FIGHTSABERS

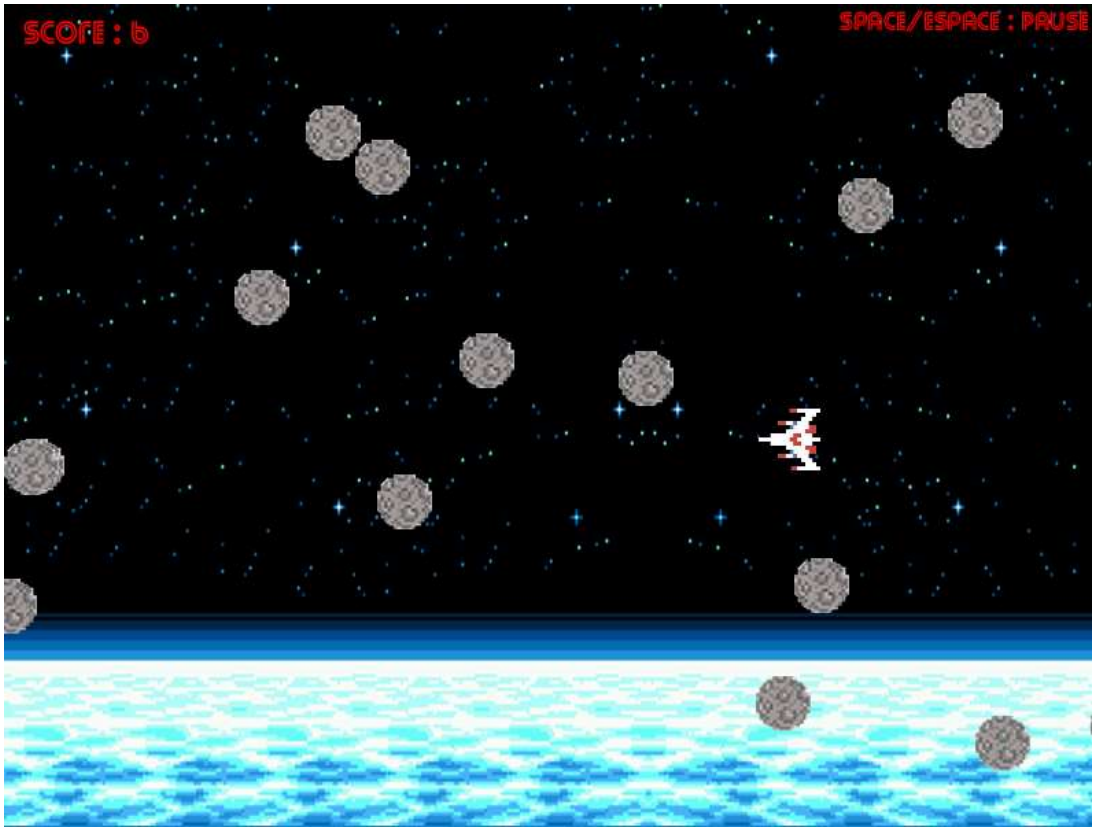
Compte rendu de projet d'ISN

Vincent MARSOT – Clément VELLU

Table des matières

Présentation du projet

Lors de l'élaboration du projet, nous nous sommes tournés vers la réalisation d'un jeu. En effet, l'univers du jeu vidéo nous était très familier, et le projet intermédiaire réalisé en milieu d'année, qui consistait à revisiter le mythique « Space Invaders ».



Ce projet nous a donner envie de poursuivre dans la création d'un jeu. Dans un premier temps nous prévoyions de réaliser un Bomberman. Dans un deuxième temps, nous avons finalement choisi de faire un jeu de tir en 1 contre 1.

Le jeu final, est légèrement différent du projet initial, en effet il ne s'agit plus d'un jeu de tir mais d'un jeu de combat au corps à corps.

Le jeu est axé sur le thème de Star Wars, auquel nous avons emprunté plusieurs aspects phare, notamment le sabre laser (l'objet en lui-même, comme les sons qui lui sont associés), les personnages (Clone Trooper), ainsi que la police d'écriture.

Définition du cahier des charges du jeu

Analyse du besoin : Au cours des premières séances, nous avons définis les fonctionnalités que l'on a jugé incontournables :

- Différents écrans (écrans d'accueil, de jeu, d'options, de crédits, de sortie)

- Déplacements des personnages simultanément au clavier, à la souris et à la manette

- Des graphismes de bonne qualité, tout en gardant un coté rétro

- Une ambiance sonore et des sons qui rappellent le coté rétro

- Un algorithme de collision précis, efficace et peu gourmand en performance

- Un jeu ayant pour thème la science-fiction (rappelant l'univers de Star Wars)

- Un jeu ayant une jouabilité se rapprochant d'un jeu standard

- Algorithme de gestion du temps (compte à rebours)

Le projet avait pour finalité d'être semblable à un jeu rétro que l'on peut trouver aujourd'hui sur le marché.

Afin de réaliser ce projet, nous avons choisi d'utiliser le langage Processing. Ce langage est issu du langage Java.

Java est un langage de programmation, née en 1995. Influencé par le C++, notamment, le langage est orienté Objet. Ce type de programmation consiste à définir et attribuer des caractéristiques à chaque briques logicielles (objets). Ces objets représentent un concept, une idée, une entité physique, … (tel qu'un animal par exemple).

Choisir le langage Processing plutôt que Java nous permettait d'accéder à une interface graphique bien plus facilement et rapidement qu'en utilisant seulement Java. C'est pourquoi nous l'avons choisi.

Nous nous sommes donc naturellement tournés vers l'IDE Processing (Integrated Development Environment).

Dans certains cas, notre connaissance du langage, ne nous permettait pas de répondre à certains de nos besoins. C'est pourquoi nous nous sommes rapidement tournés vers des sites d'informations ou d'échanges à propos de ces langages. Par exemple voici certains sites que nous avons utilisés

- https://forum.processing.org
- https://stackoverflow.com
- https://github.com

Répartition des tâches et démarche collaborative

Le tableau suivant présente la répartition des différentes tâches :

Noms / Fonctionnalités	Vincent M.	Clément V.
Sons	X	X
Graphismes / Gestion des écrans	X	
Options		X
Support Manette		X
Support Clavier/Souris	X	
Algorithme de collision		X
Jouabilité	X	X
Compteur à rebours	X	X

Afin de fonctionner le plus efficacement possible, nous avons exploité les outils qui étaient à notre disposition sur internet. Afin de communiquer nous nous sommes servis de Discord. Cette plateforme met à disposition pour ses utilisateurs, gratuitement, des serveurs vocaux, afin que ceux-ci puissent communiquer.

Enfin, afin de favoriser le partage des tâches, nous avons utilisé GitHub. Cette plateforme, fondée en 2008 (originellement connu sous le nom de « Logical Awesome LLC »), par le créateur de Linux, Linus Torvald, a pour but de faciliter le développement collaboratif, en proposant d'une part un outil d'historique et de gestion des versions.

D'autre part il propose un espace de stockage sur lequel plusieurs personnes peuvent travailler en même temps.

Cela est rendu possible grâce à un système de branche. A chaque fonctionnalité en cours de développement est associé une branche. Lorsque la fonctionnalité est terminée, elle est fusionnée au code principal.

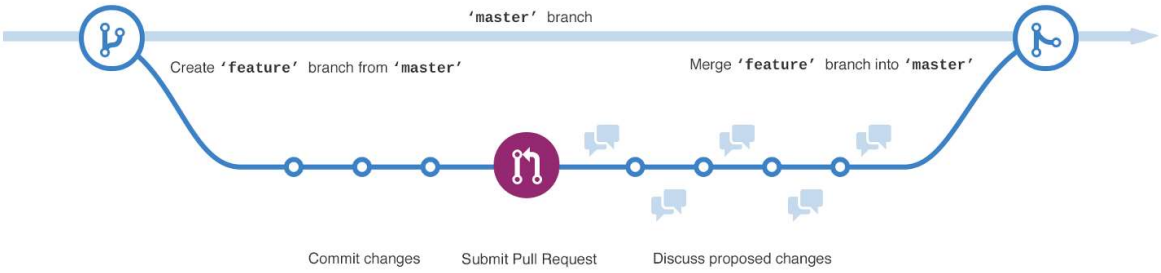


Schéma du principe de branche

Cet outil nous a été précieux, car il nous a permis de développer plusieurs fonctionnalités simultanément, ce qui a permis un gain de temps remarquable.

La plateforme nous a également permis de récupérer des « librairies ». Celles-ci sont des morceaux de code qui apportent leur lot de fonctionnalités, déjà prêtes à être utilisées.

Nous avons choisi, dans certains cas, d'utiliser des « librairies ». En effet, elles permettent de gagner du temps sur des fonctionnalités qui ne sont pas au centre du programme.

