01/06/2021

Rapport du projet de fin de semestre
Microcontrôleurs

Finette Loïc – 296932
Amor Mehdi – 296831

I. Introduction

Ce projet consiste en l'acquisition de connaissances approfondies en ce qui concerne les microcontrôleurs et leur développement dans le cadre d'un travail en groupe. Son support principal est la MCU Atmega128L qui permet l'utilisation d'autres périphériques divers (Détecteur de distance, EEPROM, etc..), dont un obligatoire étant le capteur de température 1-Wire. L'application complète est développée en assembleur AVR et doit respecter des concepts de base comme la modularité et la responsivité. Parmi les objectifs de ce projet se trouve l'étude d'un périphérique et son protocole de communication, ainsi que son interfaçage avec un microcontrôleur et son implémentation, sans oublier l'application des notions apprises en cours théorique comme les interruptions, les macros et sous-routines, traitement input/output, etc…

II. Description générale de l'application

Notre application est un jeu interactif simulant un animal de compagnie virtuel. Il donne la possibilité au joueur d'être maître d'un chien avec lequel il peut interagir grâce aux interfaces tels que le capteur de température, le buzzer et l'écran d'affichage LCD de la MCU. Le jeu commence avec un chien qu'il faut alors élever grâce aux différents outils cités plus haut ainsi qu'un portefeuille à zéro (la monnaie étant des VetterCoins). Le joueur envoie le chien à différent endroit à l'EPFL avec pour but que le chien trouve et ramène des VetterCoins (VC). On remporte le jeu si l'on obtient 255 VC ou plus. Le chien a trois stats étant les points de vie (HP), la défense (DEF), et l'amour (L). On peut perdre si le stat HP ou le stat L est à 0. Le joueur contrôlera le jeu affiché sur le LCD à l'aide des boutons poussoirs. Le menu principal contient deux possibilités, tout d'abord, il y a une boutique (SHOP) où il est possible d'acheter de la nourriture pour restaurer les HP, ou des armures pour augmenter la DEF. La deuxième

option du menu principal est d'envoyer (SEND) le chien en aventure à cinq différents endroits (STCC, ART, BS, INNOV, UNIL). L'envoie du chien à un endroit est simulée par un compte à rebours. En revenant d'une aventure, le chien aura perdu un nombre de points de vie et d'amour aléatoire Aussi, il aura ramené un nombre de VC aléatoire. Le maître peut « donner » son amour au chien en le caressant à travers le capteur de température et faire remonter le stat L. En effet, lorsque le maître place sa main sur le capteur de température et que la température augmente, le stat de L remonte. Les endroits sont listés en ordre de difficulté. Alors le STCC est l'endroit le plus facile, donc l'endroit où l'aventure va prendre le moins de temps et où le chien va recevoir le moins de dégâts et de VC. Tandis que l'UNIL est l'endroit le plus difficile. Donc l'endroit où l'aventure dure le plus longtemps et où l'on va recevoir le plus de dégâts et de VC. Afin de créer ce jeu, nous nous sommes inspirés des DAPPs (Decentralized apps) tel que Alien Worlds qui permettent aux joueurs d'obtenir de la crypto en jouant.

III. Mode d'emploi

Bouton 0 : CYCLE, permet de déplacer le curseur pour sélectionner les options
Bouton 1 : OK, confirme la sélection
Bouton 2 : BACK, permet de retourner au menu principal
Bouton 3 : RESET, fait un reset global du jeu

Le jeu s'initialise au menu principal et on débute avec le stat HP à 12, le stat DEF à 10 et le stat L à 10. En sélectionnant l'option SEND à l'aide du bouton OK on peut choisir un des cinq endroits suivant où envoyer le chien:

Endroit	**Temps d'attente (s)**	**Dégâts (DMG)**	**VC obtenables**
STCC	15	10 - 15	1 - 11
ART	22	12 - 17	3 - 13
BS	33	14 - 19	5 - 15
INNOV	40	18 - 23	9 - 19
UNIL	50	22 - 27	15 - 25

