CPNV, Centre professionnel du Nord vaudois

Projet du Simon

Candas Kuran
03/11/2023

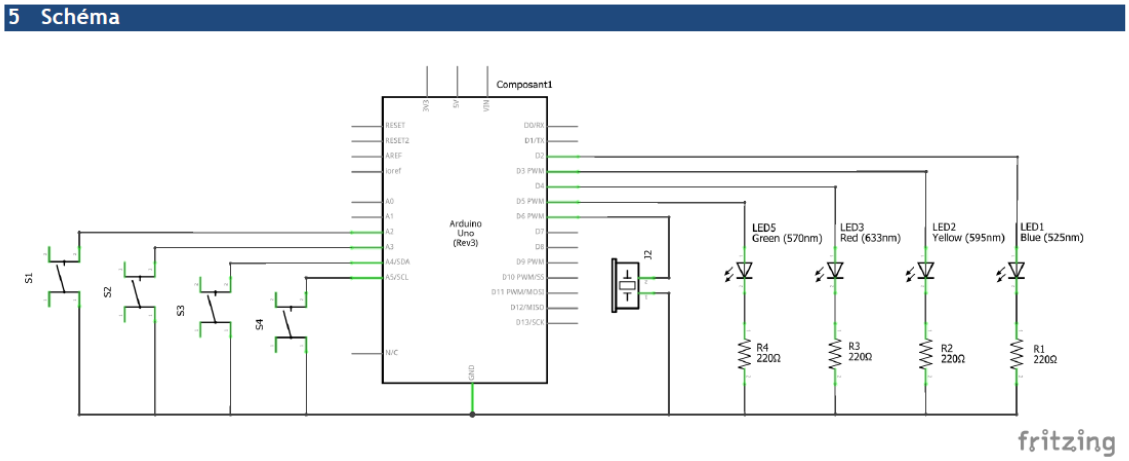
Table des matières

Explication du schéma

Le schéma illustre le montage électronique du jeu Simon. Il comprend :

- Un Arduino qui sert de cerveau au jeu.
- Quatre boutons, utilisés pour saisir la séquence.
- Quatre LEDs (Verte, Rouge, Jaune, Bleue) qui affichent la séquence à reproduire.
- Des résistances de 220Ω pour chaque LED afin de limiter le courant.
- Un buzzer

L'Arduino est connecté à chacune des LEDs et des boutons à travers des pins spécifiques. Les LEDs s'allument en fonction de la séquence générée par l'Arduino, et les boutons permettent au joueur de reproduire cette séquence.



Description du Schéma : Le schéma montre un Arduino. Les connexions de pins sont les suivantes :

LEDs:

- **LED Vert :** Connecté au Pin 2.
- **LED Rouge :** Connecté au Pin 3.
- **LED Jaune :** Connecté au Pin 4.
- **LED Bleu :** Connecté au Pin 5.

Les autres extrémités des LEDs doivent être connectées à une résistance et ensuite à la masse (GND). Cela est nécessaire pour protéger les LEDs d'un courant excessif.

Boutons :

- **Bouton Vert :** Connecté au Pin A2.
- **Bouton Rouge :** Connecté au Pin A3.
- **Bouton Jaune :** Connecté au Pin A4.
- **Bouton Bleu :** Connecté au Pin A5.

Les autres extrémités des boutons doivent être connectées à la masse (GND). Il est également recommandé d'ajouter une résistance de rappel (pull-up) à l'autre extrémité du bouton connecté au pin correspondant de l'Arduino. Cela garantit que le pin sera dans un état HIGH lorsque le bouton n'est pas pressé.

```
const int buttons[] = {A2, A3, A4, A5}; // Pins des boutons
const int leds[] = {2, 3, 4, 5};        // Pins des LEDs
```

Buzzer:

- Une extrémité du buzzer doit être connectée au Pin 6.
- L'autre extrémité doit être connectée à la masse (GND).
- Après avoir réalisé ces connexions, vous pouvez télécharger votre code sur l'Arduino.

```
const int TONE_PIN = 6;                 // Pin du buzzer
```

