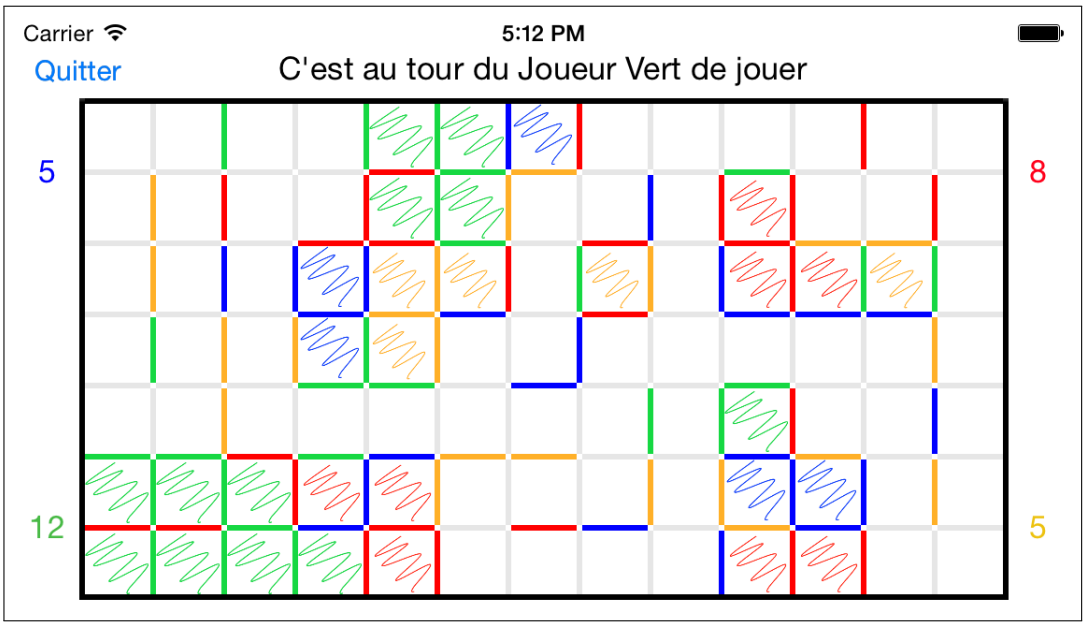
Gymnase de Burier

Création d'une Application iOS
La Pipopipette



Burier, le 23 octobre 2014

Loïs Bilat et Joachim Droz / 3M5

Remerciements

Avant de commencer, nous souhaiterions remercier **M. Salvadore** et **M. Besson** sans qui ce travail n'aurait pas pu être réalisé. Nous aimerions aussi remercier **Lennard Ludwig** qui nous a donné plusieurs conseils et idées, ainsi que les personnes suivantes pour la relecture de ce travail : **Dominique Droz, Jean-Paul Droz, Frédérique Bilat** et **Grégoire Bilat** qui a également relié ce dossier.

Table des matières

1 Introduction **1**
- 1.1 Le projet . . . 1
- 1.2 La démarche . . . 2
- 1.3 Nos objectifs . . . 2

2 Démarche **4**
- 2.1 Apprentissage du C . . . 4
  - 2.1.1 Le commentaire du code : première application . . . 4
  - 2.1.2 Ce que nous avons appris . . . 10
- 2.2 Apprentissage de l'Objective-C . . . 10
  - 2.2.1 Le commentaire du code : deuxième application . . . 10
  - 2.2.2 Ce que nous avons appris . . . 19

3 Les premiers modèles de la Pipopipette **20**
- 3.1 Le modèle 2x2 . . . 20
  - 3.1.1 Le commentaire du code : le modèle 2x2 . . . 21
  - 3.1.2 Le compte-rendu du premier modèle . . . 24
- 3.2 Le modèle 3x3 . . . 24

4 L'application finale **25**
- 4.1 L'interface et la présentation du code . . . 25
  - 4.1.1 La page d'accueil . . . 26
  - 4.1.2 La page du choix du nombre de joueurs . . . 28
  - 4.1.3 La page de la grille de jeu . . . 30
- 4.2 Le compte-rendu de cette application . . . 48
- 4.3 La démarche . . . 49

5 Conclusion **50**

6 Annexes **52**
- 6.1 Programme de répétition de livrets en C . . . 52
- 6.2 Code du répétiteur de livrets en objective-C . . . 57
  - 6.2.1 ViewController.h . . . 57
  - 6.2.2 ViewController.m . . . 57
- 6.3 Code du modèle 2x2 . . . 64
  - 6.3.1 ViewController.h . . . 64
  - 6.3.2 ViewController.m . . . 66
- 6.4 Code de l'application finale . . . 70

6.4.1 ViewController.h 70
6.4.2 ViewController.m 70
6.4.3 choixModeJeu.h 72
6.4.4 choixModeJeu.m 72
6.4.5 grille.h 74
6.4.6 grille.m 75

Chapitre 1

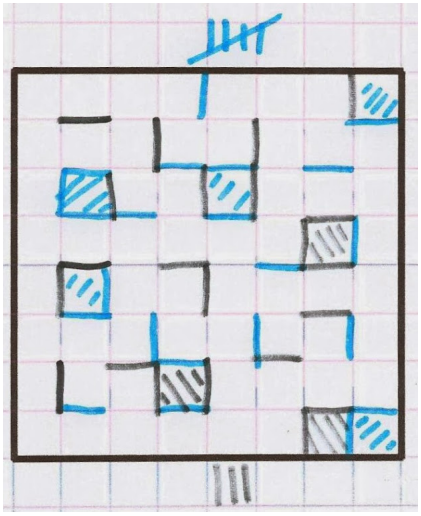
Introduction

1.1 Le projet

Durant cette année 2014, nous avons l'intention de développer une application pour appareils mobiles iOS. Nous voulons faire de la programmation, car c'est un sujet qui nous intéresse fortement, et nous voudrions savoir pour notre avenir, si travailler dans ce domaine nous plairait. Cette application serait plus précisément un jeu. Dans un premier temps, nous allons vous présenter ce jeu et détailler son but et ses règles.

Ce jeu est nommé "La Pipopipette" et a été imaginé en 1889 [1]. C'est un petit jeu relativement rapide qui se joue en général à 2 joueurs, à l'aide d'une feuille de papier quadrillée et de stylos de couleurs différentes. Pour commencer, on délimite une forme suivant le quadrillage qui sera le "terrain de jeu". Plus cette zone est grande, plus la partie sera longue. Le but du jeu est de, chacun son tour, tracer un trait le long du quadrillage de la feuille, et de réussir à être le joueur qui trace le dernier trait pour former un carré. A ce moment-là, la personne ayant fini le carré peut le remplir avec sa couleur, marque un point, et peut placer un autre trait, qui peut éventuellement lui permettre de finir un autre carré, et ainsi de suite. Une fois que la totalité de la zone délimitée au préalable est remplie, la partie se termine et le joueur qui possède le plus de points a gagné. Nous avons l'intention de créer une version mobile de ce jeu, car nous apprécions particulièrement ce concept, qui mêle observation et réflexion. Nous pensons développer cette application pour appareil de petite taille, tel que l'iPhone ou l'iPod Touch, mais peut-être que nous allons également le faire pour iPad, car la taille d'écran de cet appareil nous offrirait plus de possibilités, rendrait l'expérience de jeu meilleure et permettrait également de faire de parties plus longues.

1. Source : wikipedia.org



Exemple d'une partie de Pipopipette sur papier

1.2 La démarche

Pour parvenir à cela, nous pensons procéder de la manière suivante. Premièrement, nous allons simplement suivre le cours de programmation qui nous est proposé, dans le but d'apprendre le langage de programmation pour plate-forme iOS qui est l'Objective-C. Une fois cela fait, nous pourrons nous intéresser plus précisément à notre application pour savoir quelles fonctionnalités pourront ou non être intégrées, selon les connaissances que nous aurons à ce moment-là. Nous pensons aussi nous renseigner à d'autres endroits, selon ce que nous aurons besoin de spécifique pour créer notre jeu. Nous pensons aussi lire en parallèle un livre qui apprend les notions de bases de programmation pour iOS, et qui donne également des exemples concrets d'applications, qui sont réalisées et expliquées dans ce guide. Ce livre a été créé par le site du zéro, maintenant renommé OpenClassrooms. Pour programmer cette application, nous aurons besoin de l'application nommée Xcode, qui est le programme officiel fourni par Apple pour créer des applications pour Mac, iPhone, iPad et iPod Touch. Ce logiciel offre une interface de travail adaptée au développement d'applications iOS. Nous allons donc apprendre ce langage, réfléchir plus précisément aux fonctions que nous allons intégrer dans notre application, en fonction de ce que l'on sera capable de faire, et seulement après, nous allons commencer le développement de l'application. Quand l'application sera faite, nous allons peut-être ajouter quelques fonctions supplémentaires, telles que des bruitages, des animations, etc. Et, pendant tout ce temps, nous allons rédiger un dossier pour expliquer plus précisément notre démarche, pour présenter l'application plus en profondeur et surtout pour expliquer son code.

1.3 Nos objectifs

Nous avons plusieurs objectifs : le premier est de pouvoir être capable, après cette année de travail, de développer une application iOS par nous-même, et ce, du début à la fin, et d'arriver au final avec une application qui fonctionne bien, et qui est comme nous le voulions. Nous aimerions donc assimiler suffisamment de connaissances, et comprendre

la logique de programmation de manière à pouvoir réutiliser ces notions plus tard, éventuellement dans nos futurs métiers. C'est le but principal de ce travail. Le second objectif, qui est d'ailleurs la raison pour laquelle nous sommes deux, est de réussir à travailler ensemble, de parvenir de collaborer pour créer cette application, tout cela pour avoir un aperçu du travail en entreprise, pour nous rendre compte des possibles difficultés de cette collaboration, et pour donc nous préparer en partie à notre avenir professionnel.

Chapitre 2

Démarche

2.1 Apprentissage du C

La première étape de notre apprentissage de l'Objective-C fut l'apprentissage du C. Les quelques premières semaines de notre travail de maturité furent donc consacrées à l'apprentissage des notions élémentaires de ce langage, qui nous seront aussi utiles en Objective-C, pour développer notre application. Après quelques semaines, nous avons créé un petit programme de répétition de livrets, qui mettait en pratique ce que nous avions appris. Il faut noter que ce n'est pas vraiment une application, il n'y a pas de vraie interface. Tout se passe dans Xcode, dans une console. Nous allons vous détailler les différentes parties de ce programme ainsi que son fonctionnement ci-dessous. Le code intégral du programme se trouve à la fin, dans les annexes.

