Rapport final de Projet

Conception d'une nouvelle loi de contrôle pour tablette tactile

Soutenu par :
Pierre Fitoussi

Organisme :
L2EP - CRISTAL

Encadrant Polytech
Thomas Vantroys

Tuteur de projet:
Laurent Grisoni

Sommaire

Introduction

Le projet E-Vita s'inscrit dans la volonté de rendre plus interactif la technologie utilisée au quotidien. Depuis des années, les chercheurs réfléchissent et conçoivent différents moyens de contrôle de ces technologies.

On notera l'omniprésence de l'écran tactile qui est désormais le moyen de contrôle le plus répandu pour l'usage des smartphones, tablettes et ordinateurs.

C'est dans cette optique que la collaboration des laboratoires de recherche L2EP et CRISTAL développent depuis 2003 E-Vita, une technologie qui utilise des céramiques piézo-électriques pour qu'une fois rentrées en oscillation, la résistance de l'écran sous le doigt varie et perturbe notre sensation de toucher..

Cahier des charges

Le but de ce projet est de s'inspirer des différents programmes déjà développés sur cette technologie, afin de comprendre les besoins d'un développeur et de lui fournir une librairie qui simplifie et unifie les différentes fonctions nécessaires pour utilisé le retour tactile.

Développer une application qui reprend ces principes et mets en lumière les diverses fonctionnalités précédemment décrites.

Présentation général

Le système est en constante évolution, reprenons le tel qu'il était au début de ma mission.

Matériel

E-Vita est une tablette tactile composée des éléments suivants :

Banana Pro

Écran tactile

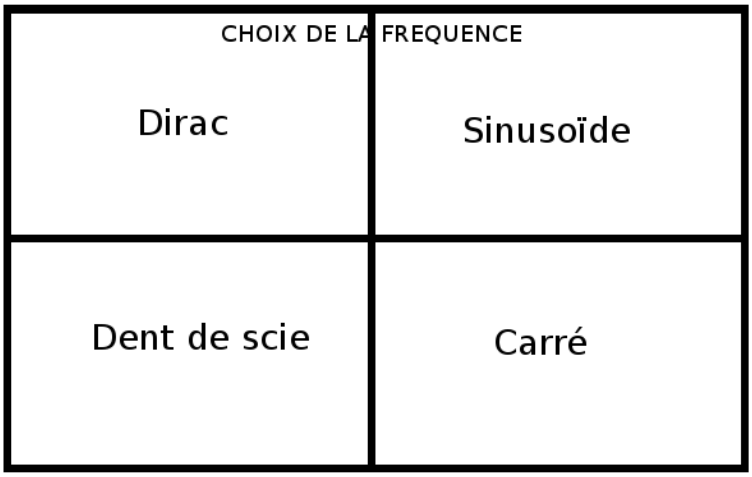
Carte contrôleur dédiée aux céramiques

Ces différentes couches communiquent entre elles sur la couche bas-niveau via différents protocoles, ainsi l'écran est contrôlé via I2C, le périphérique pilotant les céramiques quant à lui est contrôlé via SPI et le lien entre le code haut niveau et bas niveau se fait grâce au protocole de communication OSC.

Logiciel

Le code bas niveau est rédigé en C est fait donc le lien entre les divers équipements reliés.

La première présentation que l'on m'ait montré, divisait l'écran tactile en 4 parties. Ces 4 parties génèrent une forme d'onde sous le doigt grâce à la vibration de l'écran. L'écran se divisait ainsi



Afin de ressentir ces signaux sous le doigt, les motifs sont enregistrés « en dur » dans le programme C.

Sachant que plusieurs fréquences de vibrations était disponible pour chacun des 4 signaux, disons de base, cela représentait plus de 25 fichiers d'en-têtes codés dans la ROM du système.

Il y avait donc là un manque à gagner en optimisation de mémoire.

Ces fichiers étaient représentés sous la forme d'un tableau d'entier qui dénombrait pas moins de 10000 cases.

Sur le Banana Pro, un entier occupe 4 octets de mémoire, sachant que la tablette pouvait générer 4 signaux différents :

Dirac, Sinusoïde, Carré et Dent de scie

Il est possible de choisir entre 6 fréquences de vibrations, cela représente une occupation en mémoire de :

$$taille\,du\,tableau * taille\,d'un\,entier * nbr\,signaux * nbr\,de\,fréquence$$

Soit un total de 960ko d'espace mémoire occupé.

Un tableau est donc composé de 10000 entrées et représente la variation d'amplitude d'un signal.

Exemple : le tableau composant le signal carré de fréquence spatiale 50 est formé ainsi :

t[0] → t[25] = 0

T[26] → T[50] = 30

T[51] → T[75] = 0

…

L'amplitude allant de -30 à 30 pour les besoins de l'application.

Le programme principale est composé de la routine suivante :

Initialisation des variables spatiales

Initialisation du serveur/client local pour la communication OSC

Initialisation de la communication I2C relié le µP à l'écran.

Choix de l'entête à utiliser (Signal, fréquence)

Récupération des informations sur le déplacement du doigt (position, vitesse)

Dû au différentes fréquences de rafraîchissement du système complet, il est nécessaire d'utiliser un minimum de prédiction quant à la vitesse et la position du doigt.

