École Polytechnique de l'Université de Tours
64, Avenue Jean
Portalis 37200
TOURS, FRANCE
(33)2-47-36-14-14
www.polytech.univ-tours.fr

Parcours des Écoles d'Ingénieurs Polytech

Année 2023 / 2024

Console low tech Chip-8

Étudiant(e)

Johan Ondet
Raphaël Texier
Mateo Aranhita Saucia
Johan.Ondet@etu.univ-tours.fr
Raphael.Texier@etu.univ-tours.fr
Mateo.Aranhitasaucia@etu.univ-tours.fr

Encadrant

Pierre Gaucher
pierre.gaucher@univ-tours.fr

Avertissement

Ce document a été rédigé par ***Johan Ondet, Raphaël Texier et Matéo Aranhita Saucia*** susnommés les auteurs. L'École Polytechnique de l'Université François Rabelais de Tours est représentée par ***Pierre Gaucher*** susnommé le tuteur académique.
Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.
Les auteurs reconnaissent assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non-respect des lois ou des droits d'auteur.
Les auteurs attestent que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.
Les auteurs attestent ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.
Les auteurs attestent que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.
Les auteurs reconnaissent qu'ils ne peuvent diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.
Les auteurs autorisent l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Table des matières

Introduction 4
1) L'hardware 5
1.1 La liste des composants 5
1.2 Le Raspberry pi pico 6
1.3 L'écran 7
1.4 Le Lipo Shim 7
1.5 Le clavier 9
2) L'assemblage 11
2.1 Le prototype 11
2.2 Le clavier custom 12
2.3 Les branchements 15
2.4 Les impression 3D 17
2.5 L'assemblage final 18
3) L'OS 20
3.1 La librairie pour les boutons et le pinout 20
3.2 L'interface principale 20
3.3 La gestion de la carte SD 21
3.4 Animation de lancement 21
3.5 Interface de sélection de fichier 23
3.6 Logiciel de musique 24
4) L'émulateur 25
4.1 Le langage CHIP-8 25
4.2 Un jeu en CHIP-8 25
4.3 Le fonctionnement de l'émulateur trouvé sur internet et ses modifications 26
4.4 Les améliorations de l'émulateur 27
5) Et ensuite ? 28
5.1 Vers un PCB custom 28
5.2 Le niveau de batterie 28
5.3 Des inserts 28
5.4 L'ajout d'un clavier 16 touches 29
Conclusion 30
Bibliographie 31

Introduction

De nos jours, les consoles de jeux font partie intégrante de notre quotidien, devenant de plus en plus complexes et performantes. Face à cette évolution, nous avons choisi de revenir aux sources en nous penchant sur une ‘console’ plus ancienne, à la fois pour mieux comprendre leur conception et pour redécouvrir des classiques du jeu vidéo. Ainsi, notre projet consiste à concevoir et programmer une console permettant de jouer à des jeux codés en CHIP-8, tels que Pong, Bricks, Tetris, etc.

Dans ce rapport, nous détaillerons chacune des étapes de notre démarche afin de permettre à tout passionné, qu'il soit novice ou confirmé en électronique et informatique, de construire sa propre console rétro CHIP-8. Nous avons tenté de rendre ce tutoriel aussi complet et accessible que possible.

Dans un premier temps, nous présenterons les composants nécessaires à la construction de la console, en détaillant leur fonctionnement et leur coût.

Par la suite, nous aborderons la partie hardware de la console, en expliquant en détail l'installation et la configuration des différents périphériques indispensables au bon fonctionnement de notre système.

Succinctement, nous examinerons l'assemblage de ces composants, en détaillant les étapes clés pour relier chaque élément de manière durable.
Nous nous attarderons également sur l'aspect logiciel de la console, en décrivant l'OS que nous avons créé ainsi que l'émulateur nécessaire pour faire fonctionner les jeux CHIP-8.

Enfin, nous évoquerons les diverses possibilités d'amélioration de la console, ouvrant la voie à des projets futurs et des développements ultérieurs.
Nous espérons ainsi fournir à la communauté des amateurs de jeux vidéo et d'électronique un guide complet et inspirant pour la création de leur propre console rétro CHIP-8.

Vous retrouverez ainsi en annexe l’ensemble des composants, programmes et librairies utilisées.

1. L’hardware

1.1 La liste des composants

Dans le cadre de la réalisation de notre console, il nous fallait choisir les différents composants. Le choix est assez libre, néanmoins il faut respecter nos critères. Ici nous cherchons à réaliser une console qui est avant tout économique et portable, tout en ayant un maximum de fonctionnalité et une grande compatibilité. Nous allons donc vous expliquer la configuration que nous avons choisie, vous trouverez tous les composants et leur prix dans la bibliographie. Il s’agit ici d’une liste non exhaustive, les spécifications des composants seront expliquées plus en détails par la suite.

Commençons par le plus important : le microcontrôleur. Dans notre cas, il nous faut quelque chose d’assez puissant pour supporter ce qu’on va mettre dessus et assurer le bon fonctionnement des jeux à un prix moindre. Le candidat idéal pour cela était donc le Raspberry PI Pico.

Ensuite, il convient de disposer d’un écran. Il s'agit du composant le plus cher de notre console. Le plus important pour l’achat de ce composant est la résolution, le format et le mode de communication. Nous avons pour notre part choisi un écran de la marque joy it.

Il est également nécessaire d’avoir un clavier pour jouer. Dans un premier temps, nous avons choisi un clavier 16 touches de la marque joy it également car il s’agît du nombre de touches maximales nécessaire pour jouer à des jeux chip-8. Néanmoins, la plupart des jeux n’utilise que 6 touches. C’est pourquoi, nous avons plus tard créé notre propre clavier.

Afin de faire un système répliquant le système des cartouches de jeux, nous avons fait le choix de prendre un lecteur de carte SD (il est aussi possible de faire ça avec la mémoire du microcontrôleur mais l’échange de fichiers devra se faire par le logiciel Thonny).

Si l’on veut que notre console soit portative, alors, il est nécessaire de prévoir une batterie ainsi qu’un module de charge. Il existe des batteries de toutes capacités et de toutes tailles, les options sont vastes. Nous avons finalement choisi une batterie 3.7V, (bien vérifier la compatibilité avec votre module de charge). Pour ce qui en est du

module de recharge, le choix est également très libre, il faut seulement surveiller la compatibilité avec le microcontrôleur. Ici notre choix s'est porté sur un module pico Lipo Shim car il est parfaitement adapté à notre microcontrôleur.

Enfin, pour produire du son nous avons choisi d'utiliser un simple buzzer. Au départ, nous pensions utiliser un module haut-parleur avec amplificateur mais finalement notre choix s'est tourné vers un buzzer pour plusieurs raisons : celui-ci prend très peu de place et à un faible coût. De plus, le langage CHIP-8 ne permet pas de produire des sons très élaborés, un buzzer convient donc parfaitement.

1.2 Le Raspberry pi pico

Afin de commencer le projet, il nous fallait d'abord une carte pour contrôler notre système, un cerveau permettant le contrôle des autres composants. Pour cela, nous avons donc opté pour une Raspberry pi pico qui offre un large espace de stockage et une puissance très correcte à un prix minimal.

La Raspberry pi pico est un microcontrôleur programmable en micro-python ou bien en C++. Cette carte est composée d'un connecteur micro-USB, d'un processeur ARM Cortex-M0+ Dual Core à 133 MHz qui est amplement suffisant pour les programmes que nous allons mettre dessus. Elles sont également composées de 256 KB RAM, 2MB en mémoire flash en SPI, 23 entrées/sorties appelé GPIO (general purpose input/output).

Ensuite, pour utiliser cette carte il convient de la configurer à l'aide d'un ordinateur. Pour cela il nous faut nous munir d'un câble micro-USB que nous branchons à la carte et à un ordinateur. Puis, une fois connectée à l'ordinateur, il est nécessaire de choisir le langage dans lequel nous souhaitons programmer, ici nous avons choisi le micro-python. Afin d'installer celui-ci, nous avons donc installé un firmware que nous avons trouvé sur le site de la carte directement répertorié dans la bibliographie. Ensuite nous avons configuré notre environnement de développement pour du micro-python, notre choix s'est porté sur Thonny. Une fois cela effectué, il faut mettre le firmware téléchargé précédemment sur la Raspberry, il suffit de maintenir pendant 6 secondes le boutons boot de la carte tout en la connectant à l'ordinateur. Suite à cette manipulation, une fenêtre apparaît sur l'ordinateur correspondant au stockage de la carte : il faut donc y glisser le firmware précédemment installé. Une fois toutes ces manipulations effectuées, la carte est prête à être utilisée.

ATTENTION : Il faut bien prendre un câble qui permet le transfert de données et pas simplement que l'alimentation !

1.3 L’écran

L’écran qui nous a été fourni était le « 2.42" OLED DISPLAY » de chez Joy-it, voici ses spécifications :

- 2.42 pouces
- Interface I2C, SPI, sérielle 6800 et 8080
- Oled monochrome (jaune ou noir)
- 90mA de consommation
- 3 – 5V pour l’alimentation

La technologie OLED de l’écran permet de réduire la consommation lorsque les pixels ne sont pas allumés, cela permet aussi d’avoir une meilleure vision au soleil ainsi que des angles de vision plus élevés qu’avec un écran LCD standard.
Nous utiliserons l’écran avec son interface SPI, ce qui évite d’avoir à souder des pads.

1.4 Le Lipo Shim

Pour gérer l’alimentation de notre Pi Pico en mode portable nous avons choisi le module de chez Pimoroni qui est parfaitement adapté au Raspberry. Celui-ci viendra se souder directement à l’arrière du microcontrôleur. Voici ses spécifications :

- Courant de charge fixé de 220mA
- LED d’indication de charge
- ICs de protection
- Bouton marche-arrêt

Mis à part le schéma électrique, il n'y a aucune information ou datasheet sur ce module. Nous allons tout de même tenter d'expliquer brièvement son fonctionnement, que nous avons compris par sa rétro-ingénierie.

