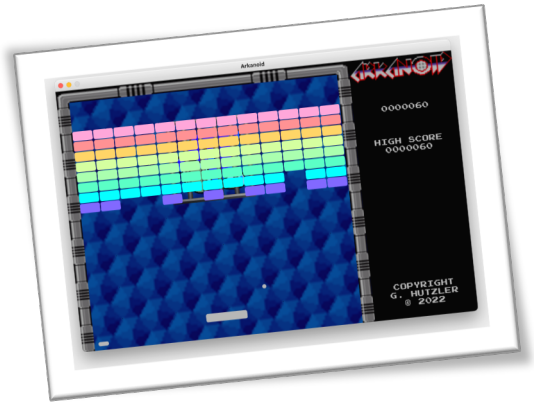
DM
Arkanoid



Objectif

L'objectif consiste à écrire un jeu ressemblant à Arkanoid, jeu d'arcade de type casse-briques développé par Taito en 1986, reprenant même principe que Breakout d'Atari Inc. (1976) (https://fr.wikipedia.org/wiki/Arkanoid) :

> *« Le joueur déplace de droite à gauche une barre horizontale censée représenter un vaisseau spatial. Ce plateau, positionné en bas de l'écran, permet de faire rebondir une balle qui va détruire des blocs situés en haut de l'écran. Le joueur passe au niveau suivant lorsque tous les blocs sont détruits (…) ; le joueur perd une vie à chaque fois qu'il laisse filer la balle au bas de l'écran. ».*

Les dix commandements de maître Yoda

⚠

1. **Ce document, avant de commencer, tu liras jusqu'au bout, et `DM_Arkanoid.pdf` pareillement**
2. Les instructions qui te guident pas à pas, tu suivras
3. Les différentes étapes de ton travail, tu conserveras
4. Ton programme, tu indenteras et commenteras
5. La signature des fonctions, tu respecteras
6. De la programmation orientée objet, tu ne feras pas
7. L'utilisation des variables globales, tu limiteras, et les variables locales préfèreras
8. Les tableaux, quand c'est possible, tu utiliseras
9. Seul, tu travailleras
10. Ce que tu fais, tu comprendras, et capable de l'expliquer, tu seras

Prise en main

- Téléchargez l'archive arkanoid.zip et décompressez-là dans un dossier sur votre ordinateur.
- Renommez `Arkanoid_v0.pde` en respectant la convention de nommage suivante : `GXX_NOM_Prenom.pde`, où `XX` doit être remplacé par :
  - `MIn` avec `n` votre n° de groupe si vous êtes dans le portail Maths-Info
  - `MPn` avec `n` votre n° de groupe si vous êtes dans le portail Maths-Physique
  - ME si vous êtes en double licence Maths-EcoB
  - BI si vous êtes en licence double diplôme Info-SdV
  - DI si vous êtes en licence double diplôme Droit-Info
- Chargez le programme dans Processing et complétez les informations vous concernant dans le bloc de commentaire en haut du programme
- Observez les différentes fonctions que vous allez devoir programmer
- Prenez connaissance du barème de notation :

$$Note_{finale} = \text{à définir…}$$

I TD1 = Qui c'est le boss ? (2 points)

I.1. Dessiner le boss

Dans l'histoire originelle, une fois que vous avez passé les 32 niveaux du jeu, vous devez finalement détruire le grand méchant « Doh ». Pour notre jeu, une fois que toutes les briques auront été détruites, vous pourrez libérer le héros de votre jeu en détruisant la cage dans laquelle il est enfermé.
Il faut donc commencer par dessiner votre héros, de taille 400x400 pixels, avant de l'enfermer en affichant par-dessus l'image de grille (« cage.png ») fournie dans le dossier data.

