Rapport SAE303 - Application interactive

Tom BOULLAY, Julian RODRIGUES,
Noah HEINRICH, Lucas MUSY,
Jocelyn MARCILLOUX-BUISSON

MMI2

Université Savoie Mont Blanc

Sommaire

1 Présentation

1.1 Rappel du sujet

Pour ce projet, nous devions créer application web ou mobile afin de contrôler un objet connecté en groupe de minimum de 2 personnes et à maxima de 5. Il y avait 4 sujets différents, le premier étant de répondre à des questions avec des boutons lumineux type ”buzzers”, le second est un objet qui indique une direction pour faire un jeu de type chasse au trésor, le troisième est un escalier ”piano” qui emet un son quand on monte les marches et le dernier est de placer des LEDs au dessus des salles et de les allumés en fonction du cours qu'il y a dans la salle. Lors de l'évaluation, une des personnes du groupe sera tirée au sort et devra expliquer le projet et répondre aux questions.

1.2 Notre projet

Nous avons créé un groupe de 5 personnes composé de BOULLAY Tom, HEINRICH Noah, MARCILLOUX-BUISSON Jocelyn, MUSY Lucas et RODRIGUES Julian. Nous avons choisi le premier sujet pour faire quelque chose avec des buzzers, mais au lieu de faire un ”question réponse”, nous avons décidé de faire un tape taupe. Nous avons eu plusieurs idées, la première était de faire un jeu avec 1 buzzer par personne qui jouait, comme on peut retrouver sur des jeux de société de type ”chasse taupe” et nous avons également eu l'idée de faire un jeu avec 8 buzzers qui se joue avec un seul joueur, mais qui pourrait se décliner en multijoueur avec un mode à 2 joueurs où chaque joueur possède 4 buzzers. La première idée fut abandonnée rapidement, car la majorité voulait faire un jeu de type ”tape taupe” et non ”chasse taupe”.

1.3 Évolution du projet

A partir du moment où nous avions choisi le thème des buzzers, nous avons immédiatement pensé à faire un jeu style arcade. Plusieurs idées sont survenues et celle retenue a été le tape taupe. Un jeu simple, 8 buzzers, un écran, quand les taupes apparaissent à l'écran, on tape sur le bon buzzer le plus vite possible. Puis des petites options supplémentaires que l'on a décidé d'intégrer ou non sont venues tout au long du projet.

- La première a été de mettre un mode multijoueur, 2 joueurs avec 4 buzzers, cette idée devait être réalisée à la fin si on avait le temps et ne devait pas primer sur les autres.
- La seconde option concerne les différentes taupes : Les gentilles taupes à ne pas taper, les méchantes taupes à taper et les taupes secrètes à notre effigie. Avec ceci une idée de multiplicateur pour chaque type de taupe.
- Ensuite nous avons ajouté un tableau des scores, qui n'est pas encore totalement terminé.
- Enfin une dernière idée nous est venue à l'esprit après nous être rendus compte d'une triche évidente sur notre jeu : ajouter un easter egg/jumpscare lorsqu'on appuie sur tous les boutons en même temps ou si on reste appuyé sur un bouton trop longtemps.

2 Partie Physique

2.1 Liste du matériel

Les éléments utilisés pour notre application interactive sont les suivants :

- Une Carte Arduino UNO,
- Une breadboard,
- Un câble USB,
- 8 buzzers,
- 25 câbles,
- Un ordinateur,
- Une boîte solide.

2.2 Structure

2.2.1 Boîte

Pour rendre le jeu jouable, il est crucial que les joueurs puissent appuyer rapidement. Pour assurer la robustesse de notre dispositif, nous avons choisi de concevoir une boîte solide.

Initialement, nous avions envisagé de modéliser une boîte en 3D pour l'imprimer par la suite, mais les dimensions n'étaient pas compatibles avec notre imprimante. Nous avons donc décidé de couper la boite en 2 et de l'imprimer en 2 parties différentes que nous pourrions rassembler par la suite.