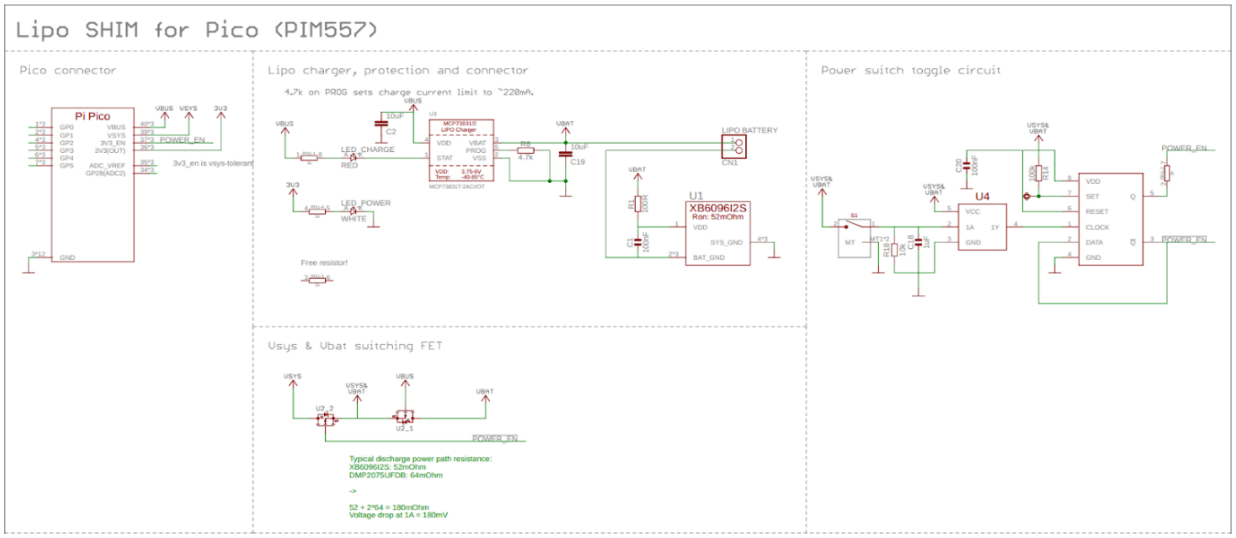


Figure 1 : Schéma électrique Lipo Shim

La figure 1 est issue du site de Pimoroni, elle représente le schéma électrique du module pico Shim. On y repère plusieurs circuits intégrés, le plus important étant le MCP73831/2 qui correspond au chargeur de batterie. La résistance R8 de 4K7 permet de fixer le courant de charge à 220mA. La “LED CHARGE” est celle qui s'allume lorsque l'on branche un câble USB.

Le deuxième CI est le XB6096I2S, celui-ci s'occupe de protéger la batterie. Il protège celle-ci contre l'inversion de polarité, la surcharge, les surintensités et les courts-circuits. Ce CI va permettre d'augmenter la durée de vie de notre batterie LiPo.

Enfin, nous pouvons repérer un D-Latch dans la partie "Power switch toggle circuit", il va nous permettre d'allumer et d'éteindre le Raspberry avec un seul bouton poussoir (S1).

On peut remarquer que le module utilise 3 broches d'alimentations du Pico, à savoir : VBUS, VSYS et 3V3. Le 3V3 est seulement utilisé par la led blanche et sert à indiquer que le Raspberry est allumé. Pour les deux autres, c'est un peu plus compliqué, pour comprendre leur utilisation nous avons inclus une photo du schéma électrique du Pico, plus précisément, la partie alimentation de celui-ci, voir figure 2.

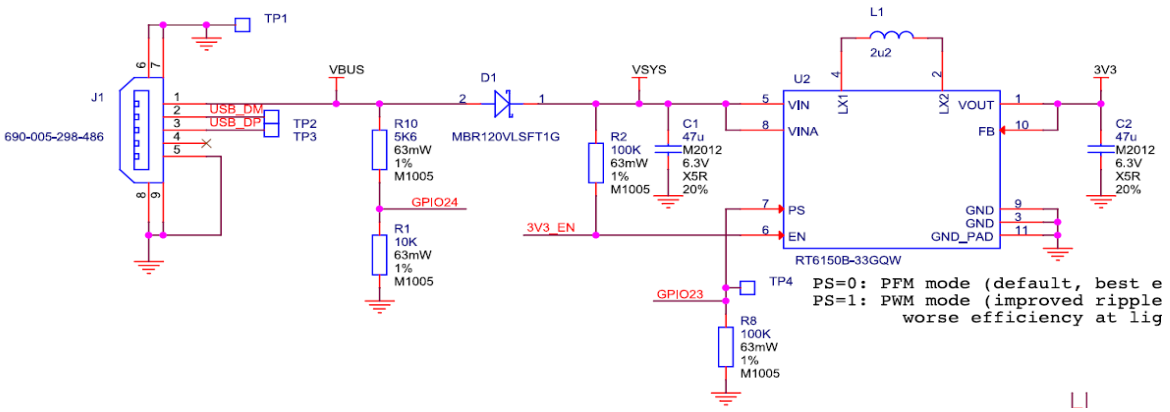


Figure 2 : Extrait d’un schéma électrique RP pico

On peut retrouver la broche VBUS qui est l'alimentation 5V de la prise USB et la broche VSYS qui se trouve après la diode schottky D1. Cette configuration est très importante car elle va nous permettre d'utiliser la prise USB du Pico pour charger notre batterie et aussi d'alimenter le Raspberry avec la batterie.

En conclusion, le LiPo shim est exclusivement réservé aux microcontrôleurs présentant cette configuration, il ne fonctionnera donc pas avec les modules du type RP2040 zero comme montré figure 3.

Figure 3 : RP 2040 pinout

1.5 Le clavier

Afin de rendre la chose fonctionnelle, sans pour autant se retrouver avec trop de fils, notre choix s'est orienté vers un clavier 16 touches avec 8 broches (au lieu de 16). Cela complique un petit peu le code mais nous permet de gagner des pins et de s'y retrouver un peu plus.

Dans un premier temps, il faut donc brancher correctement le clavier au microcontrôleur, en faisant bien attention à ne pas brancher de câbles sur les pins "GND". Une fois ce branchement effectué, sur l'IDE Thonny, nous avons donc défini nos pins à l'aide de deux listes : 'broches_colonnes' et 'broches_lignes' car le clavier n'a que 8 broches. Il faut donc analyser ligne par ligne et colonne par colonne (Nous avons choisi de travailler avec 4 lignes et 4 colonnes).

Suite à cela, nous avons conçu une matrice définissant chaque touche du clavier comme sur le schéma ci-dessous. Les lignes sont réglées en sortie, elles ont donc un niveau logique haut (1) et les colonnes sont réglées en entrée, ce qui signifie un niveaux logique bas (0). Puis dans notre programme nous avons mis les valeurs de toutes les lignes à 0. Et nous avons mesuré chaque colonne en dressant une liste de celles-ci nommée résultat.

Enfin, on regarde si la valeur minimum de résultat est de 0. Si tel est le cas, alors une touche est enfoncée, il faut donc regarder la colonne et la ligne afin de récupérer la touche dans la matrice des boutons. On remet la valeur logique de la ligne à 1 dans le cas où le bouton est maintenu enfoncé, puis, on retourne simplement le bouton qui a été pressé.

Ne pas oublier de remettre les valeurs logiques des lignes à 1 par la suite si aucune touche n'a été pressée ! Une fois ce programme terminé, pour pouvoir jouer, il faut exécuter la fonction de lecture de touches en continu.

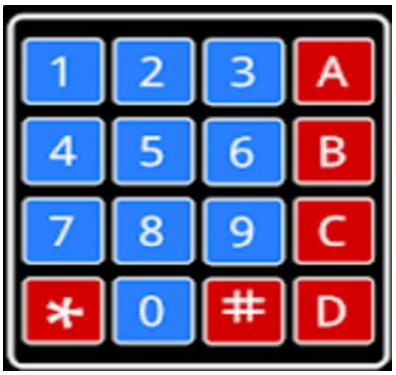


Figure 4 : Schéma d'un clavier 16 touches

2. L’assemblage

Dans l’optique de gagner un maximum de place, nous avons cherché des solutions pour réduire l’encombrement des barrettes de connexion du Pico, du lecteur SD et du futur clavier custom. Nous avons tout d’abord pensé à des barrettes coudées à 90° qui nous permettaient déjà de gagner beaucoup d’épaisseur. Néanmoins nous avons fait une observation intéressante : en enlevant la barre noire des barrettes droites et en pliant chaque pin un par un nous pouvions réduire encore l’épaisseur. Figure 5 nous avons : à droite les barrettes du commerce et à gauche notre méthode.

Figure 5 : Barrettes normales contre barrettes pliées

C’est donc cette solution que nous avons retenue pour le lecteur SD, le pico et le clavier.

2.1 Le prototype

Une fois toutes les bases faites, l’objectif est de mettre en relation les composants, c'est-à-dire les faire communiquer entre eux.

Dans un premier temps, nous avons utilisé un « shield » afin de faire nos essais. Une fois nos essais effectués on a pu repérer plusieurs problèmes. La connexion avec l'écran nous a posé quelques difficultés, car il fallait modifier deux fichiers différents (l’émulateur et le « main »). Une fois cela effectué, un autre bug était présent nous forçant à débrancher et rebrancher le câble micro-USB pour pouvoir lancer un jeu.

En jouant le problème rencontré était sur la configuration de touches. En effet, chaque jeu utilise une disposition de touches différentes, ce qui rend la matrice réalisée sur notre programme obsolète. Pour résoudre ce problème, la solution est donc d’analyser jeu par jeu les touches et de faire une matrice de touche propre à chacun.

Le dernier problème rencontré est purement ergonomique. Le clavier 16 touches étant très compact, il est compliqué d'être précis et d'appuyer sur une seule touche.

Là, nous avons donc eu l'idée "d'exploser le clavier" en diminuant le nombre de touches et en les espaçant entre elles. Ci-dessous se trouve une photo de notre premier prototype.

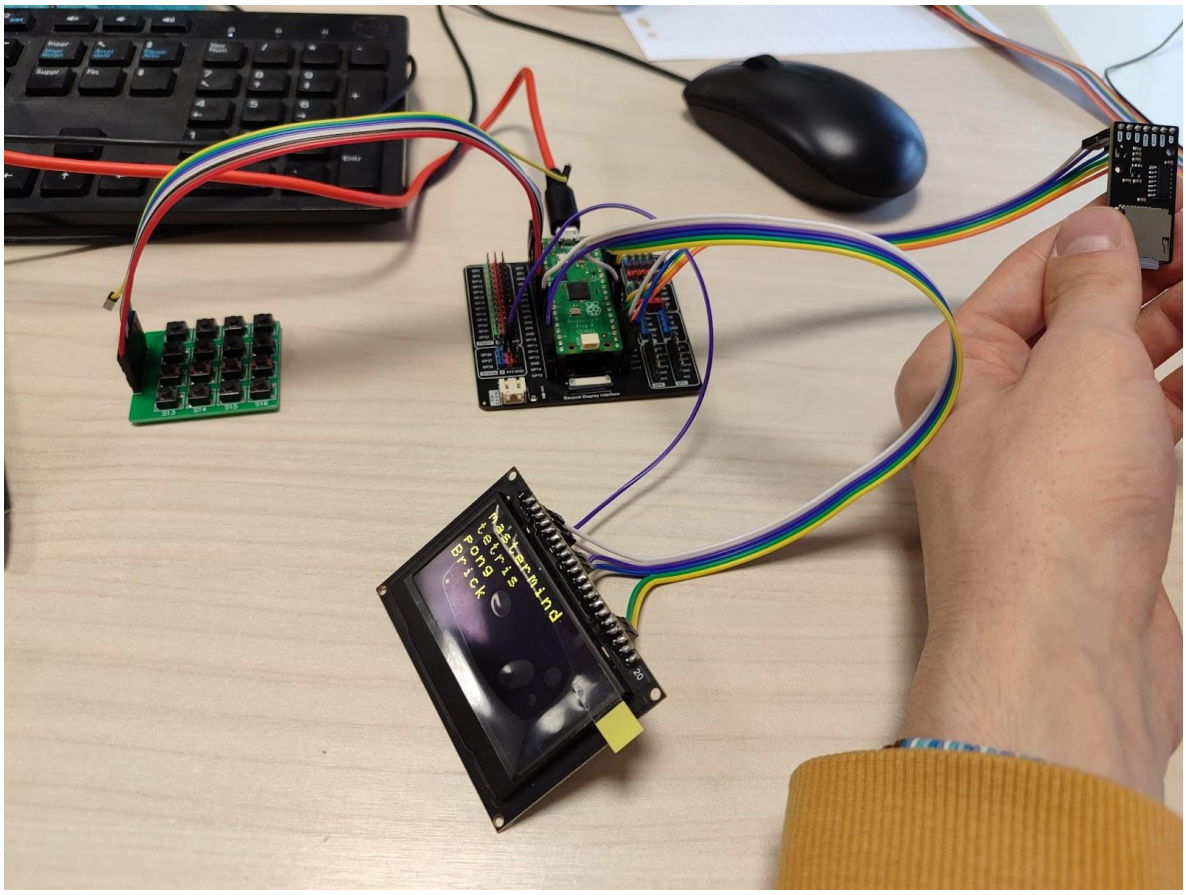


Figure 6 : Premier montage

2.2 Le clavier custom

Nous avons donc décidé de garder seulement 8 touches pour notre clavier, c'est ce qui nous a semblé optimal après avoir testé plusieurs jeux. Nous avons une croix directionnelle, un bouton A et B, un bouton start et un bouton home que nous utiliserons pour revenir rapidement au menu principal.

