# Introduction

Dans le cadre du Summer School 2, nous devons réaliser un Pong. Le but est de le créer entièrement, c’est-à-dire que nous devons concevoir la schématique de l’électronique, faire le routage, concevoir le PCB, monter et souder les composants ainsi que de concevoir le code complet du jeu.

La plaque électronique doit être conçue comme un jeu portable. Le circuit sera alimenté par deux piles AAA. Un des enjeux de ce projet est de réussir à obtenir une durée de vie la plus longue possible, de ce fait, nous devons optimiser la consommation de courant et gérer le système de mise en veille automatique.

Nous allons donc d’abord vous expliquer en détail le cahier des charges, puis nous vous présenterons notre architecture matérielle ainsi que l’implémentation du routage. Enfin, nous vous détaillerons le fonctionnement logiciel de notre jeu à l’aide de diagramme UML.

# Cahier des charges

Les règles du jeu sont assez simples. Le joueur devra jouer contre l’ordinateur. Une option de départ permet de choisir le niveau de difficulté du jeu. En fonction de cette difficulté, la vitesse de la balle augmentera plus rapidement. La vitesse de réaction de l’ordinateur est constante, ainsi, après un certain temps de jeu, l’ordinateur sera plus lent que la balle et il sera donc possible de gagner.

Dans les contraintes matérielles, nous devons utiliser un écran tactile LCD, de type NHD. Le microcontrôleur est un PIC18LF25K22. Nous devons utiliser des supports de piles AAA pour la partie alimentation.

Concernant la conception, nous utiliserons le logiciel Altium Designer pour le développement hardware et MPLAB X pour l’environnement logiciel du microcontrôleur. Le langage de programmation utilisé est le C.

# Schéma bloc

Figure 1 : Schéma bloc

Comme on peut le voir ci-dessus, le schéma bloc de notre projet est relativement simple.

Le premier bloc concerne la partie alimentation, dont nos 2 piles ainsi qu’un MOSFET permettant de protéger le circuit contre une potentielle inversion de polarité. Ainsi, les deux piles AAA vont venir alimenter la plaque avec du +3V.

La partie PIC permettra de gérer la logique d’affichage de l’écran, ainsi qu’à contrôler les MOSFET pour éteindre l’écran LCD ainsi que le rétroéclairage, afin d’économiser de l’énergie.

# Conception

Dans cette partie, nous allons aborder la conception l’architecture matérielle, ainsi que la schématique et l’implémentation du PCB. Vous trouverez en annexe le schéma complet de notre circuit et le routage.

Notre architecture matérielle a été pensée pour que la prise en main de l’utilisateur soit la plus agréable possible. Les piles et supports de piles sont placées sur le bottom du PCB, car nous souhaitons imprimer en 3D un boitier qui viendra couvrir la partie arrière du circuit pour pouvoir protéger les composants et d’offrir une bonne prise en main.

La schématique contient un système d’anti-inversion de polarité ainsi que de transistor permettant d’éteindre l’alimentation de l’écran LCD et le rétroéclairage de celui-ci, que nous allons détailler maintenant.

## Anti-inversion de polarité

Figure 2 Sécurité d'inversion de polarité

Notre carte est équipée d’un système empêchant tout problème provenant d’une inversion de polarité lors de l’installation des piles.

Cette partie fonctionne grâce à un simple MOSFET channel N, ainsi, si les piles sont branchées à l’envers, ceci nous permet de garder l’alimentation à 0V au lieu de -3V, ce qui risquerait d’endommager nos composants. Il est important de choisir un MOSFET ayant une résistance Rdson la plus faible possible, pour ainsi limiter toute consommation excessive de courant pour obtenir une durée de vie des batteries la plus longue possible.

Nous avons mis des pointes de tests pour simplifier la mesure lors des tests et de la mise en service.

## ON/OFF rétroéclairage

Figure 3 On/Off alimentation

Toujours dans l’optique d’économiser l’énergie, notre circuit possède un système de « veille ». C’est-à-dire que si l’utilisateur n’interagie plus avec l’écran LCD pendant un certain temps, le PIC va décider de couper l’éclairage de l’écran automatiquement.