Description du code

- **buttons[]** : Un tableau qui définit les pins des boutons.
- **leds[]** : Un tableau qui définit les pins des LEDs.
- **tones[]** : Les fréquences associées à chaque bouton/LED pour produire un son distinct lorsqu'ils sont activés.
- **ERROR_TONE** : La fréquence à jouer en cas d'erreur.
- **NB_COLORS** : Le nombre total de couleurs/boutons.
- **TONE_PIN** : Le pin où le buzzer est connecté.
- **MAX_SEQUENCE** : La longueur maximale de la séquence que le joueur doit suivre.

- **sequence[]** : Un tableau qui contiendra la séquence générée aléatoirement pour le joueur à suivre.
- **gameStarted** : Une variable booléenne qui indique si le jeu a commencé.

Définition des notes

Pour le jeu, nous devons importer certaines notes pour la musique de début et de fin. Pour cela, nous importons ces notes au début du projet.

```
// Définition des notes

#define NOTE_C4  262
#define NOTE_D4  294
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_G4  392
#define NOTE_A4  440
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_D5  587
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_G5  784
#define NOTE_A5  880
#define NOTE_B5  988
#define NOTE_SUSTAIN 300
```

Initialisation avec setup()

- Initialise la communication série pour la débogage et les messages.
- Configure les pins des boutons et des LEDs.
- Attend que le joueur commence le jeu avec la fonction **waitForStart()**.

Boucle principale avec loop()

Si le jeu a démarré :

- Jouer la séquence avec **playSequence()**.
- C'est le tour du joueur avec **playerTurn()**.

Fonction pour attendre le démarrage du jeu avec waitForStart()

- Attend que le joueur appuie sur le bouton A2.
- Joue une séquence d'introduction avec les LEDs.
- Joue une mélodie pour indiquer le démarrage avec la fonction **startupMusicAndLeds()**. Voici le code de cette fonction ;

```
// Fonction pour jouer la séquence de démarrage
void startupMusicAndLeds() {
  // Jouer une musique et allumer les LEDs pour indiquer le début du jeu
  for (int i = 0; i < NB_COLORS; i++) {
    tone(TONE_PIN, tones[i], 500);
    digitalWrite(leds[i], HIGH);
    delay(600);
    digitalWrite(leds[i], LOW);
  }
}
```

Indique que le jeu démarre et commence un nouveau jeu avec **startNewGame()**.

Autres fonctions notables

- **playSequence()**: Joue la séquence actuelle.

```
// Fonction pour jouer la séquence actuelle
void playSequence() {
  for (int i = 0; i < sequenceLength; i++) {
    int color = sequence[i];
    digitalWrite(leds[color], HIGH);
    tone(TONE_PIN, tones[color]);
    delay(600);
    digitalWrite(leds[color], LOW);
    noTone(TONE_PIN);
    delay(200);
  }
}

```

- **playerTurn()**: Gère le tour du joueur, vérifie si le joueur a correctement suivi la séquence ou non.

```
// Fonction pour le tour du joueur
void playerTurn() {
  // Laisser le joueur essayer de suivre la séquence
  playerStep = 0;

  while (playerStep < sequenceLength) {
    for (int i = 0; i < NB_COLORS; i++) {
      if (digitalRead(buttons[i]) == LOW) {
        if (i == sequence[playerStep]) {
          playerStep++;
          digitalWrite(leds[i], HIGH);
          tone(TONE_PIN, tones[i]);
          delay(300);
          digitalWrite(leds[i], LOW);
          noTone(TONE_PIN);
        } else {
          playError(); // Jouer un ton d'erreur si le joueur fait une
erreur
          Serial.println("Error! The game restarts.");
          startupMusicAndLeds();
          delay(1000);
          startNewGame(); // Recommencez le jeu après une erreur
          return;
        }
      }
    }
  }
```