En effet l'écran tactile à une fréquence de rafraîchissement de 60Hz.

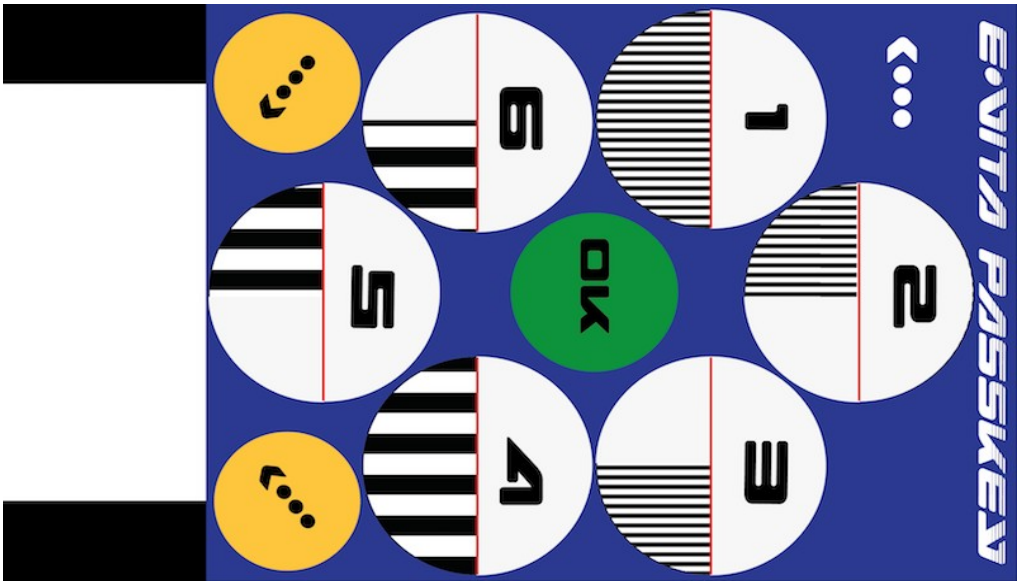
Alors que notre carte pilotant les céramiques peut monter jusqu'à 10KHz.

Animation pour application de déverrouillage

Descriptif

Un des points essentiel dans l'interface homme-machine est que la machine a besoin de se faire comprendre par utilisateur.

L'application PassKey, est une application qui permet d’enregistrer un code de verrouillage et de tenter de le déverrouiller.



Le code lui est formé de 4 motifs tactiles.

Sont présents sur la photo ci-dessus différents éléments a comprendre :

On compte six bulles numéroté de 1 à 6 contenant deux informations :

Un chiffre

Un motif tactile (représenté par les zébrures noires)

Les flèches jaunes permettent de changer la sélection en cours

Zone blanche : retour tactile

Dans la première phase de l'application l'utilisateur doit renseigné le code de déverrouillage. La zone blanche dans la partie inférieur de l'application se trouve le motif tactile correspondant à la bulle en cours de sélection.

Il n'y a pas moyen de savoir quelle bulle est sélectionnée hormis de comparer le motif présent dans la zone blanche et celui présent dans une des bulles.

L'utilisateur peut ainsi glisser son doigt dans cette zone est reconnaître le motif tactile qui y est présent.

Il valide son choix de premier motif en appuyant sur 'OK'. Il continue ainsi de suite afin d’enregistrer un code à 4 chiffres.

Afin que l'utilisateur se rende mieux compte du motif en cours de sélection, il m'a été demandé de réalisé une animation d'une bague qui effectue une rotation dans le sens indiqué par les flèches.

L'application étant développé sur Processing, la documentation général fut suffisante pour créer cette animation. Avant de mieux comprendre les fonctionnalité de Processing, j'ai réalisé une animation qui pointé un segment depuis le centre du bouton OK vers la bulle en sélection.