Ce système est réalisé à l’aide d’un MOSFET channel N, qui fonctionne comme un interrupteur. Dès lors que l’on envoie un signal de 3V (label « BackLightCmd » sur le schéma), le MOSFET va commuter et ainsi alimenter les LEDs. Cela nous permet de gérer l’alimentation du rétroéclairage de l’écran, car c’est cette partie qui consomme le plus dans notre circuit, mais aussi de couper l’alimentation générale de tout l’écran LCD.

## ON/OFF Alim LCD

Figure 4 On/Off alim LCD

Sur la figure 4, on peut voir le montage avec MOSFET qui vient couper le GND de l’écran LCD pour pouvoir l’éteindre quand on le décide.

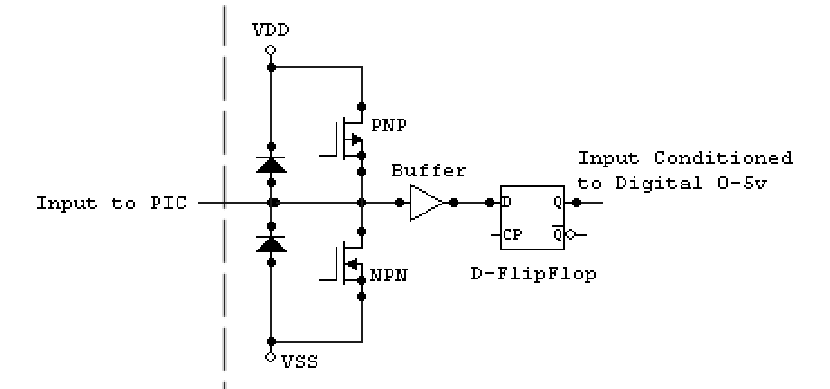


Figure 5 Internal Port

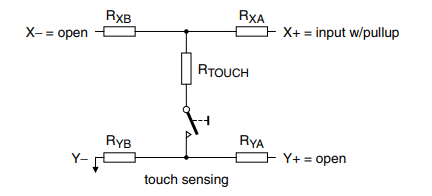
Pour pouvoir couper l’alim du LCD, il est important que toutes les PINS reliés au PIC soient mis au VCC. Etant donné que l’on vient couper le GND, si d’autres PINS connectées directement au PIC sont au GND, cela peut causer une alimentation indirecte via ces PINS dû à la structure interne comme représenté sur la figure 5. Ce phénomène sera important à prendre en compte lorsque l’on voudra mettre notre processeur en mode sleep (économie d’énergie).

## Touchscreen

Notre écran LCD possèdent 4 pins (X-,X+,Y-,Y+) qu’il va falloir connecter correctement au pic afin de pouvoir détecter une pression et calculer la position de celle-ci.

**Mode touch :**

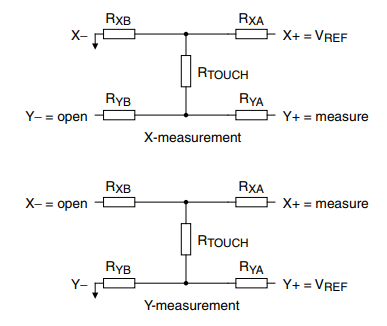
Ce mode va nous permettre de pouvoir détecter via une interruption sur le pic quand notre écran a été pressé ou relâché. Pour ce faire il est important de suivre la configuration ci-dessous.

Nous constatons que pour ce mode il faut configurer nos pins d’une certaine manière.

* Il faut une des pins X en entrée et l’autre sur une pin d’interruption avec une pull.up.
* Il faut une des pins Y en sortie au GND et l’autre en entrée.

Figure 6 Touch mode pin configuration

**Mode measure**

Ce mode va nous permettre de mesurer le positon de notre touch en X et en Y. Il y aura donc 2 mesures à faire, une par axe.

* Pour mesurer l’axe X : il nous faut une des pins Y sur une entrée analogique et l’autre en entrée. Il nous faut également une des pins X en sortie sur le VCC et l’autre en sortie sur le gnd.
* Pour mesurer l’axe Y c’est exactement le même principe que pour l’axe X, mais cette fois si ce sont les pins Y qui sont au VCC et au gnd et les pins X qui sont respectivement ouverte et sur une entrée analogique.

