOUND : charge le son grâce à plusieurs paramètres: l'objet « system », le format de la musique, et un flag.
- FMOD SYSTEM PLAYSOUND : Joue la musique.
- FMOD SET PAUSE : mets le son en pause ou non.
- FMOD SOUND RELEASE : libérer l'espace prise par la musique.
- FMOD SOUND CLOSE : fermer la musique.
- FMOD SOUND SETLOOP : répéter la musique à l'infini.

(Bien sûr, nous n'avons pas tout détaillé car le programme est lui remplit de commentaires sur les différentes fonctions, constantes et variables qui ont été utilisé.)

Lançons-nous désormais dans le vif du sujet, le code lui-même. Ce dernier est composé de plusieurs parties (headers, .c...), délimités par des fonctions.... Le code est peut-être sous SDL, mais la « source », elle, reste la même. En effet, l'utilisation des boucles for pour initialiser un tableau reste pareil, le menu est aussi fait de la même manière (case, break..).

Le code est divisé en 4 : 2 headers pour mettre le nom des constantes et la structure de certaines variables, (en effet, le code serait abominable à lire si nous avions mis tout dans un seul .c, les headers servent à espacer le projet et le rendre plus « fluide » d'une certaine manière. Les 2 .c servent eux du code principal : un .c sert au menu, l'autre au jeu lui-même.

En route !

IV – DETAILS DE CHAQUE FONCTION

Avant même de commencer à détailler celles-ci, il faut d'abord définir les petits fichier qu'on appelle Header.

Un Header est un FICHIER (dit .h) qui nous permet d'espacer le programme, en effet, cela n'est pas pratique de voir un code qui fait directement plusieurs centaines de lignes sur un seul main.c . Dans la plupart des cas possibles en terme de programmation, les **headers contiennent les prototypes des fonctions**, ce qui est bien sur le cas pour le notre.

HEADER 1 : CONSTANTE.H

Commençons par celui-ci, ce header nous sert, comme son nom l'indique, à définir les constantes utilisées pour le jeu. Pour gagner de la place, on utilise #define qui va justement définir les constantes pour ne plus les réinitialiser. Ce header sera ensuite inclut dans le jeu.c grâce à la bibliothèque #include qui va en réalité prendre le fichier pour que les constantes soit incluses dans le .c

Un point important à mentionner est à propos des conditions #ifndef et #endif. En effet, on les utilise pour les define telle que : (petite traduction algorithmique)

Si (CONSTANTE = vrai) alors

condition1, 2... → POSSIBLE

(dans ce cas, les « requis » sont les constantes.)

Pour finir, on se penche sur une des structures du C qui est le enum : en effet, dans le C, on peut créer ses propres types de variables. C'est grâce à cette structure de variable que nous pourrons justement déplacer le Sonic, car, comme explicitement dit dans le code, enum prend en fonction 4 paramètres : HAUT, BAS, DROITE, GAUCHE. Elle sera d'autant plus importante pour les IA car les programmeurs vont en réalité coder à partir de ces coordonnées si.

HEADER 2 : jeu.h

Les prototypes de fonctions sont dans ce header-ci. Nous allons bien évidemment les détailler plus tard dans le rapport, mais il faut noter qu'on inclut le header CONSTANTE: cela est naturel car les fonctions dépendent des constantes définis précedemment.

A noter également que l'on nomme nos fonctions grâce à la directive void : cet outil sert à ne pas rendre aucun résultat, comme sa traduction pourrait l'indiquer. (void = néant)

Le point le plus important à noter ici est l'utilisation de ***« struct joueur ».*** C'est littéralement la strucutre clé du code. Nous nous expliquons : comme on l'a dit, le C peut permettre aux développeurs de créer nos propres types de variables, et cela grâce à différents critères. Struct est en réalité un élément nous permettant de faire une variable à sa guise, ici donc, le joueur.

Qui dit structure dit évidemment une certaine syntaxe vis-à-vis de notre nouvelle variable créer. Effectivement, nous avons fait dépendre la variable de 3 critères : sa position en abscisse, sa position en ordonnée, et enfin le point sur lequel il se situe. Nous avons également utilisé SDL_Rect qui est la variable nous aidant à indiquer les coordonnées de l'écran à « blitter »(remplir) sur l'écran. En effet, ici Rect va dépendre de la variable position (x et y) du Sonic, nous verrons dans le jeu.c que l'on utilise en réalité un pointeur vers position pour déterminer ses coordonnées (en effet, vu que nous voulons sa position, il faut utiliser son adresse.)

Pour finir dans ce dernier header, nous devons parler de la SDL_Surface qui est en fait le « noyau » de la SDL : c'est en autre grâce à elle qu'il y a une surface dans l'écran pour y mettre diverses choses. (on dit qu'elle la charge.) Bien sur ici, on va pointer une variable que nous nommons explicitement IA qui va elle dépendre des 4 paramètres vu précedemment grâce à enum ; cela pour y affecter sa position grâce au pointeur mis dans la variable ia.