Fonction d'Accélération du Jeu

Au fur et à mesure que le joueur réussit à suivre la séquence correcte des lumières et des tons, le jeu devient plus difficile en augmentant la vitesse de la séquence. La fonction d'accélération ajuste le délai entre les lumières et les tons, rendant le jeu plus exigeant pour le joueur. Cette fonction est conçue pour augmenter l'excitation et le défi en réduisant progressivement le temps que le joueur a pour répondre.

```
void playSequence() {
  for (int i = 0; i < sequenceLength; i++) {
    int color = sequence[i];
    digitalWrite(leds[color], HIGH);
    tone(TONE_PIN, tones[color]);
    delay(currentDelay);
    digitalWrite(leds[color], LOW);
    noTone(TONE_PIN);
    delay(currentDelay / 3);
  }

  // Augmenter la vitesse du jeu tous les 3 tour
  if (sequenceLength % 3 == 0) {
    currentDelay *= 0.75; // Augmenter le vitesse de 25%.
    if (currentDelay < 100) { // Ajouter une limite inférieure pour éviter le
jeu ne devienne trop rapide,
      currentDelay = 100;
    }
  }
}
```

- **playError()**: Joue un son d'erreur et fait clignoter tous les LEDs en cas d'erreur du joueur.

```
•  void playError() {
•    tone(TONE_PIN, NOTE_E5, 200);
•    delay(250);
•    tone(TONE_PIN, NOTE_D5, 200);
•    delay(250);
•    tone(TONE_PIN, NOTE_C5, 200);
•    delay(250);
•    noTone(TONE_PIN);
•    blinkAllLeds();
•  }
•
```

- **playVictory()**: Joue une mélodie de victoire lorsque le joueur réussit à suivre la séquence maximale.

Diagramme de Flux du Jeu Simon

Début

1. **Initialisation des boutons, LEDs, tons et variables du jeu :**
    - **Boutons :** A2, A3, A4, A5
    - **LEDs :** 2, 3, 4, 5
    - **Tons :** Associés à chaque couleur
    - **Variables du jeu :** séquence, longueur de la séquence, étape du joueur, jeu démarré
2. **Attente du démarrage du jeu :**
    - Attendez que le joueur appuie sur le bouton A2 (bouton vert) pour démarrer le jeu.
3. **Séquence d'introduction des LEDs :**
    - Jouez la musique de démarrage et allumez les LEDs
4. **Générer une nouvelle séquence aléatoire pour le jeu**

```
randomSeed(analogRead();
```

5. **Boucle Principale du Jeu : Si le jeu a démarré, suivez les étapes suivantes.**

    a. Jouer la Séquence : Allumez les LEDs et jouez les tons selon la séquence actuelle.