Figure 7 Measure mode pin configuration

## Schéma LCD et PIC

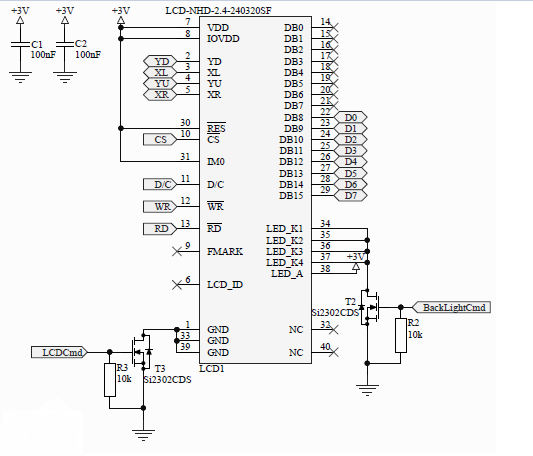
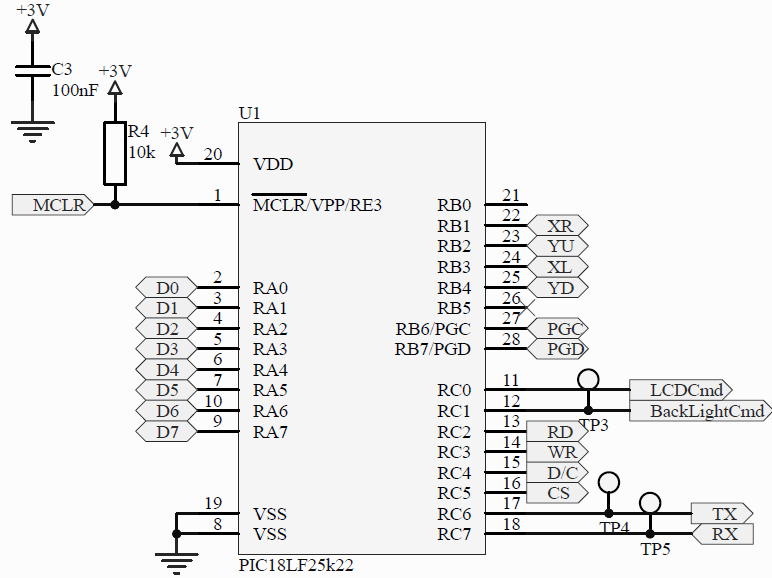
 Voici ci-dessous les connections entre notre écran LCD et notre processeur.

Figure 8 LCD connection with processor

* Les 4 pins X-Y : Ces pins sont connectés sur le pic sur le PORTB(RB1-RB4). RB1 sera utilisée comme pin d’interruption (mode touch), RB1 et RB2 comme pin analogique (mode mesure), RB3 et RB4 comme sortie ou entrée.
* Les pins D0-D8 sont nos pins de data du LCD, on utilise que 8 bit et non 16. Pour ce faire il est important de mettre la pin IM0 du LCD sur le VCC. Il faut également que ces pins de data soit tous sur le même port de notre pic (ceci est dû à la librairie du LCD qui nous est fournie), dans notre cas le PORTA.
* Les pins CS, D/C, WR et RD sont simplement relié au pic sur des I/O quelconque.
* La pin RES n’est pas relié au pic mais au VCC, car on ne compte jamais reset notre écran.
* Il faut également mettre une pull-up sur l’entrée du master reset de notre pic. Il faut aussi ajouter des condensateurs de découplage.
* Le backlight est connecté à une entrée PWM du pic.
* La pin de commande du LCD est elle aussi directement connecté au pic.

**Connecteur pickit3**

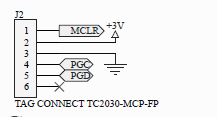


Figure 9 Pinning picKit3

Pour le connecteur de programmation nous avons opté pour le pickit3 avec un socket de type tag-connect.

# Fonctionnalités du système

## Diagramme de cas d’utilisation

Ci-dessus, voici le diagramme de cas d’utilisation de notre logiciel. Comme on peut le voir, il y a 4 parties distinctes.