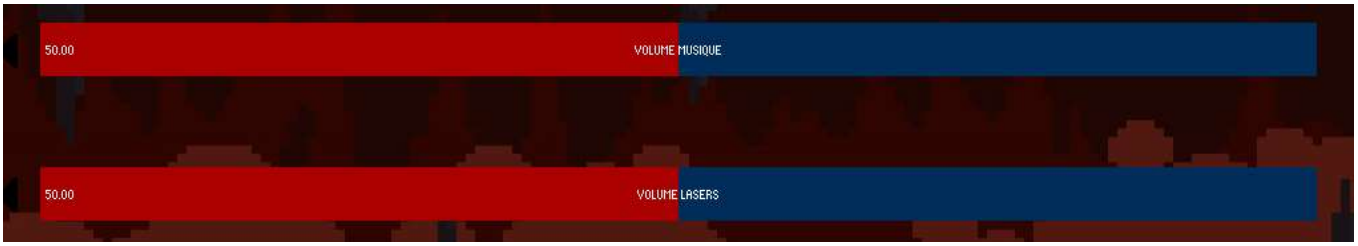
Réalisation des solutions

Options et barres de réglage

Un des critères du cahier des charges était de réaliser un menu des options dans lequel on devait pouvoir régler les volumes des différents éléments. (Les musique, les sons, …)

La meilleure solution que nous avions envisagée était l'utilisation de barres de réglage, type « Slider ». En effet elles permettent de régler une variable sur un intervalle défini. Ce qui était idéal pour notre projet, notamment pour le réglage du son (de 0% à 100%)

Pour mettre en place cette solution, nous avions dans un premier temps envisagé de créer des barres nous-même, et en interagissant avec les fonctions **mousePressed**() et **mouseRelease**(), récupérer les valeurs des différents paramètres.
Malheureusement, nous nous sommes rapidement rendu compte que la création de telles barres allait être longue et fastidieuse. Entre temps, nous avions recherché un autre moyen de le faire.
Finalement, nous avons décidé de le faire avec la librairie **cp5.**
Cette librairie permet d'importer et d'utiliser des barres, des boutons, des listes déroulantes, … de différents types. Nous avons décider de n'utiliser que les barres type « Slider ».



Barres de réglage

```
cp5.addSlider("Volume musique")
   .setPosition(width>>2,height*0.4)
   .setSize((int)(width*0.5),40)
   .setRange(0,100)
   .setValue(50)
   .setVisible(false)
   .setColorLabel(#FFFFFF)
   .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);
```

Initialisation des barres

Ces barres sont définies et initialisées dans le **setup**(). Elles permettent la gestion de plusieurs paramètres.
Leur utilisation est très simple, le joueur maintien le clique appuyé et déplace jusqu'à la valeur souhaité et relâche la souris. Ici la barre correspond au réglage du volume de la musique. La position, la taille, l'intervalle, la valeur initiale, la visibilité, la couleur et l'emplacement du nom de la barre sont définis pour chaque barre.
L'accès aux différents écrans a été fait par Vincent. Celui-ci permet de naviguer entre les différents écrans. Lorsque l'écran des options est affiché, on affiche les différentes barres de réglage.

```
cp5.getController("Vitesse Personnage 1").setVisible(true);
cp5.getController("Vitesse Personnage 2").setVisible(true);
cp5.getController("Volume musique").setVisible(true);
cp5.getController("Volume lasers").setVisible(true);
cp5.getController("Volume bruits").setVisible(true);
cp5.getController("Temps de Jeu").setVisible(true);
```

Affichage des barres

Lors de l'affichage des options certaines valeurs sont récupérées et actualisées dynamiquement grâce à la fonction **updateOptions**(). Celle-ci récupère les valeurs des barres de réglage des volumes à chaque fois que le **draw**() est appelé. Elle redéfini également les valeurs des variables de volume en même temps.

Certaines options ne sont, elles, pas actualisées dynamiquement et sont ainsi récupérée et redéfinie lorsque l'on quitte le menu dans une fonction prévue à cet effet.
Je me suis occupé d'utiliser la librairie, de créer, placer et gérer l'intégration des barres de réglage dans le jeu comme dans le code.

Gestion du son

Le cahier des charges prévoyait également l'utilisation de musiques et de sons. Lors de nos recherches sur internet, nous avons découvert une librairie développée par les bénévoles de processing, « sound ».

Comme son nom l'indique, elle répond parfaitement à notre cahier des charges puisqu'elle permet de jouer des sons, comme nous le désirions.

```
lightSaber1 = new SoundFile(this, "LightSaberHit1.mp3");
lightSaber2 = new SoundFile(this, "LightSaberHit2.mp3");
MusiqueAJE = new SoundFile(this, "MusiqueAJE.mp3");
MusiqueOptions = new SoundFile(this, "MusiqueOptions.mp3");
MusiqueCredits = new SoundFile(this, "MusiqueCredits.mp3");
```

Chargement des sons

Les sons sont chargés, et initialisés en tant qu'objet au lancement du programme dans le **setup**(), puis sont joué sous certaines conditions que nous avons imposées.

Les sons sont joués en appelant la méthode .**loop**() (le son est joué en boucle), ou la méthode .**play**() du son correspondant. Nous avons rencontré quelques difficultés avec cette librairie. En effet, lors du chargement de fichiers sons sous certain format, le jeu pouvait parfois planter. Nous avons résolu le problème en changeant le format du fichier en .mp3

Un autre problème s'est opposé au bon déroulement du projet : les sons joués en boucle ne l'étaient pas toujours. Après quelques recherches, nous nous sommes rendus compte qu'il s'agissait d'un problème de canal de son. Il fallait effectivement convertir les sons diffusés sur plusieurs canaux (stéréo), en sons diffusé sur un seul canal (mono).

Support manette

Un autre point du cahier des charges précisait la possibilité de jouer avec une manette. Ici encore, nous avons fais appel à une librairie. Il s'agit de « GameControlPlus ».

Le but était de récupérer les valeurs de chaque axe montré ici et de transcrire cette information en un déplacement ou un changement de l'axe de visée du personnage.

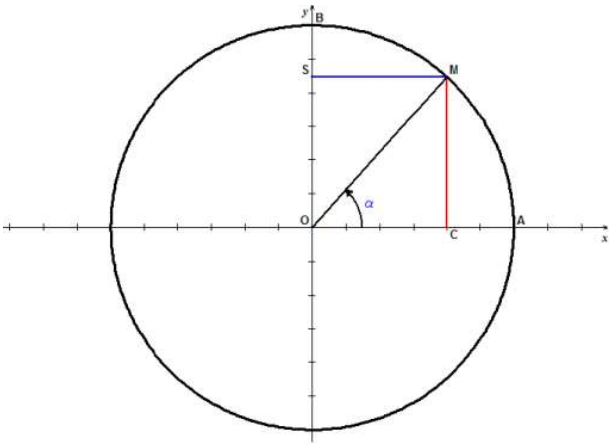
J'ai réussi sans réelle difficulté à faire la partie mouvement. Celle-ci s'explique plutôt simplement : on récupère les valeurs des axes de déplacement, puis ces valeurs servent de multiplicateurs à la vitesse de déplacement du personnage, ces premières étant comprise entre -1 et 1.

```
posXGamePad = gpad.getSlider(RightLeft).getValue();
posYGamePad = gpad.getSlider(UpDown).getValue();
```

Les valeurs se récupèrent ainsi, en indiquant l'axe voulu.

Ensuite, pour des contraintes de sortie de l'écran, on ne s'intéresse qu'à la valeur absolue de ces valeurs, puis en fonction du signe initial, on en déduit le sens du mouvement. La valeur absolue nous sert de coefficient multiplicateur. Ainsi, on peut se déplacer à différentes vitesses.

La partie visée, en revanche, m'a demandé un peu plus de travail.



En imaginant que le stick peut être déplacé au sein d'un disque de rayon 1, on réalise que les valeurs lues par les fonctions **getValue**(), nous donnent les projections sur les axes x et y du point où le stick est déplacé.

On peut déduire de ces valeurs la valeur de l'angle α grâce à la fonction réciproque de cosinus, et du signe de la projection sur l'axe y. On obtient un angle compris entre π et – π

Pour tracer le laser, on déduit l'abscisse du point d'arrivée de la ligne dessinée grâce au cosinus de cet angle et l'ordonnée grâce au sinus de cet angle.

Le jeu détecte également lorsqu'aucune manette n'est branchée car la librairie permet de faire une liste des périphériques compatible et leur attribue un type. Si aucun périphérique de type « Gamepad » n'est détecté, le jeu se lance en mode « Sans manette » et affiche un message.

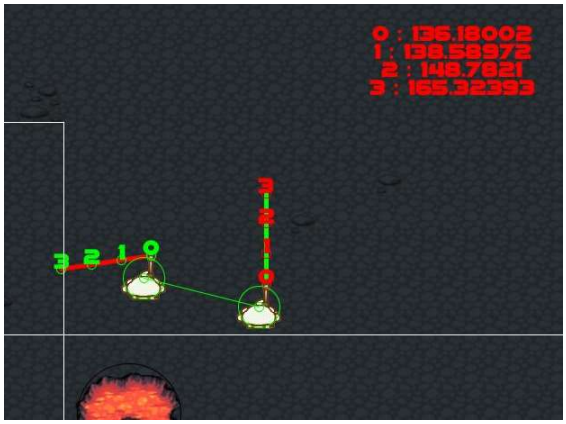
Algorithme de collision

L'algorithme de collision est séparé en deux partie. La première s'occupe de la collision entre le personnage et le décor. Des zones correspondantes aux zones de morts sont définies. Si le centre du personnage entre en contact avec les bordures de ces zones, il meurt et est replacé aléatoirement.