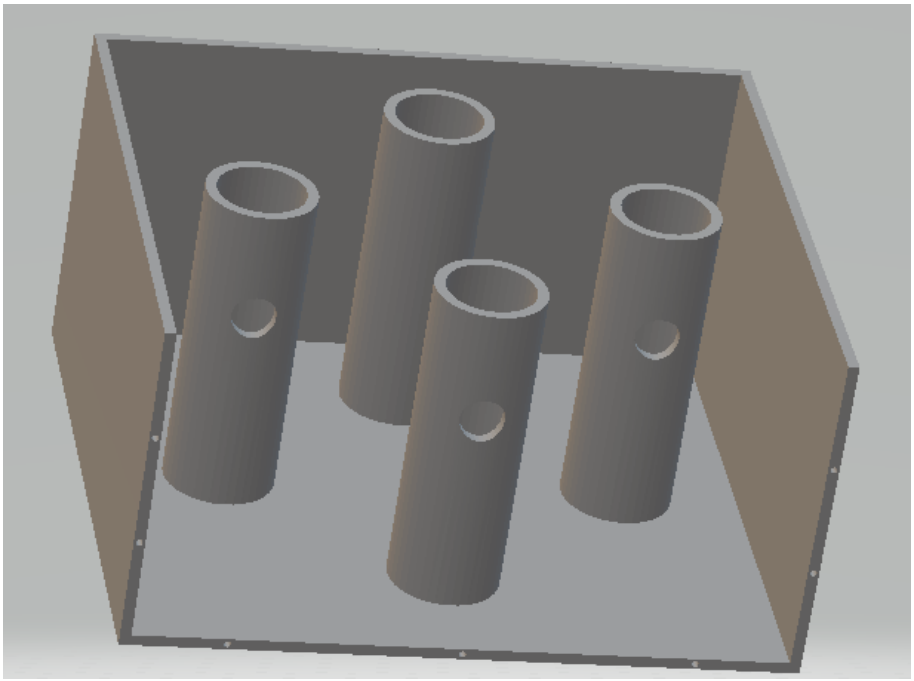

FIGURE 2.1 – Modelisation en 3d de la V1 de la boite avec ses supports

Cependant, nous nous sommes rapidement rendu compte que le coût en plastique serait très élevé pour imprimer la boite en entier, nous avons donc décidé d'imprimer seulement le couvercle.

Suite à cela, nous avons donc cherché d'autres solutions pour fabriquer la boite, une des solutions que nous avons trouvé a été de la fabriquer grâce à une découpeuse à bois.

Cependant, on ne pouvait pas créer la boite que nous voulions de cette façon, car elle était encore une fois trop grande et elle n'aurait sûrement pas été assez solide.

En fin de compte, nous avons décidé de transformer une vieille table en une boîte extrêmement robuste. Grâce à cela, nous n'avons pas eu besoin de la faire de 2 parties différentes ce qui répondait ainsi à nos besoins.

Après de nombreux ajustements, nous avons réussi à adapter le couvercle pour correspondre à la boîte en bois. En effet, nous devions bien faire attention à ce que les trous pour y faire entrer les buzzers soient suffisamment grands sans l'être trop, mais également que l'épaisseur du couvercle soit suffisant pour être résistant, mais sans que le coût ne soit trop élevé.

Nous avions également eu l'idée de faire des piliers pour soutenir le couvercle et permettre aux câbles des buzzers de ne pas trop se mélanger, cependant nous avons abandonné, car l'imprimante ne permettait pas de les faire correctement et le couvercle tient correctement sans les avoirs.

FIGURE 2.2 – Extérieur de la boite

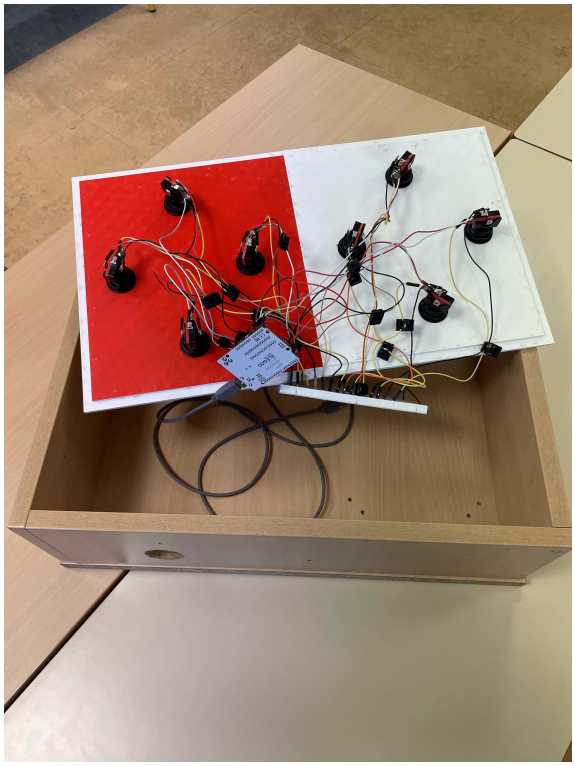

FIGURE 2.3 – Intérieur de la boite

2.2.2 Plan de cablâge

Chaque buzzer nécessite quatre connexions, donc on a besoin quatre câbles sont nécessaires. Deux d'entre eux sont destinés à contrôler les lumières du buzzer : l'un est connecté en mode analogique sur la carte pour recevoir des informations, tandis que l'autre est relié à la breadboard, qui à son tour est connectée à la masse (GND) de la carte. Les deux autres câbles sont utilisés pour le bouton du buzzer, et leur configuration est similaire à celle des câbles pour les lumières du buzzer.

