Guide d'animation

Work in progress

Guide d'animation "Pixel Shooter"

□□ Objectifs principaux :

- S'initier à la soudure à l'étain
- Découvrir les micros-controleurs

Objectifs secondaires :

- Découvrir le fonctionnement de ....

□□ Public cible :

- Ados / enfants de 11 à 14 ans
- 6 participants max

□□ Pré-requis :

- savoir utiliser clavier/souris
- pouvoir tenir un stylo "*pour par exemple vérifier la capacité à tenir un fer à souder*"

□□ Durée de l'atelier : 2h (3h avec découpe laser)

□□ Matériel à préparer

□□ Machines & Outils

- [ ] Ordinateurs X8
- [ ] Kits fer à souder X4
- [ ] Kits pinces...
- [ ]

Consommables

- [ ] Découpes de CP
- [ ] Impression 3D des 2 pièces
- [ ] Vis M4 conique (pour venir à plat du CP)
- [ ] Bande de Led WS282B
- [ ] 4 touche de claviers insérables (14x14mm)
- [ ] Arduino Nano/Micro
- [ ] Résistance de 330 à 470 ohm
- [ ] Condensateur de 470 à 1000 µf (au moins 6.5V)
- [ ]

Fichiers et liens utiles

Le projet d'origine https://github.com/4aka/Arduino-LED-Game

Le plan de soudure plan_soudure_PixelSHooter.svg

Le code Arduino ArduinoLEDGame.ino

Fichiers de découpe Laser ==> Pixel Shooter.svg

Fichiers 3D ===> Pixel Shooter coté droit.3mf Pixel shooter coté gauche.3mf

Version trou vis de 3.5mm Pixel Shooter coté droit3,5.3mf Pixel Shooter coté gauche3,5.3mf

---

Déroulé de l'animation

Introduction *20min*

L'animateur fait un tour des prénoms et lance la discutions pour casser la glace.
`"Vous connaissiez le Fablab ?"`
`"Quelqu'un a déjà pratiqué la soudure à l'étain ?`

Introduction à la soudure 20min

L'animateur **interroge** en présentant les différents outils rencontrés sur un poste de travail de soudure.
L'animateur **présente** la technique de soudure à l'étain.

> *Ne pas hésiter à mimer le geste de soudure plusieurs fois, les apprenants ont tendance à ne pas laisser assez longtemps la pane chauffer l'élément à souder !*

Soudure en autonomie 40min

L'animateur **présente** les composants ainsi que le plan de soudure papier .
Les apprenants **soudent** les composants dans l'ordre défini par le plan.
L'animateur passe pour **débloquer** les situations et **corriger** les gestes.

> *Attention aux risques de brûlure ! Prévoir de quoi les traiter.*

Clôture de l'atelier 10min

L'animateur fait **reformuler** les étapes pour valider la compréhension par tous les participants.
L'animateur **Interroge** sur les points améliorables de l'atelier.
`"Bon alors, de 1 a 5  ==> 1, j'ai perdu mon temps, 5 c'était top génial ! Vous vous situez ou ? "`

□□ Point rangement

Les postes sont nettoyés ?
Les fers sont débranchés ?
Des accus lithium (ou autres danger du genre) qui traînent ?