Tableau d'endroits

Après le temps d'attente pour l'endroit écoulé, le buzzer joue une mélodie indiquant que le chien est de retour. Ensuite, le joueur doit appuyer sur le bouton OK pour au moins deux secondes et relâcher. Cette action arrête le buzzer et génère aléatoirement un nombre de dégâts et de VC en respectant le tableau ci-dessus. Puis, les VC obtenus sont directement ajoutés au portefeuille. Les stats de HP et de L sont modifiés selon le pseudo-code suivant:

$$trueDmg \leftarrow DMG - DEF$$
$$Si\ trueDmg > 0$$
$$HP \leftarrow HP - trueDmg$$
$$L \leftarrow L - \frac{trueDmg}{2}$$

La division pour le stat L étant une division entière. Ensuite le joueur retourne au menu principal. Pour remonter son stat L, le joueur peut tout simplement placer sa main sur le capteur de température et attendre quelques secondes pour que la température monte et le stat augmente jusqu'à sa valeur maximale de 10. Si le joueur veut remonter son HP ou bien augmenter sa DEF, il doit naviguer à l'option SHOP. Les items à gauche de l'écran restaurent les points de vie et les items à droite augmentent la défense. Alors pour remonter l'HP, on peut soit acheter de la viande (MEAT) pour obtenir +3 HP ou une friandise (TRT) pour en obtenir +6. Pour la défense, le joueur peut se procurer de l'armure (ARM) pour avoir +1 DEF ou de l'armure élite (EARM) pour obtenir +4 DEF. Si le joueur n'a pas assez de VC pour acheter l'item qu'il désire, le programme affiche un message indiquant que l'achat n'a pas été effectué. Dans le cas où le joueur a suffisamment de VC, un message de confirmation est affiché sur le LCD et les stats sont modifiés. Si le stat de L ou d'HP se retrouve à 0, le joueur perd. Alors, le buzzer joue une mélodie et un message indiquant que le joueur a perdu s'affiche. Si le joueur obtient 255 VC ou plus, une mélodie différente est jouée et un message de victoire apparaît sur le LCD. Pour recommencer le jeu dans ces deux cas, il faut tout simplement appuyer sur le bouton RESET (Voir annexe pour le déroulement du jeu).

IV. Description technique de l'application et du matériel

Nous utilisons trois périphériques pour l'application : Le buzzer branché au port B, le capteur de température 1-wire branché au port E et l'affichage LCD branché au port approprié. De plus, nous utilisons les boutons poussoirs SW0 - SW3 pour que le joueur puisse interagir avec le jeu. Comme mentionné plus haut, le capteur de température est utilisé pour gérer le stat d'affection. Le buzzer est utilisé pour avertir le joueur que le chien est revenu de son aventure et qu'il est temps d'obtenir les VC. Ce périphérique est aussi utilisé lorsque le joueur gagne ou perd. Le LCD Hitachi44780U est le cœur du jeu. C'est sur cet affichage que l'on peut observer le déroulement du jeu. Les boutons poussoirs sont employés pour déplacer le curseur, sélectionner les différentes options, revenir en arrière, obtenir les VC et effectuer un reset global. Pour effectuer ce reset, nous utilisons une interruption. En effet, lorsque le joueur appuis sur le bouton poussoir SW3, on exécute l'instruction située à l'adresse 0x08. Cette instruction fait appel à la routine d'interruption *ext_int3*. Dans cette routine, une instruction *jmp reset* est placée pour effectuer un reset global. Aussi, nous utilisons une interruption pour effectuer le décompte lorsqu'on envoie le chien. Quand un endroit est sélectionné, le registre r22 est attribué la valeur du temps d'attente approprié (cf. *Tableau d'endroits*) et le timer 0 est enclenché avec un facteur de pré-division CS = 5. Avec ce facteur, l'overflow qui exécute l'interruption est réalisé à chaque seconde ce qui permet d'effectuer un décompte du registre r22 en seconde. De plus, le timer 0 est employé pour générer deux nombres aléatoires : les VC obtenus de l'aventure et les dégâts reçus. Lorsque le registre r22 est égale à 0, on change le facteur de prédivision CS à 2 pour effectuer l'overflow à chaque 0.0625 secondes. Cette