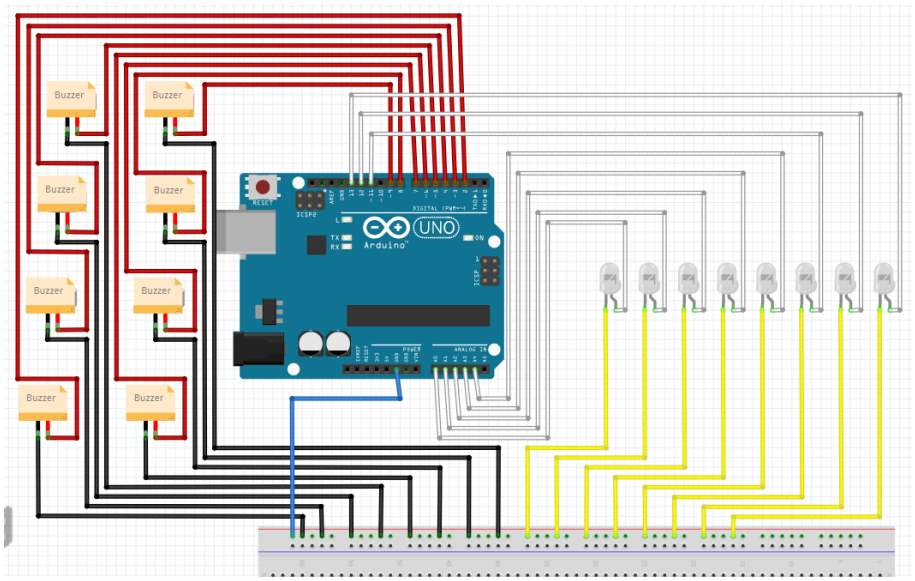

FIGURE 2.4 – Plan de cablâge du jeu 'Guaca-mole'

3 Partie Numérique

3.1 Arduino

Voici notre code Arduino, qui permet d'utiliser les buzzers de notre Tape-Taupe :

```
const int buzzerCount = 8;
const int buzzer[buzzerCount] = {2, 3, 4, 5, 6, 7, 8, 9};
const int button[buzzerCount] = {11, 12, 13, A0, A1, A2, A3, A4};
int multvalue = 1;
```

Ce code définie le nombre de buzzer à 8. Il définie ensuite toutes les pin qui sont utilisées par les buzzers et leur lumière. On initialise aussi multvalue qui permettra par la suite d'avoir un multiplicateur de score en fonction de la taupe tappé.

```
bool buzzerPressed[buzzerCount] = {false, false, false, false, false, false
    , false, false};
unsigned long buzzerPressTime[buzzerCount] = {0, 0, 0, 0, 0, 0, 0, 0};
```

On initialise une variable qui permet de checker si le buzzer est appuyé ou non. On initialise ensuite une variable qui va permettre de savoir combien de temps est pressé un bouton.

```
unsigned long startTime;
const unsigned long gameDuration = 90000;
int score = 0;
```

Ici, on définie le startTime (moment où on lance le jeu), on définie aussi la durée du jeu (ici, 1min30). Enfin, on initialise le score à 0.

```
    enum GameState { WAITING, PLAYING, FINISHED };
    GameState gameState = WAITING;
    bool startGame = false;
```

Avec enum, on défini les différents état du jeu. Puis, on initialise l'état du jeux à WAITING. Le jeux n'a pas encore commencé, on initialise donc startGame à false.

```
void setup() {
  for (int i = 0; i < buzzerCount; i++) {
    pinMode(buzzer[i], OUTPUT);
    pinMode(button[i], INPUT_PULLUP);
    digitalWrite(buzzer[i], LOW);
  }

  Serial.begin(9600);
  randomSeed(analogRead(0));
  startTime = millis();
}
```

Dans le void setup, on définie pour chaque buzzer sa LED comme une sortie et les boutons comme des entrés. On dit ensuite que toutes les LEDs sont éteintes. On initialise le startTime au moment où le jeu commence pour ensuite compter le temps écoulé depuis le début du jeu.

```
void loop() {
if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    if (command == "START") { // Si on recoit START dans le serial
      startGame = true; // On passe la variable de jeu a true
      gameState = PLAYING; // On met le stade du jeu a PLAYING
      score = 0; // On reinitialise la variable de score si jamais le jeu
          est relance
    }
}
```

Dans la loop, on viens lire les informations qu'envoi Processing. Si c'est START qui est envoyé, on lance le jeux et on change le gameState à PLAYING. Le score est initialisé à 0.

```
if(gameState == PLAYING){
  //Compte a rebours lumineux avant le debut du jeu
  delay(234);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH); // Allumer toutes les LEDs
  }
  delay(420);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW); // Eteindre toutes les LEDs
  }
  delay(520);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH); // Allumer toutes les LEDs
  }
  delay(420);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW); // Eteindre toutes les LEDs
  }
  delay(520);
```

```
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH); // Allumer toutes les LEDs
  }
  delay(420);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW); // Eteindre toutes les LEDs
  }
  delay(520);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH); // Allumer toutes les LEDs
  }
  delay(984);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW); // Eteindre toutes les LEDs
  }
  delay(1285);
```