Pour dessiner votre héros, vous devez :

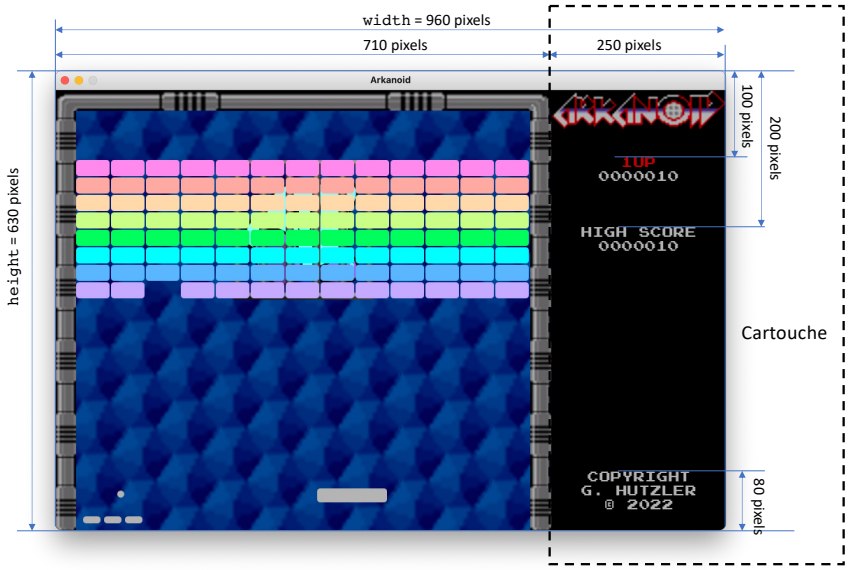
1. compléter la fonction `boss` en :
   a. utilisant au moins 20 instructions de tracé de base (point, line, rect, ellipse)
   b. utilisant au moins 10 instructions de couleurs (fill, stroke, noFill, noStroke)
   c. utilisant au moins 3 fonctions plus élaborées différentes (text, arc, quad, triangle, bezier, beginShape, etc.)
   d. commentant soigneusement votre programme
2. ajouter un appel à la fonction `boss` dans la fonction `afficheJeu`
3. ajouter un appel à la fonction `afficheJeu` dans la fonction `draw`

Pour afficher une image :

```
PImage img = loadImage(« cage.png ») ;
image(img, 0, 0) ;
```

II TD2 = On complète l'interface (2 points)

Dans la partie droite de l'image (voir schéma ci-dessous), apparaitra un « cartouche » avec le score du joueur, le meilleur score réalisé depuis le début, ainsi qu'une notice de copyright.



II.1. Afficher une ébauche du cartouche

1. Complétez la fonction `cartouche` en :
   a. dessinant un rectangle noir
   b. insérant l'image « arkanoid.png » en haut du cartouche
   c. affichant « `1UP` » et « `HIGH SCORE` »
   d. affichant les éléments de la notice de copyright (séparés verticalement de 20 pixels), en remplaçant le nom (`G. HUTZLER`) par le vôtre
2. Ajoutez un appel à la fonction `cartouche` dans la fonction `afficheJeu`

Pour afficher le texte « hello! » au centre de la fenêtre avec la police joystix.ttf :

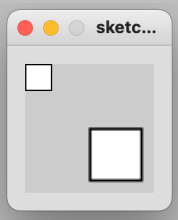
```
PFont fonte = createFont("joystix.ttf", 20);
textFont(fonte);
text("Hello!", width / 2, height / 2);
```

II.2. Afficher le cadre et repositionner le boss

1. choisissez un arrière-plan de couleur (0, 60, 130) pour l'espace de jeu
2. dessinez un cadre gris de couleur (128, 128, 128) tout autour de l'espace de jeu (voir schéma), de largeur 30 pixels
3. utilisez les fonctions `translate` et `scale` pour afficher votre boss de taille 200x200 avec le coin supérieur gauche en (255, 100). Notez que les fonctions `pushMatrix` et `popMatrix` permettent de changer temporairement l'origine du repère (avec `translate`) et l'échelle (avec `scale`), puis de revenir aux conditions de tracé initiales (voir ci-dessous).