interruption incrémente en boucle le registre r24, qui représente les dégâts, en respectant les conditions (cf. *Tableau d'endroits*) Puis, quand le joueur appuis sur le bouton OK, la boucle de r24 est arrêtée, laissant la dernière valeur enregistrée dans ce registre. L'appui du bouton enclenche une deuxième boucle, cette fois avec r23, qui représente le nombre de VC. De la même manière quand le joueur relâche le bouton OK, la boucle s'arrête et la dernière valeur est enregistrée dans r23. Ces deux valeurs sont ensuite affichées sur le LCD. Un merci à Claude Dahan pour cette idée pour générer des valeurs aléatoires.

V. Fonctionnement du programme

Veuillez trouver en annexe le schéma Top-Down du programme. De cette approche, on peut observer trois grands modules : le module "Main Menu", le module "Game Logic" et le module "Memory". Les macros et sous-routines les plus importantes se retrouvent dans le module "Game Logic" qui agit en tant que moteur de jeu.

VI. Présentation des modules

Module "Memory"

Le module "Memory" porte le nom de "SavingVar.asm" dans les fichiers du code. Puisqu'il est nécessaire de constamment mettre à jour les variables, nous avons ajouté un module qui stocke et lit des valeurs de la SRAM. On peut observer dans la "main.asm" que nous avons réservé au début du programme 5 bytes. Les 4 premiers bytes sont réservés pour les variables visibles sur le LCD, donc les variables HP, DEF, L et WALLET. Le dernier byte est réservé pour la température de référence TEMP (élaboré dans la section VII). Dans ce module "SavingVar.asm", la macro LOADSTAT prend en argument une adresse de la SRAM et en utilisant le pointeur x, place la valeur lu à l'adresse de l'argument dans le registre r19, que l'on utilise comme registre de travail. La macro SAVESTAT prend elle aussi une adresse de la SRAM en argument. En employant le pointeur z à l'adresse en argument, on stocke sur la SRAM le registre r19. Grâce à ces deux macros, nous sommes en mesure d'accéder à un des 5 bytes, le modifier et l'enregistrer sur la SRAM. Ce module contient aussi toutes les initialisations des valeurs de bases tels que le minimum de VC obtenu pour un endroit, le nombre de HP de base, etc… Ceci a été implémenté pour facilement changer les valeurs initiales et donc, mieux équilibrer le jeu.

Module Main Menu

Ce module comprend deux grandes sous-catégories "Display" et "Navigate". Le sous-module "Display" consiste à afficher le jeu sur le LCD et sera expliqué dans la section VII. Le sous-module "Navigate" permet au joueur de sélectionner les différentes options dans les différents menus. La plupart des fonctionnalités de ce sous-module se trouvent dans le code "cycle.asm". Les trois premières lignes de ce fichier réservent des octets pour les différentes positions du curseur pour les différents menus. Par exemple, pour menuArr qui contient les positions pour le menu principal, le curseur peut seulement se déplacer à la position 0x04 et

0x0a, qui sont les positions à côté de “SEND” et “SHOP” respectivement. Le changement des positions du curseur se produit dans la sous-routine cycle. Avant d’utiliser cette sous-routine, on définit le registre r28 au nombre de positions possibles pour le menu actuel. Ensuite, la sous-routine est appelée et on initialise r27 à 0 et le pointeur z à la première position du menu. Le registre r27 garde un œil sur la position actuelle du curseur. Par la suite, grâce aux instructions *lpm*, *mov a0, r0* et *rcall LCD_pos*, on déplace le curseur à la première position. Lorsque le joueur appuie sur le bouton poussoir SW0, on augmente de 1 la valeur de r27 et de z. Alors, maintenant le pointeur pointe à la deuxième position. Si la valeur de r27 est égale à la valeur de r28, cela veut dire qu’on est arrivé à la dernière position du menu et on remet le curseur à sa position initiale et r27 à 0.