Si l'état du jeu est à playing, on fait s'allumer et s'éteindre les LEDs pour simuler un départ mario kart.

```
startTime = millis(); // Lancer le chronometre du jeu

  while (millis() - startTime < gameDuration) {
    int randomBuzzer = random(0, buzzerCount);
    int randomTime = random(800, 3000);

    digitalWrite(buzzer[randomBuzzer], HIGH);
    Serial.print(buzzer[randomBuzzer]);
    Serial.print(",");
    Serial.println(digitalRead(buzzer[randomBuzzer]));

    unsigned long tapStartTime = millis();
    unsigned long buttonPressTime = 0;
    bool buttonPressed = false;
```

On lance le jeu et on regarde que la durée actuelle du jeu soit bien inférieur à la durée du jeu (1min30). Ensuite, on choisit aléatoirement un buzzer et un temps avant lequel il s'allume. Puis on l'allume et on envoi une trame pour processing. Enfin, on initialise le tapStartTime et le buttonPressedTime.

```
while (millis() - tapStartTime < 800) {
      if (digitalRead(button[randomBuzzer]) == LOW) {
        buttonPressed = true;
        digitalWrite(buzzer[randomBuzzer], LOW);
        buttonPressTime = millis(); // Enregistre le moment ou le bouton a
            ete presse
        Serial.print(buzzer[randomBuzzer]);
        Serial.print(",");
        Serial.println(2);
```

```
        break;
      }
    }
```

On regarde si la différence entre le moment ou la lumière c'est allumé sur le buzzer et le moment ou on a appuyé dessus est inférieur à 800 millisecondes. Si c'est le cas on éteint le buzzer et on prends la valeurs exact de l'appuie sur le buzzer. Puis on envoie une nouvelle trame. Le 2 permet de savoir que la touche a été appuyé au bon moment.

```
if (buttonPressed) {
      // Si des donnees sont disponibles sur le port serie
      if (Serial.available() > 0) {
        // Lis les donnees jusqu'a la fin de la ligne
        multvalue = Serial.parseInt();
        Serial.println(multvalue);
      }
      unsigned long reactionTime = buttonPressTime - tapStartTime; //
          Calcul du temps de reaction
      score += map(reactionTime, 0, 800, 100, 0)*multvalue; // Ajout du
          score en fonction du temps de reaction
      Serial.print("Score␣:␣");
      Serial.println(score); // Affiche le score dans la console pour
          Processing

      delay(random(500, 1000)); // Attendre un temps aleatoire avant d'
          allumer un nouveau buzzer
```

Si on a appuyé sur le bouton, on récupère la valeur du multiplicateur envoyé par processing. On calcul son temps de réaction et on calcul son score en fonction du temps de réaction et du multiplicateur. On renvoie la valeur de score et on attends un temps aléatoire entre 500 et 1000 millisecondes avant de réallumer un nouveau buzzer.

```
} else {
      digitalWrite(buzzer[randomBuzzer], LOW);
      Serial.print(buzzer[randomBuzzer]);
      Serial.print(",");
      Serial.println(digitalRead(buzzer[randomBuzzer]));
      delay(random(800, 2000));
    }
  }
```

Sinon on éteint le buzzer, on envoi la trame et on attends un temps aléatoire entre 800 et 2000 millisecondes avant de réallumer un nouveau buzzer

```
    bool allPressed = true;
    bool anyPressedTooLong = false;
    unsigned long currentTime = millis();
```

```
  for (int i = 0; i < buzzerCount; i++) {
      if (digitalRead(button[i]) == LOW) { // Si le bouton est appuye
          if (!buzzerPressed[i]) {
              buzzerPressed[i] = true;
              buzzerPressTime[i] = currentTime; // Enregistrer le temps de
                  pression
          } else if (currentTime - buzzerPressTime[i] > 3000) { // Verifier
               si appuye trop longtemps
              anyPressedTooLong = true;
          }
      } else {
          allPressed = false;
          buzzerPressed[i] = false;
      }
  }

  if (allPressed || anyPressedTooLong) {
      gameState = FINISHED; // Passer a l'etat FINISHED
      Serial.println("SPECIAL_END"); // Envoyer un signal special a
          Processing
      break;
  }
}
```

Ce bout de code permet de checker si tout les boutons sont appuyé en même temps ou si un bouton est appuyé trop longtemps. On utilise des variables pour calculer le temps de pression et l'état des buttons (appuyé ou non). Si tout les boutons sont tous appuyé ou un seul bouton est appuyé trop longtemps, on passee l'état du jeux à FINISHED et on envoie SPECIAL END sur le port série pour Processing.

```
    Serial.print("Temps␣ecoule␣!␣Score␣final␣:␣");
  Serial.println(score);

  // Allumer et eteindre les buzzers en alternance pour signaler la fin du
      jeu
  delay(700);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH);
  }
  delay(734);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW);
  }
  delay(177);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH);
  }
  delay(850);
  for (int j = 0; j < buzzerCount; j++) {
```

```
    digitalWrite(buzzer[j], LOW);
  }
  delay(111);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH);
  }
  delay(900);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW);
  }
  delay(125);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH);
  }
  delay(1500);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW);
  }
  delay(112);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH);
  }
  delay(925);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW);
  }
  delay(15);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], HIGH);
  }
  delay(815);
  for (int j = 0; j < buzzerCount; j++) {
    digitalWrite(buzzer[j], LOW);
  }
  delay(30);
  gameState = FINISHED; // On passe le stade du jeu a FINISHED
  Serial.println("END"); // On signale au processing que le jeu est fini
  Serial.print("Score"); // On envoie le score a processing pour qu'il
      puisse l'afficher
  Serial.print(",");
  Serial.println(score);
  startGame = false; // On dit que la partie est finie
```

On print le score final et on fait s'allumer et s'éteindre toutes les LEDs pour signifier la fin du jeu. On passe l'état du jeux à FINISHED et on l'envoie à Processing.