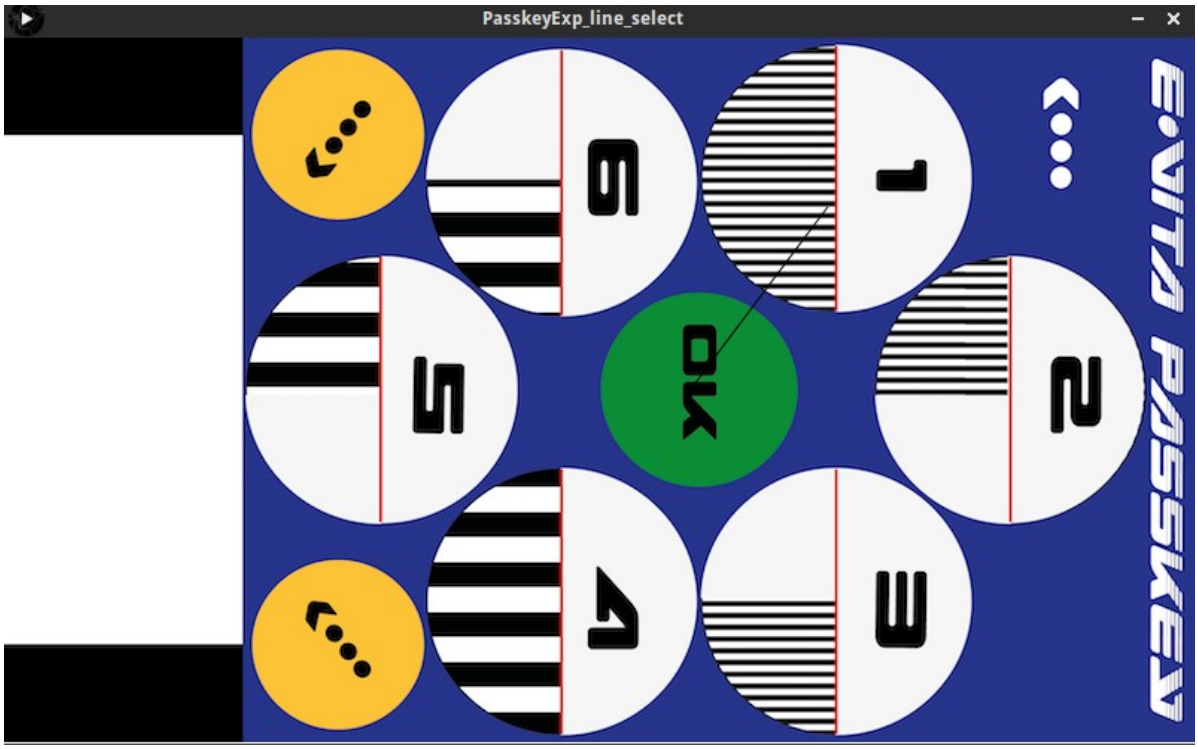


Illustration 1: Utilisation d'une barre de sélection

Cela permet à l'utilisateur d'avoir une représentation physique de quel motif est inscrit dans la zone blanche. Seulement en implémentant cette solution, on perd toute la sécurité que nous offre le retour tactile → ne pas afficher la sélection.

Solution répondant aux critère.

Afin d'utiliser au mieux le retour tactile, l'animation sera composé d'une bague circulaire et de créneau, comparable à un gouvernail de bateau.

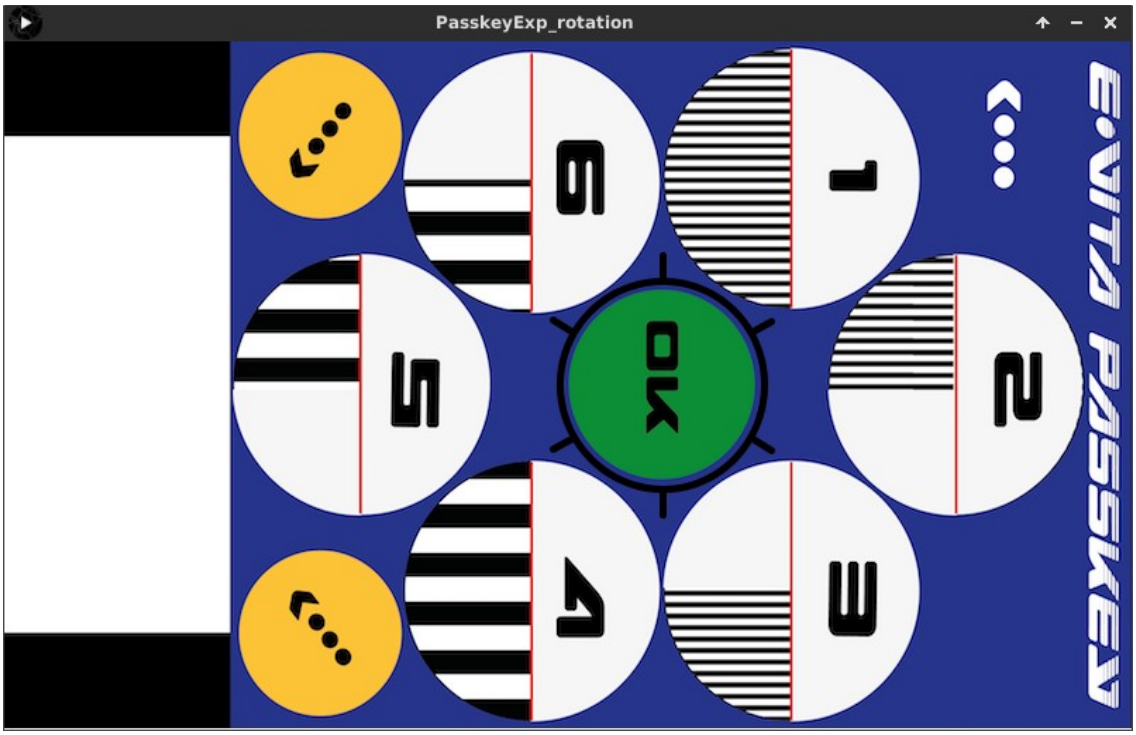


Illustration 2: Réalisation d'un gouvernail, tourne en fonction de l'appuie sur les flèches

Ce gouvernail rentre en rotation, lors d'un appui sur les flèches jaunes, il permet donc à l'utilisateur de se repérer dans la phase de déverrouillage. En effet, un fois que l'utilisateur à reconnu le motif afficher dans la zone blanche, il repère la bulle correspondante et peu changer de motif, tout en visualisant une animation qui lui permet de comprendre vers quelle bulles la sélection se dirige.