Les zones de lave au milieu de la carte sont considérées comme des cercles. La distance entre le centre personnage et le centre est calculée. Si celle-ci est inférieure au rayon du cercle le joueur meurt. La zone extérieure est contrôlée à l'aide de simples coordonnées.

L'algorithme de collision entre un sabre et le joueur se fait sur plusieurs points, placés en fonction de la longueur de l'arme et dont la nombre varie en fonction de ce même paramètre. La condition est la suivante : deux points de collisions ne peuvent être éloignés de plus de la taille du personnage (définie à 50 pixels par défaut).

Si cette condition n'est pas remplie, un point supplémentaire est ajouté, et tous les points sont replacés.



Mode débug

Ensuite, lorsque les personnages sont suffisamment proches (lorsque leur distance est au moins égale à 3 fois la taille de l'arme), pour des soucis d'optimisation, l'algorithme est déclenché.

Il calcule la distance entre chaque point et le centre de l'autre joueur. Si cette distance est inférieure ou égale au rayon de la « hitbox » du personnage, l'adversaire meurt.

Mode débug

Le mode débug m'a été très utile lors des divers tests que j'ai pu réaliser pour essayer les fonctionnalités. Il affiche quelques détails supplémentaires, comme les zones d'apparition possible après une mort, les hitbox des personnages, les zones de morts de la carte, …

Image complète du mode débug

Compteur à rebours

Le compteur à rebours est initialisé lorsque l'écran de jeu est affiché pour la première fois.

On stocke l'instant où le compteur à été initialisé grâce à la fonction **millis**(). Lorsque le programme se lance, une horloge interne est également lancée. Cette fonction permet de récupérer la valeur de cette horloge interne (en millisecondes).

Ensuite, à chaque actualisation, on récupère la valeur de l'horloge interne que l'on soustrait au temps de jeu initial, additionné au temps récupéré lors de l'initialisation.

Lorsque l'on quitte, ou que l'on met en pause le jeu, le compte à rebours n'est plus actualisé. Ce n'est qu'au retour dans l'écran de jeu, que l'on réinitialise le compteur, mais en gardant l'ancienne valeur de temps de jeu restant. Cela permet d'avoir un compte à rebours qui ne change pas pendant la pause.

Intégration et validation

Les fonctionnalités que j'ai codées se sont parfaitement intégrées au jeu final. En effet, la programmation d'une fonctionnalité était souvent suivie d'une longue phase de test, de changement, de test à nouveau, … Cela a permis d'intégrer correctement mes fonctionnalités au projet final.

Enfin, travailler en communication constante avec Vincent, nous a permis de fonctionner efficacement. En effet, nous étions coordonnés, et chaque changement fait par l'un était quasiment instantanément récupéré par l'autre grâce à GitHub.

Cela a permis une grande fluidité dans le travail, sans aucun souci d'incompréhension ou d'incohérence entre nos différentes fonctionnalités.

Bilan et perspectives

Le jeu n'étant pas parfait, il y a évidemment des améliorations possibles auxquelles nous avons pensé. Nous pourrions par exemple :

- Illustré les modifications lorsque du relâchement du Slider des options (par exemple le son du laser pourrait être joué au relâchement de la souris pour donner un ordre d'idée du volume)

- Développer nos sons et musiques nous-mêmes. En effet, les musiques utilisées n'étaient pas complétement libres de droits.

- Créer les images et les icônes nous-mêmes. De même pour les images, qui n'étaient libre de droit.

- Inclure un mode multijoueur via LAN ou via Internet

- Créer plus de modes de jeu et de cartes

Le projet nous a permis de développer une capacité de travail en groupe qui sera un atout formidable pour le futur.

Cela a également développé notre autonomie car nous avons dû apprendre certaines choses par nous-mêmes afin de réaliser des fonctionnalités que nous souhaitions.

Étant des grands fans de jeux-vidéos, nous avons pu voir une infime partie du travail qui se cache derrière nos licences préférées tout en découvrant une façon de travailler nouvelle car l'informatique reste globalement peu enseignée.

Cette expérience m'a vraiment permis de découvrir le monde de l'informatique et du développement. Malheureusement, même si la réalisation de projets ponctuels reste assez satisfaisante et plaisante, je ne pense pas personnellement poursuivre des études dans ce domaine.

Diffusion du projet

Le projet n'étant pas exclusivement réalisé avec des éléments libres de droits, le projet restera libre de toute modification par toute personne.

Il est disponible sur GitHub en accès open-source (accessible, modifiable, utilisable et distribuable par toute personne) : https://github.com/Clem103/projetJeuISN

QR-Code vers GitHub

Annexe

Imports & Variables

```
import processing.sound.*;          //Librairie permettant de jouer des
sons
import controlP5.*;                 //Librairie permettant l&apos;ajout
de barres glissantes
import org.gamecontrolplus.gui.*;   //Librairies permettant de controler
des perosnnages à la manette
import org.gamecontrolplus.*;
import net.java.games.input.*;
import java.util.List;

SoundFile lightSaber1, lightSaber2;     //Déclaration des variables de
son (Sabres laser)
SoundFile lightSaberOn, lightSaberOff;
SoundFile clicSound;
SoundFile deathmatchAnnouncer;
SoundFile MusiqueAJE;                   //Musique Accueil+Jeu+Exit
SoundFile MusiqueOptions;
SoundFile MusiqueCredits;
SoundFile darkSideVMusic;
SoundFile brightSideVMusic;
SoundFile drawMusic;
SoundFile fellInLava;

boolean isMusicAJEPlaying, isMusicOptionsPlaying, isMusicCreditsPlaying,
isDarkSideVMusicPlaying, isBrightSideVMusicPlaying, isDrawMusicPlaying,
isVictoryMusicPlaying;

int pSpeed1 = 8;              //Vitesses des personnages
int pSpeed2 = 8;
float pSpeedX2;
float pSpeedY2;

int timer;                    //Variables liées au timer
int tpsDeJeu = 120;
int tpsEcoule;
int tpsInit;
int scoreP1=0, scoreP2=0;

int spawn;   //Variable définissant la zone de spawn

int tPersonnage=50;

int xPersonnage1;                //Coordonnées du personnage1
int yPersonnage1;
int xs1,ys1,xs2,ys2,xs3,ys3,xs4,ys4;  //Coordonnées des sommets du
personnage1 (en carré) s1 = sommet haut gauche, s2 = sommet haut droit,
s3 = sommet bas droit, s4 = sommet bas gauche

int xPersonnage2, yPersonnage2;
int xS1,yS1,xS2,yS2,xS3,yS3,xS4,yS4;
float xP,yP,xC,yC,xCP,yCP,angleCurseur;    //C= curseur, P= Personnage,
CP= vecteur entre curseur et personnage
```

```
float distancePerso, distanceP1ellipseTop, distanceP1ellipseRight,
distanceP1ellipseLeft, distanceP2ellipseTop, distanceP2ellipseRight,
distanceP2ellipseLeft;
int xGp1, yGp1, xGp2, yGp2;   //Centre de gravité des personnages

boolean espace =  false;      //initialisation des touches du jeu (false
= touche non appuyée)
boolean up=       false;
boolean down=     false;
boolean left=     false;
boolean right=    false;

int screen;                   //Ecran à afficher

PFont titre, texte;           //Déclaration des polices d&apos;écriture

PImage fondAccueil,fondJeu, fondOptions, fondCredits, fondExit,
Personnage;    //Déclaration des images
PImage gameIcon, playIcon, pauseIcon, optionsIcon, creditsIcon,
returnIcon, exitIcon, speedDownIcon, speedUpIcon, volumeDownIcon,
volumeUpIcon, timeLessIcon, timePlusIcon;   //Déclaration des icones

ControlP5 cp5;                //Déclaration du controlleur (permetant la
création d&apos;une SlideBar)

float volumeM = 0.50;         //Valeur initiale en % du volume de la
musique
float volumeL = 0.50;         //Valeur initiale en % du volume du laser
float volumeB = 0.50;         //Valeur initiale en % du volume des
bruits

color sliderActiveColor = #FF0000, sliderForegroundColor=#AA0000;
//Couleurs liées au sliderBar
color gameTitleColor = #FFFF00, homeTextColor=#00FF00,
gameTextColor=#FF0000, creditsTextColor=#FF0000, exitTextColor=#FF7800;
//Couleurs liées au texte dans les différents menus

color optionsBackButtonColor = #007FFF, creditsBackButtonColor=#007FFF,
exitYesButtonColor=#FF0000, exitNoButtonColor=#FF0000,
gameModeBackButtonColor=#007FFF;    //Couleurs liées au texte dans
différents "boutons"
color optionsTextsColor = #FF0000;
color fleches = #FFFFFF;
color sliderLabelColor = #AA0000;

ControlIO control;                  //Definition des variables associées
à l&apos;utilisation d&apos;une manette
Configuration config;
ControlDevice gpad;
int UpDown = 0;
int RightLeft = 1;
int YAim = 2;
int XAim = 3;


float posXGamePad, posYGamePad;    //Positions des sticks du gamepad
float rollXGamePad, rollYGamePad;
float angleAim;                    //Angle généré entre la position du
stick et le 0
float xEndOfWeapon1, yEndOfWeapon1, xEndOfWeapon2, yEndOfWeapon2;
```

```
int weaponLength = 101;            //Taille de l&apos;arme
int distanceBetweenPoints;         //Distance entre deux points de
hitbox
int numberOfPoints=1;              //Nombre de points de la hitbox

boolean debugMode;
boolean noGamepadMode;
boolean inGame = false;
```