Il va falloir 8 boutons poussoirs de 6x6x7mm, une plaque de prototypage de 40x60mm qui correspond exactement à la taille du PCB de l'écran, une barrette de connexion de 10 pins et un buzzer.

Pour commencer, voici un extrait du schéma Kicad (figure 7) portant sur la connexion du bouton. On peut repérer une connexion commune au GND car nous utiliserons une configuration dite Pull-UP.

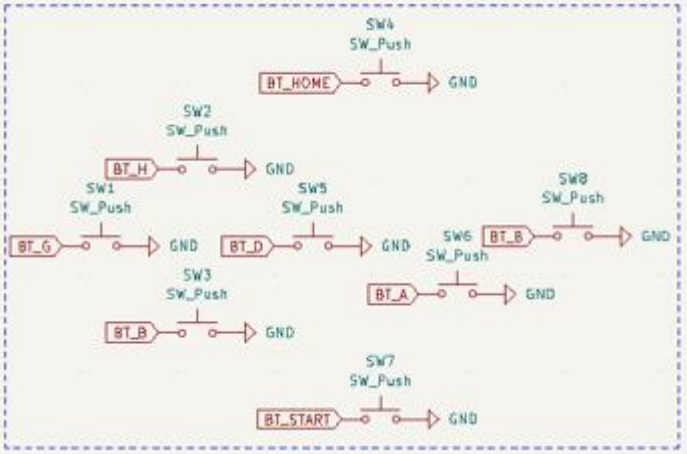


Figure 7 : Extrait du schéma Kicad

Tout d'abord, il faut choisir la meilleure configuration pour les boutons. Puis les insérer dans les trous du PCB. Voici notre configuration (figure 8).

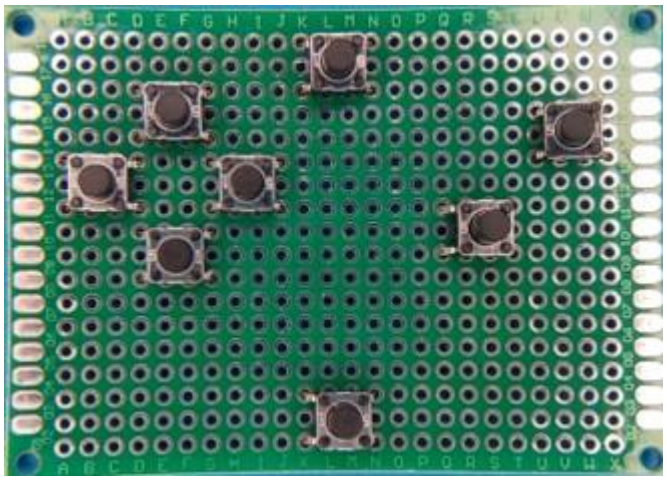


Figure 8 : Boutons placés

Puis, reliez tous les boutons ensemble au GND. Nous avons utilisé du fil de cuivre émaillé de 0.2mm (voir figure 9).

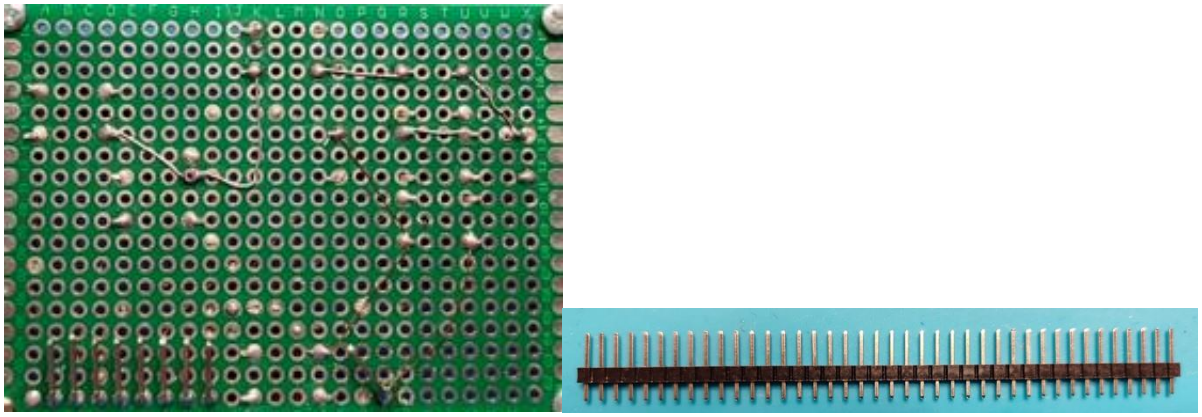


Figure 9 : Arrière du PCB et barrette

Ensuite, sur la face avant, il faut relier l'autre extrémité de chaque bouton à une broche du connecteur préalablement soudée, toujours avec du fil de cuivre émaillé. Le connecteur barrette est sur la figure 9 à droite.

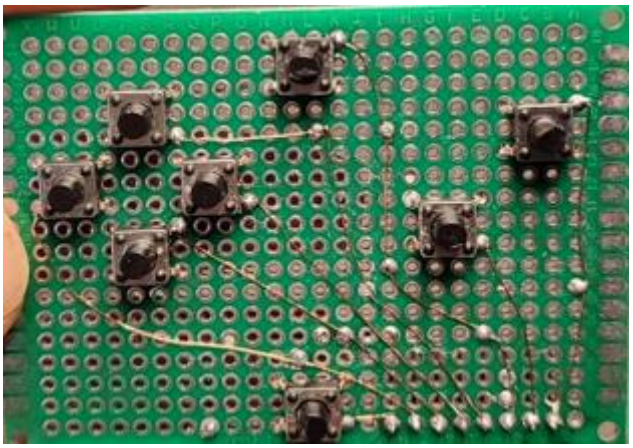


Figure 10 : Avant du PCB après routage

Voici le résultat que vous devez obtenir à quelques détails près (figure 10).

Enfin, après avoir plié ces deux pins à 90°, soudez le buzzer (figure 11).

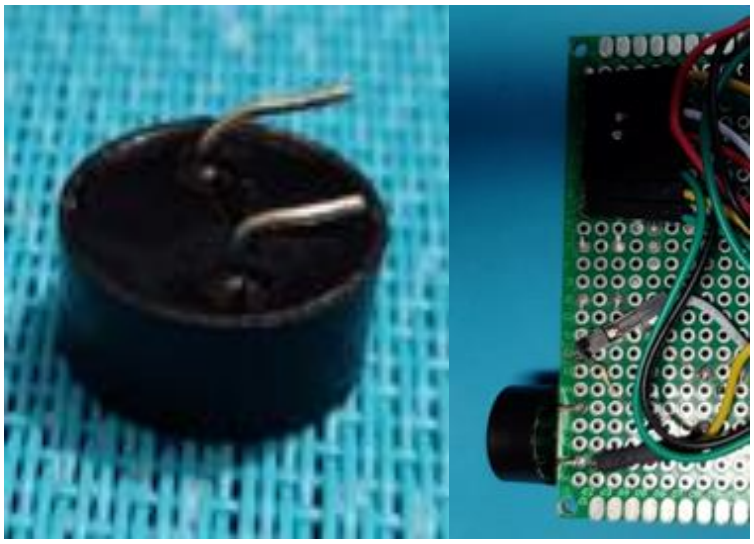


Figure 11 : Buzzer avec pin plié et PCB avec le buzzer

Reliez le pin le plus court au GND et l'autre pin à une broche pliée à 90°, comme pour les boutons.
Voici le résultat final que vous devez obtenir (figure 11, image de droite).

2.3 Les branchements

Tout d'abord, voici un schéma fait sur Kicad (figure 12) sur lequel on peut retrouver tous les composants et leurs pins correspondants. Le lecteur carte SD utilise le bus SPI n°1 et l'écran le n°0. Le reste des composants est branché au plus proche.