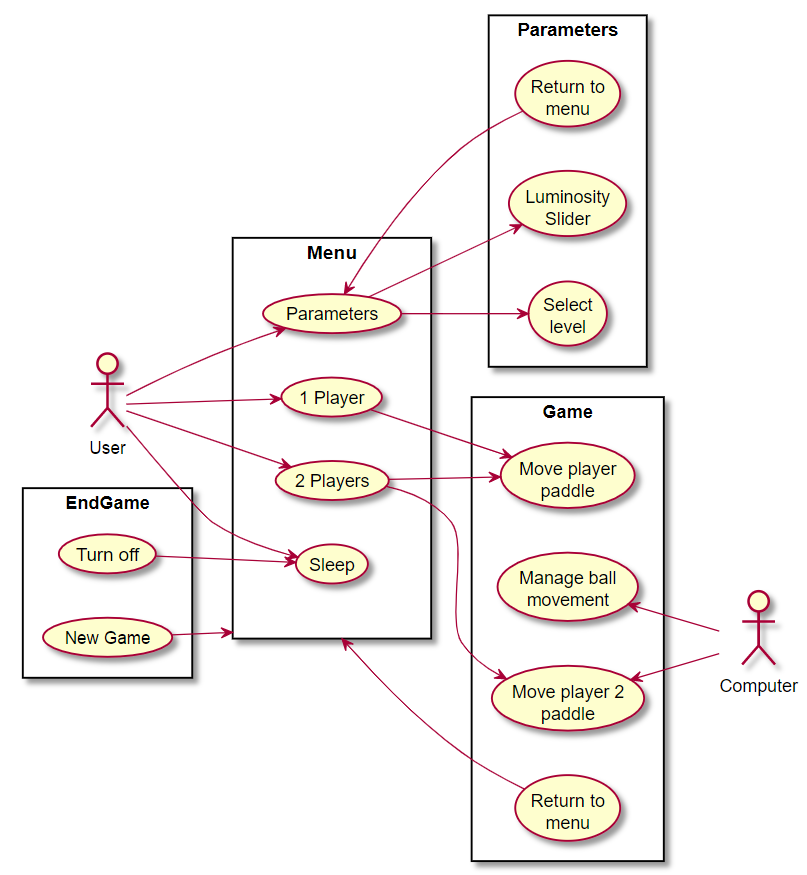


Figure 10 Use case diagram

* En premier lieu, lorsqu’on vient allumer l’écran LCD, un menu principal s’affiche. Sur ce menu on peut choisir de jouer en solo, à deux, d’aller dans les paramètres et d’éteindre le jeu.
* Dans le menu paramètre, on peut venir choisir la luminosité grâce à un slider ainsi que la difficulté du niveau que l’on souhaite jouer. Puis on peut revenir en arrière au menu principal.
* Lorsque l’on rentre dans la partie jeu, on peut jouer contre l’ordinateur si on a choisi une partie en solo, ou alors contre un autre joueur si on a choisi la partie multijoueur. La seule action possible pour le joueur est de déplacer le paddle à gauche ou à droite. L’ordinateur va venir s’occuper du déplacement de la balle ainsi que le déplacement du second paddle en partie solo. En partie duo c’est l’autre joueur qui va déplacer le deuxième paddle.
* A la fin de la partie un menu indique le vainqueur du jeu et offre la possibilitéde rejouer une partie, ce qui nous renvoie au menu principale ou d’éteindre le jeu.

## C:\Users\sebastie.metral\Desktop\SummerSchool\SummerSchool\Software\UML\SequenceDiagram.pngDiagramme de séquence

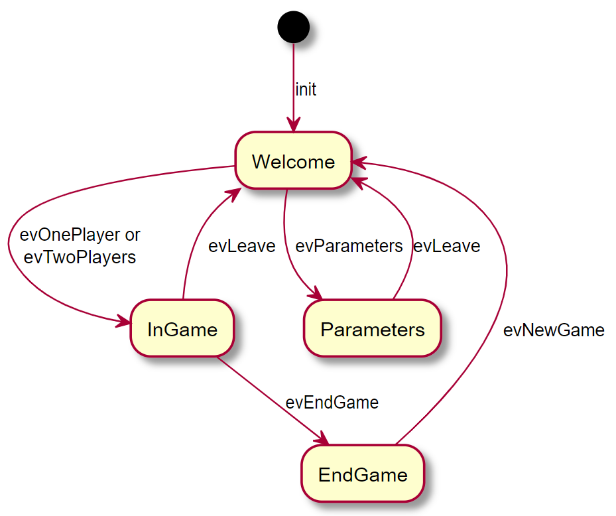
Figure 11 Sequence diagramm

Dans notre projet nous allons utiliser l’interface XF, celle-ci permet de créer des évènements et de les ajouter dans une liste. A chaque passage dans le main nous allons donc pop un ces éléments les uns après les autres afin de les distribuer aux différentes machines d’états. Ces machines d’états décideront d’elle-même des actions à réaliser en fonction de l’évènement qu’elles ont reçu.