Pour changer de repère :

```
pushMatrix() ;
translate(50, 50);
scale(2);
// affiche le grand carré gras en bas à droite
rect(0, 0, 20, 20);
popMatrix();
// affiche le petit carré fin en haut à gauche
rect(0, 0, 20, 20) ;
```



III TD3 = Le chat et la souris (2 points)

III.1. Des constantes pour délimiter le terrain de jeu

1. déclarez des constantes pour définir les coordonnées des bords du terrain (`minX`, `maxX`, `minY`) et le centre de la zone de jeu (`centreX`)

III.2. Des variables pour la raquette

La raquette doit s'afficher sous forme de rectangle aux coins arrondis, de taille 100x20 pixels, centré horizontalement sur la souris et dont le dessus est à 50 pixels du bas de la fenêtre.

1. ajoutez des constantes `raquetteY`, `raquetteL`, `raquetteH` de type entier pour définir respectivement la position du milieu de la raquette (verticalement), la largeur et la hauteur de la raquette ;
2. ajoutez une variable `raquetteX` de type entier pour enregistrer le centre de la raquette (horizontalement) ;
3. complétez la fonction `initBalleEtRaquette` pour initialiser la raquette au centre du terrain de jeu ;
4. complétez la fonction **`mouseMoved`** pour ajuster le centre de la raquette en fonction de la position de la souris ;
5. complétez la fonction `afficheRaquette` pour afficher la raquette centrée sur le point (`raquetteX`, `raquetteY`), de couleur blanche ;
6. Ajoutez un appel à la fonction `afficheRaquette` dans la fonction `afficheJeu`.

III.3. Des variables pour la balle

La balle s'affiche sous forme d'un disque blanc de diamètre 10.

1. ajoutez une constante `balleD` pour définir le diamètre de la balle ;
2. ajoutez des variables `balleX` et `ballY` de type réel pour enregistrer les coordonnées du centre de la balle ;
3. complétez la fonction `initBalleEtRaquette` pour initialiser la balle posée sur la raquette ;
4. complétez la fonction `afficheBalle` pour afficher la balle centrée sur le point (`balleX`, `balleY`), de couleur blanche ;
5. Ajoutez un appel à la fonction `afficheBalle` dans la fonction `afficheJeu`

IV TD4 = On lance la balle (2 points)

On souhaite maintenant pouvoir faire bouger la balle. On définira pour cela un vecteur vitesse (`balleVx`, `balleVy`).

IV.1. Encore des variables pour la balle

1. déclarez une constante de type entier `balleVitesse` ;
2. déclarez deux variables de type réel `balleVx` et `balleVy` ;
3. complétez la fonction `initBalleEtRaquette` pour choisir aléatoirement un angle compris entre $5\pi/4$ et $7\pi/4$ et initialisez le vecteur vitesse de la balle de manière à ce qu'il soit orienté dans la direction de l'angle choisi et avec une norme `balleVitesse`. Pour rappel :
$$vx = norme.\cos(angle)$$
$$vy = norme.\sin(angle)$$
4. déclarez deux variables de type réel `newBalleX` et `newBalleY` (elles seront utiles par la suite pour calculer les rebonds de la balle sur les briques et sur le boss) ;
5. complétez la fonction `deplaceBalle` qui met à jour `newBalleX` et `newBalleY`, la nouvelle position de la balle (cf exo IV.2 du TD 4) ;
6. complétez la fonction `miseAJourBalle` qui recopie les coordonnées (`newBalleX`, `newBalleY`) dans (`balleX`, `balleY`)
7. appelez les fonctions `déplaceBalle` et `miseAJourBalle` au début de la fonction `draw`.

V TD5-6 = On fait rebondir la balle (3 points)

Il s'agit maintenant de faire rebondir la balle sur les bords de l'écran et sur la raquette.