3.2 Processing

En parallèle de la programmation en Arduino, nous avons dû coder notre jeu en Processing afin de faire un lien entre notre jeu et le codage du matériel physique. Nous avons utiliser Processing afin de définir les actions possibles et de la réponse du jeu à l'écran. Vous avons séparé le jeu en deux parties : les Classes et Main.

3.2.1 Classes

La première chose à faire est de créer la partie Main, ainsi qu'une classe pour nos Taupes, mais concentrons nous sur cette dernière. Nous l'avons initialement nommé "mechante" car nous pensions qu'il nous faudrait une classe pour chaque type de taupe, mais finalement nous avons utilisé une méthode via une variable nommée "style".

```
public class mechante {
  float x,y;
  int cote;
  int visible=0;
  int style=1;
```

La position des taupes est en float afin d'ajuster ensuite plus facilement la position de chaque taupe. La taille de la taupe est initialisé par la variable "cote", signifiant la taille du coté de l'image carré. Une variable visible afin de définir si la taupe est affiché à l'écran ou non. La visibilité initiale est de 0, ce qui signifie qu'elle est invisible. Et la variable style afin de définir s'il s'agit d'une Taupe Méchante, d'une Taupe Gentille ou d'une des 5 Taupes Secrètes.

```
    mechante(float _x, float _y, int _cote) {
     x = _x;
     y = _y;
     cote = _cote;
   }
```

Toujours dans la classe mechante, on precise dans le constructeur de la Taupe la position sur l'axe des x et l'axe des y, ainsi que la taille de la Taupe comme ça on peut les modifier dans le code *main* plus tard.

```
    void display() {
         if (this.visible>0) {
           image(imagetaupe,x,y,cote,cote);
           noStroke();
         }
    }
}
```

On ajoute à la classe une fonction ***void display*** afin d'afficher notre Taupe. L'affichage de la Taupe est possible uniquement si la variable visible est supérieure à 0, ce qui pour rappel, signifie qu'elle est invisible.

On affiche ensuite le visuel de la taupe avec la variable ”imagetaupe”, que nous initialiserons dans le Main afin d'importer les images voulues. On place la taupe via les coordonnées et la longueur et la largeur de la taupe qui equivaut à la taille de l'image.

3.2.2 Main

Dans ce code, nous avons mis la quasi-totalité du code du jeu car c'est ici que chaque interaction est possible.

```
import processing.serial.*;
import processing.sound.*;
import java.util.Collections;

final int TITLE = 0;
final int GAME = 1;
final int END = 2;
final int SPECIAL_END = 3;

int gameState = TITLE;
```

On initialise les différents états de jeu en final, afin qu'ils ne soient pas modifiables, et on utilise l'état TITLE en état de base lors du lancement du jeu (Écran titre).

```
SoundFile start;
SoundFile end;
SoundFile music;
SoundFile jog;
SoundFile noa;
SoundFile jul;
SoundFile tom;
SoundFile luc;
SoundFile nice;
SoundFile bad;
SoundFile egg;
```

On initialise les variables des sons utilisés dans le jeu.

```
Serial port;
```

On relie le port d'Arduino et Processing.

```
mechante[] badtaupes = new mechante[8];
```

On crée un tableau des différentes Taupes affichées à l'écran, en l'occurrence 8.

```
PImage easterEgg;
PImage imageTitle;
PImage imagetaupe;
PImage imageFond;
PImage imageEnd;
```

Voici l'initialisation des images des Taupes dont nous avions parlé précédemment.

```
boolean startStarted = false;
boolean musicStarted = false;
boolean endStarted = false;
```

On crée des booléens pour les états des musiques qu'on initialise sur false afin qu'aucune se lance en même temps.

```
int finalScore = 0;
```

On initialise le Score final, qui est de 0 par défaut au début de la partie.

```
int rand(int min, int max) {
  return round(random(min, max + 1));
}
```

Une fonction ***rand*** qui permet de choisir un nombre aléatoire entre deux valeurs. La fonction round arrondi le nombre permettant d'avoir un nombre entier et le max+1 permet d'avoir la valeur maximale même après l'arrondi (ex : 99,1542 +1 → 99 +1 = 100).

```
int StyleTaupe;
```

Voici l'initialisation du style des taupes.

```
long specialEndTime = 0;
```

On initialise la durée de l'easter egg.

```
class PlayerScore {
  String name;
  int score;

  PlayerScore(String n, int s) {
    name = n;
    score = s;
  }
}
```