Main

```
void setup(){
  fullScreen();                                  //La taille de la fenêtre
remplie tout l&apos;écran
  //size(1280,720);
  frameRate(60);
  titre = createFont("PoliceTitre.ttf",1);     //Initialisation de la
police utilisée pour les titres
  texte = createFont("PoliceTexte.ttf",1);     //Initialisation de la
police utilisée pour le texte
  smooth();                                      //Rend les contours plus
lisses

  xPersonnage1 = width>>2;                       //En binaire : décalage à
droite des chiffres de 1 (0101 -> 0010). Revient ici à diviser par 2^1
==> Fludification des calculs
  yPersonnage1 = height>>1;                      //Autre ex: 11010001>>2  -
> 00110100 : division par 2^2=4. "left shift" & "right shift"

  xPersonnage2 = (width>>2)*3;                   //Positionnement des
personnages
  yPersonnage2 = height>>1;


  screen = 0;                                    //Initialisation de
l&apos;écran initial à l&apos;écran d&apos;accueil

  lightSaber1 = new SoundFile(this, "LightSaberHit1.mp3");
//Variables qui correspondent à un fichier son placé dans /data du
dossier projet (son du laser)
  lightSaber2 = new SoundFile(this, "LightSaberHit2.mp3");
  MusiqueAJE = new SoundFile(this, "MusiqueAJE.mp3");
//Musique Accueil+Jeu+ Menu Exit
  MusiqueOptions = new SoundFile(this, "MusiqueOptions.mp3");
//Musique Menu Options
  MusiqueCredits = new SoundFile(this, "MusiqueCredits.mp3");
//Musique Menu Credits
  fellInLava = new SoundFile(this, "splashInLava.mp3");
//Bruit tomber dans la lave
  lightSaberOn = new SoundFile(this, "lightSaberON.mp3");
//Bruit allumage sabre laser
  lightSaberOff = new SoundFile(this, "lightSaberOFF.mp3");
//Bruit extinction sabre laser
  clicSound = new SoundFile(this, "clicSound.mp3");
//Bruit de clic sur case
```

```
  deathmatchAnnouncer = new SoundFile(this, "deathmatchAnnouncer.mp3");
//Voix annoncant mode jeu
  drawMusic= new SoundFile(this, "DrawMusic.mp3");
//Musique égalité
  brightSideVMusic= new SoundFile(this, "BrightSideVMusic.mp3");
//Musique victoire coté lumineux
  darkSideVMusic= new SoundFile(this, "DarkSideVMusic.mp3");
//Musique victoire coté obscur
  MusiqueAJE.amp((0.125*volumeM));
   //Volume initial des musiques de fond (Volume max = 0.125, Volume
initial = 0.125*0.5)
  MusiqueOptions.amp(0.125*volumeM);
  MusiqueCredits.amp(0.125*volumeM);
  drawMusic.amp(0.125*volumeM);
  darkSideVMusic.amp(0.125*volumeM);
  brightSideVMusic.amp((0.125*volumeM));
  lightSaber1.amp(0.1*volumeL);
   //Volume initial des lasers (Volume max = 0.05, Volume initial =
0.05*0.5)
  lightSaber2.amp(0.1*volumeL);
  lightSaberOn.amp(0.25*volumeL);
  lightSaberOff.amp(0.25*volumeL);
  deathmatchAnnouncer.amp(0.3*volumeB);
  clicSound.amp(0.25*volumeB);
  fellInLava.amp(0.125*volumeB);

  gameIcon = loadImage("gameIcon.png");
//Chargement des images dans des variables
  fondAccueil = loadImage("fondAccueil.png");
  fondJeu = loadImage("fondJeu.png");
  fondOptions = loadImage("fondOptions.png");
  fondCredits = loadImage("fondCredits.png");
  fondExit = loadImage("fondExit.png");
  Personnage = loadImage("PersonnageWhite.png");
  playIcon = loadImage("PlayIcon.png");
  pauseIcon = loadImage("PauseIcon.png");
  optionsIcon = loadImage("OptionsIcon.png");
  creditsIcon = loadImage("CreditsIcon.png");
  returnIcon = loadImage("ReturnIcon.png");
  exitIcon = loadImage("ExitIcon.png");
  speedDownIcon = loadImage("SpeedDownIcon.png");
  speedUpIcon = loadImage ("SpeedUpIcon.png");
  volumeDownIcon = loadImage("VolumeDownIcon.png");
  volumeUpIcon = loadImage("VolumeUpIcon.png");
  timePlusIcon = loadImage("TimePlusIcon.png");
  timeLessIcon = loadImage("TimeLessIcon.png");

  gameIcon.resize(50,50);
//Adaptation de la taille des images à la taille de la fenêtre
  fondAccueil.resize(width,height);
  fondJeu.resize(width,height);
  fondOptions.resize(width,height);
  fondCredits.resize(width,height);
  fondExit.resize(width,height);


  surface.setIcon(gameIcon);
//Définition de l&apos;icone du jeu
  surface.setTitle("FightSabers");
//Définiton du titre du jeu dans la barre de la fenêtre
```

```
   cp5 = new ControlP5(this);
//Initialisation du controleur de paramètres
   cp5.setColorActive(sliderActiveColor).setColorForeground(sliderForegr
oundColor);    //Réglage de la couleur lors du mouse-over et couleur en
règle générale des barres


   cp5.addSlider("Vitesse Personnage 1")
//Initialisation des différentes barres avec leurs paramètres (Position,
taille, valeurMin/Max, valeur initiale, visibilité)
      .setPosition(width>>2,height*0.2)
      .setSize((int)(width*0.5),40)
      .setRange(1,20)
      .setValue(pSpeed1)
      .setVisible(false)
      .setColorLabel(#FFFFFF)
      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

   cp5.addSlider("Vitesse Personnage 2")
      .setPosition(width>>2,height*0.3)
      .setSize((int)(width*0.5),40)
      .setRange(1,20)
      .setValue(pSpeed2)
      .setVisible(false)
      .setColorLabel(#FFFFFF)
      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

   cp5.addSlider("Volume musique")
      .setPosition(width>>2,height*0.4)
      .setSize((int)(width*0.5),40)
      .setRange(0,100)
      .setValue(50)
      .setVisible(false)
      .setColorLabel(#FFFFFF)
      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

   cp5.addSlider("Volume lasers")
      .setPosition(width>>2,height>>1)
      .setSize((int)(width*0.5),40)
      .setRange(0,100)
      .setValue(50)
      .setVisible(false)
      .setColorLabel(#FFFFFF)
      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

   cp5.addSlider("Volume bruits")
      .setPosition(width>>2,height*0.6)
      .setSize((int)(width*0.5),40)
      .setRange(0,100)
      .setValue(50)
      .setVisible(false)
      .setColorLabel(#FFFFFF)
      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);

   cp5.addSlider("Temps de Jeu")
      .setPosition(width>>2,height*0.7)
      .setSize((int)(width*0.5),40)
      .setRange(10,180)
      .setValue(120)
      .setNumberOfTickMarks(18)
      .showTickMarks(false)
```

```
      .setVisible(false)
      .setColorLabel(#FFFFFF)
      .getCaptionLabel().align(ControlP5.CENTER, ControlP5.CENTER);


  control = ControlIO.getInstance(this);
  List<ControlDevice> inputs = control.getDevices();
      //On récupère les périphériques disponible dans une liste
  for(int i=0;i<inputs.size();i++){
//Pour chaque élément de la liste
     if(inputs.get(i).getTypeName()=="Gamepad")
gpad=control.getDevice(i);    //Si le périphérique est du type Gamepad,
alors on l&apos;utilise comme manette
  }

  if(gpad== null){
//Si aucun périphérique branché n&apos;est compatible, on entre en mode
débug
    noGamepadMode=true;
  }

  if(!noGamepadMode){    //Définition de la tolérance
    gpad.getSlider(UpDown).setTolerance(0.1);
    gpad.getSlider(RightLeft).setTolerance(0.12);
    gpad.getSlider(XAim).setTolerance(0.05);
    gpad.getSlider(YAim).setTolerance(0.15);
  }
  sommetsPerso();        //Calcul des sommets des personnages
  numberOfPoints=numberOfHitboxPoints();
}



void draw(){
  switch(screen) {
   case 0: ecranAccueil();        break;    //Affichage de l&apos;écran
d&apos;accueil
   case 1: ecranJeu1vs1();        break;    //Affichage de l&apos;écran
de jeu (1 contre 1)
   case 2: ecranOptions();        break;    //Affichage de l&apos;écran
des options
   case 3: ecranCredits();        break;    //Affichage de l&apos;écran
des crédits
   case 4: ecranSortie();         break;    //Affichage de l&apos;écran
de sortie
   case 5: ecranFinPartie();      break;    //Affichage de l&apos;écran
de fin de partie
  }
}


//
// Utilisation de la souris
//

void mousePressed(){    //Au moment où le click souris est enfoncé
  if (screen == 0){     //Dans l&apos;écran d&apos;accueil
    if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height/3)+40 && mouseY>(height/3)-40){
      screen=1;         // Click Souris sur Play/Jouer
      clicSound.play();
```

```
    }
    if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height>>1)+40 && mouseY>(height>>1)-40){
      screen=2;          // Click Souris sur Options
      clicSound.play();
    }
    if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.67)+40 && mouseY>(height*0.67)-40){
      screen=3;          // Click Souris sur Credits
      clicSound.play();
    }
    if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.84)+40 && mouseY>(height*0.84)-40){
      screen=4;          // Click Souris sur Sortie
      clicSound.play();
    }
  }

  if(screen == 2){      //Dans l&apos;écran des options
    if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.9)+40 && mouseY>(height*0.9)-40){      //Click sur le
bouton retour -> masquage des barres
      clicSound.play();

      cp5.getController("Vitesse Personnage 1").setVisible(false);
      cp5.getController("Vitesse Personnage 2").setVisible(false);
      cp5.getController("Volume musique").setVisible(false);
      cp5.getController("Volume lasers").setVisible(false);
      cp5.getController("Volume bruits").setVisible(false);
      cp5.getController("Temps de Jeu").setVisible(false);


      pSpeed1 =(int) cp5.getController("Vitesse Personnage
1").getValue();                                          //On récupère les
valeurs, non mises à jour à chaque image, à la sortie du menu
      pSpeed2 =(int) cp5.getController("Vitesse Personnage
2").getValue();
      tpsDeJeu =(int) cp5.getController("Temps de Jeu").getValue();
      scoreP1=0;
      scoreP2=0;

      screen=0;   //Retour à l&apos;accueil
    }

    if((mouseX<(width>>3)+100 && mouseX>(width>>3)-100 &&
mouseY<(height*0.85)+40 && mouseY>(height*0.85)-40) &&
(debugMode==true)){
      debugMode = false;
      clicSound.play();
    }

    else if((mouseX<(width>>3)+100 && mouseX>(width>>3)-100 &&
mouseY<(height*0.85)+40 && mouseY>(height*0.85)-40) &&
(debugMode==false)){
      debugMode = true;
      clicSound.play();
    }

  }

  if (screen == 3){ //Dans l&apos;écran des crédits
```

```
    if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.9)+40 && mouseY>(height*0.9)-40){
      screen=0;               //Retour à l&apos;accueil lors du click
sur Retour
      clicSound.play();
    }
  }

  if (screen == 4){ //Dans l&apos;écran de sortie
    if (mouseX<(width*0.75)+100 && mouseX>(width*0.75)-100 &&
mouseY<(height>>1)+40 && mouseY>(height>>1)-40){
      screen=0;               //Retour à l&apos;accueil lors du click
sur No/Non
      clicSound.play();
    }
    if (mouseX<(width>>2)+100 && mouseX>(width>>2)-100 &&
mouseY<(height>>1)+40 && mouseY>(height>>1)-40){
      exit();                 //Fermeture de la fenêtre lors du click
sur Yes/Oui
      clicSound.play();
    }
  }

  if(screen == 5){
    if (mouseX<(width>>2)+100 && mouseX>(width>>2)-100 &&
mouseY<(height>>2)*3+40 && mouseY>(height>>2)*3-40){
      clicSound.play();
      screen=1;               //Joueurs veulent rejouer, jeu relancé
      p1Death();
      p2Death();
      scoreP1 = 0;
      scoreP2 = 0;
    }
    if (mouseX<(width>>2)*3+100 && mouseX>(width>>2)*3-100 &&
mouseY<(height>>2)*3+40 && mouseY>(height>>2)*3-40){
      clicSound.play();
      screen=0;               //Joueurs ne veulent pas rejouer, retour à
l&apos;accueil
      p1Death();
      p2Death();
      scoreP1 = 0;
      scoreP2 = 0;
    }
  }
}

//
//Utilisation du clavier
//
void keyPressed(){  //Lorsque l&apos;on appuie sur la touche, la
variable correspondante passe à true
  switch(keyCode){
    case 32 : espace =   true; break;  //Espace
    case UP: up =        true; break;  //Flèches clavier
    case DOWN : down =   true; break;
    case LEFT : left =   true; break;
    case RIGHT : right = true; break;
    case 90: up=         true; break;  //z
    case 81: left=       true; break;  //q
    case 83: down=       true; break;  //s
    case 68: right=      true; break;  //d
```

```
 }
 if (key == ESC) {
    key = 0;                              //On supprime la possibilité de
quitter le jeu en utilisant ESC
  }
}
void keyReleased(){  //Lorsque l&apos;on relâche la touche, la variable
correspondante passe à false
   switch(keyCode){
    case 32 : espace =   false; break; //Espace
    case UP : up =       false; break; //Flèches clavier
    case DOWN : down =   false; break;
    case LEFT : left =   false; break;
    case RIGHT : right = false; break;
    case 90: up=         false; break; //z
    case 81: left=       false; break; //q
    case 83: down=       false; break; //s
    case 68: right=      false; break; //d
  }
}

void affichagePersonnages(){
//Affichage des personnages
  image(Personnage,xs1,ys1,tPersonnage,tPersonnage);
  image(Personnage,xS1,yS1,tPersonnage,tPersonnage);
}
void affichageIconesAccueil(){
//Affichage des icones dans l&apos;accueil
  image(playIcon,(width>>1)-175,(height/3)-30,60,60);
  image(optionsIcon,(width>>1)-175,(height>>1)-30,60,60);
  image(creditsIcon,(width>>1)-175,(height*0.67)-30,60,60);
  image(exitIcon,(width>>1)-175,(height*0.84)-30,60,60);
}
void affichageIconeJeu1vs1(){
//Affichage des icones en jeu
  image(pauseIcon,(width>>1)-150,height-33,25,25);
}
void affichageIconesOptions(){
//Affichage des icones dans les options
  image(speedDownIcon,(width>>2)-50,height*0.2,40,40);
  image(speedDownIcon,(width>>2)-50,height*0.3,40,40);
  image(speedUpIcon,(width/1.3),height*0.2,40,40);
  image(speedUpIcon,(width/1.3),height*0.3,40,40);
  image(volumeDownIcon,(width>>2)-40,height*0.4,40,40);
  image(volumeDownIcon,(width>>2)-40,height>>1,40,40);
  image(volumeDownIcon,(width>>2)-40,height*0.6,40,40);
  image(volumeUpIcon,(width/1.3)-10,height*0.4,40,40);
  image(volumeUpIcon,(width/1.3)-10,height>>1,40,40);
  image(volumeUpIcon,(width/1.3)-10,height*0.6,40,40);
  image(timeLessIcon,(width>>2)-40,height*0.7,40,40);
  image(timePlusIcon,(width/1.3)-10,height*0.7,40,40);

}
void affichageIconesExit(){
//Affiche des icones dans le menu de sortie
  image(exitIcon,(width>>2)+100,(height>>1)-40,80,80);
  image(returnIcon,(width*0.75)-180,(height>>1)-40,80,80);
}
```