V.1. Rebond sur le bord de l'écran

1. En respectant le bord gris de 30 pixels autour de la zone de déplacement, complétez la fonction `rebondMurs` de manière à ce que la balle rebondisse quand elle atteint le bord de la zone. Pour cela, quand vous détectez que les coordonnées (`newBalleX`, `newBalleY`) sont en dehors de la zone de déplacement, ramenez la balle au bord de la zone et inversez la vitesse horizontale ou verticale de la balle selon les cas (cf exo II.3 du TD 6).
2. Ajoutez un appel à la fonction `rebondMurs` dans la fonction `draw` après le calcul des nouvelles coordonnées de la balle.

V.2. Rebond sur la raquette

1. Complétez de même la fonction `rebondRaquette` de manière à ce que la balle rebondisse quand elle arrive sur la raquette. On considère qu'il y a rebond sur la raquette quand les conditions suivantes sont réunies :
   a. l'ancienne position de la balle est au-dessus de la partie haute de la raquette ;
   b. la nouvelle position de la balle se trouve sous la partie haute de la raquette, tout en étant horizontalement entre le bord gauche et le bord droit de la raquette. Pour cela, vous replacerez la balle sur la raquette et vous inverserez la vitesse verticale de la balle.
2. Ajoutez un appel à la fonction `rebondRaquette` dans la fonction `draw` après le calcul des nouvelles coordonnées de la balle.

V.3. Rebond sur le boss

1. Complétez la fonction `rebond` pour gérer le rebond de la balle sur le boss. Cette même fonction sera utilisée aussi pour gérer le rebond sur les briques. La fonction prend en entrée l'ancienne (x1, y1) et la nouvelle (x2, y2) position de la balle, de même que les coordonnées du rectangle avec lequel on teste le rebond (coordonnées x, y du coin supérieur gauche puis largeur et hauteur) et renvoie `true` ou `false` selon qu'un rebond est détecté ou non. Selon que vous détectez un rebond horizontal ou vertical, inversez la vitesse horizontale ou verticale de la balle (d'après les conditions ci-dessous, le rebond peut être à la fois horizontal et vertical…).
   a. le rebond sera horizontal si la nouvelle abscisse de la balle est à l'intérieur du rectangle alors que l'ancienne abscisse est à l'extérieur ;
   b. le rebond sera vertical si la nouvelle ordonnée de la balle est à l'intérieur du rectangle alors que l'ancienne ordonnée est à l'extérieur.
2. Complétez la fonction `rebondBoss` :
   a. Appelez la fonction rebond avec les coordonnées du boss pour savoir s'il y a rebond sur le boss ;
   b. Si c'est le cas, augmentez le score de 10 : vous devrez pour cela avoir déclaré une variable `score`, que vous afficherez dans le cartouche en plus des autres éléments du cartouche.
3. Ajoutez un appel à la fonction `rebondBoss` dans la fonction `draw` après le calcul des nouvelles coordonnées de la balle.
4. Modifiez la fonction `rebondMurs` pour remettre le score à 0 quand la balle touche le bas de la fenêtre (donc il n'a pas été rattrapé par la raquette).

VI TD7-8 = On soigne la présentation (2 points)

VI.1. La texture du fond

1. Complétez la fonction `pavage` de manière à dessiner le fond de la zone de déplacement de la balle. Utilisez pour cela l'image `"tile.png"` pour créer une texture en l'affichant de manière répétée à la fois horizontalement et verticalement, à l'aide de boucles for. Sur une même colonne, les éléments sont séparés verticalement de 76 pixels. Deux colonnes successives sont séparées horizontalement de 65 pixels et sont décalées les unes par rapport aux autres de 38 pixels.
2. Appelez la fonction `pavage` dans `afficheJeu` pour compléter l'interface.

VI.2. Le cadre de la zone de la balle

1. Complétez la fonction `cadre` de manière à dessiner le cadre de la zone de déplacement de la balle. Utilisez pour cela les images `"wall1.png"` et `"wall2.png"` que vous afficherez en alternance verticalement grâce à une boucle. Deux éléments identiques de type wall1 ou wall2 sont séparés verticalement de 105 pixels. Un élément de type wall2 est placé 40 pixels sous un élément de type wall1.
2. Appelez la fonction `cadre` dans `afficheJeu` pour compléter l'interface.