# Architecture logicielle

Nous allons maintenant vous expliquer l’architecture logicielle de notre programme.

## Display StateMachine



Tout d’abord nous avons une machine d’état display (figure 12). Cette machine d’état va s’occuper, pour chaque état, d’afficher le menu correspondant. Dans l’état INGAME, elle va également redessiner les éléments du jeu qui auront changer leur paramètre (balle, paddle, score, middle line).

Figure 12 display StateMachine

## TouchScreen StateMachine

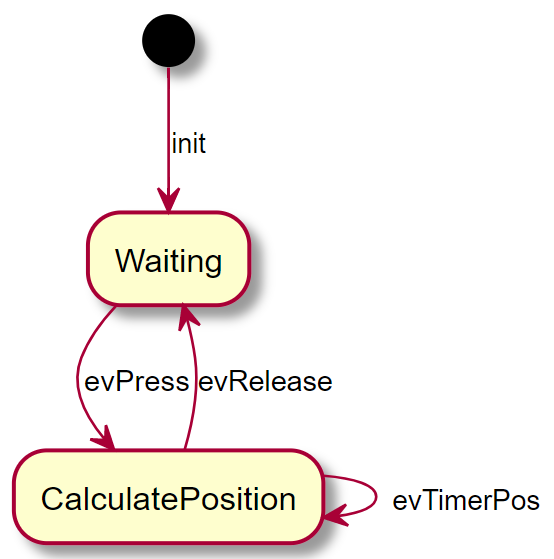
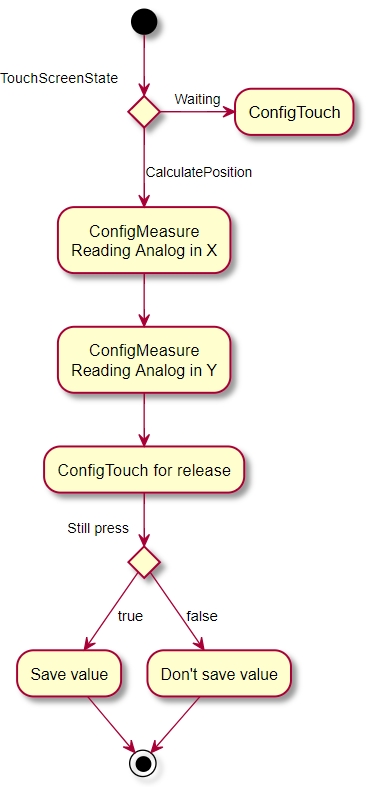


Figure 13touchScreen StateMachine

Nous avons ensuite une machine d’état qui se prénomme touchScreen qui va s’occuper de calculer la position d’une pression sur l’écran (figure 13).

Sur la figure 14, nous pouvons observer les actions que va réaliser cette machine d’état pour chacun de ses états. En bref, elle va tout d’abord attendre un évènement press qui signifie que l’utilisateur a appuyé sur l’écran. Une fois cette evPress généré, elle va calculer la position en X et en Y. Puis, elle va recalculer cette position chaque 4ms, tant que l’utilisateur appuie sur l’écran.

Figure 14 touchScreen: action to do on a state

Lorsque l’utilisateur relâche la pression sur l’écran un évènement release va être généré. Celui-ci va permettre à notre machine d’état d’arrêter de calculer la position chaque 4 ms et de se remettre en état WAITING, afin d’attendre un nouvelle evPress.

## C:\Users\sebastie.metral\Desktop\SummerSchool\SummerSchool\Software\UML\Interrupt.pngInterruption

Voici le diagramme représentant la partie des interruptions de notre code. Ce diagramme est composé de deux partie distincts.