Ecran

```
//
// Définition de l&apos;écran d&apos;accueil
//

void ecranAccueil(){
  if(isMusicOptionsPlaying){          //On passe par des boolean pour
détecter lorsqu&apos;une musique est entrain d&apos;être jouée (pour
éviter de la jouer plusieurs fois en même temps)
    MusiqueOptions.stop();            //Lors du retour à l&apos;ecran
d&apos;accueil, toutes les musiques s&apos;arretent (si elles sont
entrain d&apos;être jouées)
    isMusicOptionsPlaying = false;
  }
  if(isMusicCreditsPlaying){
    MusiqueCredits.stop();
    isMusicCreditsPlaying = false;
  }
  if(isVictoryMusicPlaying){
    darkSideVMusic.stop();
    brightSideVMusic.stop();
    isVictoryMusicPlaying = false;
  }
  if(isDrawMusicPlaying){
    drawMusic.stop();
    isDrawMusicPlaying = false;
  }

  cursor();
  background(fondAccueil);

  if(!isMusicAJEPlaying){            //Et la musique de l&apos;accueil
est jouée en boucle (.loop();)
    MusiqueAJE.loop();
    isMusicAJEPlaying=true;
  }

  noFill();
  stroke(0,0,0);
  rectMode(CENTER);
  textFont(titre,75);
  textAlign(CENTER);
  fill(gameTitleColor);
  text("FightSabers",width>>1,height/5);
  textFont(titre,28);
  noFill();

  affichageIconesAccueil();

  if(debugMode) text("Debug mode activated",width>>1,28);
  else if(noGamepadMode) text("No suitable device found, no gamepad mode
activated",width>>1,28);

  if((mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height/3)+40 && mouseY>(height/3)-40)){       //Souris sur PLAY
/ JOUER
```

```
    fill(255,50);
  }
                        //Remplissage (ou non) de la case avec blanc un
peu transparent
  else noFill();
  rect(width>>1,height/3,200,80);
//Réalisation de la case

  if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height>>1)+40 && mouseY>(height>>1)-40) {     //Souris sur
Options
    fill(255,50);
  }
                        //Remplissage (ou non) de la case avec blanc un
peu transparent
  else noFill();
  rect(width>>1,height>>1,200,80);
//Réalisation de la case

  if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.67)+40 && mouseY>(height*0.67)-40) { //Souris sur
Copyrights
    fill(255,50);
  }
                        //Remplissage (ou non) de la case avec blanc un
peu transparent
  else noFill();
  rect(width>>1,height*0.67,200,80);
//Réalisation de la case

  if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.84)+40 && mouseY>(height*0.84)-40) { //Souris sur Fin
    fill(255,50);
  }
                        //Remplissage (ou non) de la case avec blanc un
peu transparent
  else noFill();
  rect(width>>1,height*0.84,200,80);

  fill(homeTextColor);
  text("Play",width>>1,(height/3)+10);
  text("Options",width>>1,(height>>1)+10);
  text("Credits",width>>1,height*0.67+10);
  text("Exit",width>>1,height*0.84+10);
  noFill();
}

//
//Définition de l&apos;écran de Jeu 1 contre 1
//

void ecranJeu1vs1(){
  if(!inGame){
    initTimer();
    inGame = true;
    lightSaberOn.play();
    deathmatchAnnouncer.play();
  }
  if(isVictoryMusicPlaying){
    darkSideVMusic.stop();
    brightSideVMusic.stop();
```

```
    isVictoryMusicPlaying = false;
  }

  if(isDrawMusicPlaying){
    drawMusic.stop();
    isDrawMusicPlaying = false;
  }
  background(fondJeu);                    //Apparition de la carte de jeu
  if(!isMusicAJEPlaying){
    MusiqueAJE.loop();
    isMusicAJEPlaying=true;
  }

  if(!noGamepadMode){
    bougerPersonnageGamepad(pSpeed2);
    viseeGamepad();
  }
  bougerPersonnageClavier();            //Appel des fonctionnalités de
jeu
  viseeSouris();
  sommetsPerso();
  affichagePersonnages();
  affichageIconeJeu1vs1();
  checkHitbox(numberOfPoints);
  updateTimer();
  dispTimer();

  if(debugMode){
    debugHitboxPerso();
    debugHitboxMap();
    debugRespawn();
  }
  fill(gameTextColor);
//Affichage des scores
  stroke(255,0,0);
  textFont(texte,20);
  fill(#890000);
  text("DarkSide points : " + scoreP1,width>>2,height-15,20);
  fill(#00FF00);
  text("BrightSide points : " + scoreP2,(width>>2)*3,height-15,20);
  noFill();
  fill(#FFFFFF);
  text("Space/Espace : Pause",width>>1,height-15,20);
//Création des informations pour le retour accueil
  textFont(texte,25);
  noFill();

  if(espace){                                          //Si on appuie
sur espace, l&apos;écran d&apos;accueil est ouvert (mise en pause du
jeu)
    screen=0;
    tpsDeJeu = timer;
    inGame=false;
    lightSaberOff.play();
  }

  if(timer<=0){
    screen=5;
    inGame = false;
    lightSaberOff.play();
  }
```

```
}

//
// Définition de l&apos;écran de fin de partie
//

void ecranFinPartie(){
 MusiqueAJE.stop();
 isMusicAJEPlaying = false;
 background(fondJeu);
 affichagePersonnages();
 textFont(titre,50);
 fill(gameTitleColor);
 textAlign(CENTER);
 stroke(0,0,0);

 if(scoreP1==scoreP2){      //Affichage des scores finaux
   text("It&apos;s a draw !\n The scores are " + scoreP1 + "
point(s)!",width>>1,height>>2);
   if(!isDrawMusicPlaying){
      drawMusic.play();
      isDrawMusicPlaying=true;
   }
 }
 if(scoreP1<scoreP2){
   text("The BrightSide won\n Its score is " + scoreP2 + "
point(s)",width>>1,height>>2);
   if(!isVictoryMusicPlaying){
      brightSideVMusic.play();
      isVictoryMusicPlaying=true;
   }
 }
 if(scoreP1>scoreP2){
   text("The DarkSide won\n Its score is " + scoreP1 + "
point(s)",width>>1,height>>2);
   if(!isVictoryMusicPlaying){
      darkSideVMusic.play();
      isVictoryMusicPlaying=true;
   }
 }

 fill(gameTextColor);
 text("Do you want to play again?",width>>1,height*0.4);
 noFill();
 rectMode(CENTER);
 if(mouseX<(width>>2)+100 && mouseX>(width>>2)-100 &&
mouseY<(height>>2)*3+40 && mouseY>(height>>2)*3-40) {
   fill(255,50);
 }
 else noFill();
 rect(width>>2,(height>>2)*3,200,80);

 if(mouseX<(width>>2)*3+100 && mouseX>(width>>2)*3-100 &&
mouseY<(height>>2)*3+40 && mouseY>(height>>2)*3-40){
   fill(255,50);
 }
 else noFill();
 rect((width>>2)*3,(height>>2)*3,200,80);

 textFont(texte,22);
 fill(exitYesButtonColor);
```

```
 text("Yes",width>>2,(height>>2)*3+10);
 text("No",(width>>2)*3,(height>>2)*3+10);
}

//
//Définition de l&apos;écran des options
//

void ecranOptions(){
  if(isMusicAJEPlaying){
    MusiqueAJE.stop();
    isMusicAJEPlaying=false;
  }
  background(fondOptions);                              //Apparition du fond
  if(!isMusicOptionsPlaying){
    MusiqueOptions.loop();
    isMusicOptionsPlaying=true;
  }
  cp5.getController("Vitesse Personnage 1").setVisible(true);
//On affiche les barres définies dans le setup{}
  cp5.getController("Vitesse Personnage 2").setVisible(true);
  cp5.getController("Volume musique").setVisible(true);
  cp5.getController("Volume lasers").setVisible(true);
  cp5.getController("Volume bruits").setVisible(true);
  cp5.getController("Temps de Jeu").setVisible(true);

  updateOptions();
  affichageIconesOptions();                            //Affichage des
icones des options

  fill(optionsTextsColor);                             //Création bouton
retour accueil
  fill(optionsBackButtonColor);
  textFont(texte,22);
  text("Back / Retour",width>>1,height*0.915);
  if(debugMode){
    fill(#58FF00);
    text("Debug ON",width>>3,height*0.862);
  }
  else{
    fill(#FF001E);
    text("Debug OFF",width>>3,height*0.862);
  }
  if(mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.9)+40 && mouseY>(height*0.9)-40) {  //Bouton retour à
l&apos;écran d&apos;accueil (avec un remplissange semi-transparent au
mouse-over)
    fill(255,50);
  }
  else noFill();
  rect(width>>1,height*0.9,200,80);

  if(mouseX<(width>>3)+100 && mouseX>(width>>3)-100 &&
mouseY<(height*0.85)+40 && mouseY>(height*0.85)-40) {
    fill(255,50);
  }
  else noFill();
  rect(width>>3,height*0.85,200,80);
//Case "debug"
}
```

```
//
//Définition de l&apos;actualisaiton dynamique du menu des options
//

void updateOptions(){   //Ces paramètres sont mis à jour à chaque frame
tant que l&apos;on est sur l&apos;écran des options

  volumeM=(cp5.getController("Volume musique").getValue())/100;
//Réglage des volumes
  MusiqueAJE.amp(0.125*volumeM);
  MusiqueOptions.amp(0.125*volumeM);
  MusiqueCredits.amp(0.125*volumeM);

  volumeL=(cp5.getController("Volume lasers").getValue())/100;
  lightSaber1.amp(0.05*volumeL);
  lightSaber2.amp(0.05*volumeL);
  lightSaberOn.amp(0.25*volumeL);
  lightSaberOff.amp(0.25*volumeL);

  volumeB=(cp5.getController("Volume bruits").getValue())/100;
  clicSound.amp(0.25*volumeB);
  deathmatchAnnouncer.amp(0.3*volumeB);
  fellInLava.amp(0.125*volumeB);
}

//
//Définition de l&apos;écran des crédits
//

void ecranCredits(){
  if(isMusicAJEPlaying){
    MusiqueAJE.stop();
    isMusicAJEPlaying=false;
  }
  background(fondCredits);
  if(!isMusicCreditsPlaying){
    MusiqueCredits.loop();
    isMusicCreditsPlaying=true;
  }
  textAlign(CENTER);
  textFont(titre,50);
  fill(gameTitleColor);
  text("Credits",width>>1,height*0.1);
  textFont(texte,30);
//Création bouton retour accueil
  fill(0);
//Couleurs en dégradé
  text("Graphism Management : MARSOT Vincent",width>>1,height*0.25);
  fill(51);
  text("Sound Management : MARSOT Vincent & VELLU
Clément",width>>1,height*0.35);
  fill(102);
  text("Keyboard + Mouse Support : MARSOT
Vincent",width>>1,height*0.45);
  fill(153);
  text("Gamepad Support : VELLU Clément",width>>1,height*0.55);
  fill(204);
  text("Hitbox Algorithm : VELLU Clément",width>>1,height*0.65);
  fill(255);
  text("Gameplay : VELLU Clément & MARSOT
Vincent",width>>1,height*0.75);
```

```
  fill(creditsBackButtonColor);
  textFont(texte,22);
  text("Back / Retour",width>>1,height*0.9+10);
  if (mouseX<(width>>1)+100 && mouseX>(width>>1)-100 &&
mouseY<(height*0.9)+40 && mouseY>(height*0.9)-40) fill(255,50);
  else noFill();
  rect(width>>1,height*0.9,200,80);
}

//
//Définition de l&apos;écran de sortie
//

void ecranSortie(){
  background(fondExit);                        //Affichage du fond écran
fermer jeu
  affichageIconesExit();                       //Affichage des icones
écran fermer jeu
  if(!isMusicAJEPlaying){
    MusiqueAJE.loop();
    isMusicAJEPlaying=true;
  }
  textAlign(CENTER);
//Affichage question validation
  fill(exitTextColor);
  textFont(texte,30);
  text("Êtes-vous sûr de vouloir quitter ?",width>>1,height/3);

  if (mouseX<(width>>2)+100 && mouseX>(width>>2)-100 &&
mouseY<(height>>1)+40 && mouseY>(height>>1)-40) {  //Bouton Yes
    fill(255,50);
  }
  else noFill();
  rect(width>>2,height>>1,200,80);
  textFont(texte,30);
  fill(exitYesButtonColor);
  text("Yes",width>>2,(height>>1)+10);

  if (mouseX<(width*0.75)+100 && mouseX>(width*0.75)-100 &&
mouseY<(height>>1)+40 && mouseY>(height>>1)-40) { //Bouton No
    fill(255,50);
  }
  else noFill();
  rect(width*0.75,height>>1,200,80);
  fill(exitNoButtonColor);
  text("No",width*0.75,(height>>1)+10);
}
```