2.1.1 Le commentaire du code : première application

Nous avons construit notre programme de manière à avoir une fonction `main` qui représente le menu principal de notre application, et qui, après une interaction de l'utilisateur, lance une autre fonction. Nous allons commencer par détailler la fonction `main`, qui se trouve à la fin du code.

```
char programme;
```

Dans cette ligne, on initialise la variable `programme` du type chaîne de caractères pour pouvoir l'utiliser par la suite dans la fonction principale. Elle sera utilisée pour stocker la lettre entrée par l'utilisateur. Nous ne la mettons pas dans la fonction `main` pour qu'elle soit accessible depuis n'importe quelle autre fonction.

```
int main(int argc, const char * argv[]){
    printf("Bienvenue dans le répétiteur de livrets !\n");
    printf("\n");
    printf("Menu principal :\n");
    printf("\n");
    printf("Appuyez sur 'l' pour afficher un livret\n");
    printf("Appuyez sur 't' pour lancer le mode test\n");
    printf("Appuyez sur 'e' pour lancer le mode exercice\n");
    printf("Appuyez sur 'q' pour quitter le programme\n");
```

Toutes les lignes commençant par `printf` ci-dessus sont là pour afficher dans la console Une sorte de menu principal du programme. Le joueur a le choix entre différents types d'exercices. Selon la lettre sur laquelle il appuie, le programme va lancer une des autres fonctions, que nous allons vous détailler plus tard.

```
    while (1) {

        scanf("%c",&programme);
        if (programme == 'q') {
            printf("Au revoir et à bientôt\n");
            break;
        }
        if (programme == 'l') {
            livret();
            continue;
        }
        if (programme == 't') {
            test();
            continue;
        }
        if (programme == 'e') {
            exercice();
            continue;
        }
        printf("Menu principal :\n");
        printf("\n");
        printf("Appuyez sur 'l' pour afficher un livret\n");
        printf("Appuyez sur 't' pour lancer le mode test\n");
        printf("Appuyez sur 'e' pour lancer le mode exercice\n");
        printf("Appuyez sur 'q' pour quitter le programme\n");
    }
     return 0;
}
```

Cette boucle infinie teste quelle lettre l'utilisateur a entré. Pour cela, on utilise `scanf` qui stocke dans la variable `programme` la lettre entrée. Selon le choix de l'utilisateur, le programme lance la fonction correspondante. Dans le cas ou on a appuyé sur "q", le programme s'arrête grâce à la ligne `break`. Et à la fin de cette boucle le programme réaffiche le menu.

Maintenant, nous allons rapidement expliquer l'intérêt de la fonction suivante, qui sera utile dans les autres parties de ce programme. Il faut savoir qu'en C, il existe une fonction qui permet de créer des nombres aléatoires. Mais malheureusement, elle ne permet pas de définir des bornes, c'est-à-dire un nombre maximum et minimum. Cette fonction, qui dépend de `a` et `b`, les deux bornes, va permettre de transformer le nombre aléatoire créé par la fonction `rand` en un nombre aléatoire situé entre nos deux bornes. Nous avons nommé cette fonction `rand_a_b`.

```
int rand_a_b(int a, int b){
    int nbr;
    nbr = rand() % (b-a) + a;
    return nbr;
}
```

Nous allons maintenant vous expliquer le fonctionnement de la fonction livret, qui permet d'afficher une table de multiplication choisie par l'utilisateur.

```
int livret(){
    printf("Vous avez choisi d'afficher un livret\n");
    printf("\n");
    printf("Quel livret voulez-vous afficher ?\n");
    printf("\n");
```

Dans la première partie de la fonction livret, on dit à l'utilisateur qu'il a choisi le mode livret pour ensuite lui demander quel livret il veut afficher.

```
    int numeroLivret;
    scanf("%d",&numeroLivret);
    printf("Table des livrets de %d :\n",numeroLivret);
    printf("\n");
    int i = 1;
    for (i=1; i<21; i++) {
        printf("%2.d * %d = %d\n",i,numeroLivret,i*numeroLivret);
    }
    printf("\n");
    return 0;
   }
```

Dans cette deuxième partie de la fonction livret, on regarde quel livret l'utilisateur veut voir pour ensuite lui afficher les vingt premiers calculs de ce livret à la console. Par exemple, s'il entre "1", la console affichera : `1 x 1 = 1, 2 x 1 = 2, 3 x 1 = 3`, etc, en revenant à la ligne après chaque calcul.

A présent, nous allons vous parler d'une des fonctions les plus importantes : le mode exercice, qui permet à l'utilisateur de s'exercer aux livrets en boucle, en choisissant son niveau de difficulté.

```
int nbrEntreExercice;

int exercice(){
    printf("Mode exercice :\n");
    printf("\n");
    int a;
    int b;
```

```
int nombreAleatoire1;
int nombreAleatoire2;
int difficulte;
printf("Choisissez la difficulté (0 pour facile, 1 pour moyen, 2 pour
difficile et 3 pour personnalisé): ");
scanf("%d",&difficulte);
printf("\n");
```

Ici, on commence par déclarer toutes les variables qui seront utilisées et on demande à l'utilisateur de choisir son niveau de difficulté, représenté par un nombre stocké dans la variable `difficulte`.

```
if (difficulte == 0) {
    a = 1;
    b = 13;
    printf("Mode facile\n");
}
if (difficulte == 1) {
    a = 2;
    b = 21;
    printf("Mode moyen\n");
}
if (difficulte == 2) {
    a = 13;
    b = 21;
    printf("Mode difficile\n");
}
if (difficulte == 3) {
    printf("Mode personnalisé\n");
    printf("Choisissez le plus petit livret de l'exercice : ");
    scanf("%d",&a);
    printf("Choisissez le plus grand livret de l'exercice : ");
    scanf("%d",&b);
    b++;
}
```

Maintenant, le programme va regarder la valeur du nombre entré par l'utilisateur, et selon cette valeur, il va changer les bornes des nombres qui seront utilisées pour les livrets, ce qui change la difficulté. Si on choisit le mode personnalisé, on demande alors à l'utilisateur de choisir les bornes lui-même. Les bornes sont stockées dans les variables `a` et `b`.

```
srand(time(NULL));
printf("\n");
printf("Entrez 0 pour revenir au menu principal\n");
printf("\n");
while (1) {
```

```
        nombreAleatoire1 = rand_a_b(a,b);
        nombreAleatoire2 = rand_a_b(a,b);
        int resultat = nombreAleatoire1 * nombreAleatoire2;
        printf("%d * %d = ",nombreAleatoire1,nombreAleatoire2);
        scanf("%d",&nbrEntreExercice);
        if (resultat==nbrEntreExercice) {
            printf("La réponse est juste\n");
        }
        else if (nbrEntreExercice == 0){
            printf("\n");
            printf("Vous avez quitté\n");
            printf("\n");
            break;
        }
        else{
            printf("Faux, la réponse est %d\n",resultat);
        }
    }
    return 0;
}
```

Voici à présent la partie la plus importante de cette fonction. La première ligne de cette partie `(srand(time(NULL))`, est une particularité qui permet de rendre les nombres le plus aléatoirement possible [1]. Cette ligne permet de trouver un nombre le plus au hasard possible en se servant de l'heure et la date. Après cela, on entre dans une boucle infinie et on définit deux nombres aléatoires, en fonction des bornes décidées, qui seront les deux termes des livrets affichés. On calcule la réponse, on affiche la question dans la console, et on scanne la réponse qu'entre l'utilisateur. S'il entre "0", la fonction s'arrête en on retourne au menu principal et si la réponse est juste, on le lui dit et la boucle recommence.

Pour finir, voici la fonction qui gère le mode test.

```
int nbrEntreTest;

int test(){
    printf("Mode Test\n");
    printf("\n");
    int a;
    int b;
    int difficulte;
    printf("Choisissez la difficulté (0 pour facile, 1 pour moyen,
    2 pour difficile et 3 pour personnalisé): ");
```

Au début de cette fonction, on introduit toutes les variables dont nous aurons besoin. Puis, comme pour le mode exercice, on affiche à l'écran le message qui demande à l'utilisateur de choisir la difficulté du mode.

1. En informatique, les nombres ne peuvent pas être aléatoires. En général, l'ordinateur se base sur une grande liste de nombre générés aléatoirement par un autre moyen.

```
scanf("%d",&difficulte);
if (difficulte == 0) {
    a = 1;
    b = 13;
    printf("Mode facile\n");
}
if (difficulte == 1) {
    a = 2;
    b = 21;
    printf("Mode moyen\n");
}
if (difficulte == 2) {
    a = 13;
    b = 21;
    printf("Mode difficile\n");
}
if (difficulte == 3) {
    printf("Mode personnalisé\n");
    printf("Choisissez le plus petit livret du test : ");
    scanf("%d",&a);
    printf("Choisissez le plus grand livret du test : ");
    scanf("%d",&b);
    b++;
}
```

Toute cette partie est la même que pour le mode exercice, on définit la difficulté des calculs.

```
int x = 0;
int vrai = 0;
int faux = 0;
int nombreAleatoire1;
int nombreAleatoire2;
int n;
printf("Combien de calculs voulez-vous faire ?\n");
scanf("%d",&n);
srand(time(NULL));
```

Ici, On déclare les variables dont nous aurons besoin. Les variables `vrai` et `faux` donneront le nombre de réponses justes et le nombre de réponses fausses. On demande ensuite à l'utilisateur combien de calculs il veut faire et on stockera ce nombre dans la variable `n`.

```
for (x = 0; x<n; x++) {
    nombreAleatoire1 = rand_a_b(a,b);
    nombreAleatoire2 = rand_a_b(a,b);
    int resultat = nombreAleatoire1 * nombreAleatoire2;
    printf("%d * %d = ",nombreAleatoire1,nombreAleatoire2);
```

```
        scanf("%d",&nbrEntreTest);
        if (resultat==nbrEntreTest) {
            vrai++;
        }
        else{
            faux++;
        }
```

Dans cette boucle du programme qui s'exécute autant de fois que l'utilisateur l'aura voulu, on affiche la question à l'écran avec les deux nombres aléatoires. Si le résultat est correct, on ajoute un à la variable qui compte le nombre de réponses justes. Si le résultat est incorrect, on fait de même mais avec la variable du nombre de réponses fausses.

```
    }
    printf("\n");
    printf("Il y a %d réponse(s) juste(s) et %d réponse(s) fausse(s)\n",
    vrai,faux);
    printf("\n");
    printf("Fin du test\n");
    printf("\n");
    return 0;
}
```

Pour finir, on affiche le nombre de réponses justes et fausses, et la fonction s'arrête ce qui nous ramène au menu principal.

2.1.2 Ce que nous avons appris

Grâce à cette petite application, nous avons appris les bases du langage C. Nous avons appris, entre autres, à utiliser les chaînes de caractères, les variables et la génération de nombres aléatoires. Grâce à d'autres petits exercices que nous avons faits, nous avons également vu lors de l'apprentissage de ce langage le fonctionnement d'autres éléments comme par exemple les tableaux. Nous allons maintenant poursuivre notre apprentissage de la programmation avec un nouveau langage : l'Objective-C.

2.2 Apprentissage de l'Objective-C

Une fois les bases du C acquises, nous avons appris l'Objective-C. Ce langage est basé sur le C, mais a été adapté par Apple. Il permet de développer des applications pour Mac, iPhone, iPod Touch et iPad. C'est donc ce langage qui sera utilisé pour créer notre projet. Après quelques cours pendant lesquels nous avons appris les rudiments de ce langage, nous sommes passés au développement de notre première application en Objective-C.

