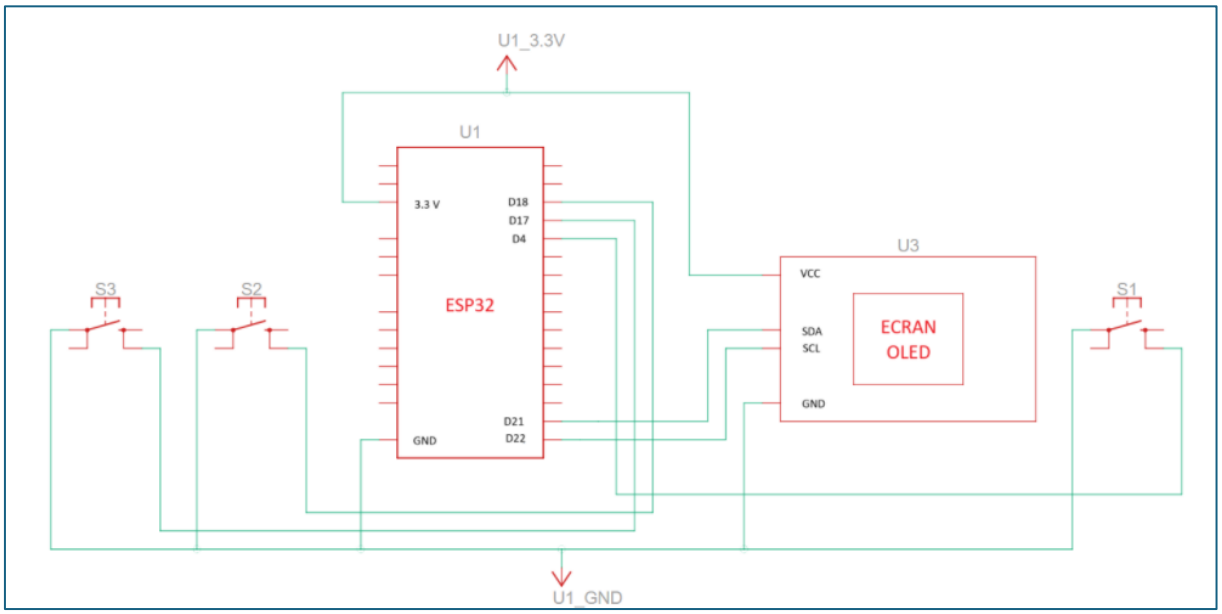
TAMAMONSTRA

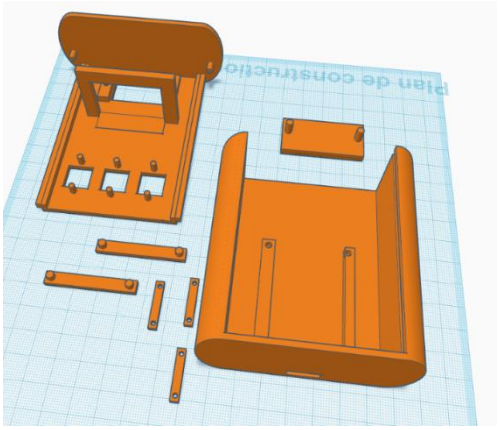
Les composants :

- Des fils électriques : permettent de connecter les différents composant à la carte.
- Une carte ESP32 : Le cœur du dispositif, assurant le traitement et la communication.
- Trois boutons poussoirs : actionne différentes actions dans le jeu.
- Un écran oled : affichage du jeu.
- Un boitier modélisé avec Tinkercard

Schéma électronique :

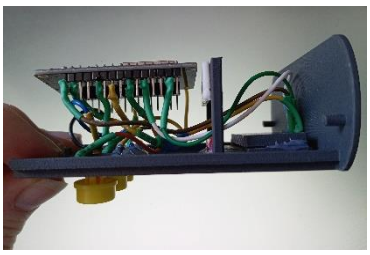
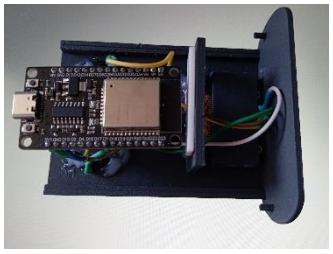


Modélisation du boitier :



- Le boitier est composé de deux parties facilitant la fermeture du boitier. On peut les faire glisser pour assurer une fermeture optimale.
- Les différentes barres permettent de caler les différents éléments contenus dans le boitier.
- Trois trous pour les trois boutons et une dernier pour le l'écran.
- Les deux barres dans la partie basse du boiter sert a placer la carte ESP32 et l'Arche permet de caler l'ESP32 une fois le boitier fermé. Un trou pour faire passer les fils.

Photographie de l’intérieur et de l’extérieur du boitier :

Les bibliothèques utilisées :

Preferences.h

Description :

Cette bibliothèque est utilisée pour stocker et récupérer des données non volatiles sur les microcontrôleurs ESP32. Elle permet de sauvegarder des valeurs qui persistent après un redémarrage ou une coupure d'alimentation.