Mouvement

```
void bougerPersonnageClavier(){
```

```
  if(up && (yPersonnage1>=0))                    yPersonnage1-=pSpeed1;
//Mouvement vers le haut (on soustrait la vitesse (en pixel) sur y) ssi
le personnage n&apos;est pas sur le bord haut et que la touche "up" est
enfoncée
  if(down && (yPersonnage1<height-tPersonnage))  yPersonnage1+=pSpeed1;
//Mouvement vers le bas (on additionne la vitesse (en pixel) sur y) ssi
le personnage n&apos;est pas sur le bord bas et que la touche "down" est
enfoncée
  if(left && (xPersonnage1>=0))                  xPersonnage1-=pSpeed1;
//Mouvement vers la gauche (on soustrait la vitesse (en pixel) sur x)
ssi le personnage n&apos;est pas sur le bord gauche et que la touche
"left" est enfoncée
  if(right && (xPersonnage1<width-tPersonnage))  xPersonnage1+=pSpeed1;
//Mouvement vers la droite (on additionne la vitesse (en pixel) sur x)
ssi le personnage n&apos;est pas sur le bord droit et que la touche
"right" est enfoncée
}



void bougerPersonnageGamepad(int pSpeed2){  //Mouvement du personnage à
la manette

  pSpeedX2 = pSpeed2;
  pSpeedY2 = pSpeed2;

  posXGamePad = gpad.getSlider(RightLeft).getValue();    //Récupération
des valeurs des axes X et Y de la manette
  posYGamePad = gpad.getSlider(UpDown).getValue();

  pSpeedX2=pSpeedX2*abs(posXGamePad);                    //Si le stick
est décalé complétement dans une direction la vitesse dans cette
direction est maximale
  if(posXGamePad >=0.93 || posXGamePad <=-0.93) pSpeedX2 = pSpeed2;
  pSpeedY2=pSpeedY2*abs(posYGamePad);
  if(posYGamePad >=0.93 || posYGamePad <=-0.93) pSpeedY2 = pSpeed2;

  if(posXGamePad <= 0 && (xPersonnage2>=0))
xPersonnage2-=pSpeedX2;  //en fonction de la position du stick, on
augmente (ou diminue) la vitesse dans une direction
  if(posXGamePad >= 0 && (xPersonnage2<=width-tPersonnage))
xPersonnage2+=pSpeedX2;
  if(posYGamePad <= 0 && yPersonnage2>=0)
yPersonnage2-=pSpeedY2;
  if(posYGamePad >= 0 && (yPersonnage2<=height-tPersonnage))
yPersonnage2+=pSpeedY2;
}
```