2.2.1 Le commentaire du code : deuxième application

Cette première application en Objective-C permet à l'utilisateur de faire du calcul mental. Après avoir choisi le mode de calcul (multiplication, division, addition ou sous-

traction) et le niveau de difficulté (facile, moyen ou difficile), une question lui est posée aléatoirement. Une fois la réponse entrée, l'application se charge de nous dire si la réponse inscrite est correcte ou erronée et nous corrige si notre réponse ne correspond pas à la solution. Il est également possible de voir ses statistiques, qui nous indiquent le nombre de réponses justes et fausses faites, celles qui ne sont pas correctes et nous donne en plus une petite phrase qui dépend du pourcentage de véracité de nos réponses.

Interface de l'application

Ci-dessous, nous allons détailler le fichier principal de l'application, dans lequel la majeure partie code est écrit.

```
int nombreAleatoire1;
int nombreAleatoire2;
int a = 2;
int b = 12;
int multiple;
int resultat;
int reinitialiser;
int zero = 0;
int provisoire;
float vrai = 0;
float faux = 0;
NSString *texteFaux = @"";
NSString *texteAjout;
NSString *listeLivretsFaux = @"";
```

Dans cette partie, on commence par déclarer les variables qui nous seront utiles dans tout le code. `nombreAleatoire1` et `nombreAleatoire2` sont les nombres qui seront générés aléatoirement pour créer les livrets demandés. `a` et `b` sont les bornes des livrets, autrement

dit le maximum et le minimum des nombres qui nous seront demandés. Les variables `vrai` et `faux` comptent le nombre de réponses juste et fausses. `mulitple` et `resultat` sont deux variables utilisées pour vérifier la réponse entrée par l'utilisateur. `reinitialiser` et `zero` sont deux variables dont nous expliquerons l'utilité le moment venu. Nous ferons de même avec `texteFaux`, `texteAjout` et `listeLivretsFaux` , qui seront utiles pour la partie des statistiques.

```
int rand_a_b(int a, int b){
    int nbr;
    nbr = rand() % (b-a) + a;
    return nbr;
}
```

Là, on retrouve la même fonction que celle utilisée lors de la création de l'application de répétition de livrets en C, elle permet de générer un nombre aléatoire entre deux bornes.

```
- (void)viewDidLoad{

    [super viewDidLoad];
    srand(time(NULL));

    UIAlertView *bienvenue = [[UIAlertView alloc] initWithTitle:@"Bonjour"
        message:@"Bienvenue dans notre application de répétition de calcul
        mental\n\nApplication crée par Loïs et Joachim dans le cadre de leur
        Travail de Maturité" delegate:nil cancelButtonTitle:@"Continuer"
        otherButtonTitles:nil];
    [bienvenue show];

    [self suivant:nil];
    self.statistiques.enabled = NO;
}
```

Ici, c'est ce qui se passe au démarrage de l'application. On commence par afficher un pop-up [2] pour souhaiter la bienvenue à la personne qui utilise l'application. Après cela, on désactive le bouton menant aux statistiques, on appelle la fonction `suivant`, qui va générer un nouveau calcul.

Ce qui va suivre se passe lorsqu'un on appuie sur un des boutons pour changer la difficulté. Ces boutons sont tous dans une même "barre d'onglets".

```
- (IBAction)niveau:(id)sender {
    self.onglets.enabled = YES;
    if (self.onglets.selectedSegmentIndex == 0){
        if (self.mode.selectedSegmentIndex == 0 ||
            self.mode.selectedSegmentIndex == 1){
            a = 2;
```

2. Un pop-up est une notification apparaissant à l'écran

```
            b = 12;
        }
        else if (self.mode.selectedSegmentIndex == 2 ||
            self.mode.selectedSegmentIndex == 3){
            a = 10;
            b = 100;
        }
    }
```

Cette boucle regarde quel niveau de difficulté est choisi par l'utilisateur et quel type de calculs il souhaite faire. Pour cela, on utilise `self.onglets.selectedSegmentIndex` qui regarde quel partie de la "barre d'onglets" de difficulté est sélectionnée et on en déduit le niveau de difficulté (ici le mode 0 correspond au niveau facile) Ensuite, on regarde le mode de calcul choisi avec `self.mode.selectedSegmentIndex`. Si on fait des multiplications ou des divisions (`self.mode.selectedSegmentIndex` vaut zéro ou un), on définit 2 et 12 comme bornes. Mais si on a choisi de faire des additions ou des soustractions, ces bornes sont 10 et 100.

```
    else if (self.onglets.selectedSegmentIndex == 1){
        if (self.mode.selectedSegmentIndex == 0 ||
            self.mode.selectedSegmentIndex == 1){
            a = 6;
            b = 16;
        }
        else if (self.mode.selectedSegmentIndex == 2 ||
            self.mode.selectedSegmentIndex == 3){
            a = 100;
            b = 1000;
        }
    }
    else if (self.onglets.selectedSegmentIndex == 2){
        if (self.mode.selectedSegmentIndex == 0 ||
            self.mode.selectedSegmentIndex == 1){
            a = 12;
            b = 20;
        }
        else if (self.mode.selectedSegmentIndex == 2 ||
            self.mode.selectedSegmentIndex == 3){
            a = 1000;
            b = 10000;
        }
    }
```

Ici on fait de même pour les niveaux moyen (`self.onglets.selectedSegmentIndex == 1`) et difficile (`self.onglets.selectedSegmentIndex == 3`)

```
    [self suivant:nil];
}
```

Cette dernière ligne appelle la fonction nommée `suivant` que nous décrirons plus tard. Elle permet de créer un nouveau calcul et de l'afficher.

```
- (IBAction)statistiques:(id)sender {
    NSString *texte = @"";
    self.vraiFaux.text = @"";

    float pourcentage = vrai/(vrai+faux);
    if (pourcentage >= 0.95) {
        texte = @"Vous êtes un génie du calcul mental";
    }
    else if (pourcentage >= 0.8 && pourcentage < 0.95){
        texte = @"Vous êtes un pro du calcul mental";
    }
    else if (pourcentage >= 0.6 && pourcentage < 0.8){
        texte = @"Vous maîtrisez bien le calcul mental";
    }
    else if (pourcentage >= 0.4 && pourcentage < 0.6){
        texte = @"Il y a encore un peu de travail à faire";
    }
    else if (pourcentage >= 0.2 && pourcentage < 0.4){
        texte = @"C'est pas votre jour!";
    }
    else if (pourcentage >= 0.1 && pourcentage < 0.2){
        texte = @"ToDoList: réviser les maths";
    }
    else {
        texte = @"Le calcul mental c'est pas votre truc!";
    }
```

Toute cette partie du programme fonctionne avec le pourcentage de réponses justes entrées par l'utilisateur. Selon ce pourcentage, le programme affiche une petite phrase correspondant au pourcentage de justes. Par exemple si on a fait vingt questions de livrets dont deux réponses qui étaient fausses (ce qui fait nonante pourcents de réponses justes), le programme affiche : "Vous êtes un pro des livrets".

```
    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:
        @"Statistiques" message:[NSString stringWithFormat:
        @"%@%.f%@%.f%@%@%@%@",@"Entrez 0 comme réponse pour effacer
        les statistiques\n\nVous avez fait ",vrai,@" réponse(s)
        juste(s) et ",faux,@" réponse(s) fausse(s) depuis le
        lancement de l'application\n\n",texte,listeLivretsFaux,
        texteFaux] delegate:self cancelButtonTitle:@"Fermer"
        otherButtonTitles:nil];
    [alert show];
}
```

Un pop-up s'affiche lorsqu'on appuie sur le bouton `statistiques`. Il contient le nombre de réponses justes et le nombre de réponses fausses depuis la dernière réinitialisation ou depuis le lancement de l'application. Il contient aussi tous les livrets répondus faux, afin que l'on puisse les répéter.

```
- (IBAction)entree:(id)sender {

    self.verifier.enabled = YES;
    [self.verifier setTitle:@"Vérifier la réponse"
        forState:UIControlStateNormal];
    self.vraiFaux.text = @"";
    int n = [self.entree.text  integerValue];
    if (n == 0) {
        if (zero == 0) {
            [self.verifier setTitle:@"Rénitialiser les
             statistiques" forState:UIControlStateNormal];
            reinitialiser = 1;
        }
    }
    else {
        zero = 1;
        reinitialiser = 0;
    }
}
```

Dans cette partie du code, on active tout d'abord le bouton `verifier` qui permettra de vérifier la réponse. On met aussi à jour le texte `vraiFaux` qui nous dira si la réponse est vraie ou fausse. Ensuite, on enregistre ce qui est écrit dans la variable `n`. Si l'utilisateur entre la valeur "0" dans le champ de texte, le bouton `verifier` devient le bouton pour réinitialiser les valeurs. La variable `reinitialiser` qui passe à "1" permettra de lancer la fonction adéquate plus tard. Au contraire, s'il n'entre pas zéro, on met la variable `zero` à 1 pour que, même si l'utilisateur efface sa réponse (ce que le programme considérerait normalement comme zéro) le bouton `reinitialiser` n'apparaisse pas. Cette variable est réinitialisée lorsque qu'on appuie sur le bouton "Suivant" ou "Vérifier".

```
- (IBAction)suivant:(id)sender {

    self.entree.text = @"";
    self.verifier.enabled = NO;
    [self.verifier setTitle:@"" forState:UIControlStateNormal];

    self.vraiFaux.text = @"";
    nombreAleatoire1 = rand_a_b(a,b+1);
    nombreAleatoire2 = rand_a_b(a,b+1);
    multiple = nombreAleatoire2 * nombreAleatoire1;

    if (self.mode.selectedSegmentIndex == 0)
    {
```

```
        self.livret.text = [NSString stringWithFormat:
        @"%d%@%d",nombreAleatoire1,@" x ",nombreAleatoire2];
    }
```

Intéressons-nous maintenant à ce qui se passe si on appuie sur `Suivant`. Toute cette partie se lance aussi à chaque fois qu'un changement de mode ou de difficulté s'effectue. On commence par créer deux nombre aléatoires, juste après avoir modifié quelques textes de l'interface, pour s'adapter à la situation. La variable `multiple` est utilisée juste après dans le code, pour afficher les divisions. Si on a choisi de faire une multiplication, on affiche le calcul.

```
    else if (self.mode.selectedSegmentIndex == 1){
        self.livret.text = [NSString stringWithFormat:
            @"%d%@%d",multiple,@" / ",nombreAleatoire1];
    }
    else if (self.mode.selectedSegmentIndex == 2){
        self.livret.text = [NSString stringWithFormat:
         @"%d%@%d",nombreAleatoire1,@" + ",nombreAleatoire2];
    }
```