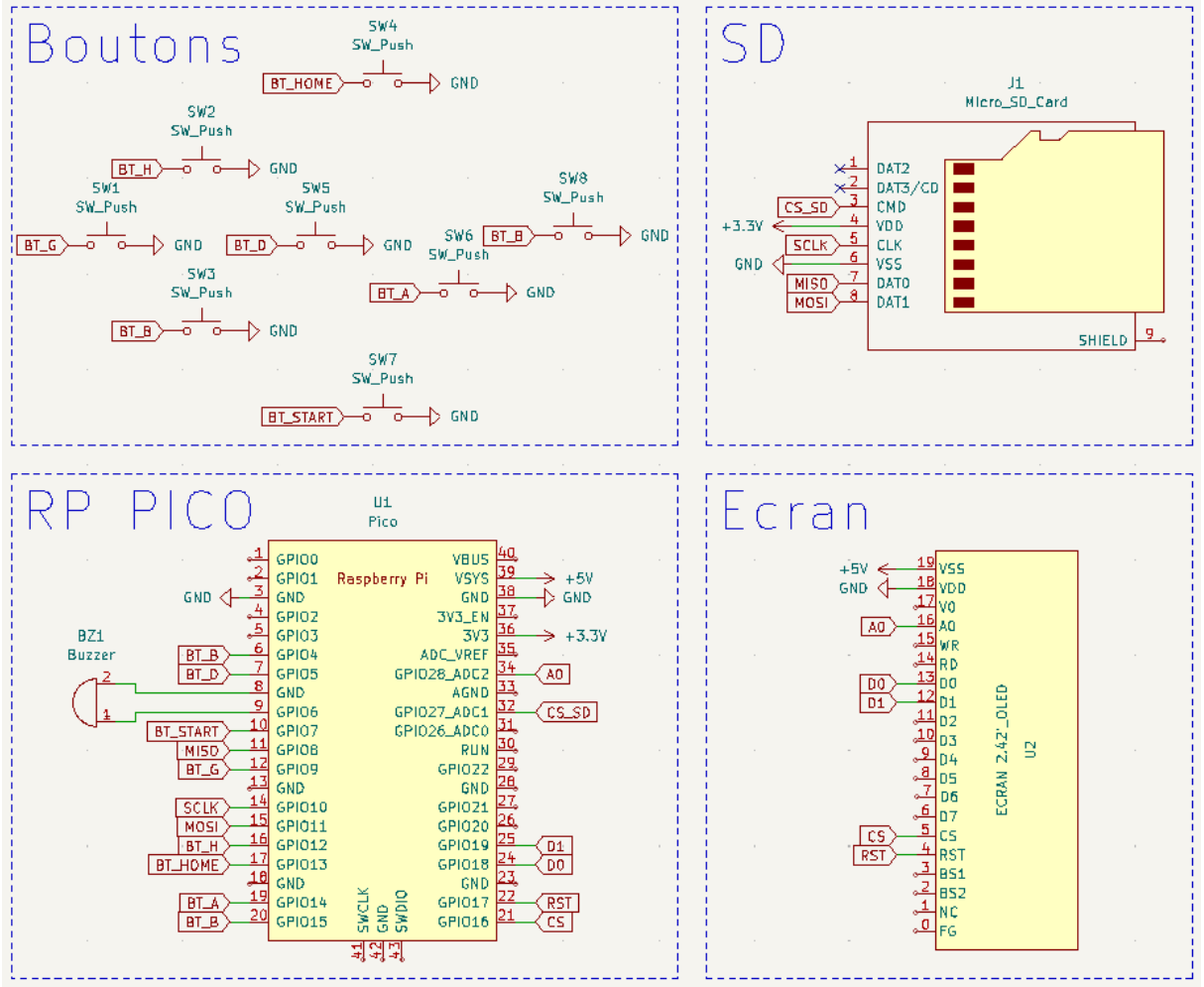


Figure 12 : Schéma électrique complet Kicad

Il faut commencer par souder les barrettes de connexion au Pico, de la même manière que pour le clavier, voir figure 13.

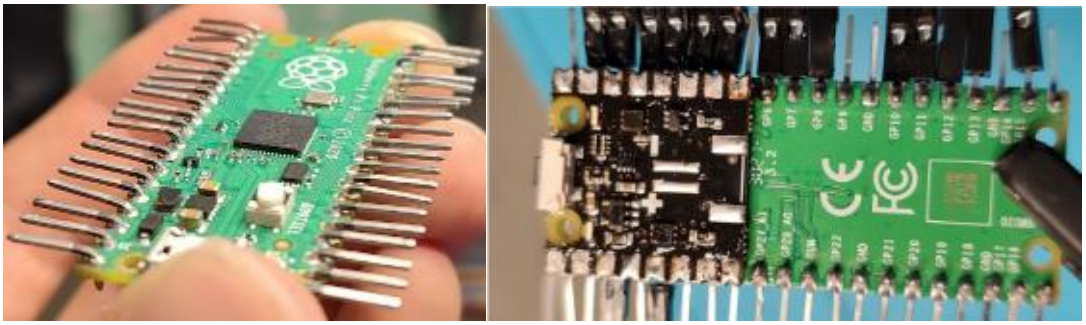


Figure 13 : Pico pin plié avec module de charge

Ensuite, soudez le module chargeur de batterie à l'arrière du Raspberry, contrairement à la figure 13, nous vous conseillons de débrancher tous les fils. Attention à ne pas se tromper de sens ! Le bouton doit se trouver du même côté que la prise USB. Ensuite, dessoudez la prise JST blanche présente pour la remplacer par le connecteur plus fin fourni avec la batterie. Il faut évidemment brancher le fil rouge sur le + et le fil noir sur le - pour éviter de court-circuiter la batterie. Puis recouvrez toute la surface avec du kapton pour éviter d'éventuels courts-circuits. Comme montré sur la figure 14.

Figure 14 : Module de charge avec connecteur rouge

Il ne reste plus qu'à relier tous les composants entre eux, pour cette étape nous avons choisi d'utiliser des fils de prototypage sur les conseils de notre tuteur pour augmenter la réparabilité en cas de panne. Nous aurions pu tout souder mais cela aurait été embêtant si un composant avait rendu l'âme. Pour ce qui est des fils de connexion, trois tailles différentes de fils femelle-femelle seront nécessaires :

- 17 fils de +/-5 cm
- 5 fils de +/-10 cm
- 2 fils de +/-15 cm

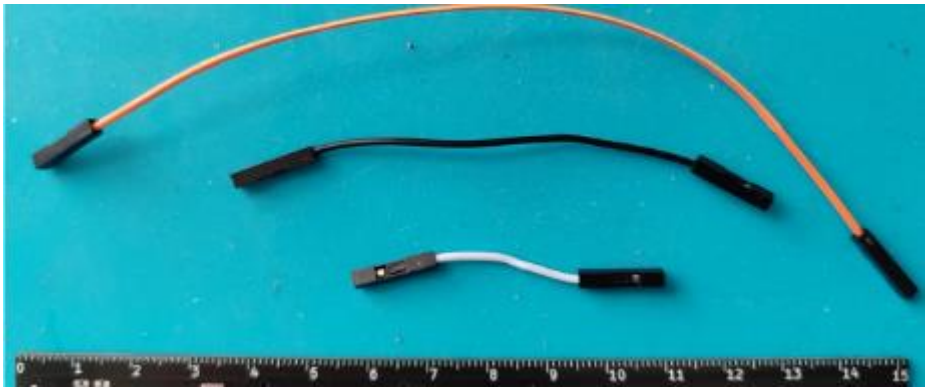


Figure 15 : Les différentes taille de fil

A partir de ce moment, deux choix s'offrent à vous, soit vous décidez de suivre le tutoriel, soit vous commencez par fixer les composants au boîtier puis vous les reliez. Nous avons choisi la deuxième option car nous avons dû modéliser le boîtier au fur et à mesure, c'était donc plus simple pour nous de ne pas tout débrancher entre chaque assemblage. Donc libre à vous de passer directement à la section suivante ou de rester ici !

Après avoir suivi le schéma de branchement fait sur Kicad voici le résultat que vous devez obtenir (figure 16).

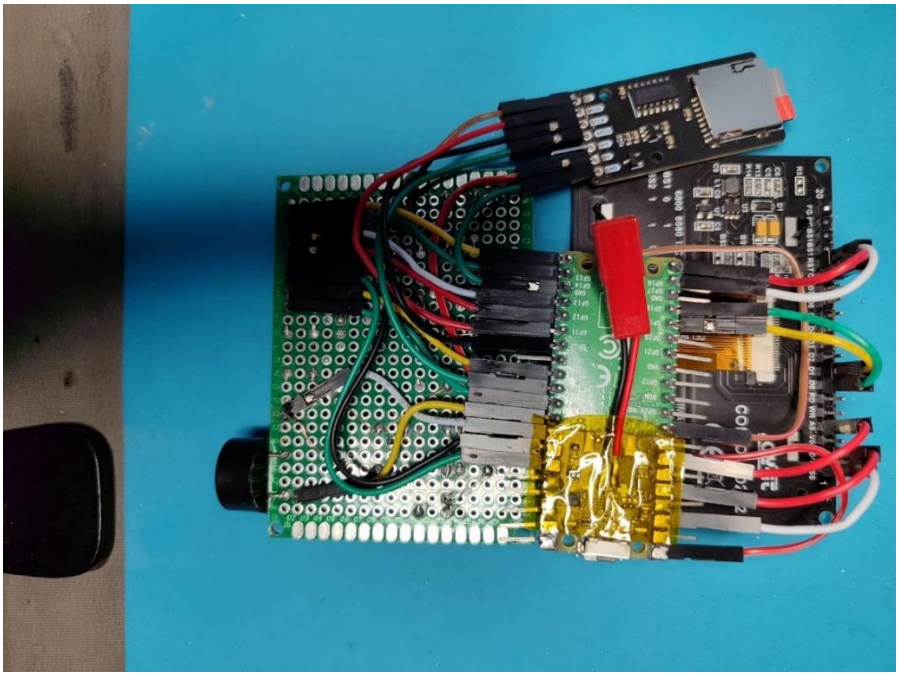


Figure 16 : Tous les composants connectés

Vous n'oublierez pas de couper les 4 pins les plus à gauche de l'écran, comme sur les figures 16 et 18, cela sera utile plus tard pour laisser passer le lecteur SD.

2.4 Les impressions 3D

Nous avons souhaité modéliser notre propre boîtier afin que celui-ci soit parfaitement adapté à tous les composants et puisse être le plus compact possible. Le boîtier se compose de plusieurs parties :

- La partie haute sur laquelle se fixe l'écran, le support pour le pico et le clavier
- 5 boutons, dont une croix directionnelle
- Un support pour fixer le pico et le lecteur carte SD à la partie haute
- Une partie basse qui permet de refermer le boîtier
- Un cache qui vient se clipser à l'arrière de la console

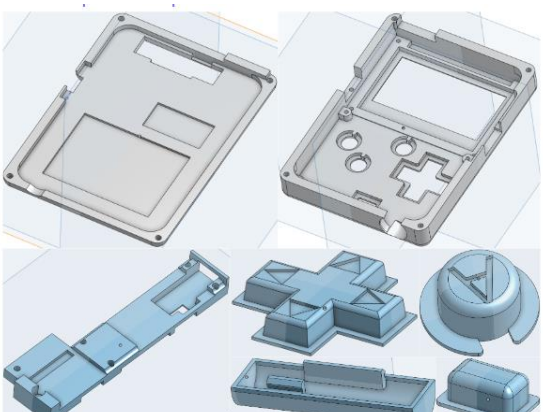


Figure 17 : Pièces fait sur Onshapes

Il faut imprimer toutes les pièces avec seulement un support d'adhésion sauf le support Pico qui demandera des supports standards en plus.

2.5 L’assemblage final

La fixation des composants au boîtier se fait avec des vis de 4 mm de long et 2 mm de large, il en faut 13 en tout, mais 11 suffisent car le pico peut se fixer avec seulement 2 vis placées en diagonale. Enfin, la partie basse se fixe à la partie haute à l’aide de vis de 3mm de large et de +- 5 mm de long. Voici comment procéder.
Vous devez commencer par fixer l’écran au boîtier à l’aide de 4 vis de 2*4 mm et disposer les boutons aux différents emplacements correspondants. (figure 18 à gauche).