Visée

```
void viseeSouris(){
  xC=mouseX;                                          //On récupère
la coordonnée sur X du curseur
```

```
  yC=mouseY;                                                    //On récupère
la coordonnée sur Y du curseur
  xP=xs2-16.2;                                                  //On récupère
la coordonnée sur X du manche du sabre
  yP=ys2;                                                       //On récupère
la coordonnée sur Y du manche du sabre
  xCP=xC-xP;                                                    //On calcule
la coordonnée sur X du vecteur représentant la direction du laser
  yCP=yC-yP;                                                    //On calcule
la coordonnée sur Y du vecteur représentant la direction du laser
  angleCurseur = atan2(yCP,xCP);                                //On récupère
l&apos;angle entre l&apos;abcisse et le vecteur(laser)
  xEndOfWeapon1 = xP+weaponLength*cos(angleCurseur);      //On définit
la coordonnée sur X de la fin du laser
  yEndOfWeapon1 = yP+weaponLength*sin(angleCurseur);      //On définit
la coordonnée sur Y de la fin du laser
  strokeWeight(5);                                              //On définit
l&apos;épaisseur du laser
  stroke(#FF0000);                                              //On définit
la couleur du laser
  line(xP,yP,xEndOfWeapon1,yEndOfWeapon1);                //On fait
apparaître le laser
  strokeWeight(1);                                              //On remet
l&apos;épaisseur des traits à 1 pour les cases
  noStroke();
}

void viseeGamepad(){
  rollXGamePad=gpad.getSlider(XAim).getValue();     //On récupère les
valeurs du stick de visee de la manette
  rollYGamePad=gpad.getSlider(YAim).getValue();

  double angleTemp = Math.acos(rollXGamePad);       //On en deduit
l&apos;angle par rapport à l&apos;axe x
  if(rollYGamePad>0){
      angleAim = (float)(-(angleTemp));             //Si la valeur
trouvée sur l&apos;axe Y du stick est négatif, l&apos;angle l&apos;est
aussi
  }
  else angleAim = (float)(angleTemp);
  strokeWeight(5);
  xEndOfWeapon2 = xS2-17+weaponLength*cos(angleAim);
  yEndOfWeapon2 = yS2-weaponLength*sin(angleAim);
  stroke(#00FF00);
  line(xS2-17,yS2,xEndOfWeapon2,yEndOfWeapon2);     //On dessine le laser
  strokeWeight(1);
}
```