Pour ce faire j'ai implémenté dans la fonction draw() le code ci-dessous :

```
if (!creatingCode){
  translate(240, 333);
  noFill();
  strokeWeight(5);
  ellipseMode(CENTER);
  ellipse(0,0,145,145);
  if (frameCount < nb_frame +60){
     phi = direction * ((nb_frame-frameCount)/60.) * PI/3;
  }else{
    direction =0;
  }
  //Creation des segments
  for (int i=0; i<6; i++){
    pushMatrix();
    rotate(phi + i*PI/3);
    line(75,0,90,0);
    popMatrix();
  }
}
```

Illustration 3: Fonction d'affichage du gouvernail

Cette portion de code s'occupe de d'afficher un cercle autour du bouton OK.

La variable *direction* est modifié à chaque appuie sur une des deux flèches jaunes. Elle incrémente remise à zéro à 0 à chaque fin de rotation.

La rotation est rendu possible grâce au combo pushMatrix()/popMatrix() qui me permet d'afficher 6 segments séparés d'un angle de pi/3. La variable phi correspond à la phase du premier segment, c'est cet élément qui réalise réellement la rotation.

Le **frameRate** étant fixé à 60, la rotation d'un cran à l'autre mets 1 seconde à se réalisé.

Librairie

Afin de rendre plus claire et simple la création d'application utilisant le retour tactile. Il m'a été demandé en me servant des programmes développéx par l'équipe MINT, de réaliser une librairie pour qu'un futur développeur ait à sa disposition un panel de fonctions lui permettant d'utiliser le retour tactile.

Pour ce faire, j"ai récupéré le fichier TextureControl.cpp ainsi que son header associé qui m'a servi de base pour la création de la librairie qui se compose de 2 fichiers *libTextureControl.c* et *libTextureControl.h*

Pour adapté ce programme en librairie, il a fallut modifié le header. En effet, en C, pour permettre la bonne utilisation d'une librairie il faut lister les prototypes des méthodes publique de cette dernière dans l'en-tête «libTextureControl.h ». (Annexe A)

Les méthodes inscrites dans le header permettent de :

startEvita()/stopEvita() : Début/Fin utilisation de la librairie

setTaxtel*() : Définie une zone sur l'écran, ou sera présent un motif vibratoire.

setTexture*() : Définie un motif vibratoire

enable/disableTaxTel() : Active/Désactive la vibration pour un TaxTel donné.

Les fonction callback permettent, si elle sont appelées avant startEvita(), de redéfinir les actions à mener lors de l’interaction tactile

get*() : Méthode de récupération des différentes informations lors de l'utilisation de l'écran tactile.

Méthode d’accès publiques

Ces méthodes sont utilisées si l'utilisateur ne souhaite pas redéfinir la méthode d'entré de base.

9 fonctions ont été crées, afin de récupérer et de traiter les informations de position du doigt ainsi que les états possibles (Appuie/Relâche/Glisse).

Ces fonctions permettent un accès direct aux différentes variables et pourrons donc être utilisées dans un langage de plus haut niveau L'idée derrière tout cela est bien entendu d'arriver à développer cette technologie de retour tactile sur des systèmes embarqués Android..

Fonction de Callback

En réalisant cette bibliothèque, je me suis rendu compte qu'un développeur aura besoin de définir différents moyens de contrôle. En effet, le développeur peut à sa guise redéfinir les méthodes de contrôle de l'application. Il peut soit utiliser l'écran tactile, soit utilisé le protocole OSC dans le but d'une utilisation de la librairie en mode serveur, ou tout simplement vouloir utiliser sa souris pour remplacer le doigt.

Pour ce faire, l'utilisateur passe en paramètre de ces fonctions ces propres méthodes d'acquisitions.

Si ces fonctions ne sont pas redéfinies avant l'appel à startEvita(), le programme utilise de base le tactile et l'utilisateur devra se servir des méthodes d'accès get*() par exemple lors de l’utilisation de la bibliothèque avec un langage de plus haut niveau.

Ces fonctions de callback vont venir redéfinir dans la librairie les méthodes d'acquisition de la position du doigt, du calcul de la vitesse et de la gestion des états possibles du doigt sur l'écran.

Mutex

Afin de garder en mémoire les différentes textures et taxtel (comprendre zone ou se situe un motif tactile), des structures on été créées.

Ces structures sont modifiés grâce aux fonctions setTexture(), setTaxtel(), enableTaxTel(), disableTaxTel().

La fonction MyISR(), est quant à elle appelée toute les 18ms. Cette fonction est un thread qui permet de détecter la présence du doigt sur l'écran tactile et viens parcourir les tableaux de structure afin de déterminer si le doigt est présent dans un TaxTel, et d 'envoyer l'ordre de vibration au céramiques.

Afin d'éviter les conflits d'accès aux ressources, un mutex est verrouillé dès qu'une utilisation des structures est envisagé.