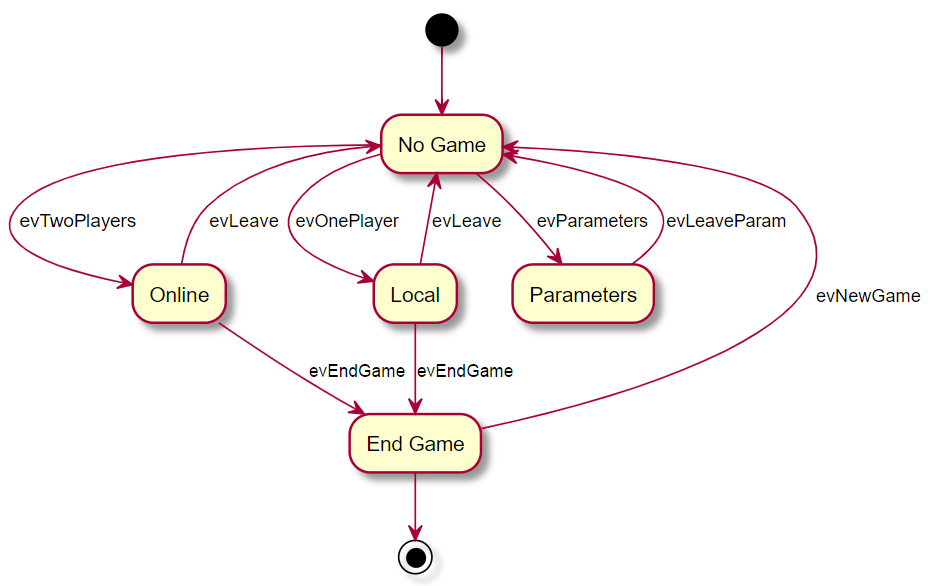
1. Une partie concernant l’interruption timer. Cette interruption est générée chaque 1ms dans notre cas. Elle va permettre de décrémenter les index de nos évènement time présent dans la queue de timer du XF.
2. Une interruption sur RB1 qui va détecter une pression ou un relâchement de pression sur l’écran. Ceci peut être fait en intervertissant le flanc de détection de l’interruption.

Pression 🡺 Flanc descendant

Relâchement 🡺 Flanc montant

Figure 15 interrupt diagramm

## gameController StateMachine



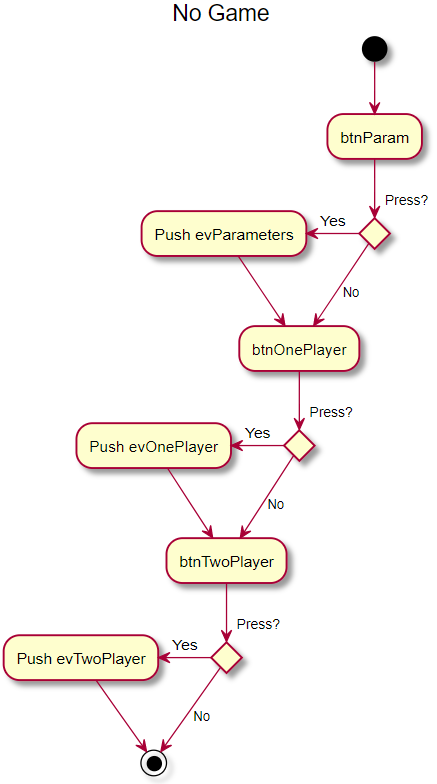
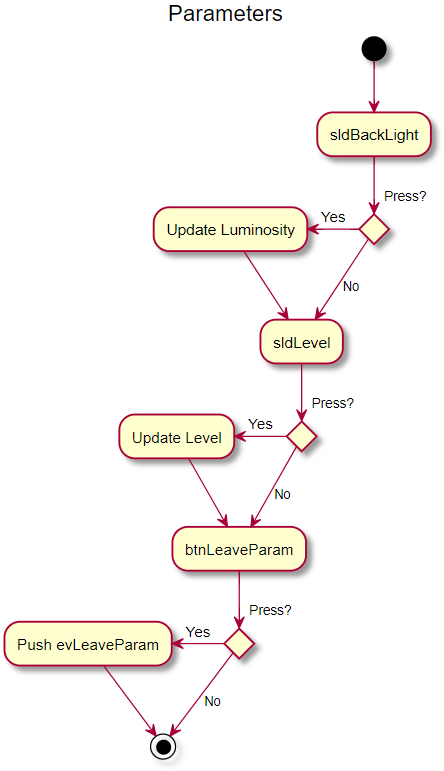
Cette machine d’état va nous gérer toute la logic de notre jeu, que ce soit les actions des boutons des différents menu ou bien le déplacement des éléments du jeu.