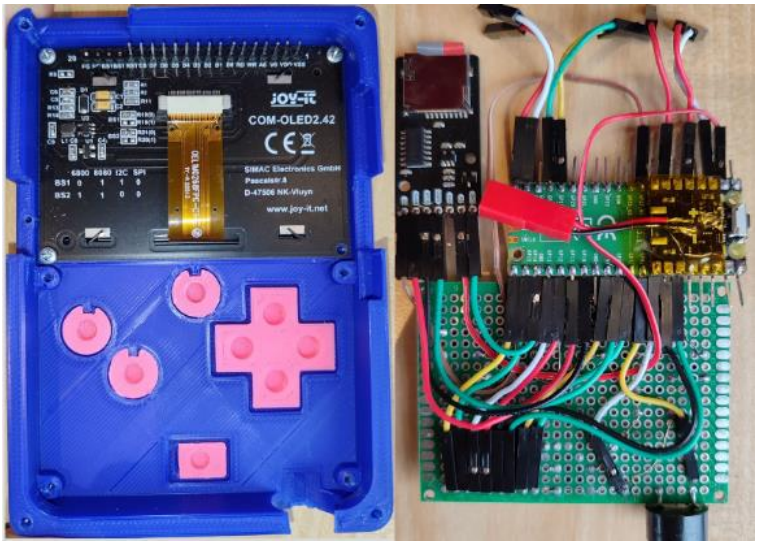


Figure 18 : Boitier avec écran et boutons / Tous les composants sans l'écran

Débranchez l’écran. L’ensemble clavier + pico + lecteur SD doit ressembler à la figure 18 (à droite).

Vous devez ensuite insérer cet ensemble dans le boîtier. Puis, fixez le clavier à l’aide des 4 mêmes vis que pour l’écran

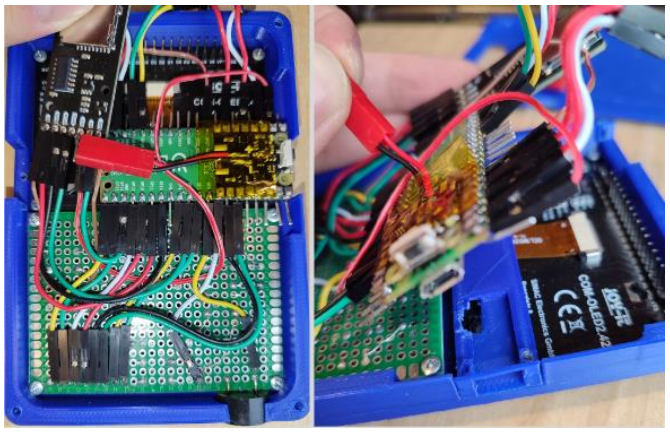


Figure 19 : Boitier avec tous les composants / on soulève le pico

Après avoir soulevé le Pico, glissez son support et fixez-le au boîtier voir figure 19. De même, fixez le lecteur SD au support comme montré sur la figure 20.

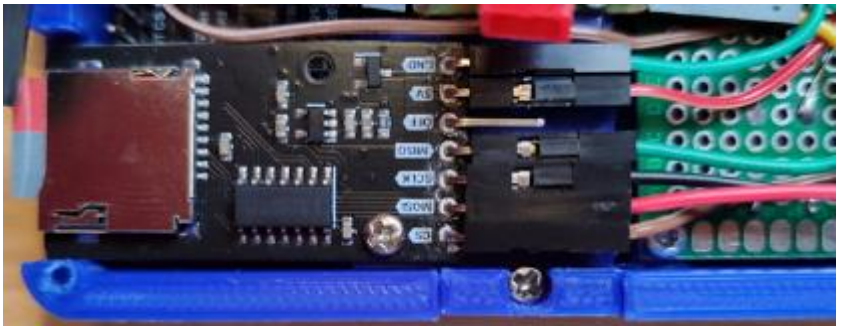


Figure 20 : Lecteur de la carte SD et vis de fixation du support

Vous fixerez ensuite le Raspberry au support avec deux vis, cela suffit. (figure 21)

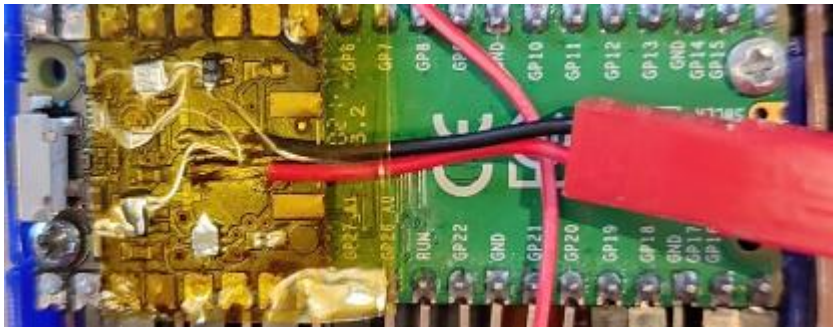


Figure 21 : Fixation du Raspberry Pi Pico

Enfin, vous terminerez l'assemblage en branchant la batterie et l'écran, puis en refermant le boîtier à l'aide de 4 vis de 3*5 mm, voir figure 22

Figure 22 : Batterie connecté / Console avec le couvercle

Vous n'oublierez pas de clipser le cache à l'arrière du boîtier !

3 L'OS

3.1 La librairie pour les boutons et le pinout

Pour unifier l'utilisation des boutons dans tous les programmes nous avons créé une librairie "bouton" contenant une fonction "lecture_touche()" qui renvoie une seule touche pressée.

D'autre part, pour éviter d'avoir à modifier les pins de chaque composant dans tous les fichiers en cas de changement, nous avons inclus un fichier dit "pinout" dans lequel chaque broche de chaque composant est répertoriée avec son numéro de pin correspondant.

3.2 L'interface principale

Nous avons essayé de soigner un maximum l'interface utilisateur pour la rendre la plus agréable et intuitive possible. L'interface principale est donc constituée de 4 icônes de 42x42 pixels (figure 23) que nous avons dessinés sur Photopea puis convertis dans un format bitmap à l'aide d'un script python présent dans la librairie de l'écran.

Figure 23 : Icones utilisées pour l'interface

Chaque icône correspond à une application, l'icône C8 correspond à l'émulation des jeux CHIP-8, l'icône en forme d'engrenage aux paramètres, l'icône en forme de note de musique correspond au logiciel de musique et enfin l'icône PY correspond à l'exécution des fichiers python.

Figure 24 : Image de notre interface

Toujours dans le but de rendre l'utilisation ergonomique, nous avons programmé des animations de transitions avec 'effet 3D'. Nous avons obtenu ce résultat en modifiant la taille de chaque rectangle et sa position en fonction du temps avec une boucle for. A cela viennent s'ajouter deux triangles animés en forme de flèches qui nous invitent intuitivement à appuyer sur la croix directionnelle.

3.3 La gestion de la carte SD

La carte sd a été l'une des choses les plus simples à mettre en place. Nous avons utilisé une librairie trouvée sur le Github de micro-python (lien en bibliographie). L'exécution des jeux et la lecture des fichiers se font grâce à d'autres librairies. On initialise la sd puis avec la librairie « uos » on crée le répertoire "sd/" afin de lire la liste des fichiers dans un dossier lors de l'affichage du menu de sélection de fichier.

Sur la carte SD, il y a un fichier « settings.cfg » permettant de stocker les paramètres de la console. Ensuite, il y a 4 dossiers :

- « Games » contenant les jeux et leurs configurations de touches
- « Music » contenant les musiques créées avec le logiciel
- « Python » contenant les fichiers python
- « Saves » contenant les sauvegardes de chaque jeu

Ces différents fichiers peuvent également être stockés directement sur la mémoire du Pico dans le cas d'une configuration sans lecteur de carte micro SD.

3.4 Animation de lancement

Afin d'avoir une meilleure esthétique, nous avons décidé d'ajouter une petite animation lors du démarrage de la console. Nous avons ainsi pensé à plusieurs styles d'animations. La première idée était un point au centre qui s'étend afin de remplir l'écran. Nous avons vite abandonné celle-ci. L'idée consistait à faire apparaître 4 carrés qui s'étendaient vers les bords de l'écran chacun dans une direction différente jusqu'à combler l'entièreté de l'écran. Le problème était que les carrés se créent dans un seul sens et ainsi c'était donc impossible.

Ainsi, nous avons décidé de seulement afficher un écran allumé complet et patienter légèrement avant d'afficher le logo de la console. Nous avons choisi une nouvelle police pour le logo du nom de la console et une autre police pour la "marque" Polytech Inc.

Nous avons également ajouté la même animation d'apparition du texte que celle présente sur l'interface de sélection de fichier (dont nous parlerons un peu plus tard). Ensuite, nous avons décidé de faire un menu principal qui découle de l'écran de démarrage. Ainsi, l'écran principal est juste l'écran de démarrage avec des couleurs inversées et sans le logo Polytech Inc. Après avoir attendu légèrement encore (moins d'une seconde) un texte "PRESS START" s'affiche avec une nouvelle police et encore une fois l'animation de l'interface de sélection de fichier. Évidemment, il suffit d'appuyer sur le bouton Start afin d'accéder à l'interface principale.

Voici notre principale source d’inspiration pour cette animation de lancement :

Figure 25 : Ecran de démarrage de la PlayStation 1 (1995)

3.5 Interface de sélection de fichier

Pour l'interface de sélection de fichier, nous avons utilisé la librairie uos. Nous avons tout d'abord créé trois fonctions afin d'alléger le code. Une fonction "afficher" qui sert à l'affichage correct lors d'un changement de page. Une fonction "launch" qui reprend la commande pour lancer les jeux et ajoute un petit temps de chargement avec un écran complètement allumé pour rajouter un peu d'esthétique et éviter que l'utilisateur soit directement lancé dans le jeu sans s'y être préparé.

Enfin, la fonction "changement_ligne" va permettre de mettre en surbrillance le jeu actuellement en sélectionné (afin de savoir sur lequel nous sommes positionnés). Cette fonction permet plus particulièrement le changement de ligne lorsque nous changeons notre sélection. Une fois toutes ces fonctions créées, nous allons commencer à scanner la carte sd afin de connaître le nom de chaque fichier présent sur celle-ci. Tout d'abord, nous avons créé une liste appelée "menu" qui contient à chaque indice le nom d'un unique jeu.

Nous avons également initialisé une variable intermédiaire appelée "fichiers_jeux" qui sera également un tableau contenant à chaque indice le nom d'un unique jeu (nous verrons un peu plus tard pourquoi). En utilisant uos, nous scannons notre dossier (le répertoire sd/games) et ajoutons uniquement les fichiers se terminant par "8" afin de vérifier si leur extension est bien « ch8 » pour chip-8. Nous avons décidé de vérifier uniquement le dernier caractère car quasi aucune extension se termine par "8" mais il est également possible d'élargir cette vérification à ".ch8" afin d'éviter tout conflit. Nous ajoutons alors à la liste "fichiers_jeux" le jeu.

Également si le nom du fichier dépasse 20 caractères, on l'ajoute à la liste "menu" ce fichier avec "..." à la place des 3 derniers caractères. La liste "fichiers_jeux" sert à avoir le nom complet du fichier car la fonction qui permet de lancer le jeu demande le nom complet du fichier. La liste "menu" sert uniquement à un affichage esthétique. Nous allons créer une nouvelle liste appelée "menu_sans_emplacements_vide" qui contient les mêmes éléments que "menu". Elle nous servira également plus tard pour le système de changement de page.