Cette classe permet de stocker le score du joueur tout le long de la partie, et de leur attribuer au nom du joueur.

```
ArrayList<PlayerScore> topScores = new ArrayList<PlayerScore>();
```

On initialise un tableau qui correspond au tableau des scores avec de fin de partie.

```
void setup() {
  fullScreen();
  port = new Serial(this, "COM3", 9600);
  port.bufferUntil('\n');
  imageTitle = loadImage("img/title_bcknd.png");
  imageFond = loadImage("img/fond.png");
  imageEnd = loadImage("img/ecranscore.png");
```

On setup l'aire de Jeu en choisissant la taille de la fenetre, les ports utilisés pour la boîte à buzzer et on charge les images de fond.

```
  badtaupes[0] = new mechante(200, 625, 150);//blancG
  badtaupes[1] = new mechante(425, 825, 150);//rougeG
  badtaupes[2] = new mechante(425, 425, 150);//bleuG
  badtaupes[3] = new mechante(650, 625, 150);//vertG
  badtaupes[4] = new mechante(1120, 625, 150);//vertD
  badtaupes[5] = new mechante(1345, 825, 150);//rougeD
  badtaupes[6] = new mechante(1345, 425, 150);//bleuD
  badtaupes[7] = new mechante(1570, 625, 150);//blancD
```

On place les Taupes dans les différents trous prévus, en précisant en commentaire à quel buzzer les coordonnées correpondent.

```
  music = new SoundFile(this, "son/musique_1mn30.wav");
  start = new SoundFile(this, "son/son_debut.wav");
  end = new SoundFile(this, "son/son_fin.wav");
  nice = new SoundFile(this, "son/taupe_gentille.wav");
  jog = new SoundFile(this, "son/taupe_Jogulin.wav");
  jul = new SoundFile(this, "son/taupe_Julian.wav");
  luc = new SoundFile(this, "son/taupe_Lucas.wav");
  bad = new SoundFile(this, "son/taupe_mechante.wav");
  noa = new SoundFile(this, "son/taupe_Noah.wav");
  tom = new SoundFile(this, "son/taupe_Tom.wav");
  egg = new SoundFile(this, "son/easterEgg.wav");
```

On donne à chaque variable son fichier audio correspondant.

```
  topScores.add(new PlayerScore("Tom", 2450));
  topScores.add(new PlayerScore("Oscar", 3154));
  topScores.add(new PlayerScore("Oscar", 3220));
}
```

On clôture le ***setup*** par l'affichage du tableau des scores finaux, ici en dur par faute de temps.

```
void draw() {
  switch (gameState) { // Switch pour changer entre les differents ecrans
      de jeu
    case TITLE: // Si jamais on est sur l'ecran titre
      displayTitleScreen(); // On utilise la fonction pour afficher l'ecran
          d'accueil
      break;
    case GAME: // Si jamais on est sur le jeu
      background(imageFond); // On change l'image de fond
      for (int i = 0; i < 8; i++) { // On initialise toutes les taupes
        badtaupes[i].display();
      }
```

On a créé une fonction ***draw*** en précisant ce qu'elle doit affiché en fonction de l'etat du jeu. Si l'etat du jeu est "GAME" donc que la partie commence, toutes les Taupes sont initialisés.

```
if (!start.isPlaying() && !startStarted){ // Si jamais la musique start n'
    est pas en train de jouer et qu'elle n'a pas encore ete jouee
        start.play(); // On joue la musique
        startStarted = true; // On dit qu'elle a ete jouee
      }

      if (!start.isPlaying()){ // Si jamais la musique start n'est pas
          jouee
        if (!music.isPlaying() && !musicStarted){ // Si jamais la musique
            music n'est pas jouee et qu'elle n'a pas encore ete jouee
          music.play(); // On joue la musique music
          musicStarted = true; // On dit qu'elle a ete jouee
        }
        if (!end.isPlaying() && !music.isPlaying() && musicStarted && !
            endStarted && gameState != SPECIAL_END){ // Si la musique end n
            'est pas jouee et que la musique music n'est pas jouee et que
            la musique music a deja ete jouee et que la musique end n'a pas
             ete jouee
          end.play(); // On joue la musique end
          endStarted = true; // On dit qu'elle a ete jouee
        }
      }
      break;
```

Ce morceau de code permet de lancé les musiques du jeu au bon moment, en fonction de l'état des musiques précédentes et l'état de jeu.

```
    case SPECIAL_END:
      background(easterEgg);
        if (!egg.isPlaying() && millis() - specialEndTime >= 5000) {
            gameState = END;
        }
        break;
```

```
    case END: // Si jamais on est sur l'ecran de fin
      displayEndScreen(); // On utilise la fonction pour afficher l'ecran
          de fin
      break;
  }
}
```