On applique la même chose si on est en mode division ou addition. Dans le cas de la division, on utilise la variable multiple vue auparavant pour avoir un calcul avec une réponse entière.

```
    else if(self.mode.selectedSegmentIndex == 3){
        if (nombreAleatoire1 < nombreAleatoire2) {
            provisoire = nombreAleatoire2;
            nombreAleatoire2 = nombreAleatoire1;
            nombreAleatoire1 = provisoire;
        }
        else if(nombreAleatoire1 == nombreAleatoire2){
            [self suivant:nil];
        }
        self.livret.text = [NSString stringWithFormat:
         @"%d%@%d",nombreAleatoire1,@" - ",nombreAleatoire2];
    }
}
```

Pour le mode soustractions, il y a une petite subtilité à noter : comme sur le clavier des chiffres sur iPhone il n'y a pas le signe moins, il faut feinter en mettant toujours le nombre le plus grand devant pour ne pas avoir une réponse négative. Si le premier nombre est plus petit, on inverse les deux nombre en utilisant la variable `provisoire`. De plus, si les deux nombre sont les mêmes, on rappelle la méthode `suivant` pour régénérer deux nombres. Passons maintenant à la ce qui se passe quand on appuie sur le bouton `Vérifier`.

```
- (IBAction)verifier:(id)sender {
    zero = 0;
    self.statistiques.enabled = YES;
```

Cette partie du programme désactive le blocage fait auparavant qui empêchait qu'une réponse effacée soit considérée comme `0` par le programme. Par la même occasion, on réactive le bouton pour afficher les statistiques, vu que l'utilisateur a entré une réponse.

```
if (reinitialiser == 1) {
        vrai = 0;
        faux = 0;
        self.vraiFaux.textColor = [UIColor darkGrayColor];
        self.statistiques.enabled = NO;
        self.verifier.enabled = NO;
        [self.verifier setTitle:@"" forState:UIControlStateNormal];
        self.entree.text = @"";
        reinitialiser = 0;
        texteFaux = @"";
        listeLivretsFaux = @"";
        [self suivant:nil];
        self.vraiFaux.text = @"Statistiques réinitialisées";
    }
```

Rappelez-vous, plus tôt dans le programme, lorsque l'utilisateur avait entré `0`, la variable `reinitialiser` avait été égalée à un. Et ici lorsque `reinitialiser` est égal à un, toutes les variables du nombre de réponses justes et du nombre de réponses fausses ainsi que le texte contenant toutes les réponses fausses sont remis à zéro. On désactive par la même occasion le bouton des statistiques et celui pour vérifier la réponse. Et grâce à `[self suivant:nil]` on affiche un calcul selon le mode choisi après avoir tout réinitialisé. On affiche également "Statistiques réinitialisées".

```
else {
    int m = [self.entree.text integerValue];
```

Si on est dans un cas normal, c'est-à-dire que l'utilisateur a entré une réponse, on déclare la variable `m` comme étant cette réponse.

```
    if (self.mode.selectedSegmentIndex == 0) {
        resultat = nombreAleatoire2 * nombreAleatoire1;
    }
    else if (self.mode.selectedSegmentIndex == 1){
        resultat = multiple / nombreAleatoire1;
    }
    else if (self.mode.selectedSegmentIndex == 2){
        resultat = nombreAleatoire1 + nombreAleatoire2;
    }
    else if (self.mode.selectedSegmentIndex == 3){
        resultat = nombreAleatoire1 - nombreAleatoire2;
    }
```

On définit ensuite la réponse à la question en fonction du mode choisi par l'utilisateur.

```
if (m != resultat) {

    listeLivretsFaux = @"\n\nListe des calculs faux :\n\n";
    if (self.mode.selectedSegmentIndex == 0) {
        texteAjout = [NSString stringWithFormat:
        @"%d%@%d%@%d%@%d%@",nombreAleatoire1,@" x ",
        nombreAleatoire2,@" = ",resultat,@"\n
        (Vous avez mis ",m,@")\n\n"];
    }
    else if (self.mode.selectedSegmentIndex == 1) {
        texteAjout = [NSString stringWithFormat:
        @"%d%@%d%@%d%@%d%@",multiple,@" / ",
        nombreAleatoire1,@" = ",resultat,@"\n
        (Vous avez mis ",m,@")\n\n"];
    }
    else if (self.mode.selectedSegmentIndex == 2) {
        texteAjout = [NSString stringWithFormat:
        @"%d%@%d%@%d%@%d%@",nombreAleatoire1,@" + ",
        nombreAleatoire2,@" = ",resultat,@"\n
        (Vous avez mis ",m,@")\n\n"];
    }
    else if (self.mode.selectedSegmentIndex == 3) {
        texteAjout = [NSString stringWithFormat:
        @"%d%@%d%@%d%@%d%@",nombreAleatoire1,@" - ",
        nombreAleatoire2,@" = ",resultat,@"\n
        (Vous avez mis ",m,@")\n\n"];
    }
    texteFaux = [texteFaux stringByAppendingString:texteAjout];
    [self suivant:self];
    self.vraiFaux.text = [NSString stringWithFormat:
    @"%@%d",@"Faux, la réponse est ",resultat];
    self.vraiFaux.textColor = [UIColor redColor];
    faux++;
}
```

Toute cette partie du programme s'exécute uniquement lorsque l'utilisateur entre une réponse fausse. Dans ce cas-là, on définit `texteAjout` comme un texte ayant la réponse fausse écrite à l'intérieur. Après cela, on modifie le texte `texteFaux` en y ajoutant cette réponse fausse. Ce texte sera affiché dans le pop-up des statistiques. De plus, on informe l'utilisateur que sa réponse est fausse et on lui indique la bonne solution.

```
else {

    [self suivant:nil];
    self.vraiFaux.text = @"La réponse est correcte";
    self.vraiFaux.textColor = [UIColor greenColor];
```

```
            vrai ++;
        }
    }
}
@end
```

Pour finir, cette partie ne s'exécute que lorsque la réponse entrée par l'utilisateur est correcte. Dans ce cas-là, le programme recrée un calcul en appelant la méthode `suivant` et informe que la réponse est correcte. Il ajoute aussi un à la valeur du nombre de réponses justes.

2.2.2 Ce que nous avons appris

Grâce à cette application, nous avons appris les rudiments de la programmation en Objective-C, que ce soit le langage en lui-même, le fonctionnement global d'Xcode ou la création d'interfaces avec le Storyboard [3]. Nous savons maintenant par exemple utiliser les pop-up, les tableaux, les sélecteurs, activer ou non des boutons, et modifier des textes. La majorité de ces éléments nous seront utiles pour développer notre application finale.

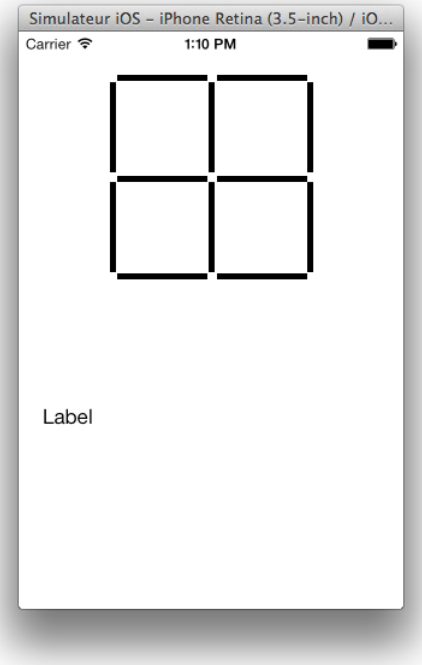
3. Le Storyboard est une page dans Xcode grâce à laquelle on peut créer l'interface d'une application facilement en y disposant différents éléments.

Chapitre 3

Les premiers modèles de la Pipopipette

3.1 Le modèle 2x2

Pour commencer, nous avons voulu créé notre jeu en petit format. Nous avons donc créé une application avec un maximum de 4 points par partie uniquement dans le but de tester différents codes et voir si l'application fonctionnait avec ces différents codes.



Interface du premier modèle.

Notre première idée pour coder cette application était de considérer chaque trait comme étant un bouton, et à chaque fois qu'on appuyait sur un bouton il changeait de couleur en fonction de si c'était le joueur rouge ou le joueur bleu qui avait appuyé. Mais cette méthode n'était pas très efficace et comportait pleins d'inconvénients. Nous allons vous présenter le code de cette application, puis nous allons faire un petit compte rendu sur les problèmes rencontrés.

3.1.1 Le commentaire du code : le modèle 2x2

```
- (void)viewDidLoad
{
    [super viewDidLoad];
    c1 = 0;
    c2 = 0;
    c3 = 0;
    c4 = 0;
    tour = 0;

    nombreDePointsBleu = 0;
    nombreDePointsRouge = 0;

}
```

Dans cette première partie de notre programme, nous déclarons les variables dont nous aurons besoin. `c1, c2, c3, c4` étant la valeur de chaque carré entre les boutons. Cette valeur compte le nombre de traits ayant été appuyés autour du carré en question. Elle est nulle au début tout comme le nombre de point de chaque joueur.

```
- (void) fonctionBords:(int)cX andOther: (UIButton *) btnX andOther:
(UIImageView *) carreX andOther: (UILabel *) texte {
    if (tour == 0) {

        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor blueColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"bleu.jpg"];
            nombreDePointsBleu ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge, @" Pour Rouge"];

        }
        else if (cX != 4) {
            tour = 1;
        }
    }
    else if (tour == 1) {
        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor redColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"rouge.jpg"];
            nombreDePointsRouge ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge, @" Pour Rouge"];
```

```
        }
        else if (cX != 4) {
            tour = 0;
        }
    }
}
```

Toute cette partie du programme est une fonction. Elle n'est utilisée que lorsque le joueur appuie sur un bouton du bord. Si le nombre `tour` est à zéro c'est au tour du joueur bleu de jouer. Au contraire si elle vaut un, c'est au tour du joueur rouge de jouer. Une fois qu'un joueur appuie sur un bouton, on désactive ce bouton grâce à la commande `btnX.enabled = NO` et on colorie ce bouton de la couleur du joueur qui a appuyé dessus. Si cette action fait passer la valeur du carré adjacent à quatre (ce qui correspond à la fermeture d'un carré) on ajoute un point au joueur qui a fermé le carré. Puis on affiche dans une zone de texte le score actuel. Sinon, on fait passer la valeur de `tour` à un. Dans la deuxième partie du programme on fait exactement la même chose lorsque c'est le deuxième joueur qui joue.

```
- (void) fonctionInterieur:(int)cX andOther: (int)cY andOther:
(UIButton *)btnX andOther: (UIImageView *)carreX andOther:
(UIImageView *)carreY andOther:(UILabel *)texte{
    if (tour == 0) {
        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor blueColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"bleu.jpg"];
            nombreDePointsBleu ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
           nombreDePointsRouge, @" Pour Rouge"];
        }
        if (cY == 4) {
            carreY.image = [UIImage imageNamed:@"bleu.jpg"];
            nombreDePointsBleu ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge, @" Pour Rouge"];
        }
        if (cX != 4 && cY != 4){
            tour = 1;
        }
    }
    else if (tour == 1) {
        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor redColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"rouge.jpg"];
```

```
            nombreDePointsRouge ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge, @" Pour Rouge"];
        }
        if (cY == 4) {
            carreY.image = [UIImage imageNamed:@"rouge.jpg"];
            nombreDePointsRouge ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge, @" Pour Rouge"];
        }
        if (cX != 4 && cY != 4){
            tour = 0;
        }
    }
}
```