Figure 16 gameController StateMachine

En figure 17 nous pouvons voir les actions effectuées par l’état NOGAME. Cette état correspond au menu de notre jeu. Il nous permet de commencer le jeu, d’accéder au paramètre ou encore d’éteindre le jeu.

Figure 17 gameController: action to do on a NOGAME state



En figure 18, nous voyons l’état paramètre. Celui-ci nous permet de modifier les paramètres du jeu tel que la luminosité ou bien le niveau de difficulté.

Il va donc gérer les slidebar correspondante, ainsi que le bouton permettant de revenir au menu principal.

Figure 18 gameController: action to do on PARAMETERS state

En figure 19, nous voyons le fonctionnement de notre jeu en mode local. Il est séparé en 3 partie distinct.

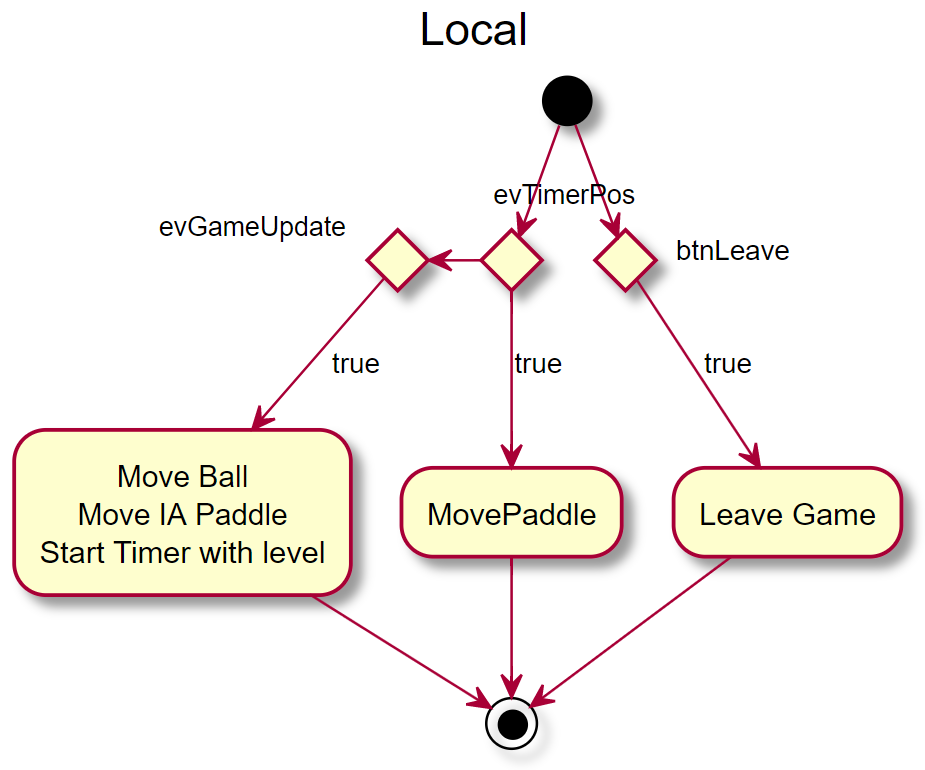
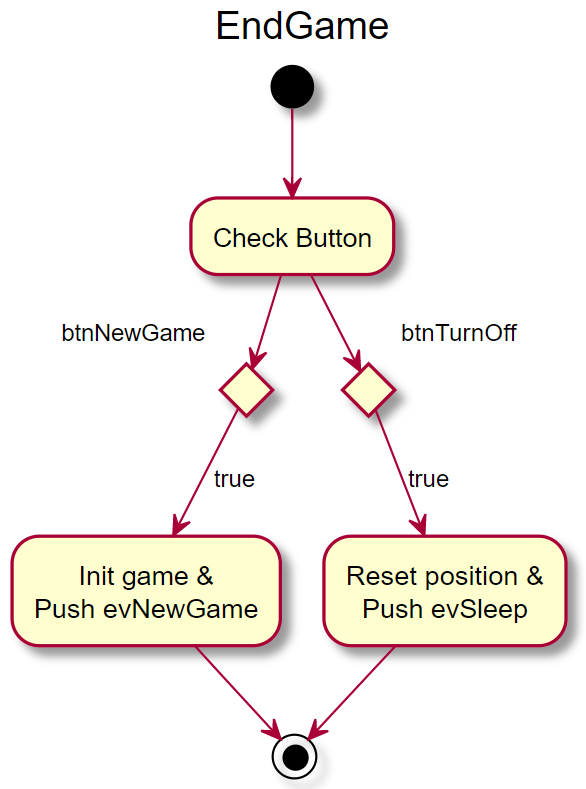
1. Un timer qui s’effectue chaque 10,20 ou 30 ms en fonction du niveau de difficulté va nous permettre de déplacer la balle ainsi que de déplacer le paddle IA. Pour le déplacement de la balle une méthode permet de vérifier toutes les collisions du jeu (mur, paddle, défaite).
2. A chaque fois qu’un calcul de position est effectué (voir figure 14), on va déplacer notre paddle.
3. Enfin, un bouton permettant de quitter le jeu et de retourner au menu principal a été implémenté.