Nous allons ajouter un grand nombre d'emplacements vides à liste "menu" également pour l'esthétique des pages : si la page ne contient pas le même nombre que la précédente, l'alignement n'est pas correct. Une fois tout cela créé, nous allons afficher les 6 premiers éléments de "menu" avec l'animation du menu principal. S'ensuit un nombre de conditions qui dépendent chacune de la touche pressée: si c'est la touche A, lance le jeu; si c'est droite, efface l'écran et affiche instantanément les éléments de "menu" 7 indices plus loin (ex: 0 à 6 devient 7 à 13); si c'est gauche, même principe mais 7 indices avant (vérifie également que ne soyons pas sur la première page); si c'est haut, inverse le fond et la couleur de la police de sélection actuelle (jaune->noir et noir->jaune) et fait l'inverse pour l'élément du dessus (si nous sommes tout en haut de la liste, va directement au dernier élément et fait le même procédé) et enfin si c'est bas, même procédé que pour la touche haut mais vers le bas (si l'élément actuelle est le plus bas, va directement en haut de la liste).

3.6 Logiciel de musique

Nous avons terminé le projet en avance. Après avoir installé le buzzer dans la console et avoir testé quelques jeux, une idée nous est venue. Pourquoi ne pas créer un petit logiciel dans le style du Song Maker de Chrome Music Lab mais avec un style sonore plus rétro ? Et c'est ce que nous avons fait. L'interface principale de notre logiciel se présente comme un grand tableau de 60 divisions temporelles par 32 notes, chaque case mesure 2x2 pixels afin que cela ne soit pas trop petit. A gauche nous avons essayé de reproduire les touches d'un clavier pour rendre plus facile l'utilisation du logiciel aux novices. Voir figure 26.

Figure 26 : Extrait du logiciel de musique

Pour se déplacer sur la grille il suffit d'utiliser la croix directionnelle puis de cliquer sur A pour poser une note (ou effacer celle existante). Après avoir posé toutes les notes, il faudra cliquer sur le bouton start pour lancer la lecture de la musique. Comme sur un piano, à chaque note correspond une fréquence à laquelle va vibrer le buzzer pour produire la note désirée (comme le montre la figure 27). Une grande barre verticale vient nous indiquer la note en cours.

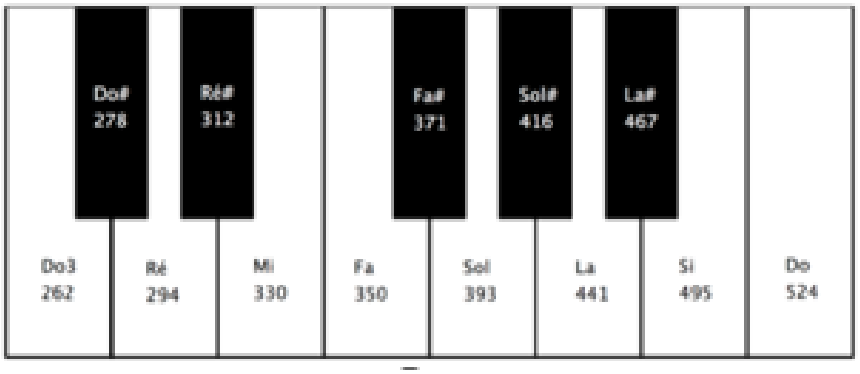


Figure 27 : Clavier avec les fréquences associées à chaque note

Si la vitesse de lecture est trop élevée, trop basse ou que vous voulez sauvegarder votre musique, il faudra cliquer sur B pour ouvrir le menu. Après avoir cliqué sur "sauvegarder", 10 emplacements s'offrent à vous. Le son est ensuite sauvegardé dans un fichier nommé son_N avec N correspondant à l'emplacement.

4 L’émulateur

4.1 Le langage CHIP-8

Ici, le langage choisi est le « Chip-8 », nous avons privilégié l’utilisation de ce langage pour diverses raisons. Notamment pour sa simplicité qui survole un petit peu la plupart des notions importantes dans l’univers du codage. Le Chip-8 est un excellent langage pour débuter, il ne possède pas autant d'instructions que la plupart des langages modernes mais offre quand même une grande liberté malgré tout. Il permet de se familiariser avec le domaine pour les amateurs souhaitant commencer. Les fonctionnements du langage Chip-8 en détails vous sont indiqués dans l’annexe 2 et 3

4.2 Un jeu en CHIP-8

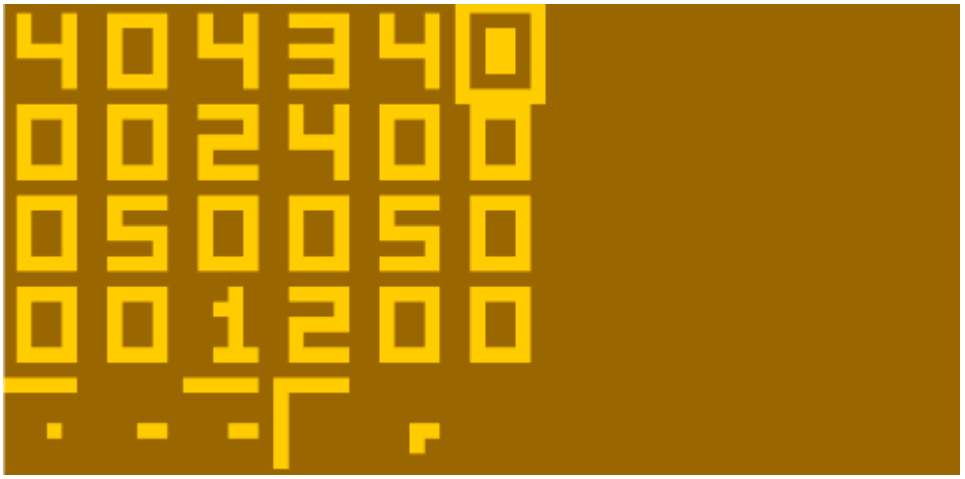


Figure 28 : Extrait du mastermind

Afin de s’approprier le langage, nous avons tout d’abord commencé par programmer un jeu simple en CHIP-8. Pour cet exercice, nous avons choisi le très célèbre ‘Mastermind’. Dans ce jeu, l'ordinateur choisit une combinaison de 4 pièces choisies parmi 6 couleurs différentes. Le but du jeu est de deviner la combinaison avec le moins d’essais possibles. A chaque essai, l'ordinateur indique s’il y a des pièces de bonne couleur et si elles sont bien placées. Voir figure 28 pour une capture d’écran du jeu.

Le jeu a été codé sur OCTO sur ordinateur pour pouvoir être débogué rapidement. Comme nous sommes sur un écran monochrome, les couleurs ont été remplacées par des chiffres mais le principe reste le même. Pour tirer un nombre au hasard, nous

avons utilisé l'instruction « random » intégrée au langage CHIP-8. La saisie du joueur se fait avec la croix directionnelle, gauche/droite pour changer la valeur et haut/bas pour changer de puissance de 6. Enfin après avoir appuyé sur 'A', si la combinaison est la bonne, l'écran affiche 'WIN' sinon, un petit carré en dessous de la combinaison va venir se remplir comme sur la figure 29.

Figure 29 : Notation combinaison

Chaque côté du carré extérieur représente un chiffre bon et bien placé. Chaque petit carré à l'intérieur représente un chiffre bon mais mal placé. Nous avons dû utiliser ce système pour parer la résolution très faible de 64x32 pixels supportée par le CHIP-8.

4.3 Le fonctionnement de l'émulateur trouvé sur internet et ses modifications

Tout d'abord, l'émulateur commence par lire le fichier du jeu et copie tout son contenu dans la liste 'memory' servant de RAM. Le « Program Counter »(PC) est initialisé avec la valeur 0x200 qui correspond à la première adresse RAM qui sera lue. Ensuite, la fonction 'execute' va lire l'opcode présent à l'adresse 0x200 et effectuer les actions correspondantes à celui-ci. Puis le PC sera modifié en conséquence et la fonction 'execute' exécutera la prochaine instruction qui elle-même modifiera le PC…

A la base, l'émulateur était conçu en python pour fonctionner avec la librairie Pygame sur pc, il a donc fallu adapter ce programme pour notre plateforme.
Tout d'abord, nous avons dû remplacer la gestion de l'écran, ce qui revenait à changer les "self.screen" en "display" mais nous avons rencontré un problème, à chaque cycle, l'émulateur effaçait tout l'écran puis l'affichait pixel par pixel.

Cette méthode passait en pygame mais sur notre faible microcontrôleur c'était injouable. Nous avons donc supprimé cette fonction et comme la commande "sprite" est la seule commande en CHIP-8 à permettre l'affichage nous avons décidé de directement afficher les pixels sur l'écran lors de l'appel de cette fonction.

Ce fut un succès et les jeux étaient beaucoup plus fluides. Ensuite nous avons modifié la manière de gérer les entrées. Puis ce fut le tour du « timer », pour cela nous nous sommes appuyés sur la fonction « tick_us() » de la librairie « utime » qui permet de renvoyer le temps en microsecondes depuis le dernier allumage, nous l'utiliserons pour créer le timer de 16.67 millisecondes entre chaque frame. Enfin, pour le son, nous avons simplement adapté la classe relative au son en ayant précédemment précisé la fréquence du buzzer à savoir 500Hz d'après la documentation sur le CHIP-8.

4.4 Les améliorations de l’émulateur

La première amélioration que nous avons effectuée fut l’ajout d’une fonction de sauvegarde de l’état du jeu. Lorsque l’on appuie sur le bouton home, le fichier ROOT/saves/NomJeu.sav est ouvert (ou créé s’il n'existe pas). Dans ce fichier va être enregistré dans l’ordre, les 4Ko de mémoire RAM, les 16 registres 8bit, le registre 16bit, le Program Counter de 16bit, le timer de délai 8bit, les 64x32 pixels de l’écran et le stack composé de valeurs 16bit.
Ensuite, au démarrage du jeu, la console demande à l’utilisateur s’il souhaite charger la sauvegarde précédemment créée. Si l’utilisateur accepte, la console va rouvrir le même fichier puis répéter l’opération précédente dans l’ordre inverse.

Dans l’optique de rendre compatible un maximum de jeux, nous avons aussi ajouté une fonction permettant de créer ses propres configurations de touches dans un fichier Python. Il suffit de créer un fichier « .py » avec comme nom, le nom du jeu pour lequel la configuration s’applique, par exemple : “Tetris.py” pour Tetris.ch8
Dans le fichier, il faudra copier le contenu de « default.py », qui contient le dictionnaire « Keydict », le même que dans l’émulateur. Enfin, il ne reste plus qu'à modifier les valeurs en CHIP-8 correspondantes à chaque touche.

La dernière amélioration est l’ajout d’un booléen ’pause’ qui permet de stopper l'exécution du programme lorsque la touche ‘start’ est pressée et ainsi de mettre pause à celui-ci. Nous avons également ajouté un logo pause qui vient s’afficher afin d’avoir un visuel pour l’utilisateur.