Hitbox

```
void sommetsPerso(){
  xS1=xPersonnage2;              //Calcul des coordonées des sommets de
l&apos;image
```

```
  yS1=yPersonnage2;              //s1 = sommet haut gauche, s2 = sommet
haut droit, s3 = sommet bas droit, s4 = sommet bas gauche
  xS2=xPersonnage2+tPersonnage;
  yS2=yPersonnage2;
  xS3=xPersonnage2+tPersonnage;
  yS3=yPersonnage2+tPersonnage;
  xS4=xPersonnage2;
  yS4=yPersonnage2+tPersonnage;

  xs1=xPersonnage1;              //Personnage 1
  ys1=yPersonnage1;
  xs2=xPersonnage1+tPersonnage;
  ys2=yPersonnage1;
  xs3=xPersonnage1+tPersonnage;
  ys3=yPersonnage1+tPersonnage;
  xs4=xPersonnage1;
  ys4=yPersonnage1+tPersonnage;
}

void checkHitbox(int numberOfPoints){
  if(hasP1WeaponCollided(numberOfPoints)){
    p2Death();
    lightSaber1.play();
  }
  else if(hasP2WeaponCollided(numberOfPoints)){
    p1Death();
    lightSaber2.play();
  }
  else if(isP1InBackground()){
   p1LavaDeath();
  }
  else if(isP2InBackground()){
   p2LavaDeath();
  }
}

int numberOfHitboxPoints(){  //Définition du nombre de points de la
hitbox en fonction de la longueur du sabre
  distanceBetweenPoints = weaponLength;
  while(distanceBetweenPoints >= tPersonnage){
    numberOfPoints++;
    distanceBetweenPoints = weaponLength/numberOfPoints;
  }
  return numberOfPoints;
}

boolean hasP1WeaponCollided(int numberOfPoints){  //On check pour chaque
point si il est en collision avec le personnage2

  xGp1 = (xs1 + xs2 + xs3 + xs4)/4;
  yGp1 = (ys1 + ys2 + ys3 + ys4)/4;
  xGp2 = (xS1 + xS2 + xS3 + xS4)/4;
  yGp2 = (yS1 + yS2 + yS3 + yS4)/4;

  distancePerso = sqrt(pow((xGp1 - xGp2),2) + pow((yGp1 - yGp2),2));

  if(distancePerso<weaponLength*3){
    for (int i=0;i<=numberOfPoints;i++){
       float xHitboxPoint =
xP+weaponLength*cos(angleCurseur)*i/numberOfPoints;
```

```
        float yHitboxPoint =
yP+weaponLength*sin(angleCurseur)*i/numberOfPoints;
        float distancePointPerso = sqrt(pow((xGp2 - xHitboxPoint),2) +
pow((yGp2 - yHitboxPoint),2));

        if(debugMode){
        stroke(#00FF00);
        ellipse(xHitboxPoint,yHitboxPoint,10,10);
        fill(#00FF00);
        text(i,xHitboxPoint,yHitboxPoint);
        text(i+" : " + distancePointPerso, (width>>3), (height>>2)+i*20);
        noFill();
        }

        if(distancePointPerso <= tPersonnage*27/50){
          if(debugMode){
            fill(#FF0000);
            text(i,xHitboxPoint,yHitboxPoint);
            text(i+" : " + distancePointPerso, (width>>3),
(height>>2)+i*20);
            noFill();
          }
          return true;
        }
    }
  }
  return false;
}

boolean hasP2WeaponCollided(int numberOfPoints){      //On check pour
chaque point si il est en collision avec le personnage1

  xGp1 = (xs1 + xs2 + xs3 + xs4)/4;
  yGp1 = (ys1 + ys2 + ys3 + ys4)/4;
  xGp2 = (xS1 + xS2 + xS3 + xS4)/4;
  yGp2 = (yS1 + yS2 + yS3 + yS4)/4;

  distancePerso = sqrt(pow((xGp1 - xGp2),2) + pow((yGp1 - yGp2),2));

  if(distancePerso<weaponLength*3){
    for (int i=0;i<=numberOfPoints;i++){
        float xHitboxPoint = xS2-
17+weaponLength*cos(angleAim)*i/numberOfPoints;
        float yHitboxPoint = yS2-
weaponLength*sin(angleAim)*i/numberOfPoints;
        float distancePointPerso = sqrt(pow((xGp1 - xHitboxPoint),2) +
pow((yGp1 - yHitboxPoint),2));

        if(debugMode){
          stroke(#FF0000);
          ellipse(xHitboxPoint,yHitboxPoint,10,10);
          fill(#FF0000);
          text(i,xHitboxPoint,yHitboxPoint);
          text(i+" : " + distancePointPerso, (width*0.875),
(height>>2)+i*20);
          noFill();
          noStroke();
        }

        if(distancePointPerso <= tPersonnage*27/50){
          if(debugMode){
```

```
            fill(#00FF00);
            text(i,xHitboxPoint,yHitboxPoint);
            text(i+" : " + distancePointPerso, (width*0.875),
(height>>2)+i*20);
            noFill();
          }
          return true;
        }
    }
  }
  return false;
}

boolean isP1InBackground(){        //On check les collisions avec le
décor de chaque personnage

  if(xGp1+tPersonnage*23/50 >= width-width*0.0195 || xGp1-
tPersonnage*23/50 <= width*0.0195){
    return true;
  }
  else if(yGp1+tPersonnage*23/50 >= height-height*0.0325 || yGp1-
tPersonnage*23/50<=height*0.0325){
   return true;
  }
  distanceP1ellipseTop = sqrt(pow(xGp1 - width*0.414,2) + pow(yGp1 -
height*0.260,2));
  distanceP1ellipseRight = sqrt(pow(xGp1 - width*0.688,2) + pow(yGp1 -
height*0.635,2));
  distanceP1ellipseLeft = sqrt(pow(xGp1 - width*0.1875,2) + pow(yGp1 -
height*0.76,2));

  if(distanceP1ellipseTop <= (width*0.06)/2){
    return true;
  }
  else if(distanceP1ellipseRight <= (width*0.06)/2){
    return true;
  }
  else if(distanceP1ellipseLeft <= (width*0.06)/2){
    return true;
  }
  else return false;
}

boolean isP2InBackground(){
  if(xGp2+tPersonnage*23/50 >= width-width*0.0195 || xGp2-
tPersonnage*23/50 <= width*0.0195){
    return true;
  }
  else if(yGp2+tPersonnage*23/50 >= height-height*0.0325 || yGp2-
tPersonnage*23/50<=height*0.0325){
   return true;
  }
  distanceP2ellipseTop = sqrt(pow(xGp2 - width*0.414,2) + pow(yGp2 -
height*0.260,2));
  distanceP2ellipseRight = sqrt(pow(xGp2 - width*0.688,2) + pow(yGp2 -
height*0.635,2));
  distanceP2ellipseLeft = sqrt(pow(xGp2 - width*0.1875,2) + pow(yGp2 -
height*0.76,2));

  if(distanceP2ellipseTop <= (width*0.06)/2){
    return true;
```

```
  }
  else if(distanceP2ellipseRight <= (width*0.06)/2){
    return true;
  }
  else if(distanceP2ellipseLeft <= (width*0.06)/2){
    return true;
  }
  else return false;
}
```

Death

```
void p1Death(){          //fonction appelée lorsque le personnage 1
meurt
  int[] personnage1 = respawn(xPersonnage1,yPersonnage1);  //Assignation
des coordonées au personnage 1
  xPersonnage1 = personnage1[0];
  yPersonnage1 = personnage1[1];
  scoreP2++;                                              //Le score du
joueur 2 augmente
}

void p2Death(){          //fonction appelée lorsque le personnage 2
meurt
  int[] personnage2 = respawn(xPersonnage2,yPersonnage2);
//Assignation des coordonées au personnage 2
  xPersonnage2 = personnage2[0];
  yPersonnage2 = personnage2[1];
  scoreP1++;                                               //Le score
du joueur 1 augmente
}

void p1LavaDeath(){                                        //Fonction
appelé lors de la chute dans la lave
  fellInLava.play();
  p1Death();
}

void p2LavaDeath(){                                        //Fonction
appelé lors de la chute dans la lave
  fellInLava.play();
  p2Death();
}

int[] respawn(int xPersonnage,int yPersonnage){            //Fonction
appelé lorsque qu&apos;un joueur meurt
  spawn = (int)random(3);                                  //On
randomise la zone de spawn
  switch(spawn){
    case 0 : {                                             //Au sein de
la zone de spawn, on randomise la pos du personnage à la réapparition
```

```
      xPersonnage=(int)random(width>>5,width*0.35125);
      yPersonnage=(int)random(height>>4,height*0.6825);
      int[]personnage={xPersonnage,yPersonnage};
      return personnage;
    }
    case 1 : {
      xPersonnage=(int)random(width*0.25,width*0.65);
      yPersonnage=(int)random(height/3,height*0.88);
      int[]personnage={xPersonnage,yPersonnage};
      return personnage;
    }
    case 2 : {
      xPersonnage=(int)random(width*0.6,width*0.96);
      yPersonnage=(int)random(height>>4,height*0.5525);
      int[]personnage={xPersonnage,yPersonnage};
      return personnage;
    }
    default : {
    int[] personnage={width>>1,height>>1};                //Si la valeur
aléatoire ne fontionne le cas par défaut place les personnages au centre
    return personnage;
    }
  }
}
```

Debug

```
void debugHitboxPerso(){    //Affichage d&apos;informations
supplémentaires à propos de la hitbox des personnages
  sommetsPerso();
  if(distancePerso<=(weaponLength*3)){
   stroke(0,255,0);
  }
  else stroke(255,0,0);
  noFill();
  ellipseMode(CENTER);
  ellipse(xGp1,yGp1,10,10);
  ellipse(xGp2,yGp2,10,10);
  ellipse(xGp1,yGp1,tPersonnage*46/50,tPersonnage*46/50);
  ellipse(xGp2,yGp2,tPersonnage*46/50,tPersonnage*46/50);

  textSize(20);
  text(distancePerso,width>>1,height>>1);
  line(xGp1,yGp1,xGp2,yGp2);
}

void debugHitboxMap(){      //De même pour les hitbox des éléments du
décor
  if(isP1InBackground()) stroke(#0057FF);
  else if (isP2InBackground()) stroke(#5EFF03);
  else stroke(#030303);
  rectMode(CENTER);
  rect(width>>1,height>>1,width-width*0.0390,height-height*0.0651);
```

```
  ellipseMode(CENTER);
  ellipse(width*0.414,height*0.260,width*0.06,width*0.06);  //ellipse
top
  ellipse(width*0.688,height*0.635,width*0.06,width*0.06);  //ellipse
right
  ellipse(width*0.1875,height*0.76,width*0.06,width*0.06);  //ellipse
left
  noStroke();
}

void debugRespawn(){    //Affichage des zones de respawn
  rectMode(CORNER);
  stroke(#FFFFFF);
  rect(width>>5,height>>4,width*0.32,height*0.62);
  rect(width*0.25,height/3,width*0.4,height*0.55);
  rect(width*0.6,height>>4,width*0.36,height*0.49);
  noStroke();
}

/* Zones blanches = Zones de spawn
*  Contours noir = hitbox du décor
*  Points numérotés = points de hitbox du sabre laser
*  Cercle autour des personnages = hitbox des personnages
*  Valeurs affichées à gauche, au milieu et à droite = distance entre ch
aque point de hitbox et le personnage adverse(à gauche et à droite)
*  distance entre les 2 personnages (au mileur)
*/
```

Timer

```
void updateTimer(){      //Le compte à rebours est mis à jour chaque
frame
  timer = tpsDeJeu - ((millis())/1000 - tpsInit);  //Temps de jeu =
temps écoulé depuis le début, mis à jour lorsque le jeu est mis en pause
}

void initTimer(){
  tpsInit = millis()/1000;  //appelé 1 seul fois lorsque le mode 1v1
commence
}

void dispTimer(){  //Affiche en haut de l&apos;écran le temps restant
  int dispTimer = timer;
  fill(#FFFFFF);
  text("Temps restant : "+dispTimer+"s",width>>1,height>>5);
  noFill();
}
```