VII On fait un « vrai » jeu ! (2 points)

VII.1. On met du gros son

Pour améliorer l'intérêt du jeu, on pourra ajouter du son. L'exemple suivant permet de jouer un fichier son (puis de l'arrêter quand on appuie sur une touche) :

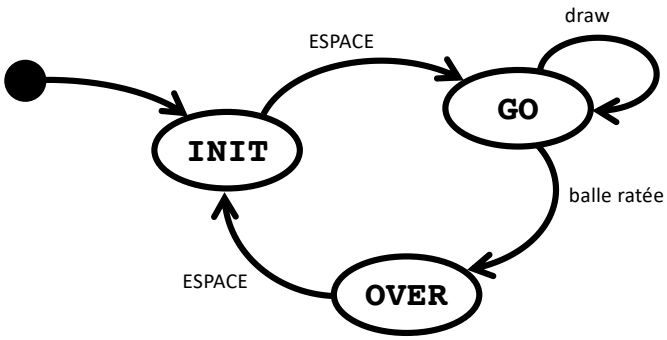
Pour jouer/arrêter le son « intro.mp3 » :

```
import processing.sound.*;
SoundFile intro;

void setup() {
  intro = new SoundFile(this, "intro.mp3");
  intro.play();
}

void draw() {
  if (keyPressed)
    intro.stop();
}
```

1. Installez la bibliothèque « Sound ». Pour cela, allez dans le menu « Tools » (ou « Outils ») du logiciel Processing, sélectionnez « Manage tools » puis cliquez sur l'onglet « Libraries ». Dans le champ de recherche « Filter », entrez « Sound », puis cliquez sur la ligne « Sound | Provide a simple way to work with audio ». Cliquez enfin sur « Install » pour lancer l'installation.
2. Chargez les fichiers sons fournis dans des variables distinctes.
3. Lancez la lecture des sons quand les évènements correspondants sont détectés (rebond sur un mur, rebond sur la raquette).

VII.2. On distingue les étapes

Il est important de savoir où on en est dans le jeu (initialisation, victoire, défaite). Pour cela, on utilisera une variable dont la valeur correspondra à l'étape de jeu en cours. Les différentes étapes s'enchaîneront selon le schéma suivant (les ellipses correspondent aux différentes étapes du jeu, les flèches correspondent aux évènements faisant passer d'un état à l'autre) :



1. Déclarez une variable globale `etat`.
2. Déclarez des constantes (`INIT`, `GO`, `OVER`) pour identifier les différentes étapes possibles et attribuez-leur des valeurs entières croissantes (0, 1, 2, …).
3. Initialisez `etat` à la valeur `INIT`.
4. Complétez la fonction `afficheEcran` qui affichera un écran avec un message à destination du joueur (message choisi à l'appel de la fonction). L'écran ressemblera à l'image ci-dessous, si le message choisi est `"GET READY"` : en plus de l'affichage des éléments du jeu, on affiche l'image Arkanoid, le message en gros et en plus petit le message de relance `"PRESS <SPACE> TO START"`.

5. Modifiez la fonction draw pour tenir compte de l'état du jeu : les déplacements de la balle ne doivent être affichés que quand on est dans l'état GO. Si on est dans l'état `INIT`, on affiche un écran avec le message `"GET READY"` et on joue le son intro.mp3.
6. Complétez la fonction `keyPressed`, de manière à relancer le jeu quand on appuie sur la touche ESPACE : quand on détecte qu'on a appuyé sur la touche ESPACE, si on est dans l'état `INIT`, on passe dans l'état GO (en modifiant la variable `etat`), on arrête le son d'intro, on joue le son letsgo.mp3, et on attend 3,2 secondes (voir la fonction delay).