Module Game Logic

Les responsabilités principales de ce module sont d’envoyer le chien en aventure, de mettre à jour les stats, de vérifier si le joueur a gagné ou perdu et l’achat des items. On le retrouve dans le code “Logic.asm” et “Timer.asm”. Quand on envoie le chien, la sous-routine tick_tock de “Timer.asm” est appelée. Comme mentionné dans la section IV, le décompte et la génération aléatoire des nombres sont effectués à l’aide d’interruptions. Puis, la sous-routine get_new_stat de “Logic.asm” est appelée. Cette sous-routine utilise la macro LOADSTAT pour obtenir les valeurs de HP, DEF et L et effectue les vérifications et les manipulations mentionnées dans le pseudo-code de la section III. De plus, cette sous-routine vérifie si le joueur a perdu en comparant le nombre de dégâts reçus et les HP avec l’instruction *cp r19, r24* (r19 = HP, r24 = dégâts). Si r24 est plus grand que r19 on effectue un branchement à l’étiquette “game_over” qui va indiquer au joueur qu’il a perdu. Ce même branchement est effectué de la même manière avec le stat L. Donc si le stat L est à 0 on branche à game_over. La dernière fonction de cette sous-routine est d’ajouter les VC à la variable WALLET. Ceci est réalisé en créant une boucle *for* qui incrémente à chaque itération la valeur de WALLET. Le nombre d’itérations est égal aux nombre de VC obtenu aléatoirement. On incrémente en utilisant l’instruction *INC_LIM r19, 255* pour que le stat WALLET ait comme valeur maximale 255. La boucle contient une condition qui vérifie si 255 VC ont bien été obtenus, et si oui, effectue un branchement à l’étiquette winner. La dernière grande responsabilité de ce module est traitée dans la sous-routine proccess_trans qui est appelée lorsque le joueur essaye d’acheter un item. Le code de l’item que désire acheter le joueur est enregistré dans le registre r22. On associe 0 à meat, 1 à l’armure, 2 à treat et 3 à l’armure élite. En utilisant la macro if_far qui agit comme une instruction *if (@0 == @1) then go to @2* on peut aisément effectuer un branchement à l’étiquette de l’item que désire acheter le joueur.

```
if_far r22, 0, meat
if_far r22, 1, armour
if_far r22, 2, treat
if_far r22, 3, elite
```

Ensuite la macro BUY est appelé qui vérifie si le joueur a assez d'argent pour acheter l'item voulu. Dans le cas contraire, un message indiquant que le solde n'est pas suffisant s'affiche et la macro EXIT_SHOP réalise les pops nécessaires et un ret. Dans le cas où l'achat est possible, un message de confirmation est affiché et la macro ADD_STAT est exécutée. Cette macro, augmente le stat associé à l'item jusqu'à une valeur maximale. Les valeurs maximales sont les suivantes: HP_MAX = 12, DEF_MAX = 20.

VII. Description de détail de l'accès aux périphériques

Hitachi44780U

Le LCD est sans aucun doute le périphérique le plus important de notre application. Pour l'utiliser nous avons importé les fichiers "lcd.asm" et "printf.asm". Le module "Display" se trouve dans les codes "main.asm" et "Logic.asm". Le périphérique est initialisé grâce à la sous-routine *LCD_init* dans le reset. La sous-routine met les bits de SRE et SRW10 à 1, efface ce qui est affiché et active le mode deux lignes. Puis, dans la main, en début de programme, tout ce qui est affiché est effacé avec *LCD_clear* et le curseur retourne à sa position initiale grâce à *LCD_home.* On utilise *LCD_clear* car la main tourne en boucle et l'affichage doit fréquemment être mis à jour. Le développement du module "Display" a commencé en affichant les strings constants. L'instruction *PRINTF LCD* permet cela. Puis, afin de facilement écrire à n'importe quel endroit du LCD, nous avons créé une macro LCD_pos_cursor qui prend en argument une position et déplace le curseur à cette position avec l'instruction *LCD_pos*. Nous voulons faire en sorte que le joueur puisse voir le curseur et l'utiliser pour sélectionner les divers options. Alors, nous avons ajouté l'instruction *LCD_blink_on.* La position du curseur est mise à jour dans la sous-routine cycle. Puisque le programme doit constamment mettre à jour les informations, un *WAIT_MS 100* se trouve entre chaque *LCD_clear* pour avoir une bonne fréquence de rafraîchissement. Afin d'afficher les divers stats, la sous-routine write_stats_to_LCD est appelée. Pour chaque stat, la sous-routine exécute la macro LOADSTAT pour charger r19 avec la valeur du stat. Ensuite, la macro PRINTSTAT prenant comme argument une position, imprime le registre r19 à la position donnée. Une technique similaire est employée lorsqu'il faut afficher les prix des items dans le shop. La sous-routine show_prices est exécutée qui, pour chaque item, utilise la macro PRICE_PRINT prenant en argument un prix et une position pour par la suite afficher ce prix à la position donnée.