5 Et ensuite ?

Nous avons relevé le défi et prouvé qu'il était possible de construire de A à Z et à moindre coût une console portable sur laquelle on peut faire tourner des jeux codés en CHIP-8. Néanmoins le projet ne s'arrête pas là…

5.1 Vers un PCB custom

Pour réduire encore la taille et le coût de la console, il pourrait être intéressant de concevoir un circuit imprimé (PCB) sur lequel viendrait se fixer tous les composants. Cela nous permettrait, entre autres, de souder des boutons « CMS » (composant monté en surface) qui feraient gagner encore de l'épaisseur. De plus cela enlèverait l'encombrement des fils et rendrait l'assemblage plus simple.

5.2 Le niveau de batterie

Il serait bon de pouvoir connaître le niveau de la batterie. Nous avons trouvé un lien vers un programme d'exemple sur le site du module LiPo Shim. Nous pourrions par exemple avoir une icône de batterie sur l'écran d'accueil nous indiquant la charge.

5.3 Des inserts

Nous voulions utiliser des inserts pour fixer plus durablement les composants au boîtier mais compte tenu de l'encombrement des inserts cela n'a pas pu se faire. De plus, nous avons remarqué que les composants étaient solidement fixés et ne bougeaient pas. D'autre part, nous avons observé que la fixation entre les deux parties du boîtier se dégradait rapidement, il serait donc intéressant de se pencher sur ce sujet et de modifier la partie basse du boîtier pour permettre l'intégration des inserts.

5.4 L’ajout d’un clavier 16 touches

Au départ, nous avions choisi un clavier 16 touches pour jouer puis par soucis d’ergonomie et d’esthétique, nous avons préféré éclater le clavier en diminuant le nombre de touches. Néanmoins, cela pose problème pour jouer à certains jeux qui sont plus rares mais qui nécessitent 16 touches. C’est pourquoi, nous avons pensé à une solution pratique : la possibilité de brancher un clavier. Cela permettrait d’avoir un clavier 16 touches qui n'entache pas l’esthétique, qui peut être plus grand donc plus économique, plus personnel et qui pourrait même nous servir en tant que clavier pour un deuxième joueur dans un second temps.

Conclusion

Pour conclure, la réalisation d'une console à moindre coût est donc possible. Comme vous avez pu le constater, ce défi relevé a été une franche réussite. En plus d'avoir créé notre propre console, nous nous sommes également permis de la personnaliser à notre guise. Le micro-python sur le Raspberry offre une telle liberté qu'il est possible de presque tout réaliser dessus, que l'on passe par le micro-python ou par le C++ pour tirer parti de toute la puissance du microcontrôleur.

Le rapport que nous avons pu vous proposer ne représente finalement qu'un exemple destiné à tout public afin qu'il puisse s'y représenter et reproduire une console qui lui plaît. Le budget total de notre console a donc été d'environ 57 € (sans compter le clavier 16 touches), Cependant, ce prix peut varier en prenant des composants qui correspondent à vos besoins et votre budget. Pour baisser le prix notamment, il est aussi possible de commander vos composants sur d'autres sites d'électroniques.

Enfin, la réalisation de ce projet peut mener à l'émulation d'autres consoles plus développées et plus récentes avec un écran couleur tel que la Gameboy, la NES etc.

Bibliographie

1) L'hardware

Vidéo installation du micro-python -
https://www.youtube.com/watch?v=8UzVhRCeHoE&ab_channel=Ncomet

Installation micro-python -
https://micropython.org/download/RPI_PICO/

2) L'assemblage

Onshape -
https://www.onshape.com/fr/

3) L'OS

Librairie ecran -
https://github.com/rdagger/micropython-ssd1309

Librairie SD -
https://github.com/micropython/micropython-lib/tree/master/micropython/drivers/storage/sdcard

Song Maker -
https://musiclab.chromeexperiments.com/Song-Maker

Photopea -
https://www.photopea.com

4) L'émulateur

Documentation CHIP-8 -http://devernay.free.fr/hacks/chip8/C8TECH10.HTM

Emulateur Python - https://github.com/AlpacaMax/Python-CHIP8-Emulator

OCTO - https://github.com/JohnEarnest/Octo

ROMS - https://github.com/kripod/chip8-roms

ROMS TEST - https://github.com/Timendus/chip8-test-suite

Comptes rendus hebdomadaires

Compte rendu n°1

Cette semaine, nous avons, dans un premier temps, installé le micro-python sur la Raspberry. Nous avons également programmé le clavier et assigné les différentes touches à des lettres.

Ensuite nous avons fait fonctionner l'écran en le branchant et le configurant. Après cette étape, il nous a été possible de trouver un émulateur python sur internet et de l'importer sur notre microcontrôleur

Au programme de la semaine prochaine : assemblage de tous les composants et établir la communication entre eux. Enfin, continuer à adapter notre émulateur à notre machine en micro-python

Compte rendu n°2

Au cours de la semaine, nous avons créé un menu de sélection de jeux et programmé les touches afin de pouvoir naviguer dans ce menu.

En mettant les composants en relation, nous avons pu remarquer des problèmes de compatibilités des touches sur les différents jeux. Il nous a aussi été difficile de mettre en place le système de changement de pages sur le menu.

La semaine prochaine, il faut coder les touches individuellement sur chaque jeu, améliorer l'ergonomie et l'esthétique du menu et ajouter des jeux sur la carte SD

Compte rendu n°3

Pendant la semaine, nous avons réalisé une première maquette de la console en 3D sur Onshape. Une commande de différents composants a également pu être effectuée.

Nous avons aussi, mis en place une animation de démarrage et éclater le clavier pour résoudre les problèmes d'ergonomie.

Enfin, la semaine prochaine, nous mettrons en place un bouton « Home », la possibilité de sauvegarder sa partie et le lancement de fichier de configuration associé à chaque jeu individuellement.

Compte rendu n°4

Lors de cette semaine, nous avons implémenté le système de sauvegarde, créé une option permettant de faire son propre paramétrage de touche sur chaque jeu et modélisé la partie haute du boitier de la console.

Pour la semaine prochaine, nous corrigerons quelques petits bugs liés à la sauvegarde, ajouterons le logo Polytech au lancement des jeux, regarderons s'il est possible de mettre des interrupts pour les touches, implémenterons le son à l'aide d'un buzzer et terminerons la configuration du bouton « Home ».

Compte rendu n°5

Lors de cette dernière semaine, nous avons terminé l'impression de la console, rajouté du son avec un buzzer et programmé une fonction pause sur l'émulateur.
Nous avons également amélioré l'interface, mis la carte sous batterie et créé un fichier pinout qui permet de modifier tous les pins en un fichier et rapidement.

Il reste à faire, le réglage des inputs avec des interrupts, l'affichage du niveau de la batterie et avancer sur le rapport.

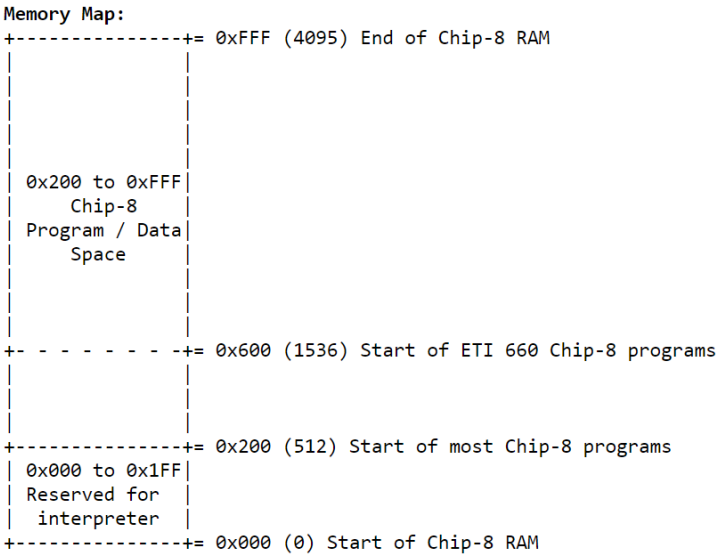
Annexe 1 : *Les composants :*

- Écran (**20,99€**) - https://joy-it.net/en/products/COM-OLED2.42_
- Lecteur SD (**5,80€**) - https://www.gotronic.fr/art-lecteur-carte-microsd-upesy-37485.htm
- RP Pico (**5,45€**) - https://www.gotronic.fr/art-carte-raspberry-pi-pico-33027.htm
- LiPo shim (**12,50€**) - https://www.gotronic.fr/art-module-pico-lipo-shim-pim557-36985.htm#complte_desc
- Batterie (**7,10€**) - https://www.gotronic.fr/art-accu-lipo-3-7-vcc-800-mah-l453350-31713.htm
- Matrice 16 boutons (**5,90€**) - https://www.gotronic.fr/art-module-matriciel-16-bp-25648.htm
- Bouton à l'unité (**0,30€**) - https://www.gotronic.fr/art-bp-subminiature-krs0610-4269.htm
- Carte (PCB) pour placer ses boutons (**1,00€**) - https://www.gotronic.fr/art-carte-d-essai-ee4060-21092.htm
- Buzzer (**1,90€**) - https://www.gotronic.fr/art-buzzer-sv12-5hq-30818.htm

Annexe 2 : *Plus de détails sur le CHIP-8 (1/2) :*

Notre console est faite pour supporter des jeux programmés en CHIP-8, ce langage est un langage de programmation datant de 1978 et ayant été développé par Joseph Weisbecker. Ce langage a beaucoup été utilisé sur les ordinateurs que l'on qualifiait un peu comme "fait soi-même", comme par exemple le "ETI 660", le "COSMAC VIP" ou encore le "DREAM 6800". Ces ordinateurs étaient faits pour utiliser une télévision en guise d'écran, avec entre 1 et 4 KB d'espace RAM. Ces ordinateurs utilisaient des claviers 16 touches configurés en hexadécimal (de 0 à F). Le système prenait 512 bytes et les programmes en hexadécimal étaient encore plus petits. Le but du CHIP-8 est donc d'offrir un large panel de possibilités pour un matériel limité en performance et en espace. Néanmoins ce langage se popularisera vraiment dans les années 90, où il sera utilisé sur la calculatrice graphique HP48 avec sa déclinaison le CHIP-48. Ensuite ce langage a connu de nombreuses modifications et dérives qui ont mené à la création de différents interpréteurs tels que : MS-DOS, Windows 3.1, Amiga, HP48, MSX, Adam et ColecoVision. Voyons maintenant, les spécifications de ce langage, pour cela intéressons-nous d'abord à la mémoire.

Celle-ci peut aller jusqu'à 4 Kilobytes soit 4096 bytes (de 0x000 à 0xFFF), les 512 premiers bytes sont alloués à l'interpréteur original et ne doivent pas être utilisés pour un programme(de 0x000 à 0x1FF). La plupart des programmes commencent donc après ces 512 bytes, mais il existe certains cas où le programme ne commence qu'à 1536 bytes comme pour le ETI 660. Ici se trouve un schéma résumant l'allocation de la mémoire en Chip-8.