L'animateur **rappelle** le contexte Fablab et **clôture.**

Le programme à televerser

Ne pas hésiter à rajouter une ligne dans le setup() :

```
FastLED.setBrightness()    #luminosité des LEDs de 0 à 255
```

```
 #include <FastLED.h>


// --- Hardware set up ---

#define NUM_LEDS 32        // led count

#define LED_PIN 6

#define GREEN_BTN 4

#define RED_BTN 2

#define BLUE_BTN 3

#define LEVEL_UP_BTN 5


CRGB leds[NUM_LEDS];


// Game colors

CRGB colors[3] = {CRGB::Red, CRGB::Blue, CRGB::Green};


// --- Game vars ---

int currentLevel = 1;
```

```
int successfulHits = 0;


// Move timers

unsigned long lastEnemyMoveTime = 0;

unsigned long lastShotMoveTime = 0;


// Vars for buttons (Check var state)

bool lastRedState = HIGH;

bool lastGreenState = HIGH;

bool lastBlueState = HIGH;

bool lastLevelUpState = HIGH;

unsigned long lastDebounceTime = 0;

const int debounceDelay = 50;


// Object structures

struct FallingPixel {

  int pos;

  CRGB color;

  bool active;

};


struct ShotPixel {
```

```
  int pos;
  CRGB color;
  bool active;
};

// Increase objects value
const int MAX_OBJECTS = 40;
FallingPixel enemies[MAX_OBJECTS];
ShotPixel shots[MAX_OBJECTS];

int spawnDistanceCounter = 0;
int nextSpawnDistance = 0;

void setup() {
  Serial.begin(9600);


  FastLED.addLeds<WS2812B, LED_PIN, GRB>(leds, NUM_LEDS);
  FastLED.clear();
  FastLED.show();

  pinMode(GREEN_BTN, INPUT_PULLUP);
  pinMode(RED_BTN, INPUT_PULLUP);
```

```
  pinMode(BLUE_BTN, INPUT_PULLUP);
  pinMode(LEVEL_UP_BTN, INPUT_PULLUP);

  resetGame();
}

void loop() {
  handleButtons();

  // Speed define
  int enemySpeedDelay = 1000 / (currentLevel + 1);
  int shotSpeedDelay = enemySpeedDelay / 2; // Shot twise faster

  bool needsDraw = false;

  // Shot speed
  if (millis() - lastShotMoveTime >= shotSpeedDelay) {
    moveShots();
    checkCollisions();
    lastShotMoveTime = millis();
    needsDraw = true;
  }
```

```
  // Enemies speed
  if (millis() - lastEnemyMoveTime >= enemySpeedDelay) {
    moveEnemies();
    checkCollisions();
    spawnEnemies();
    lastEnemyMoveTime = millis();
    needsDraw = true;
  }

  if (needsDraw) {
    drawGame();
  }

  // Auto level up
  if (successfulHits >= 50 && currentLevel < 5) {
    levelUp();
    successfulHits = 0;
  }
}
```

```
void handleButtons() {
  bool readingRed = digitalRead(RED_BTN);
  bool readingGreen = digitalRead(GREEN_BTN);
  bool readingBlue = digitalRead(BLUE_BTN);
  bool readingLevelUp = digitalRead(LEVEL_UP_BTN);

  if (millis() - lastDebounceTime > debounceDelay) {
    if (readingRed == LOW && lastRedState == HIGH) {
      shoot(CRGB::Red);
      lastDebounceTime = millis();
    }
    if (readingGreen == LOW && lastGreenState == HIGH) {
      shoot(CRGB::Green);
      lastDebounceTime = millis();
    }
    if (readingBlue == LOW && lastBlueState == HIGH) {
      shoot(CRGB::Blue);
      lastDebounceTime = millis();
    }
    if (readingLevelUp == LOW && lastLevelUpState == HIGH) {
```

```
      if (currentLevel < 5) levelUp();
      lastDebounceTime = millis();
    }
  }

  // Keep current state for the next loop
  lastRedState = readingRed;
  lastGreenState = readingGreen;
  lastBlueState = readingBlue;
  lastLevelUpState = readingLevelUp;
}

void shoot(CRGB color) {
  for (int i = 0; i < MAX_OBJECTS; i++) {
    if (!shots[i].active) {
      shots[i].pos = 1; // 1 LED HIGH on the base
      shots[i].color = color;
      shots[i].active = true;
      break;
    }
  }
```

```
}


void spawnEnemies() {
  spawnDistanceCounter++;
  if (spawnDistanceCounter >= nextSpawnDistance) {
    for (int i = 0; i < MAX_OBJECTS; i++) {
      if (!enemies[i].active) {
        enemies[i].pos = NUM_LEDS - 1;
        enemies[i].color = colors[random(0, 3)];
        enemies[i].active = true;
        spawnDistanceCounter = 0;
        nextSpawnDistance = random(1, 6);
        break;
      }
    }
  }
}


void moveEnemies() {
  for (int i = 0; i < MAX_OBJECTS; i++) {
    if (enemies[i].active) {
```

```
      enemies[i].pos--;



      // If enemy touches the base
      if (enemies[i].pos <= 0) {
        drawGame();
        delay(300);
        resetGame();
        return;
      }
    }
  }
}

void moveShots() {
  for (int i = 0; i < MAX_OBJECTS; i++) {
    if (shots[i].active) {
      shots[i].pos++;
      if (shots[i].pos >= NUM_LEDS) {
        shots[i].active = false;
      }
    }
```

```
  }
}

void checkCollisions() {
  for (int e = 0; e < MAX_OBJECTS; e++) {
    if (!enemies[e].active) continue;

    for (int s = 0; s < MAX_OBJECTS; s++) {
      if (!shots[s].active) continue;

      if (enemies[e].pos == shots[s].pos || enemies[e].pos == shots[s].pos - 1) {

        if (enemies[e].color == shots[s].color) {
          enemies[e].active = false;
          shots[s].active = false;
          successfulHits++;
        } else {
          CRGB oldColor = enemies[e].color;
          CRGB newColor;
          do {
```

```
            newColor = colors[random(0, 3)];

          } while (newColor == oldColor);



          enemies[e].color = newColor;

          shots[s].active = false;

        }

      }

    }

  }

}


void levelUp() {

  currentLevel++;

  FastLED.clear();

  fill_solid(leds, NUM_LEDS, CRGB::White);

  FastLED.show();

  delay(100);

  FastLED.clear();

  FastLED.show();

}
```

```
void resetGame() {

  currentLevel = 1;

  successfulHits = 0;

  spawnDistanceCounter = 0;

  nextSpawnDistance = random(1, 6);



  for (int i = 0; i < MAX_OBJECTS; i++) {

    enemies[i].active = false;

    shots[i].active = false;

  }



  // Show Game over

  FastLED.clear();

  fill_solid(leds, NUM_LEDS, CRGB::Red);

  FastLED.show();

  delay(500);

  FastLED.clear();

  FastLED.show();

}



void drawGame() {
```

```
  FastLED.clear();



  // First pixel

  leds[0] = CRGB(20, 20, 20);



  for (int i = 0; i < MAX_OBJECTS; i++) {

    if (enemies[i].active && enemies[i].pos > 0 && enemies[i].pos < NUM_LEDS) {

      leds[enemies[i].pos] = enemies[i].color;

    }

    if (shots[i].active && shots[i].pos > 0 && shots[i].pos < NUM_LEDS) {

      leds[shots[i].pos] = shots[i].color;

    }

  }



  FastLED.show();

}
```

Révision #14
Créé 2026-07-16 08:56:11 UTC par Florian
Mis à jour 2026-07-17 12:36:52 UTC par Florian