Buzzer

Le buzzer est employé pour annoncer le retour du chien et jouer une mélodie quand le joueur perd ou gagne. Ce périphérique est employé dans les codes "Logic.asm" et "Timer.asm". Pour utiliser le buzzer, on a implémenté des tableaux de notes en utilisant les notes mises à disposition dans le fichier "sound.asm" et en réservant des octets pour toutes ces notes. Quand on veut jouer une mélodie, on initialise le pointeur z à la première valeur du tableau et on stocke cette valeur dans r0 grâce à *lpm*. Ensuite, le pointeur z est incrémenté d'une unité afin

qu’il pointe vers la prochaine note de la séquence. Puis, on vérifie si r0 est égal à 0. Si oui, nous sommes arrivés à la fin du tableau et l’on peut effectuer un branchement pour réinitialiser le pointeur z à la première note dans le tableau pour jouer la mélodie en boucle. Si non, on place la valeur de r0 dans a0 et on initialise la durée d’une note dans le registre b0. On emploie ces deux registres car ce sont les registres de travail de la sous-routine sound, qui fait jouer le son. Dans cette sous-routine, le pin 2 du port E est rapidement inversé pour produire un son.

Capteur de température

Comme mentionné précédemment, le capteur de température est utilisé pour remonter le stat d’affection L. Ce module “Check Temp” est réalisé grâce au code “Temperature.asm” et le fichier “wire1.asm”. La macro READ_TEMP du fichier “Temperature.asm” est la macro permettant de lire la température. La macro fonctionne en envoyant tout d’abord un pulse de reset. Puis grâce à la commande skipROM, on indique au master que l’on a seulement un seul slave (le capteur de température) sur la ligne. La conversion de la température est ensuite enclenché avec convertT. En lisant les bits temporaires du readScratchpad et avec l’appel de wire1_read, on peut stocker les bits représentant la température lue par le capteur dans les registres a0 et a1. Au début de la main, la sous-routine get_init_temp est appelée. Elle a pour but d’effectuer trois lectures de la température ambiante et d’enregistrer la dernière valeur lue plus 4 dans la variable TEMP. Cette température va être utilisée comme température de référence. On effectue trois lectures pour essayer de minimiser l’erreur. De la datasheet on peut voir qu’ajouter +4 au LSB est équivalent à ajouter +0.25 degré Celsius. On prend cette valeur comme référence car on veut seulement que le stat L augmente quand il y a un assez grand changement de température (quand le joueur place sa main sur le capteur) et non quand la température ambiante fluctue légèrement. Par après, quand la sous-routine update_love est appelée le stat d’affection est chargé dans r23 grâce à LOADSTAT LOVE et *mov r23, r19* et la température de référence est chargée dans r24 grâce à LOADSTAT TEMP et *mov r24, r19.* Puis, on appelle la macro READ_TEMP et l’on va comparer la valeur de a0 et la température de référence dans r24. Si a0 est plus grand, on augmente r23 jusqu'à sa valeur limite de 10. À chaque incrémentation, les instructions *mov r19, r23* et SAVESTAT LOVE sont utilisés pour mettre à jour le stat d’affection. À plusieurs reprises lors de l’exécution du programme, un branchement est effectué à soft_r qui appelle la sous-routine get_init_temp. Ceci a été fait pour souvent mettre à jour la température de référence et réduire le risque que le capteur de température ne fonctionne pas correctement.