En parcourant le code du programme on remarque que chaque accès au tableau de structure TaxTel ou Texture est entouré d'un verrou:

```
void setTextureCos(int textureId,float offset,float amplitude,float period, char* speedFunc){
#ifdef DEBUG
    printf("Texture: %i, Type: cos, offset= %f, amplitude = %f, period = %f\n", textureId, offs
#endif
    pthread_mutex_lock(&mutex);
    if (textureId<MAXTEXTURES){
        Textures[textureId].UsAmplFun = UsCosine;
        Textures[textureId].offs = offset;
        Textures[textureId].ampl = amplitude;
        Textures[textureId].sp = period;
        if (strcmp(speedFunc, "vertical") == 0)
            Textures[textureId].UsAngleSpeedFun = VertASpeed;
        else
            if (strcmp(speedFunc, "horizontal") == 0)
                Textures[textureId].UsAngleSpeedFun = HorizASpeed;
            else
                Textures[textureId].UsAngleSpeedFun = LineicASpeed;
    }
    pthread_mutex_unlock(&mutex);
}
```

Illustration 4: Exemple d'utilisation du mutex

Makefile

Afin de compiler cette librairie facilement, j'ai réalisé un Makefile qui automatise, la compilation de la librairie soit en statique soit en dynamique.

Une option est également disponible pour « installer » la librairie, cette fonction va venir copier les fichier .a ou .so ainsi que le fichier .h de la librairie dans les dossier /usr/lib/ et /usr/include.

```
1  CC=gcc
2  AR=ar
3  EXEC=main
4  SOURCE=libTextureControl.c
5  OBJ=libTextureControl.o
6  STATIC=libTextureControl.a
7  DYNAMIC=libTextureControl.so
8  HEAD=libTextureControl.h
9
10
11 clean:
12     rm -rf *.o *.a *.so *.out
13
14 dynamic:$(DYNAMIC)
15     $(CC) main.c -L/home/demo/Documents/pfitouss/ -I/home/demo/Documents/pfitouss/ -lTextureControl -lwiringPi -lpthread -llo -lm
16
17 $(DYNAMIC):$(OBJ)
18     gcc -shared -o $(DYNAMIC) $(OBJ)
19
20 $(OBJ):$(SOURCE)
21     gcc -c -fPIC $(SOURCE)
22
23 install:
24     sudo mv $(STATIC) /usr/lib
25     sudo mv $(DYNAMIC) /usr/lib
26     sudo mv $(HEAD) /usr/include
27
28 static:$(STATIC)
29     $(CC) main.c -L/home/demo/Documents/pfitouss/ -I/home/demo/Documents/pfitouss/ -lTextureControl -lwiringPi -lpthread -llo -lm
30
31 $(STATIC):$(OBJ)
32     $(AR) rcs $(STATIC) $(OBJ)
33
```

Illustration 5: Makefile de la librairie

Ainsi 4 commandes sont disponibles pour compiler la librairie :

make clean : Efface les fichiers .o et .out inutile à l'utilisation de la librairie.

make static : Créer le fichier libTextureControl.a, librairie statique.

make dynamic : Créer le fichier libTextureControl.so, librairie dynamique

make install : Copie les fichiers dans les dossier /usr/include et /usr/lib afin d 'installer la librairie sur l'ordinateur.

Application – Jeu Vidéo

Afin de démontrer que la librairie est fonctionnel, on m'a demandé de réaliser une application qui se sert de la librairie et qui met en valeur le principe de retour tactile. Je me suis donc directement tourné sur une application ludique.

Je me suis souvenue d'avoir jouer à jeu, Memory, qui consistait à disposer devant l'utilisateur un nombre paire de cartes à jouer, face verso. Sur le recto de ses cartes sont disposés des symboles, un même symbole se trouve sur deux cartes à la fois.

Le but du jeu étant de reformé les paires de cartes ayant le même symbole.

Vous ne pouvez retourner que 2 cartes à la fois.

Si les symboles correspondent, les cartes sont retirée du jeu et si ils ne correspondent pas, elle sont retournées face verso.

Vous l'aurez deviné de par le nom c'est un jeu de mémoire où il faut localiser les cartes de même symboles. Et ici les symboles sont remplacés par des motifs vibratoires.

Afin de l'expliquer je vais décomposer le programme en plusieurs parties:

Initialisation du jeu

Afin de démarrer une partie, il faut tout d'abord charer la libraire SDL, SDL_Image et SDL_TTF. Ces librairies permettent la réalisation en C, d'interface graphique en mode fenêtre, l'utilisation d'images et l'utilisation de polices afin d'imprimer du texte à l'écran.

Il faut bien évidemment charger en mémoire les différentes images utilisé tout au long de la partie ainsi que les texture au moyen de la fonction setTextureRect(…).

Différentes variable globales sont utilisé :

isMenu, isPause, isHelp, isWin, sont des booléens qui permettent au programme de se situer dans la partie, conditionne l'affichage à l'écran et les actions à réaliser lors d'un appuie ou d'un clic.

D'autre variables quant à elles servent de compteur. Il faut un compteur d'essai, de paires trouvées, de cartes sélectionnées ainsi que le nombre de tick pour le timer.

Le jeu est comporte 18 cartes. il faut initialisé chaque élément de la variable 'tableJeu[][]' en leurs attribuant une position sur l'écran, la face de la carte et un motif tactile en respectant les conditions suivantes :

-Il faut 18/2 motifs tactiles.