Voici à présent la deuxième fonction de notre programme. Cette fonction n'est utilisée que lorsqu'un joueur appuie sur un des quatre boutons au centre. On retrouve presque la même fonction que précédemment, sauf que cette fois, il y a deux boutons adjacents à chaque trait. Et c'est pour cette raison qu'on vérifie deux conditions : si le premier carré adjacent a une valeur de quatre puis si le deuxième carré adjacent a une valeur de quatre. Puis, si aucune des deux conditions n'est remplie, on change la valeur de `tour` pour que l'autre joueur puisse jouer.

```
- (IBAction)actBtn1:(id)sender {
    [self fonctionBords:c1 andOther:_btn1 andOther:_carre1
    andOther:_texte];
    }
```

Voici maintenant l'endroit où notre première fonction est utilisée : dans la partie de code qui ne se déclenche uniquement lorsqu'un joueur appuie sur le premier bouton horizontal en haut à gauche. Cette fonction est reprise pour chaque bouton auquel il n'y a qu'un carré adjacent. C'est le même code pour tous les boutons cités précédemment en changeant à chaque fois le nom du bouton ainsi que le carré adjacent au trait.

```
- (IBAction)actBtn3:(id)sender {
    [self fonctionInterieur: c1 andOther: c3 andOther: _btn3
    andOther: _carre1 andOther: _carre3 andOther: _texte ];
}
```

Voilà à présent où apparait notre deuxième fonction. Elle est utilisée lorsque le joueur appuie sur des quatre boutons qui ont deux carrés adjacents. Ce même code est repris pour les trois autres boutons cités auparavant en changeant à chaque fois le nom du bouton ainsi que le nom des carrés adjacents.

3.1.2 Le compte-rendu du premier modèle

Le résultat est plutôt mitigé : l'application en elle-même fonctionne mais plusieurs gros inconvénients nous empêchent de continuer par cette méthode pour arriver à une application fonctionnelle. Premier gros inconvénient de notre application : la hitbox [1] des boutons est trop petite et nous devons appuyer plusieurs fois au même endroit pour arriver exactement sur le bouton. Ce qui rend le jeu long et pas amusant. Deuxième gros problème de cette méthode : il faut déclarer tous les boutons un par un dans le code ainsi que coder une action pour chaque bouton. Dans notre application 2x2, le problème n'est pas si dérangeant que ça vu qu'il n'y a que douze boutons. Mais si nous voulons faire une application de 5x5 ou même de 6x12, nous devrons alors passer beaucoup d'heures dessus pour, au final, pas grand-chose. Et c'est à cause de ces inconvénients que nous avons continué à chercher des solutions pour la future application. A ce moment-là, nous nous sommes rendu compte que le développement de cette application n'allait pas être aussi simple qu'on le pensait. Notre idée de base pour faire fonctionner ce jeu n'étant vraiment pas optimisée, nous avons dû repenser en entier la manière dont on allait créer cette application

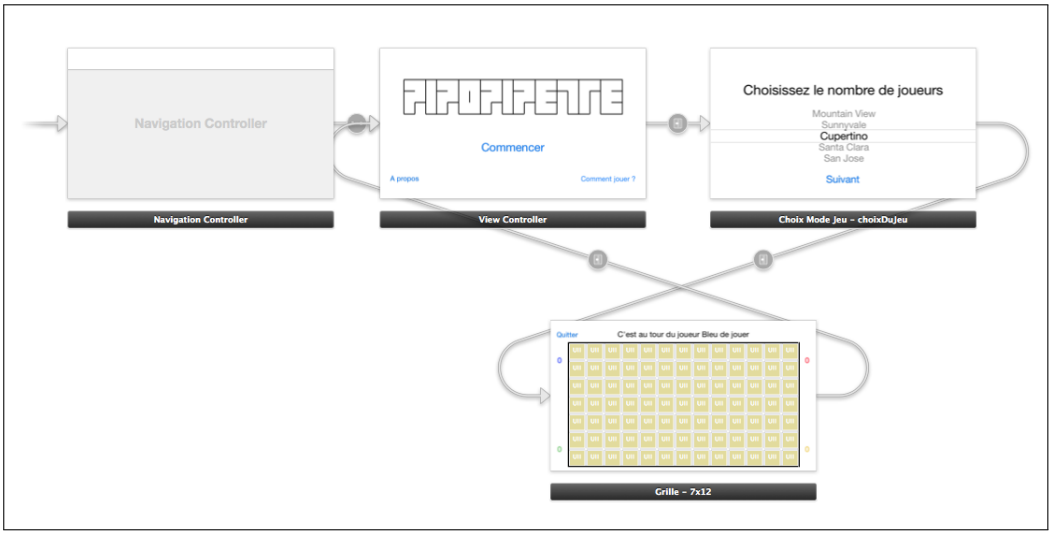
3.2 Le modèle 3x3

Au vu du résultat mitigé de notre première idée, nous avons décidé de continuer sur un modèle légèrement plus grand de notre jeu en essayant de résoudre les inconvénients qu'avait notre première application. Notre premier gros inconvénient était la hitbox des boutons. Pour corriger ce problème nous avons remplacé les boutons par un détecteur de position. Grâce à ceci, nous pourrons déterminer le trait le plus proche d'où le joueur appuie et ainsi supprimer les boutons. Nous avons créé une grille de jeu fonctionnant de cette manière dans une application toute simple. Cela nous a permis de créer le code de la grille de jeu. Nous n'allons pas vous présenter cette application, car son code est plus ou moins le même dans notre application finale.

1. Hitbox : zone de collision dans laquelle il faut appuyer pour activer le bouton. Le mot francophone est "masque de collision".

Chapitre 4

L'application finale



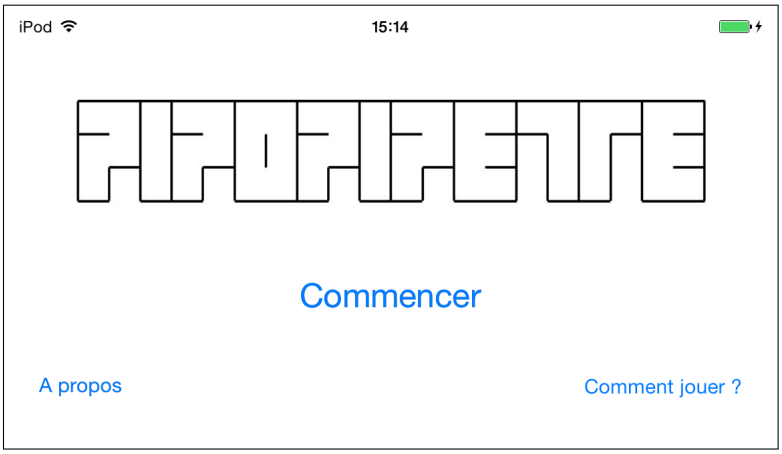
Voici à quoi ressemble le Storyboard

4.1 L'interface et la présentation du code

Tout d'abord nous avons fait un schéma sur papier pour savoir comment nous allions placer nos éléments sur l'application. Une fois ce schéma fini, nous avons placé tous nos éléments puis nous avons commencé à coder l'interface. Mais avant cela nous avions déjà réfléchi à comment allait fonctionner l'application, en faisant une esquisse du code sur papier. Nous avons commencé par créer les éléments les plus importants de l'application. Une fois cela fait, nous avons ajouté quelques fonctionnalités supplémentaires, comme par exemple les sons et le mode un joueur. Nous avons également créé un icône pour l'application, en utilisant un simple programme d'édition d'image.

L'icône de l'application

4.1.1 La page d'accueil



Page d'accueil

Ceci est la page qui s'affiche lorsqu'on lance l'application. C'est l'accueil, à partir duquel s'offrent à l'utilisateur plusieurs choix. Il peut soit commencer une partie, soit avoir différentes informations, telles que les règles du jeu ainsi que quelques informations sur l'application en elle-même.

Nous allons donc commencer par expliquer le code de cette page, c'est à dire le fichier `ViewController.m`.

```
- (BOOL) shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
toInterfaceOrientation {
    return NO;
}
```

Donc cette première partie de notre application finale, on commence par définir la rotation de l'application en mode paysage. On a mis notre application dans ce sens car cela nous paraissait plus intuitif et plus pratique de faire le jeu en mode paysage.

```
- (void)viewWillAppear:(BOOL)animated
{
    [self.navigationController setNavigationBarHidden:YES animated:YES];
}

- (void)viewWillDisappear:(BOOL)animated
{
    [self.navigationController setNavigationBarHidden:NO animated:YES];
}
```

Avec la manière dont nous avons fait notre application, des barres de navigations apparaissaient à chaque page. Sur cette barre en question, on affiche en général le titre de la page. Or, cette barre n'étant pas nécessaire sur la page d'accueil, nous avons décidé de l'enlever, mais nous voulions néanmoins la garder pour la page se trouvant après. Nous

avons donc, dans cette partie du programme, définit deux méthodes pour gérer cette barre. La première, `viewWillAppear` masque cette barre quand la page apparaît, et la deuxième, `viewWillDisappear` la réactive lorsqu'on quitte la page d'accueil.

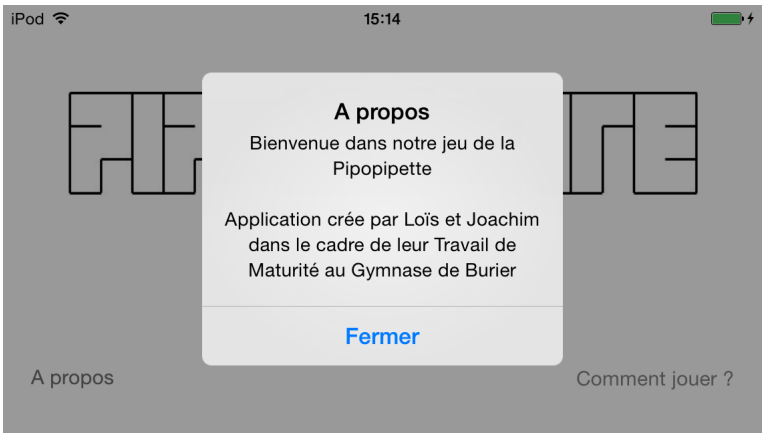
```
- (void)viewDidLoad
{
     [self.navigationItem setBackBarButtonItem:[[UIBarButtonItem alloc]
         initWithTitle:@"Retour" style:UIBarButtonItemStyleBordered target:
         nil action:nil]];

}
```

La méthode d'après définit le bouton retour que l'on trouvera sur la page suivante. Par défaut, ce bouton se nomme "Back", nous avons donc voulu le renommer en "Retour".

```
- (IBAction)aPropos:(id)sender
{
    UIAlertView *aPropos = [[UIAlertView alloc] initWithTitle:@"A propos"
        message:@"Bienvenue dans notre jeu de la Pipopipette\n\nApplication
        crée par Loïs et Joachim dans le cadre de leur Travail de Maturité
        au Gymnase de Burier" delegate:nil cancelButtonTitle:@"Fermer"
        otherButtonTitles:nil];
    [aPropos show];
}
```

La méthode présentée ici est déclenchée lorsque l'utilisateur appuie sur le bouton "A propos" en bas à gauche du menu principal de notre application. Lorsque le joueur appuie sur ce bouton, un pop-up s'affiche dans une notification expliquant par qui et dans quel but cette application a été créée.