7. Modifiez à nouveau la fonction `rebondMurs` pour passer dans l'état `OVER` quand on rate la balle :
   a. en plus de repasser le score à 0, on joue le son gameover.mp3, et on modifie la valeur de la variable `etat` ;
   b. du coup, il faut modifier la fonction `draw` pour afficher un écran avec le message `"GAME OVER"` quand on est dans l'état correspondant ;
   c. il faut également modifier la fonction `keyPressed` pour repasser dans l'état `INIT` si on appuie sur la barre d'espace alors qu'on était dans l'état `OVER`.
8. Ajoutez une variable `highScore` pour enregistrer le meilleur score de la partie en cours. Cette variable sera mise à jour chaque fois que le score la dépasse, et elle sera affichée dans le cartouche à l'endroit prévu à cet effet.

VIII Ça casse des briques… (3 points)

VIII.1. On ajoute des briques

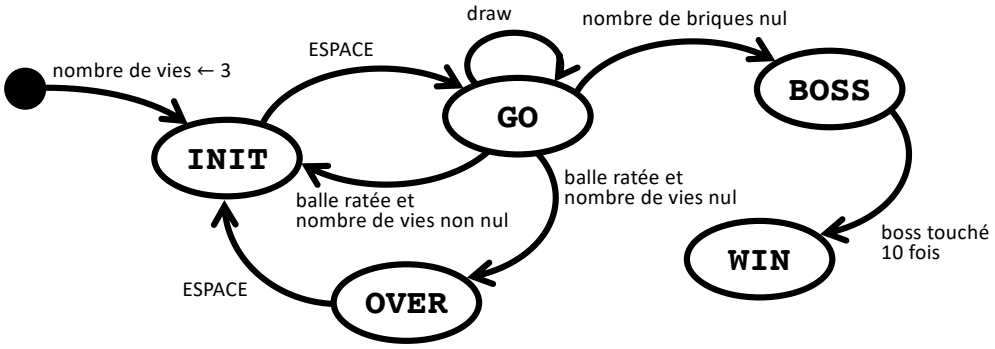
1. Déclarez des constantes `nbLignes` et `nbColonnes` pour enregistrer le nombre de lignes (8) et de colonnes (13) de briques, ainsi qu'une variable `nbBriques` initialisée à 104 (8 * 13).
2. Déclarez des constantes `briqueL` et `briqueH` pour enregistrer la largeur (50) et la hauteur (25) d'une brique.
3. Déclarez des tableaux d'entiers `briquesX` et `briquesY` pour stocker les abscisses et ordonnées des briques du jeu.
4. Déclarer un tableau `couleurs` pour enregistrer la couleur de chaque ligne. Pour la ligne i, on peut choisir par exemple la couleur (i*40, 50, 100) dans le mode couleur HSB.
5. Complétez la fonction `initBriques` pour créer les tableaux déclarés ci-dessus et initialiser les coordonnées des briques et les couleurs des lignes.
6. Ajoutez une fonction `afficheBriques` pour afficher les lignes de briques et appelez cette fonction dans `afficheJeu`.

VIII.2. On casse les briques

1. Complétez la fonction `rebondBriques` pour calculer le rebond sur les briques : pour chaque brique du tableau, appelez la fonction `rebond` pour tester s'il y a rebond entre la balle et la brique ; si un rebond est détecté, modifiez les coordonnées de la brique en (-100, -100) pour la faire disparaître, jouez le son brique.mp3, augmentez le score de 10, mettez à jour le `highScore` au besoin et décrémentez `nbBriques` (pour savoir quand on aura détruit toutes les briques).
2. Modifiez la fonction `draw` en appelant la fonction `rebondBriques` plutôt que la fonction `rebondBoss`.

IX En bonus… soyons pros ! (3 points)

1. Vous pouvez ajouter la possibilité d'avoir plusieurs vies.
2. Vous pouvez ajouter de nouvelles étapes dans le jeu : quand toutes les briques sont détruites, on peut s'occuper du boss et le détruire en le touchant une dizaine de fois selon le schéma page suivante.


draw
nombre de briques nul
ESPACE
nombre de vies ← 3
BOSS
GO
INIT
balle ratée et
nombre de vies nul
balle ratée et
nombre de vies non nul
boss touché
10 fois
WIN
ESPACE
OVER