-Un motif tactile est présent sur exclusivement 2 cartes

-Disposition aléatoires des cartes et de leur motif

C'est la fonction initJeu() qui s'occupe de l'initialisation de la table de jeu.

Pour représenter les cartes, j'ai définie plusieurs structures et énumération(Annexe B),

Pour ce qui est de la disposition des cartes j'ai divisé le plateau en 3 lignes et 6 colonnes, en me servant de la résolution de l' écran il était facile de les disposés agréablement.

En revanche, la gestion de l'aléatoire est ce qui m'a posé le plus de difficultés dans cette fonction. Vu que je me sers d'un tableau à deux dimensions et que je dois respecter les règles énoncées plus haut quant à l'attribution des motifs, il aura fallu créer une fonction spéciale pour s'assurer du bon respect des règles. L'astuce ici c'est de remarqué qu'un tableau est parcouru de manière itératif, éléments après éléments, il est donc très difficile de générer un chiffre aléatoire tout en étant sûr qu'il apparaît 2 fois et uniquement 2 fois, tout en ayant aucune redondance dans le placement des motifs sur les cartes.

En créant un tableau temporaire de 18 cases, stockant 2 fois les entiers de 1 à 9 et en venant le mélanger, on peut ensuite parcourir notre tableau 2D de manière itératif et d'y stocker la valeur du tableau temporaire. On obtient donc un tableau 2D qui aléatoirement reçoit 2 fois le même motif sans pour autant observé une redondance de placements des motifs sur les cartes.

Interface Graphique

L'interface graphique se compose de plusieurs fenêtres en fonction de l'avancement du joueur dans le programme. C'est la fonction « affichage() » qui conditionne les informations à faire apparaître à l'écran. Cette fonction est appelée une fois à partir de la fonction « main() » du programme et à la fin de chaque appel de la fonction actionListener().

Il a également fallut créer des design de cartes, choisir un fond pour le jeu et créer le graphisme des boutons (Annexe D & E)

Menu

Le menu est la première fenêtre généré par le programme, elle permet de ne pas démarrer le jeu immédiatement en proposant 2 actions :

Play : Lance une partie de Memory

Quit : Quitte le Jeu

Illustration 6: Menu du jeu

Pause

Le bouton Pause accessible durant une partie, permet comme son nom l'indique de mettre le jeu en pause et propose 3 actions à l'utilisateur :

-Continue : Reprend la partie où elle s'est arrêtée

-Help : Affiche l'écran d'aide

-Quit : Quitte le jeu

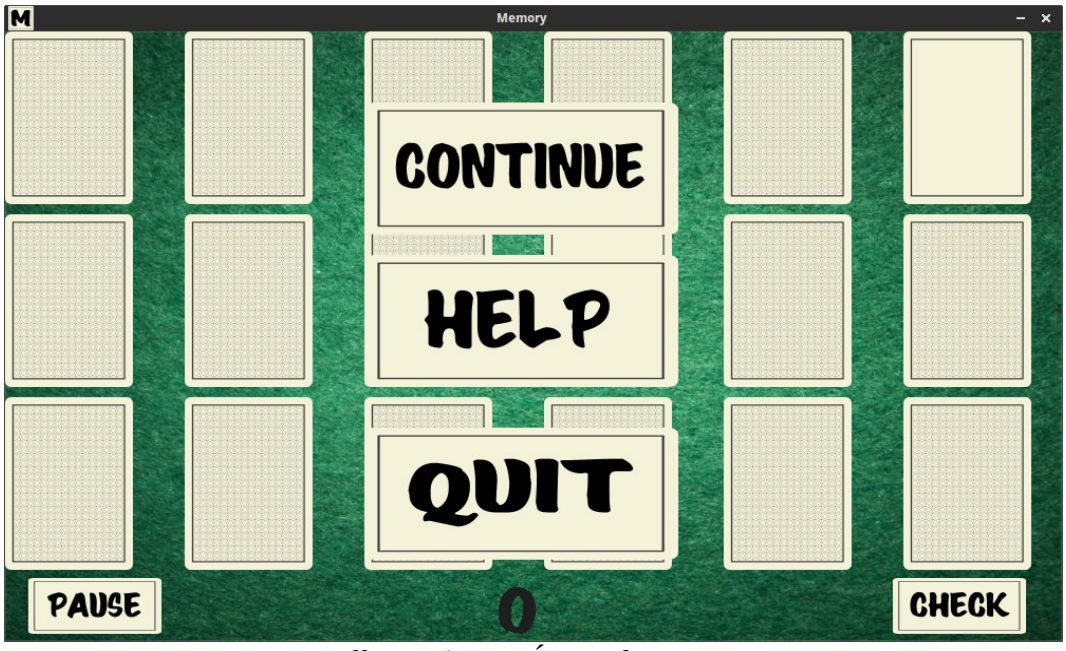


Illustration 7: Écran de pause

Help

Le menu Help est un affichage texte des règles et du principe du jeu, permettant à l'utilisateur novice de comprendre le fonctionnement du programme.

Un bouton Continue est présent, afin de sortir du menu d'aide.