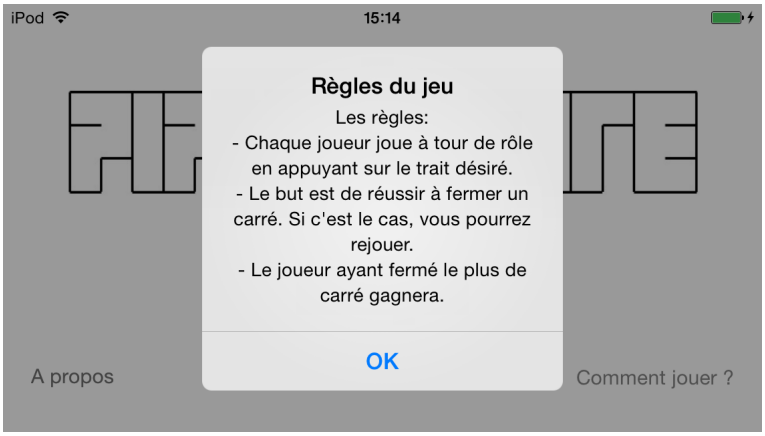


Le pop-up "A propos"

```
- (IBAction)commentJouer:(id)sender
{
    UIAlertView *commentJouer = [[UIAlertView alloc] initWithTitle:
        @"Règles du jeu" message:@"Les règles: \n- Chaque joueur joue à
```

```
        tour de rôle en appuyant sur le trait désiré. \n- Le but est de
        réussir à fermer un carré. Si c'est le cas, vous pourrez rejouer.
        \n- Le joueur ayant fermé le plus de carré gagnera." delegate:nil
        cancelButtonTitle:
        @"OK" otherButtonTitles:nil];
    [commentJouer show];
}
```

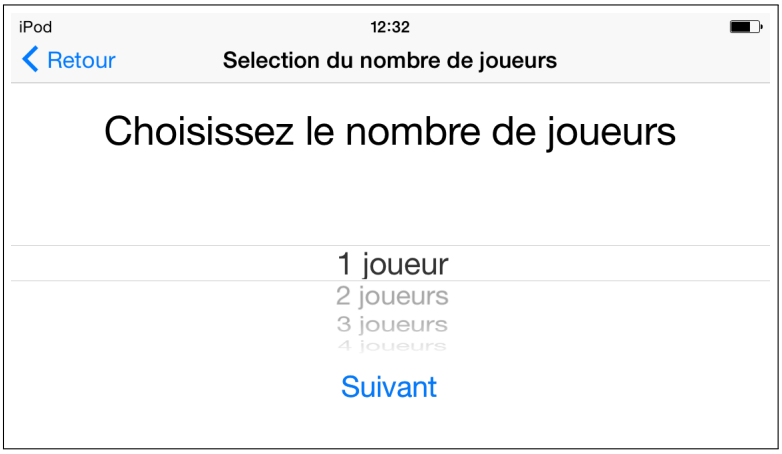
C'est le même principe dans cet extrait-ci, cette méthode se déclenché lorsque l'utilisateur appuie sur le bouton "Comment jouer ?" en bas à droite du menu principal de notre application. Lorsque le joueur appuie sur ce bouton, un pop-up s'affiche dans une notification expliquant les règles du jeu.



Le pop-up affichant les règles du jeu

4.1.2 La page du choix du nombre de joueurs

Deuxièmement, nous allons commenter la fenêtre qui s'ouvre lorsqu'on appuie sur le bouton jouer dans la fenêtre précédente. Cette fenêtre nous permet de choisir le nombre de joueurs pour une partie.



La page permettant de choisir le nombre de joueurs

Le code de cette page se trouve dans le fichier `choixModeJeu.m`.

```
- (void)viewDidLoad
{
    self.navigationItem.title = @"Selection du nombre de joueurs";
    self.tabModes = [[NSArray alloc] initWithObjects: @"1 joueur",
        @"2 joueurs", @"3 joueurs", @"4 joueurs", nil];

    self.modeDeJeu = 1
}
```

Dans cette partie du programme, on donne le nom de la page qui sera affichée sur la barre de navigation en haut, puis on crée un tableau avec les noms des modes de jeu avec deux, trois ou quatre joueurs. Ce tableau sera utile pour le sélecteur, c'est à dire l'élément dans l'interface qui permet de choisir le nombre de joueurs. De plus, on définit le nombre de joueurs par défaut à 1.

```
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;
{
    return 1;
}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:
    (NSInteger)component;
{
    return 4;
}

-(NSString *)pickerView:(UIPickerView *)pickerView titleForRow:(NSInteger)
    row forComponent:(NSInteger)component
{
    return [self.tabModes objectAtIndex:row];
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)
    row inComponent:(NSInteger)component
{
    switch(row)
    {
        case 0:
            _modeDeJeu = 1;
            break;
        case 1:
            _modeDeJeu = 2;
            break;
        case 2:
            _modeDeJeu = 3;
            break;
```

```
        case 3:
            _modeDeJeu = 4;

    }
}
```

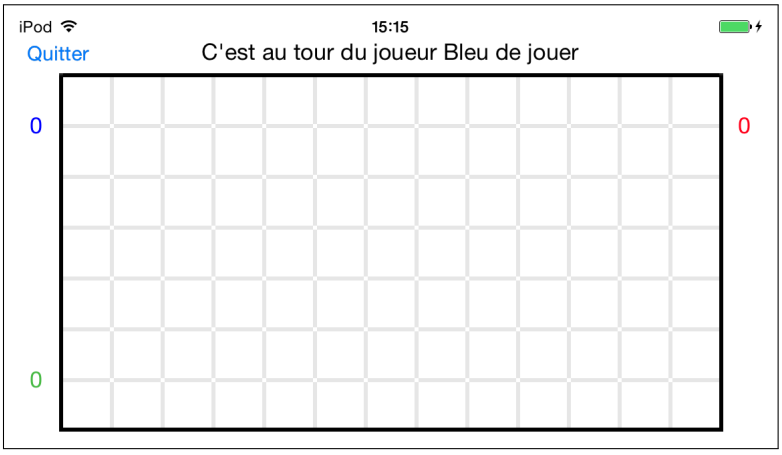
Dans cette partie, on initialise le sélecteur. La première méthode initialise le nombre de colonnes (dans notre cas une seule), on définit ensuite le nombre de lignes de ce sélecteur dans la deuxième méthode (dans notre cas quatre) et enfin on donne le nom de chaque ligne en fonction du tableau créé précédemment avec le nombre de joueurs. La dernière méthode définit le nombre de joueurs en fonction de la position du sélecteur. Par exemple, si le sélecteur passe en position deux, le mode de jeu choisi sera celui avec deux joueurs.

```
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    grille *destination = [segue destinationViewController];
    destination.nombreDeJoueurs = _modeDeJeu;
}
@end
```

Dans le bout de code ci-dessus, on fait en sorte que la page de la grille connaisse la valeur de `modedejeu`. Cette variable permettra au code présent avec la grille de savoir le nombre de joueurs, il pourra donc s'adapter à cela en conséquence.

4.1.3 La page de la grille de jeu

Nous allons passer à la partie la plus importante de notre application, à savoir la grille de jeu. A la base, nous avions créé deux grille ; celle-ci ainsi qu'un grille de 3x3. Cette petite grille en question nous a été utile pour créer le code. Après coup, nous avons décidé d'enlever cette version de la grille, car il n'y avait aucun intérêt à jouer sur une grille aussi petite. Mais avant de le détailler le code, il faut d'abord comprendre le principe que nous avons utilisé pour réaliser cette application. Voici, ci-dessous, à quoi ressemble la grille de jeu.



La grille de jeu

Cette grille est composée de plusieurs objets. Tous les traits du quadrillage ainsi que les carrés entre ceux-ci sont des images. Nous les avons placés à la main, une par une. C'est le moyen le plus simple, bien qu'il soit un peu fastidieux. Nous avons ensuite créé trois `Outlet collection` qui servent à mettre des objets dans des tableaux. Un de ceux-ci se nomme `traits`. Nous avons mis dans ce tableau toutes les images représentant les traits du quadrillage. Les deux autres, `premierCarre` et `deuxiemeCarre` vont permettre de savoir quels carrés vides sont à côté de quels traits. Nous avons donc ajouté les images correspondantes aux carrés dans ces deux tableaux de manière à ce que le premier élément du tableau `premierCarre` soit un des carrés adjacents au premier élément du tableau `traits`. Mais également de manière à ce que le premier élément du tableau `deuxiemeCarre` soit le second des carrés adjacents au premier élément du tableau `traits`. Nous avions donc grâce à ces trois tableaux une sorte de base de données qui nous permettait de savoir quel carré est à côté de quel trait. Nous avons fait de cette manière car notre idée était de faire en sorte que lorsqu'on appuie sur un des traits, on augmentait la valeur attribuée aux carrés adjacents de un, tel que, lorsqu'on aura appuyé sur les quatre traits adjacents, le programme remplira le carré de la couleur du joueur ayant fermé le carré.

Maintenant, nous allons commenter le code qui s'exécute lorsque l'on fait une partie.

Le fichier grille.h

Ce fichier contient plusieurs variables et objets utiles au fonctionnement du code. Nous allons les détailler afin de mieux comprendre le code et le fonctionnement du programme.

```
#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>


@interface grille : UIViewController <AVAudioPlayerDelegate>

{
    AVAudioPlayer * audioPlayer;
}
```

Ici, nous avons ajouté quelques lignes de manière à pouvoir mettre des sons dans l'application.

```
@property (nonatomic) int nombreDeJoueurs;

@property (strong, nonatomic) IBOutletCollection(UIImageView)
    NSMutableArray *traits;

@property (strong, nonatomic) IBOutletCollection(UIImageView)
    NSMutableArray *premierCarre;

@property (strong, nonatomic) IBOutletCollection(UIImageView)
```

```
    NSMutableArray *deuxiemeCarre;

@property NSMutableArray *tableauDistances;

@property NSMutableArray *scores;
```

Comme son nom l'indique, la variable `nombreDeJoueurs` contient le nombre de joueurs. Les trois objets suivants sont les tableaux dont nous avons parlé auparavant.

```
@property (weak, nonatomic) IBOutlet UILabel *scoreBleu;
@property (weak, nonatomic) IBOutlet UILabel *scoreRouge;
@property (weak, nonatomic) IBOutlet UILabel *scoreVert;
@property (weak, nonatomic) IBOutlet UILabel *scoreJaune;
@property (weak, nonatomic) IBOutlet UILabel *texte;

@property NSMutableArray *podium;
@property NSString *nomDuMeilleur;
@property UIImageView *traitUnJoueur;
@property UIImageView *traitUnIA;

@end
```

Les quatre premiers objets sont des textes affichant le nombre de points de chaque joueur. Quant à l'objet nommé `texte`, c'est un texte qui indique quelle personne doit jouer. `podiumGrande` et `nomDuMeilleurGrande` sont utilisées pour les scores à la fin de la partie. Les quatre derniers objets sont pour le mode un joueur (contre l'IA [1]).