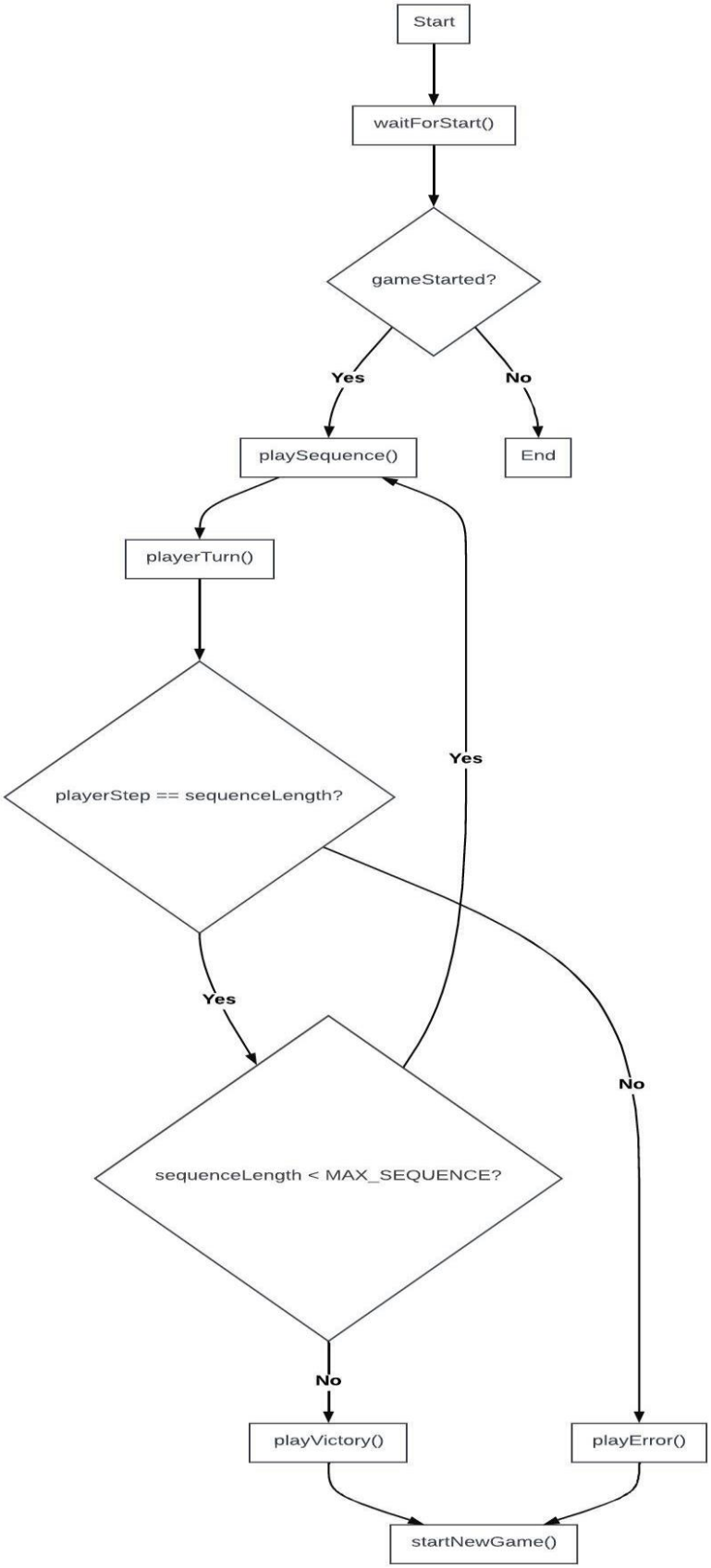
    b. Tour du Joueur : Le joueur essaie de reproduire la séquence en appuyant sur les boutons.
    - Pour chaque pression sur un bouton, vérifiez si elle correspond à l'étape actuelle de la séquence.
    - Si c'est correct, passez à l'étape suivante.
    - Sinon, jouez un ton d'erreur, affichez la séquence LED d'erreur et redémarrez le jeu.

    c. Vérifier la progression du joueur :
    - Si le joueur suit avec succès la séquence, augmentez la longueur de la séquence si elle n'est pas à son maximum.
    - Si la longueur de la séquence est à son maximum, affichez la séquence de victoire, jouez les tons de victoire et redémarrez le jeu.
6. **Erreur :**
    - En cas d'erreur, jouez le ton d'erreur et faites clignoter toutes les LEDs.

7. **Victoire :**

    - Si le joueur termine la séquence maximale, jouez un ton de victoire et faites clignoter les LEDs de manière festive.

Fin du jeu


Start
waitForStart()
gameStarted?
Yes
No
playSequence()
End
playerTurn()
playerStep == sequenceLength?
Yes
Yes
No
sequenceLength < MAX_SEQUENCE?
No
playVictory()
playError()
startNewGame()

Conclusion

Le "Projet Simon" est une réalisation technique qui vise à reproduire le jeu classique "Simon", un jeu de mémoire basé sur des séquences de couleurs et de sons. Grâce à des boutons physiques et des LEDs, les joueurs sont mis au défi de mémoriser et de reproduire une séquence croissante de signaux. Ce projet illustre l'interaction entre le matériel et le logiciel pour créer une expérience ludique et pédagogique. C'est un excellent exemple de la manière dont la technologie peut être utilisée pour recréer des jeux classiques tout en offrant l'opportunité d'apprendre les principes de la programmation et de l'électronique.