On clôture la fonction ***draw*** avec le cas de l'affichage de l'easter egg si l'etat du jeu est "SPECIAL_END".

```
void keyPressed() {
  if (gameState == TITLE && key == ' ') { // Si jamais on est sur l'ecran
      titre et si la touche espace est pressee
    gameState = GAME; // On passe l'etat de jeu a "GAME"
    // On reinitialise les variables des musiques qu'on utilise pour lancer
        le jeu
    startStarted = false;
    musicStarted = false;
    endStarted = false;
    finalScore = 0;
    // On envoie a l'arduino que le jeu est lance pour lancer le jeu
    port.write("START\n");
  } else if (gameState == END && key == ' ') { // Si on est sur l'ecran de
      fin et qu'on appuie sur espace
    gameState = TITLE; // On retourne sur l'ecran titre
  }
}
```

La fonction ***keyPressed*** est utilisé pour changer l'état de jeu en dehors des parties, comme pour lancer la partie ou revenir à l'écran titre après l'affichage des scores. Ces actions sont faites pour que le joueur puisse lancer sa partie quand il est prêt ou qu'une autre partie puisse être relancé sans encombre.

```
void displayTitleScreen() {
  background(imageTitle); // On affiche l'ecran titre
}
```

Dès que cette fonction est utilisé, on affiche l'écran titre (une image).

```
void displayEndScreen() {
  background(imageEnd);
  fill(255);
  textSize(60);
  text(finalScore, width - 385, height / 2 + 140);
```

```
  // Triez la liste par ordre decroissant des scores
  Collections.sort(topScores, (a, b) -> b.score - a.score);

  // Affichez les trois meilleurs scores
  for (int i = 0; i < min(3, topScores.size()); i++) {
    PlayerScore ps = topScores.get(i);
    text(ps.name + ": " + ps.score, 290, height / 2 + 110 + i * 150);
  }
}
```

Voici la fonction qui affiche l'écran de fin. Cela affiche le tableau des scores et le score final du joueur.

```
void hasard(mechante taupe) {
  float randimage = rand(1, 100);
  println(randimage);
  if (randimage >= 1 && randimage <= 80) {
    taupe.style = 1;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/badtaupe.png");
  }
```

La fonction ***hasard*** permet de choisir aléatoirement le style de la Taupe qui va apparaître et de charger le visuel de la taupe correspondante. Ici, la Taupe Méchante a 80% de chance d'apparaître.

```
else if (randimage > 80 && randimage < 96) {
    taupe.style = 2;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/goodtaupe.png");
  }
```

La Taupe Gentille a 15% de chance d'apparaître.

```
else if (randimage == 96) {
    taupe.style = 11;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/tomtaupe.png");
  } else if (randimage == 97) {
    taupe.style = 12;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/lucastaupe.png");
  } else if (randimage == 98) {
    taupe.style = 13;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/juliantaupe.png");
  } else if (randimage == 99) {
```

```
    taupe.style = 14;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/noahtaupe.png");
  } else if (randimage == 100) {
    taupe.style = 15;
    StyleTaupe = taupe.style;
    imagetaupe = loadImage("img/jocelyntaupe.png");
  }
}
```

Enfin, chaque Taupes Secrètes a une chance de 1% d'apparaître, cela représente un pourcentage d'apparition compris entre 96 et 100, ce qui équivaut à une probabilité de 5% de rencontrer une Taupe Secrète.

```
void badtaupeVisible(int index, int value) {
  badtaupes[index].visible = value;
}
```

Cette fonction permet d'afficher ou non chaque Taupe.

```
void serialEvent(Serial port) {
  String serialStr = port.readStringUntil('\n');
  serialStr = trim(serialStr);

  if (serialStr.equals("SPECIAL_END")) {
        stopAllSounds();
        easterEgg = loadImage("img/specialEnd.png");
        egg.play();
        gameState = SPECIAL_END;
        specialEndTime = millis();
    }
```

La fonction ***serialEvent*** permet d'écouter les données entrantes et réagit en conséquence, déclenchant différentes actions en fonction des messages reçus dans la trame. Ici, si Arduino envoie "SPECIAL_END" dans la trame, alors l'easter egg se lance.

```
  if (serialStr.equals("END")) { // Si jamais on recoit END dans le serial
    gameState = END; // On passe l'etat du jeu a END
  }
```

Si la trame Arduino renvoie "END", on change l'état du jeu et la partie s'arrete.

```
 if (serialStr.startsWith("Score")) { // Si jamais le serial commence par "
     Score"
    String[] parts = split(serialStr, ',');
    if (parts.length > 1) {
```

```
      finalScore = int(parts[1]); // On recupere le score envoye depuis l'
          arduino
    }
  }
```

Cette fonction permet d'afficher le score dans la console et de la stocker pour pouvoir l'afficher à l'écran final du tableau des scores.

```
// On verifie la trame recue d'Arduino avant de l'utiliser
  int values[] = int(split(serialStr, ','));
  if (values.length == 2) {
    print(values[0]);
    print(",");
    println(values[1]);

    for (int i = 2; i < 10; i++) {
      int index = i - 2;
      if (values[0] == i && values[1] == 1) {
        hasard(badtaupes[index]);
        badtaupeVisible(index, 1);
      } else if (values[0] == i && values[1] == 0) {
        badtaupeVisible(index, 0);
      } else if (values[0] == i && values[1] == 2) {
        badtaupeVisible(index, 0);
        score();
      }
    }
  }
}
```