Illustration 8: Affichage des règles du jeu

Win

L'écran « Win » affiche le score du joueur, c'est à dire le nombre de paires qu'il a formé durant la partie ainsi que le temps qu'il a mis à toutes les retrouver..

L'écran propose de jouer à nouveau via le bouton « Restart » ou bien de quitter le jeu via le bouton « Quit ».

Illustration 9: Écran de fin de partie

Jeu

L'écran jeu est l'écran principale de l'application, c'est sur cet écran que le joueur verra les cartes face verso et pourra essayer de les combiner en paire.

L’utilisateur devra valider chaque paire à l'aide du bouton « CHECK »

L'affichage vérifié dans le tableau « tableJeu[][] » la face de la carte afin d'afficher le verso, le recto ou ne rien afficher si la carte fait partie des paires formés.

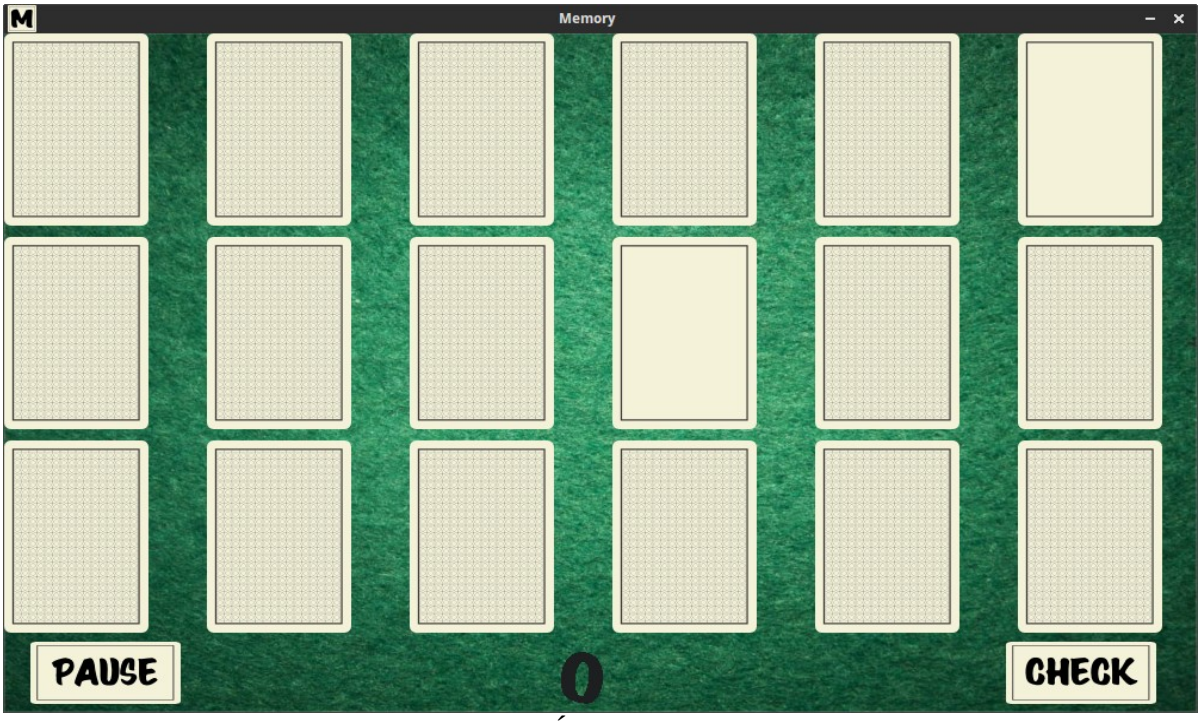


Illustration 10: Écran principale du jeu

Gestion des événements

Maintenant que nous avons vu comment est généré et conditionné l'interface graphique, il faut permettre à l'utilisateur de jouer. J'ai utilisé les fonctions de callback de la librairie afin que le doigt soit assimilé à une souris. Cela m'a permis d 'éviter les problème de multiple captation généré par le doigt.

Les fonctions initTouch(), sendTouchPressed(), sendTouchRelease(), sendTouchDragged(), et destroyTouch(), sont donc les fonctions que l'on se servira lorsque MyISR() (fonction de la librairie) devra retransmettre les informations propres à la position et l'état du doigt.

La fonction « actionListener() » est la boucle principale du programme, elle attend qu'une action soit réalisé par l'utilisateur tant que la variable « continuer » est à 1.

Différentes actions sont disponible en fonction des menus, la fonction « actionListener() » est donc également conditionné par les variable « isWIn », « isMenu » etc . (Voir Annexe C)

La fonction va donc dans un premier temps vérifier s'il s'agit d'un clique souris ou de l'appuie d'une touche du clavier.En fonction de la fenêtre en cours d’utilisation, on repère si le clic est effectué sur une carte, un bouton, ou sur la croix afin de fermer le programme.

Conclusion

Ce projet m'aura beaucoup apporté, tout d'abord j'ai eu le privilège de pouvoir travailler sur une technologie innovante et prometteuse, raison pour lesquelles j'ai choisi ce projet.

J'ai pu me rapproché davantage du monde de la recherche en me rendant dans les bâtiments de l'IRCICA, afin de discuter du projet et de leurs projets avec différents chercheurs.