L'un de nos fichiers principales est bien évidemment mis sous forme de .c, on va directement présenter notre main qui est en réalité la présentation de notre menu mis sous forme de SDL. Le menu se fait directement sous forme de case/break avec jonctions entre FMOD et SDL.

Méthodes d'approche :

Pour coder ce jeu, nous nous sommes inspirés du Color Wars qui lui mets en avant 4 IA ainsi de remporter une bataille navale. De plus, nous nous y sommes pris 2 semaines à l'avance de la séance de lancement de projet afin d'acquérir assez de bagage technique vis-à-vis de la SDL. Chaque membre devait fournir un travail spécifique concernant le code. Nous nous sommes également inspirés de vidéos et de conseils sur Internet à propos de comment un IA doit agir selon les différents jeux possibles, cela nous a grandement aidé dans la mesure où aucun d'entre nous ne savait vraiment pas comment s'y prendre. A la fin, nous étions en symbiose complète avec le travail que nous fournissions et cette coopération nous a permis, avec les leçons déjà acquises (tableau 2d, notions OpenClassroom...) de fournir ce travail ici présent.

NOS FONCTIONS ;

Dans les fonctions que nous avons codées les noms de variables utilisées sont toutes explicites.

FONCTION JEU :

Comme dis précédemment Player est une structure de variable qui nous permet de faire déplacer Sonic de 4 manières différentes: Haut, Bas, Gauche, Droite. On initialise la carte qui a de nombreux paramètres préalablement définit dans les headers. Bien évidemment nous utilisons dans cette fonction **la SDL: SDL** surface va nous permettre de charger l'écran en pointant sur l 'adresse de l'ia pour que cette dernière se déplace. Le dynamisme de la SDL s'accompagne de plusieurs évènements:

SDL EVENT va alors nous servir à crée un événement afin qu'il perdure dans l'affichage; cela se traduit par une autre utilisation de SDL RECT qui va prendre en paramètre la position du joueur vis a vis de l'obstacle. Selon la position de l'IA l'image doit changer c'est pourquoi on utilise IMG_load qui permet d'assigner une image a chaque direction . De même pour les frites, les obstacles et les burgers qui sont affiché grâce a IMG_load. On utilise SDL_SetColorKey pour la transparence des IA pour que Sonic soit transparent au VIDE.

Nous avons ensuite utiliser 2 boucles for imbriqué pour parcourir les case du tableau et les remplir de noir afin de crée un fond noir derrière le Sonic. SDL_FillRect est utilisé pour remplir de noir l'écran. L'ia apparaît sur l'écran en abscisse 5 et en ordonnée 3. De plus l'image du sonic est orienté vers le bas au départ. Deux boucles for imbriquées sont réutiliser afin de sauvegarder la position du joueur(son abscisse et son ordonnée). On utilise while(continuer) ,continuer est la variable nous servant à continuer l'évènement.

C'est une boucle dite INFINI.

On initialise un switch avec plusieurs cas : nous avons utilisé «event.key.keysym.sym» nous permettant de savoir quelle touche est enfoncée au moment où le joueur est sur

une case. Pour cela, nous avons décidé d'utiliser une condition «si» afin de placer le joueur sur un point pour qu'il prenne par exemple une frite ou un hamburger. A noter les -1 et +1 nécessaire pour la position du joueur car par exemple si la position est orienté vers le haut, l'ordonnée dans le tableau prendra -1 donc la position du joueur augmente de 1. Évidemment, on break si le joueur rencontre un obstacle car ce n'est pas une nourriture. Même chose pour les autres cas et directions.

On utilise timerApparition afin de faire apparaître les frites ,obstacles et burger a même intervalles de temps. DE même pour leurs disparition

Deux boucles for imbriqué sont utilisé pour parcourir le tableau ensuite on utilise SDL BlitzSurface pour remplir le VIDE de frite, burger, obstacle et de sa position.

SDL_FreeSurface permet dans notre cas de libère la surface sur les différentes variables.

Pour finir on utilise une boucle for et encore une fois SDL_FreeSurface pour libérer la surface prise par l'IA pour pas avoir des données en masse .

LA FONCTION MOVE PLAYER :

Cette fonction a pour utilité de permettre aux joueurs de se déplacer sur la carte comme son nom l'indique. Elle prend pour paramètres la carte c'est-à-dire sa longueur et sa largeur puis la direction et SDL_rect indiquant les coordonnées grâce à un pointeur ici nommé position pour remplir la surface (la blitter). On utilise la condition switch avec la variable principale direction, permettant de bouger le joueur: la direction dépend de deux paramètres ici nommé x et y. Dans le switch on énumère 4 cas selon la direction souhaité: Bas, Haut, Droite, Gauche. Comme il s'agit d'un tableau ayant pour origine la case (du tableau) en haut à gauche. Lorsque le joueur joue vers le HAUT les coordonnées de y diminue de 1. C'est pourquoi dans la case HAUT, on utilise une condition si qui permet de ne pas dépasser la taille du tableau définis. Pour le cas du HAUT il faut que la position->y – 1 soit inférieur à 0.Si cette condition est bien vérifié on peut donc effectuer le déplacement du joueur d'une case plus haut dans le tableau. On réalise la même chose pour la direction bas seulement les coordonnées augmentent de 1.Pour la direction droite et gauche c'est l'élément x qui varie: Pour la gauche x diminue de 1 et pour la droite il augmente de 1 .