On clôture le ***serialEvent*** par la liaison entre entre les Taupes et les buzzers. Sachant que sur le tableau des Taupes, elles sont listés de 1 à 8 et que les buzzers sont branchés sur les ports 2 à 10, on effectue une soustraction simple pour que les deux tableaux s'alignent.

```
void stopAllSounds() {
    start.stop();
    music.stop();
    end.stop();
    nice.stop();
    jog.stop();
    jul.stop();
    luc.stop();
    bad.stop();
    tom.stop();
}
```

Voici une fonction simple permettant d'arrêter tous les sons en cours.

```
void score(){
  int multiplicateur;
  if (StyleTaupe==1){
    bad.play();
    multiplicateur = 1;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }
```

La fonction ***score*** qui permet de mettre en place un multiplicateur, le son correspondant à la taupe et la bonne valeur en commentaire en fonction du style de la Taupe tapée. Ici, l'exemple est donnée pour la Taupe de style n°1, c'est-à-dire la Taupe Méchante.

```
else if (StyleTaupe==2){
    nice.play();
    multiplicateur = -3;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }
```

Ici, la même chose mais pour la Taupe Gentille.

```
else if(StyleTaupe==11){
    tom.play();
    multiplicateur = 5;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }else if(StyleTaupe==12){
    luc.play();
    multiplicateur = 5;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }else if(StyleTaupe==13){
    jul.play();
    multiplicateur = 5;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }else if(StyleTaupe==14){
    noa.play();
    multiplicateur = 5;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }else if(StyleTaupe==15){
    jog.play();
    multiplicateur = 5;
    String data = str(multiplicateur) + "\n";
    port.write(data);
  }
}
```

Et enfin on clôture la dernière fonction, donc la fonction du ***score*** on fait ma même chose mais pour les Taupes Secrètes. Chacune a son propre son mais elles ont toutes le même multiplicateur (x5).

4 Pistes d'amélioration

4.1 Structure du code

Certains éléments de la structure du code, surtout sur Processing, ne sont pas conventionnels et méritent d'être retravaillés. Cela est essentiellement dû au fait que nous avons chacun codé, puis assemblé nos parties (Tom a fait le code Arduino, Lucas a fait la base du Processing...). De ce fait, le code n'est pas très homogène et à certains endroits, certaines parties de code sont floues. Concernant le côté conventionnel, il y a des erreurs, les principales étant l'utilisation de mauvais types de variables (des INT à la place De BOOLEAN et des FLOAT à la place de INT), et des noms de variables/classes incohérents (classe méchante à la place de taupe ou variable cote plutôt que size).

Nous aurions dû simplement relire le code et l'harmoniser, ainsi qu'ajouter des commentaires.

4.2 Gameplay

4.2.1 Multijoueur

A la base du projet, nous avions pensé à intégrer un mode multijoueur à notre jeu. Ce mode multijoueur a été mis au second plan et devait être ajouté un fois sûr que le mode solo marchait correctement. Le multijoueur n'a pas été implémenté par manque de temps. Nous voulions faire un mode multijoueur à 2 joueurs en 1 contre 1.

4.2.2 Séries

Cette idée nous a été donnée le jour de la présentation de notre projet comme un retour. Une des pistes d'amélioration est donc l'ajout de série, qui consiste donc en un multiplicateur ajouté au score, ou des points bruts ajoutés à la fin, augmentant en fonction du nombre de taupes méchantes tapées d'affilé. C'est-à-dire simplement que si on tape plusieurs taupes méchantes à la suite, on a plus de points.

4.3 Tableau des scores

Nous avons dans notre code des parties dédiées à la gestion et à l'apparition d'un tableau des scores sur l'écran de fin. Si l'apparition fonctionne parfaitement, la gestion n'a pas été implémentée car elle ne marchait pas.

Les données de notre tableau des scores étaient donc codées en dur : nous changions le tableau des scores depuis le code lorsqu'un joueur faisait un meilleur score.

Nous voulons faire en sorte que lorsqu'un joueur arrive à faire un score le classant dans les 3 meilleurs joueurs, il peut rentrer son nom pour que son nom suivi de son score apparaisse dans le tableau des scores lors de la partie suivante.

4.4 Responsive

Les derniers éléments de code ont été fait par Noah sur son ordinateur. Ainsi pour les tests, l'ajustement des taupes (l'endroit où l'image des taupes apparait pour correspondre aux terriers sur l'image de fond) a été prévu pour correspondre à l'écran du PC de Noah, vu que les variables positions de l'image des taupes ont étés mises en brut. Donc sur d'autres ordinateurs, les taupes n'apparaissent pas au bon endroit.

Il faut donc que nous retravaillons le Responsive pour que le jeu marche sur tous les écrans.