J'ai également renforcé mes compétences dans le langage de programmation C et découvert l'utilisation de Processing.

Je me suis familiarisé avec les outils tel que GIT, MAKEFILE et la bibliothèque d'interface graphique SDL.

Malgre le fait que j'ai mis de côté l'étude de la représentation fréquentielle des motifs de vibration,je pense que les objectifs concernant la réalisation de la librairie ainsi que d'une application utilisant cette dernière et mettant en valeur la technologie du retour tactile sont atteints.

Cependant il serait intéressant de vérifier le bon fonctionnement de la bibliothèque avec des langages de haut, en utilisant par exemple le *Java Native Interface* pour Java .

Annexe A – Fichier en-tête « libTextureControl.h »

```
//#define DEBUGPRINTF
//#define DEBUG
#define PICC


#ifdef PICC
#include <wiringPi.h>
#include <wiringPiSPI.h>
#endif

/*
    INIT
*/


int startEvita();
void stopEvita();

/*
SET TEXTURE / TAXTEL
*/


void setTaxtTelRect(int id,int textureId, int x1, int y1, int x2, int y2);
void setTaxtTelEllipse(int id,int textureId, int x1, int y1, int x2, int y2, int r);
void disableTaxTel(int id);
void enableTaxTel(int id);


void setTextureRect(int textureId,float offset,float amplitude,float period,float ratio,char* speedFunc);
void setTextureCos(int textureId,float offset,float amplitude,float period, char* speedFunc);

/*
    CALLBACK GESTION METHODE D'ENTREE
*/
void registerCallbackPressed(void (*myfonction) (int x, int y));
void registerCallbackDragged(void (*myfonction) (int x, int y));
void registerCallbackRelease(void (*myfonction)());
void registerCallbackSpeed(void (*myfonction) (int speedX, int speedY));
void registerCallbackTexture(void (*myfonction) (int signal, int period));

/*
    MÉTHODE DE RECUPERATION DES VALEURS
    CODAGE HAUT NIVEAU

*/

int getX();
int getY();
int getPressed();
int getDragged();
int getReleased();
int getSpeedX();
int getSpeedY();
int getPeriod();
int getSignal();
```

Annexe B - Structure & Énumération

```
17  typedef enum Face Face;
18  enum Face
19  {
20      RECTO, VERSO, OUT
21  };
22
23
24  typedef struct Carte Carte;
25  struct Carte
26  {
27      SDL_Rect position;
28      Face face;
29      unsigned short int texture;
30
31  };
32
33  typedef struct Coord Coord;
34  struct Coord
35  {
36      int ligne;
37      int colonne;
38  };
```

Annexe C – actionListner() (extrait)

```
503                     if (isMenu)
504                     {
505                         //JOUER OU QUITTER
506                         if (controlEvent.x > 380 && controlEvent.x <661)
507                         {
508                             if(controlEvent.y > 90 && controlEvent.y < 290 )
509                             {
510                                 //BUTTON PLAY
511                                 isMenu = 0;
512                                 initJeu();
513                                 #ifdef DEBUG
514                                     printf("action listener MENUs\n");
515                                 #endif
516 
517                                 //affichage();
518                             }
519 
520                             if(controlEvent.y > 320 && controlEvent.y < 520)
521                             {
522                                 continuer = 0;
523                             }
524 
525                         }
526                         //affichage();
527                     }
528                     else if(isHelp)
529                     {
530                         //MENU AIDE
531                         if(controlEvent.y > 460 && controlEvent.x > 690)
532                         {
533                             //BUTTON CONTINUE
534                             isHelp = 0;
535                         }
536                     }
537                     else if (isPause)
538                     {
539                             if (controlEvent.x > 340 && controlEvent.x < 637)
540                             {
541                                 if (controlEvent.y > 70 && controlEvent.y < 200)
542                                 {
543                                     //BUTTON CONTINUER
544                                     isPause = 0;
545                                     //affichage();
546                                 }
547                                 if (controlEvent.y > 220 && controlEvent.y < 350)
548                                 {
549                                     //BUTTON HELP
550                                     isHelp = 1;
551                                 }
552                                 if (controlEvent.y > 390 && controlEvent.y < 520)
553                                 {
554                                     //BUTTON QUIT
555                                     continuer = 0;
556                                 }
557                             }
558                         //affichage();
559                     }
```

Annexe D – Graphiques du jeu

Illustration 11: Carte Verso

Illustration 13: Icône du jeu

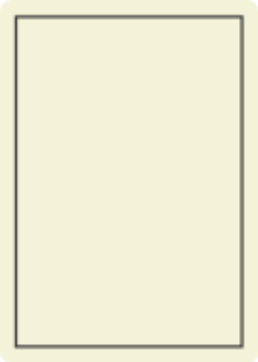


Illustration 12: Carte Recto

Illustration 14: Bouton "Quit"

Illustration 15: Bouton "Play"

Illustration 16: Bouton "Continue"

Illustration 19: Bouton "Pause"

Illustration 17: Bouton "Help"

Illustration 18: Bouton "Check"

Annexe E – Fond d'écran du jeu

Illustration 20: Fond d'écran du jeu