Le fichier grille.m

Maintenant, nous allons détailler le fichier contenant le code principal de la grille.

```
int indexMin;
int tour = 0;
int nombreDePointsBleu = 0;
int nombreDePointsRouge = 0;
int nombreDePointsVert = 0;
int nombreDePointsJaune = 0;
int tailleImage;
int nombreDeJoueurs;
int placeDuMeilleur;
int placeLibre = 0;
int positionIA;
int tourDeLIA = 1;
int dejaFait;
bool timerAllume;
int compteur;
```

1. Intelligence artificielle qui joue à la place d'un joueur

Dans cette première partie du programme on déclare les variables dont nous aurons besoin mais nous nous y intéresserons plus amplement lorsque nous les utiliserons. Nous les déclarons toutes ici pour pouvoir les utiliser plus facilement. Nous préférons avoir le plus de place possible pour créer l'interface.

```
- (void)viewWillAppear:(BOOL)animated
{
    [self.navigationController setNavigationBarHidden:
        YES animated:YES];
}
```

Avec le code ci-dessus on masque la barre d'état pour qu'elle n'apparaisse pas dans cette partie de l'application.

```
- (void)viewDidLoad {

    [super viewDidLoad];

    AudioSessionInitialize (NULL, NULL, NULL, (__bridge void *)self);
    UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback;
    AudioSessionSetProperty (kAudioSessionProperty_AudioCategory,
        sizeof (sessionCategory), &sessionCategory);
    NSData *soundFileData;
    soundFileData = [NSData dataWithContentsOfURL:[NSURL fileURLWithPath:
        [[NSBundle mainBundle] pathForResource:@"ding.mp3" ofType:NULL]]];
    audioPlayer = [[AVAudioPlayer alloc] initWithData:
        soundFileData error:NULL];
    audioPlayer.delegate = self;
    [audioPlayer setVolume:1.0f];

    self.podiumGrande = [[NSMutableArray alloc]init];
    self.scores = [[NSMutableArray alloc] init];
    self.tableauDistances = [[NSMutableArray alloc] init];

    if (self.nombreDeJoueurs == 2) {
        self.scoreVert.text = @"";
        self.scoreJaune.text = @"";
    }
    else if (self.nombreDeJoueurs == 3) {
        self.scoreJaune.text = @"";
    }
    else if (self.nombreDeJoueurs == 1){
        self.scoreVert.text = @"";
        self.scoreJaune.text = @"";
    }

    srand(time(NULL));
```

```
    if (self.nombreDeJoueurs == 1) {
        UIAlertView *infoCouleurs = [[UIAlertView alloc] initWithTitle:
            @"Information" message:[NSString stringWithFormat:@"Vous êtes
            le joueur Bleu, le joueur Rouge est geré automatiquement."]
            delegate: nil cancelButtonTitle:@"OK" otherButtonTitles:nil,
            nil];
        [infoCouleurs show];
    }
}
```

Cette partie s'exécute lorsque la grille s'affiche. Les huit premières lignes définissent un son qui est joué lorsqu'un carré est complété. On initialise les tableaux pour les scores de fin de partie. Ensuite, on modifie l'interface selon le nombre de joueurs. Par exemple, s'il n'y a que deux joueurs, on cache le score des joueurs vert et jaune, qui sont respectivement le troisième et le quatrième joueurs. Lorsque le mode un joueur est choisi, on affiche un pop-up donnant des informations supplémentaires sur le mode un joueur.

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    if (timerAllume == FALSE) {
        dejaFait = 0;

        UITouch *touch = [touches anyObject];
        CGPoint point = [touch locationInView:self.view];
        CGFloat pointX = point.x;
        CGFloat pointY = point.y;
        int x = pointX;
        int y = pointY;
        int positionXImage;
        int positionYImage;
```

Dans cette méthode, on va capter l'endroit où le joueur a touché l'écran. La première partie de cette méthode consiste simplement à déclarer les variables contenant les positions de l'endroit où on appuie ainsi que les positions des carrés. Nous expliquerons plus tard la raison pour laquelle nous remettons la variable `dejaFaitGrande` à zéro. La condition `timerAllume == FALSE` empêchera le joueur d'appuyer lorsque ce sera au tour de l'IA de jouer.

```
        for (UIImageView *image in _traits) {

            CGPoint positionBase = image.frame.origin;
            positionXImage = positionBase.x + image.frame.size.width/2;
            positionYImage = positionBase.y + image.frame.size.height/2;
            tailleImage = image.frame.size.width/2 +
                image.frame.size.height/2;
```

```
        float distance = sqrtf(pow(positionXImage - x, 2) +
            (pow(positionYImage - y, 2)));
        NSNumber *distanceNombre = [NSNumber numberWithFloat:distance];
        [_tableauDistances addObject:distanceNombre];
    }
```

Dans cette partie, on parcourt le tableau contenant tous les traits pour trouver leurs cordonnées. La variable `distance` contient la distance entre le centre d'un trait et l'endroit où on a appuyé. On ajoute cette valeur dans le tableau `tableauDistances`.

```
    int valeurMin = 1000;
    for (NSNumber * intMin in _tableauDistances) {

        int valeurActuelle = [intMin intValue];

        if (valeurActuelle < valeurMin) {

            valeurMin = valeurActuelle;
        }
    }

    for (NSNumber *laDistance in _tableauDistances){

        if (dejaFait == 0) {

            if (valeurMin == [laDistance intValue] & valeurMin <
                tailleImage){

                dejaFait = 1;

                indexMin = [_tableauDistances indexOfObject:laDistance];

                [[_premierCarre objectAtIndex:indexMin] setTag:
                    ([[_premierCarre objectAtIndex:indexMin]
                    tag ] + 1)];
                [[_deuxiemeCarre objectAtIndex:indexMin] setTag:
                    ([[_deuxiemeCarre objectAtIndex:indexMin]
                    tag ] + 1)];

                NSInteger tag1 = [[_premierCarre objectAtIndex:
                    indexMin] tag ];
                int numeroTag1 = tag1;

                NSInteger tag2 = [[_deuxiemeCarre objectAtIndex:
                    indexMin] tag ];
                int numeroTag2 = tag2;
```

La première boucle `for` cherche la valeur la plus petite du tableau `tableauDistances`. La variable `valeurMin` stocke cette valeur. Dans la deuxième boucle, on regarde la position de cette valeur et on stocke cet emplacement dans la variable `indexMin`. Dans certains cas, si on appuyait à équidistance de deux traits, le code contenu dans deuxième boucle `for` s'exécutait deux fois. Si le code de cette boucle s'est déjà exécuté une fois, elle ne pourra plus se réexécuter une deuxième fois. La variable `dejaFait` qu'on a remis à zéro en appuyant sur un trait permet au code contenu dans cette boucle de se réexécuter.

Dès lors, on peut savoir sur quel trait le joueur a voulu appuyer. Puis on ajoute un au tag [2] des deux carrés adjacents à ce trait. Ce tag est la valeur dont nous parlions auparavant qui va augmenter de un à chaque fois que l'on appuie sur un des traits à côté d'un carré. On utilise donc ce tag comme une variable associée à chaque carré. Plus loin dans le code, une condition va regarder la valeur de ce tag, et si celui-ci vaudra 4, cela voudra dire que tous les traits adjacents auront été appuyés.

```
if (_nombreDeJoueurs == 2) {

    [self fonction2Joueurs:numeroTag1 andOther:
        numeroTag2 andOther: [_traits objectAtIndex:
        indexMin] andOther:[_premierCarre objectAtIndex:
        indexMin] andOther:[_deuxiemeCarre objectAtIndex:
        indexMin]];
}
else if (_nombreDeJoueurs == 3) {

    [self fonction3Joueurs:numeroTag1 andOther:
       numeroTag2 andOther: [_traits objectAtIndex:
       indexMin] andOther:[_premierCarre objectAtIndex:
       indexMin] andOther:[_deuxiemeCarre objectAtIndex:
       indexMin]];
}
else if (_nombreDeJoueurs == 4) {

    [self fonction4Joueurs:numeroTag1 andOther:
        numeroTag2 andOther: [_traits objectAtIndex:
        indexMin] andOther:[_premierCarre objectAtIndex:
        indexMin] andOther:[_deuxiemeCarre objectAtIndex:
        indexMin]];
}
```

Dans ce bout de code, on regarde combien de joueurs il y a, puis on exécute la fonction correspondante. Nous allons maintenant vous détailler ces fonctions. Elles servent à colorer les carrés ainsi que les traits en fonction de ce que font les joueurs. Elles se trouvent au début du fichier `grille.m`.

2. Un tag est un nombre entier qui peut être utilisé pour reconnaître des objets visuels dans l'application.

```
- (void) fonction2Joueurs:(int)cX andOther: (int)cY andOther:
    (UIImageView *)image andOther: (UIImageView *)carreX andOther:
    (UIImageView *)carreY {

    if (leTour == 0) {

        image.image = [UIImage imageNamed:@"bleu.jpg"];

        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 1;
            self.texte.text = @"C'est au tour du Joueur Rouge de jouer";
        }
    }
    else if (leTour == 1) {
        image.image = [UIImage imageNamed:@"rouge.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 0;
            self.texte.text = @"C'est au tour du Joueur Bleu de jouer";
        }
    }
    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];
}
```

Cette partie est la fonction principale pour le mode deux joueurs. Si la variable `tour` est à zéro, c'est au joueur bleu de jouer. Et si le tag d'un carré obtient la valeur de quatre, autrement dit que ce carré est complété, le nombre de points du joueur bleu augmente d'un. Au contraire, si aucun carré n'est complété, la valeur de `tour` passe à un et c'est alors au joueur rouge de jouer. Dans le cas où le joueur rouge complète un carré, alors ce sera son nombre de points qui augmentera d'un. Mais s'il ne complète pas de carré, la valeur de `tour` repasse à zéro et c'est à nouveau au joueur bleu de jouer. De plus, on joue le son définit précédemment lorsqu'un carré est rempli. Puis on actualise le score à l'écran.