LA FONCTION APPARITION FRITE :

Cette fonction permet l'apparition des frites sur la carte. L'apparition des frites se réalisent de manière hasardeuse grâce à la fonction rand, et il peut y avoir 100 frites qui peuvent apparaître au total. On utilise 3 boucles for imbriqué, la première nous permettant de choisir une position aléatoire en x et en y dans le tableau. Les deux autres parcourent le tableaux et une condition if vérifie si les coordonnées de la frite est bien dans le tableau. On remplace la case vide correspondant aux coordonnées et on la remplace avec l'image de la frite.

FONCTION APPARITION FRITE :

Cette fonction comme son nom l'indique permet de faire disparaître les frites de l'écran. Elle prend en paramètre la hauteur et la largeur du tableau. On initie tout d'abord le nombre de frites disparaissant. On souhaite faire disparaître un nombre aléatoire de frites pour cela on réutilise rand. La première boucle for permet de choisir des coordonnée aléatoirement .Ensuite la deuxième boucles for et la troisième nous permettent de parcourir le tableau grâce au variables itératives j et k et de vérifier si les coordonnées trouvés correspondent à une case occupée par une frite. Si cette condition est vérifiée on remplace l'image de frite par une case vide.

FONCTIONS APPARITION BURGER, DISPARITION BURGER, APPARATION OBSTACLE

Ces deux fonctions sont a peut près similaire aux deux dernière puisque le principe est le même. Dans notre jeu les burgers valent plus de point pour les joueurs. Le seul élément qui diffère est le nombre d'apparition des burgers sur la carte. Sachant qu'ils sont plus avantageux pour celui-ci le nombre de burger est donc restreint.

V – EXPERIENCES ACQUISES

Il a fallu travailler **rapidement et efficacement** à la fois, dans la mesure où nous étions très pris de vitesse par tout ce qu'il nous restait à faire et cela hors-Programmation en C. Les méthodes de travail étaient également réfléchit très promptement, surtout sur la répartition du travail. Par exemple, l'un s'est occupé de la FMOD, un autre de la SDL... et le groupe sur le code lui-même. Nous avons rencontré des problèmes sur notre idée source de base qui était la SDL elle-même, avec un peu de détermination, nous avons pu installer SDL et FMOD et nous lancer sur le projet qui lui avait également de nombreux soucis à régler. *(Raison pour laquelle nous avons décidé de répartir celui-ci en plusieurs headers et .c)* ------

Si la répartition des taches fut en quelque soit compliqué, le travail lui-même était ***un réel chaos*** *entre le code, la SDL... l'idée elle-même a mis du temps pour se « solidifier », nous étions d'abord parti sur un Tetris, puis un Puissance 4, puis un Pong et enfin ce jeu dont nous sommes fiers du produit.*

Si l'affichage de la SDL était plutôt simple à réaliser (« *simple* », c'est-à-dire dès lors où nous avions compris l'installation pour le faire sur les autres ordinateurs.), faire déplacer une IA et « blitter » (remplir) la surface de différentes images était un sujet complètement à part. Effectivement, il nous a fallu de nombreuses heures d'activité en groupe pour pouvoir faire fonctionner notre code ; nous avions même ~~pour idée d'abandonner le projet~~ mais avec de la persévérance, nous avons réussi à produire quelque chose que nous considérons comme relativement décent et nous en sommes fiers.

Ce projet nous a permis également, et cela est un des points les plus importants, à nous former à s'adapter peu importe « l'environnement » dans lequel nous pouvons travailler, et aussi en dépend de la situation dans laquelle nous sommes (période de « *rush* », *deadline* courte...).

On pense également d'une façon ou d'une autre, ce projet nous a fait murir sur de nombreux d'aspects et qu'il nous a permis de progresser dans le C. Mais aussi sur des points fondamentaux : organisation, méthodes de travail...

Bien évidemment, pour cela, nous avions besoin de deux éléments essentiels : un ***papier et un stylo.***

VI – LIEN VERS TRAILER

Bien sûr, avant même de vous envoyer ce que vous lisez en ce moment même, nous avons fait une présentation en classe pour présenter notre jeu qui s'accompagnait d'un powerpoint et d'une petite bande annonce. Cela étant et vous l'aurez remarquez, le gameplay prit en trailer n'était pas le produit final du jeu.

Lien : https://www.youtube.com/watch?v=30PJ47TKK_g&feature=youtu.be

GOOD GAME !