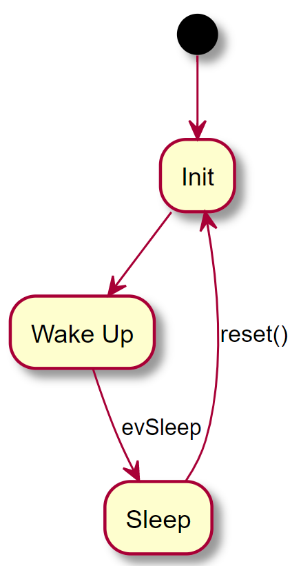
Figure 19 gameController: action to do on LOCAL state



Cette partie concerne l’état ENDGAME. Cette état est actif lorsqu’un des deux joueurs a perdu la partie. Il permet un retour au menu principal ou bien tout s’implement d’éteindre le jeu.

Figure 20 gameController: action to do on ENDGAME state

## Sleep StateMachine

Cette machine d’état permet d’éteindre notre jeu, grâce à la méthode Sleep() du pic18, lorsqu’on a pas effectué une pression sur l’écran depuis 30 sec.

La méthode Reset(), permet quant à elle de redémarrer complètement notre jeu.

Pour redémarrer notre jeu il suffit d’effectuer une pression sur l’écran car en mode sleep les interruptions sont toujours active. Une pression sur l’écran va donc continuer à générer des interruptions. Une fois l’interruption généré, le pic va tout d’abord effectuer la ligne de code suivant celle où il s’était arrêté/endormi avant d’effectuer le code dans l’interruption.

En mettant donc la ligne Sleep() suivie de la ligne Reset(), on peut éteindre/endormir notre jeu et le réveiller grâce à une pression sur l’écran

Figure 21 sleep StateMachine

# Tests et mise en service

Voici ci-dessous le récapitulatif des tests effectués sur notre projet tout au long de notre travail.

|  |  |
| --- | --- |
| **Récapitulation des tests** |  |
|  |  |
| **Test ID** | **Description** |
| Hardware 01 | Ce test permet de vérfier l'état du PCB après la fabrication |
| Hardware 02 | Ce test permet de vérifier si le PCB n'a pas de court-circuit |
| Hardware 03 | Ce test permet de vérifier la protection contre l'inversion de polarité |
| Software 01 | Ce test permet de vérifier si MPLAB reconnait le PIC |
| Software 02 | Test du Menu Welcome |
| Software 03 | Test du menu parametre |
| Software 04 | Test du sleep |
| Software 05 | Test du menu 1 player |
| Software 06 | Test du menu endgame |
| Software 07 | Test augmentation vitesse avec niveau de difficulté |
| Software 08 | Test mode sleep en jeu |

# Conclusion

## Analyse des résultats

## Conclusion

# Annexe