ANNEXES

Déroulement du jeu

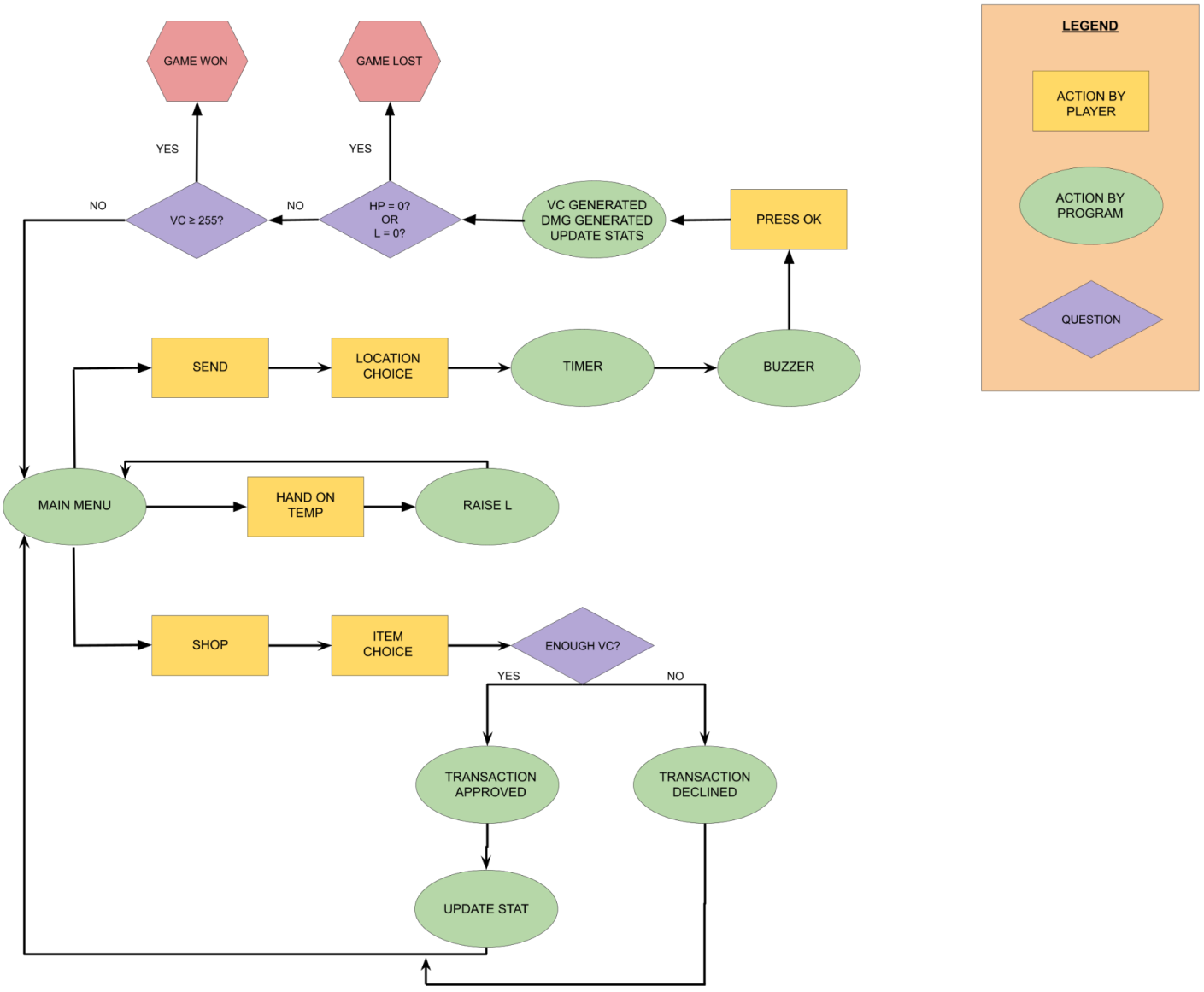

Schéma Top-Down