Utilisation :

Elle est idéale pour sauvegarder des configurations, des paramètres de l'utilisateur, des états de l'application, etc.
Exemple :
preferences.begin("my-app", false) : Initialise les préférences
preferences.putInt("counter", 42) : Stocke la valeur entière `42` avec la clé "counter" dans les préférences
int counter = preferences.getInt("counter", 0) : Récupère la valeur entière associée à la clé "counter" depuis les préférences
preferences.end() : Ferme l'accès aux préférences

SPI.h :

Description :

La bibliothèque SPI (Serial Peripheral Interface) facilite la communication avec des périphériques compatibles SPI. SPI est un protocole de communication rapide et synchrone utilisé pour relier des microcontrôleurs à divers composants comme des mémoires, des écrans, des capteurs, etc.

Utilisation :

Utilisée pour des communications rapides entre le microcontrôleur et des périphériques SPI.

Wire.h:

Description :

La bibliothèque Wire permet de communiquer avec des périphériques utilisant le protocole I2C (Inter-Integrated Circuit). I2C est un protocole de communication bidirectionnel à deux fils souvent utilisé pour relier des capteurs, des écrans, des EEPROMs, etc.

Utilisation :

Utilisée pour interagir avec des périphériques I2C.

Adafruit_GFX.h:

Description :

La bibliothèque Adafruit GFX est une bibliothèque graphique universelle utilisée pour dessiner des formes, des textes et des images sur une variété d'écrans. Elle fournit des fonctions pour dessiner des lignes, des rectangles, des cercles, du texte, etc.

Utilisation :

Elle est utilisée avec des bibliothèques spécifiques à des écrans pour faciliter la création d'interfaces graphiques
Exemple :

display.begin(SSD1306_SWITCHCAPVCC, 0x3C) : Initialise l'écran OLED
display.clearDisplay() : Efface le contenu actuel de l'écran
display.setTextSize(1) : Définit la taille du texte à 1
display.setTextColor(WHITE) : Définit la couleur du texte en blanc.
display.setCursor(0, 0) : Positionne le curseur à la coordonnée (0, 0)
display.print("Hello, world!") : Affiche la chaîne de caractères "Hello, world!" à la position actuelle du curseur sur l'écran.
display.display() : Met à jour l'écran

Adafruit_SSD1306.h :

Description :

Cette bibliothèque est spécifiquement conçue pour les écrans OLED utilisant le pilote SSD1306. Elle repose sur Adafruit GFX pour les fonctions de dessin et ajoute des fonctions spécifiques pour contrôler les écrans OLED.
Utilisation :

Utilisée pour contrôler les écrans OLED SSD1306, en fournissant des fonctions pour initialiser l'écran, dessiner des pixels, des textes, des formes, etc.

Les fonctions créées :

23 fonctions créer :

void recuperation() : récupère différentes données stockées dans la carte grâce à la bibliothèque Preferences.

void sauvegarde() : sauvergarde dans la carte différentes données grâce à la bibliothèque Preférences.

void recuperationStat() : récupère les données concernant l'état du monstre dans la carte grâce à la bibliothèque Preferences.

void menu() : affiche et gère la navigation dans lu menu : dormir manger, sortir, accès aux stats, laver, soigner.

void mettreStatsOk() : permet à la fin d'une boucle de loop de d'éviter que les stats du tamagoshi ne dépasse 100 ou descende au-dessous de 0 .

void finCicle() : mets a jour les stats du monstre toutes les deux secondes et retirant des points de stats ou en ajoutant.

void fenetreStat() : affiche et gère la navigation des stats du monstre. Et affiche chaque stats grâce à une barre plus ou moins rempli selon le nombre de points. Affiche également le temps de jeu.

void devantMaison() : affiche et gère le choix de sortie entre aller au magasin (en ville), aller lancer le jeu des pommes et partir à l'aventure dans la forêt.

void aventure() : affiche et permet de lancer un combat

void combat() : affiche et gère le combat en laissant un choix de prendre une potions, lancer une attaque ou fuir.

void prendrePotion() : affiche et gère la prise de potion lors d'un combat. Soit une potion de vie, soit une potion d'attaque.

void recompenses() : affiche et gère les récompense accordée après un combat gagné, de manière aléatoire, gain d'argent et potion aléatoire.

void chercherPomme() : gère et affiche le jeu pour la récupération des pommes. On peut déplacer un panier à gauche ou a droite pour attraper des pommes qui tombent aléatoirement d'un arbre.

void pauseJeu() : permet de faire une pause en récolte de pomme.

void pommesRecuperees() : affiche et gère les récompense accordée a la fin de la récolte des pommes. Le nombre de pomme vendu est aléatoire et si elles ne sont pas gardées, le joueur les garde pour nourrir son monstre, chaque pomme confère 2nrt.

void magasin() : fonction pour rentrez dans le magasin.

void interieur() : fonction qui permet de choisir entre acheter de la nourriture et de la potion.

void achatFood() : fonction qui affiche et qui gère l'achat de nourriture

void achatPotions() : fonction qui affiche et qui gère l'achat de potion.

Void mourir() : fonction qui gère la mort du monstre en retirant tout ce que stocke la carte et en la redémarrant grâce à la bibliothèque Préférences.

void manger() ,void dormir(), void laver(), void soigner()

Conclusion :

TAMAMONSTRA est un projet complexe intégrant divers composants matériels et une série de fonctions logicielles permettant une expérience de jeu riche et interactive. La carte ESP32, combinée à un écran et des boutons poussoirs, donne vie à un Tamagotchi virtuel que l'on peut nourrir, soigner, et avec lequel on peut jouer à divers mini-jeux. Le boîtier modélisé avec Tinkercad assure une finition soignée et professionnelle.