Annexe 3 : *Plus de détails sur le CHIP-8 (2/2) :*

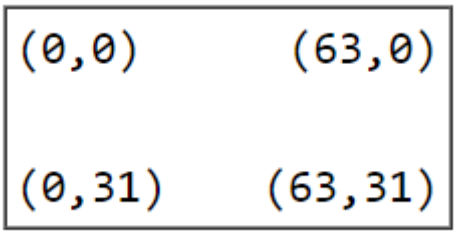
Après avoir vu la mémoire voyons maintenant les registres qui sont simplement les variables qui vont se trouver au sein du programmes pour stocker une valeur. Ils sont souvent aux nombres de 16, de 8 bits maximum de stockage. Ces registres sont souvent nommés sous la forme : « *Vx* » où « *x* » représente une valeur hexadécimale entre 0 et F. Il existe aussi un registre de 16 bits nommé “I” qui est utilisé pour stocker des adresses mémoire, donc souvent seuls les 12 premiers bits sont utilisés. Ce registre exceptionnel n’est pas le seul, le registre « VF » est également un registre spécifique qui ne doit pas être utilisé dans un programme car il sert de signal d’alerte pour certaines instructions. Dans le même genre, le chip-8 se réserve également deux autres registres, un pour le son et un autre pour le temps. On peut aussi trouver des “pseudo-registres” qui ne sont pas accessibles par les programmes. Il y a d'abord le Program Counter (PC) qui doit faire 16 bits et qui est utilisé pour stocker l’adresse de l'exécution en cours. Ensuite, il y a le stack pointer (SP) qui peut-être de 8 bits et qui sert à garantir la bonne exécution des sous programmes en gérant l’ordre notamment.

Ensuite, concernant pour le clavier, il est composé de 16 touches qui sont représentées par des valeurs hexadécimales comme sur le schéma :

1	2	3	C
4	5	6	D
7	8	9	E
A	0	B	F

Ce schéma est souvent modifié afin d’adapter les commandes à chaque plateforme ou jeu.

Et alors, pour l’affichage ? Il faut savoir que la résolution utilisée par le Chip-8 est souvent du 64x32 pixel en monochrome sous le format :



Où ici (0,0), (63,0), (63,31) et (0,31) représente des coordonnées nécessaires pour gérer l’affichage sur l’écran. Pour d’avantages d’informations, vous pouvez aller dans la bibliographie.

Annexe 4 : *Modification dans l'émulateur (1/6) :*

Initialement, l'émulateur était conçu en python pour fonctionner avec la librairie Pygame sur pc, il a donc fallu adapter le programme pour notre plateforme.

Ajout d'importations :

```
from machine import Pin, SPI
from ssd1309 import Display
from utime import ticks_ms
from buttons import lecture_touche
```

Ces librairies permettront de gérer l'écran et les timers.

Suppression des lignes 123 et 124 :

```
pygame.init()
pygame.time.set_timer(pygame.USEREVENT+1, int(1000 / 60))
```

La deuxième ligne est un timer de 16.67ms qui était utilisé pour simuler le timer du CHIP 8

Dictionnaire des entrées (ligne 129) :

```
self.keyDict = {
    49 : 1,
    50 : 2,
    51 : 3,
   …….
    99 : 0xb,
    118 : 0xf
}
```

Les codes à gauche correspondent aux codes ASCII renvoyés par Pygame et les codes à droite aux codes hexadécimaux des touches du CHIP 8.

Taille des pixels (ligne 158) :

```
self.size = 10
```

Cette valeur correspond à la taille des pixels dessinés par les programmes, ici comme nous utilisons un écran de 128*64 pour une résolution de 64*32 du CHIP 8 nous mettons cette constante à 2. Ce qui donnera des carrés de 2 pixels de côté.

Annexe 5 : *Modification dans l'émulateur (2/6) :*

Initialisation de l'écran (lignes 161-163) :

```
self.screen = pygame.display.set_mode([width * self.size, height * self.size])
self.screen.fill(self.oneColor)
pygame.display.flip()
```

On a supprimé la première ligne puis remplacé les deux dernières par :

```
display.clear()
display.present()
```

qui sont des instructions similaires mais issues de la librairie ssd1309

Modification de la gestion des touches et du timer (lignes 599 – 618) :

```
for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()
```

Si le bouton de fermeture de la fenêtre est cliqué alors quitter le programme, on supprimera ces deux lignes

```
        elif event.type == pygame.USEREVENT+1
            self.delayTimer.countDown()
```

Utilisation du timer de 16.67ms pour décrémenter le timer en CHIP 8.

```
        elif event.type == pygame.KEYDOWN:
            try:
                targetKey = self.keyDict[event.key]
                self.keys[targetKey] = True
            except: pass
```

Contrôle des touches pressées.

```
        elif event.type == pygame.KEYUP:
            try:
                targetKey = self.keyDict[event.key]
                self.keys[targetKey] = False
            except: pass
```

Annexe 6 : *Modification dans l'émulateur (3/6)* :

Contrôle des touches relâchées.

Tout d'abord on vient rajouter ligne 125 :

```
self.lastime = ticks_ms()
self.lastKey = '0'
```

ces variables nous serons utiles plus tard

```
 if ticks_ms() - self.lastime >= (1000/60):
        self.lastime = ticks_ms()
        self.delayTimer.countDown()
```

Vient remplacer l'ancienne fonction permettant de gérer le timer.

Pour la gestion des touches, on va réutiliser la librairie buttons, plus précisément sa fonction « lecture_touche() »

```
self.pin_value = lecture_touche()
```

On commence par récupérer la touche pressée.

```
    if self.lastKey != self.pin_value:
        targetKey = self.keyDict[self.lastKey]
        self.keys[targetKey] = False
```

Si la dernière touche pressée est différente de la touche actuelle, alors on met la dernière touche à 0.

```
    if self.pin_value != None:
        targetKey = self.keyDict[self.pin_value]
        self.keys[targetKey] = True
        self.lastKey = self.pin_value
```

Si la touche actuelle est différente de None alors on met l'état de celle-ci à 1, et elle devient la dernière touche pressée.

Annexe 7 : *Modification dans l'émulateur (4/6)* :

Suppression de la fonction d'affichage (lignes 630 – 640) :

Voici la fonction d'affichage prévue pour Pygame

```
    def display(self):
        for i in range(0, len(self.grid)):
            for j in range(0, len(self.grid[0])):
                cellColor = self.zeroColor

                if self.grid[i][j] == 1:
                    cellColor = self.oneColor


*               pygame.draw.rect(self.screen, cellColor, [j * self.size, i * self.size, self.size, self.size], 0)
        pygame.display.flip()
```

Concrètement self.grid est une matrice de 64*32 constituée de 0 et de 1 représentant l'état de chaque pixel, la fonction display() lit tous les éléments de la matrice et dessine chaque pixel avec l'état correspondant. On pourrait modifier cette fonction en remplaçant la ligne marquée d'un astérisque par :

```
display.fill_rectangle( j * self.size,  i * self.size, self.size, self.size, invert=True/False)
```

Mais après avoir testé nous nous sommes rendu compte que cette méthode était beaucoup trop lente car elle impliquait de redessiner l'entièreté de l'écran à chaque frame ce qui prenait +/- 4 secondes.

Nous sommes donc remontés ligne 541 où il y avait la fonction permettant de mettre à jour la matrice « self.grid » lorsqu'un sprite était dessiné à l'écran.

```
for i in range(len(spriteBits)):
            for j in range(8):
                try:
                    inversion = False
                    if self.grid[Vy + i][Vx + j] == 1 and int(spriteBits[i][j]) == 1:
                        collision = True
                        inversion = True


                    if int(spriteBits[i][j]) == 1:
```

Annexe 8 : *Modification dans l'émulateur (5/6) :*

```
        display.fill_rectangle((Vx + j) * self.size, (Vy + i) * self.size, self.size, self.size,
invert=inversion)

                self.grid[Vy + i][Vx + j] = self.grid[Vy + i][Vx + j] ^ int(spriteBits[i][j])
            except:
                continue
display.present()
return collision
```

Nous avons rajouté les lignes en vert qui servent à dessiner les pixels du sprite sous forme de carré de 2px de côté. Sans oublier le display.present() qui permet d'envoyer le buffer à l'écran et donc d'afficher les modifications.

La fonction clear (lignes 557-560) a elle aussi été modifiée :

```
    def clear(self):
        display.clear()
        display.present()

        for i in range(len(self.grid)):
            for j in range(len(self.grid[0])):
                self.grid[i][j] = 0
```

On a simplement rajouté la fonction display.clear()

Annexe 9 : *Modification dans l'émulateur (6/6) :*

Modification du mainLoop() (lignes 620-628) :

```
def mainLoop(self):
    clock = pygame.time.Clock()

    while True:
        clock.tick(300)
        self.keyHandler()
        self.soundTimer.beep()
        self.execution()
        self.display()
```

La fonction clock.tick() est une fille de pygame.time.Clock(), clock.tick(300) est une instruction permettant de limiter le framerate à 300 fps, nous ne l'utiliserons pas et elle peut donc être supprimée. De même pour « self.display() » qui peut être supprimée étant donné que la fonction self.display() à été supprimée précédemment.

Lancement du jeu (ligne 643) :

```
chip8.readProg(sys.argv[1])
```

A la base, l'émulateur est censé se lancer par ligne de commande et l'utilisateur doit entrer le nom de la rom après celui du programme. Ici, pour l'instant nous allons simplement remplacer cette ligne par

```
chip8.readProg("games/Tetris.ch8")
```

Où "games" est le nom du dossier sur le Raspberry contenant les jeux et « Tetris » peut être remplacé par le nom du jeu auquel on souhaite jouer.

Console low tech Chip-8

Résumé

Nous avons entrepris la conception et la programmation d'une console portable rétro CHIP-8, offrant ainsi la possibilité de jouer à des classiques du jeu vidéo. Elle se base sur un Raspberry Pi Pico et utilise un écran OLED monochrome de 2.42 pouces Notre rapport détaille chaque étape de la création, de la sélection des composants, au câblage de ces derniers, en passant par la programmation du logiciel. Destiné tant aux novices qu'aux experts en électronique et informatique, ce guide non-exhaustif vise à permettre à chacun de construire sa propre console rétro, ouvrant ainsi la voie à une expérience de jeu personnalisée et enrichissante.

Mots-clés

Electronique, impression 3D, rétro, jeux-vidéo, console, émulation, interface utilisateur, faible budget, économique, Mécanique, impression 3D, micro-python, Informatique embarqué, musique, CHIP-8.

Abstract

We built a retro console based on the famous and affordable Pi Pico from raspberry. It uses a 2.42 inches monochrome OLED display and is fully customizable since we basically made everything from scratch. The whole program is written in python which makes it easily understandable for beginners. Of course, the console is battery-powered so you can bring it everywhere with you to play your favorite games on the go. In this report, we tried to summarize everything for you to make your very own console at home. You only need a few cheap components and Thonny installed on your computer to get started.

Keywords

electronic, 3D printing, retro-gaming, handheld, emulation, GUI, low-tech, cheap, embedded computing, Mechanical, 3D printing, micro-python, low budget, music, CHIP-8.

Encadrant académique	Étudiant(e)s
Pierre Gaucher	*Johan Ondet* *Raphaël Texier* *Matéo Aranhita Saucia*