```
- (void) fonction3Joueurs:(int)cX andOther: (int)cY andOther:
    (UIImageView *)image andOther: (UIImageView *)carreX andOther:
    (UIImageView *)carreY{

    if (leTour == 0) {
        image.image = [UIImage imageNamed:@"bleu.jpg"];

        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 1;
            self.texte.text = @"C'est au tour du Joueur Rouge de jouer";
        }
    }
    else if (leTour == 1) {
        image.image = [UIImage imageNamed:@"rouge.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 2;
            self.texte.text = @"C'est au tour du Joueur Vert de jouer";
```

```
        }
    }
    else if (leTour == 2) {
        image.image = [UIImage imageNamed:@"vert.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 0;
            self.texte.text = @"C'est au tour du Joueur Bleu de jouer";
        }
    }
    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];
    self.scoreVert.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuVert];
}
```

Cette partie ne se déclenche que s'il y a 3 joueurs. C'est exactement le même principe qu'à deux joueurs. Si un joueur complète un carré, on ajoute un point à son score. Mais si ce même joueur ne ferme pas de carré, on change la valeur de `tour` et c'est donc au joueur suivant de jouer. La couleur du joueur numéro trois est le vert.

```
- (void) fonction4Joueurs:(int)cX andOther: (int)cY andOther:
    (UIImageView *)image andOther: (UIImageView *)carreX andOther:
    (UIImageView *)carreY {

    if (leTour == 0) {
        image.image = [UIImage imageNamed:@"bleu.jpg"];

        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreBleu.jpg"];
```

```
            nombreDePointsDuBleu ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 1;
            self.texte.text = @"C'est au tour du Joueur Rouge de jouer";
        }
    }
    else if (leTour == 1) {
        image.image = [UIImage imageNamed:@"rouge.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 2;
            self.texte.text = @"C'est au tour du Joueur Vert de jouer";
        }
    }
    else if (leTour == 2) {
        image.image = [UIImage imageNamed:@"vert.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 3;
            self.texte.text = @"C'est au tour du Joueur Jaune de jouer";
        }
    }
    else if (leTour == 3) {
        image.image = [UIImage imageNamed:@"jaune.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreJaune.jpg"];
            nombreDePointsDuJaune ++;
```

```
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreJaune.jpg"];
            nombreDePointsDuJaune ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 0;
            self.texte.text = @"C'est au tour du Joueur Bleu de jouer";
        }
    }
    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];
    self.scoreVert.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuVert];
    self.scoreJaune.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuJaune];
}
```

Cette partie est le code pour le mode quatre joueurs, le code est pareil qu'avec deux ou trois joueurs. La couleur du joueur numéro quatre est le jaune. Maintenant, nous allons continuer où nous nous étions arrêtés avant de décrire ces fonctions.

```
                else if (_nombreDeJoueurs == 1){

                    _traitUnJoueur = [_traits objectAtIndex:
                        indexMin];
                    _traitUnJoueur.image = [UIImage imageNamed:
                        @"bleu.jpg"];

                    if ([[_premierCarre objectAtIndex:indexMin] tag] == 4) {

                        UIImageView *carreUnJoueur =
                            [_premierCarre objectAtIndex:indexMin];
                            carreUnJoueur.image = [UIImage imageNamed:
                            @"CarreBleu.jpg"];
                        nombreDePointsDuBleu ++;
                        [audioPlayer play];
                    }

                    if ([[_deuxiemeCarre objectAtIndex:indexMin]tag] == 4) {

                        UIImageView *carreDeuxJoueur =
                            [_deuxiemeCarre objectAtIndex:indexMin];
                            carreDeuxJoueur.image = [UIImage imageNamed:
```

```
                @"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
            [audioPlayer play];
        }

        if ([[_premierCarre objectAtIndex:indexMin] tag] != 4 &&
            [[_deuxiemeCarre objectAtIndex:indexMin]tag] != 4) {

            tourDeLIA = 1;
        }

        [_premierCarre removeObjectAtIndex:indexMin];
        [_deuxiemeCarre removeObjectAtIndex:indexMin];
        [_traits removeObjectAtIndex:indexMin];

        self.scoreBleu.text = [NSString stringWithFormat: @"%d",
            nombreDePointsDuBleu];
        self.scoreRouge.text = [NSString stringWithFormat: @"%d",
            nombreDePointsDuRouge];
```

Si on est en mode un joueur, on commence par colorer le trait sur lequel le joueur a appuyé. Si le joueur ferme un carré, on le colorie en bleu, on joue un son et on ajoute un point au joueur. Mais s'il n'a pas fermé de carré, on met la variable `tourDeLIA` à un, ce qui signifie que c'est à l'IA de jouer. On supprime les références du trait touché et de ses carrés adjacents de leurs tableaux, pour qu'on ne puisse par réactiver ce trait. Puis, on actualise les scores sur l'interface.

```
        if (tourDeLIA == 1) {

            self.texte.text = @"C'est au tour de l'IA de jouer";

            timerAllume = TRUE;

            NSTimer *timer = [NSTimer
               scheduledTimerWithTimeInterval: 1.0
               target: self selector:@selector(onTick:)
               userInfo: nil repeats:YES];
        }
        else if ([_traits count] == 0) {
            [self fin];
        }
    }
```

Si c'est au tour de l'IA de jouer, on change le texte en haut de l'écran puis on passe la variable `timerAllume` à `TRUE` de manière à ce que le joueur ne puisse pas appuyer sur un

trait lors du tour de l'IA. Ce blocage s'effectue au début de cette méthode. Après cela, on lance un minuteur [3] qui va appeler la méthode `onTick` toutes les secondes. Si ce n'est pas au tour de l'IA de jouer, et que tous les carrés ont été remplis, on appelle la méthode `fin`, que nous vous décrirons plus tard. Nous allons maintenant vous détailler la méthode `onTick` .

```
- (void) onTick:(NSTimer *)timer {

    placeLibre = 0;

    for (UIImageView *traitVide in _premierCarre) {
        if ([traitVide tag] == 3) {
            placeLibre = 1;
            break;
        }
    }

    if (placeLibre == 0) {
        for (UIImageView *autreTraitVide in _deuxiemeCarre) {
            if ([autreTraitVide tag] == 3) {
                placeLibre = 2;
                break;
            }
        }
    }
```

Dans cette méthode, on commence par regarder dans les tableaux contenant les carrés s'il y a un carré avec un tag de trois. Si c'est le cas dans le premier tableau, on passe la variable `placeLibre` à un. On passe cette valeur à deux dans le cas où ce carré se trouve dans le deuxième tableau. Par défaut, cette valeur vaut zéro, donc, s'il n'y a pas de carré avec un tag de trois, cette valeur reste à zéro.

```
    if (placeLibre == 1) {
        for (UIImageView *celleATrois in _premierCarre) {
            if ([celleATrois tag] == 3) {
                positionIA = [_premierCarre indexOfObject:celleATrois];
                break;
            }
        }
    }

    else if (placeLibre == 2) {
        for (UIImageView *celleATrois in _deuxiemeCarre) {
            if ([celleATrois tag] == 3) {
                positionIA = [_deuxiemeCarre indexOfObject:celleATrois];
                break;
```

3. Qui équivaut au mot anglais "timer" que nous utiliserons plus souvent

```
        }
    }
}

else {
    compteur = 0;
    positionIA = (arc4random() % [_traits count]);
    while ([[_premierCarre objectAtIndex:positionIA] tag] == 2 ||
        [[_deuxiemeCarre objectAtIndex:positionIA] tag] == 2) {

        positionIA = (arc4random() % [_traits count]);
        compteur ++;
        if (compteur == 300) {
            break;
        }
    }
}
```

Dans ce bout de code, on cherche, s'il existe, l'index du carré avec un tag de trois dans son tableau. On stocke cet index dans la variable `positionIA`. Dans le cas contraire, on définit une position aléatoire. Si le carré choisi aléatoirement a une valeur de deux, on en choisit un nouveau au hasard. Si au bout de trois cents essais on a trouvé que des carrés avec une valeur de deux ou plus, on prend une position aléatoire. On peut considérer, après cet important nombre d'essais, qu'il ne reste pas de carré ayant une valeur inférieure à deux.

```
[[_premierCarre objectAtIndex:positionIA] setTag:([[_premierCarre
    objectAtIndex:positionIA] tag ] + 1)];

[[_deuxiemeCarre objectAtIndex:positionIA] setTag:([[_deuxiemeCarre
    objectAtIndex:positionIA] tag ] + 1)];

_traitUnIA = [_traits objectAtIndex:positionIA];
_traitUnIA.image = [UIImage imageNamed:@"rouge.jpg"];

if ([[_premierCarre objectAtIndex:positionIA] tag] == 4) {
    UIImageView *carreUnIA = [_premierCarre objectAtIndex:positionIA];
    carreUnIA.image = [UIImage imageNamed:@"CarreRouge.jpg"];
    nombreDePointsDuRouge ++;
    [audioPlayer play];
    tourDeLIA = 1;

}
if ([[_deuxiemeCarre objectAtIndex:positionIA]tag] == 4) {
    UIImageView *carreDeuxIA = [_deuxiemeCarre objectAtIndex:positionIA];
    carreDeuxIA.image = [UIImage imageNamed:@"CarreRouge.jpg"];
    nombreDePointsDuRouge ++;
```

```
        [audioPlayer play];
        tourDeLIA = 1;
    }

    else if  ([[_premierCarre objectAtIndex:positionIA] tag] != 4 &&
        [[_deuxiemeCarre objectAtIndex:positionIA]tag] != 4){
        tourDeLIA = 0;
    }

    [_premierCarre removeObjectAtIndex:positionIA];
    [_deuxiemeCarre removeObjectAtIndex:positionIA];
    [_traits removeObjectAtIndex:positionIA];

    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];

    if ([_traits count] ==  0) {
        [timer invalidate];
        timerAllume = FALSE;
        [self fin];
    }
    else if (tourDeLIA == 0) {
        [timer invalidate];
        timerAllume = FALSE;
        self.texte.text = @"C'est au joueur Bleu de jouer";
    }
}
```

Ici, on commence par ajouter un au tag des carrés adjacents au trait que l'IA a activé, puis on colore ce trait en rouge. Si l'IA a fermé un carré, on le colorie en rouge, on joue le son, on ajoute un au score et on met la variable `tourDeLIA` à un de manière à ce que l'IA rejoue. Dans le cas où l'IA ne ferme aucun carré, cette variable repasse à zéro, l'IA ne va donc pas rejouer et ce sera donc au tour du joueur. Puis on enlève les références, comme précédemment. L'avant dernière condition fait en sorte que la partie se finisse si la grille est complètement remplie. Pour cela, on désactive le timer de manière à ce que la méthode faisant jouer l'IA ne se relance pas, on passe la variable `timerAllume` à `FALSE` et on exécute la méthode `fin` que nous décrirons plus tard. Si ce n'est pas la fin de la partie mais que l'IA a fini de jouer, on arrête le timer, on passe la variable `timerAllume` à `FALSE` et on rechange le texte en haut de l'écran.

```
if (_nombreDeJoueurs != 1) {
    [_premierCarre removeObjectAtIndex:indexMin];
    [_deuxiemeCarre removeObjectAtIndex:indexMin];
    [_traits removeObjectAtIndex:indexMin];
```

```
                        if ([_traits count] == 0) {
                            [self fin];
                        }
                    }
                }
            }
        }
        [_tableauDistances removeAllObjects];
    }
}
```

Revenons maintenant à notre méthode principale, `touchesBegan`. Si on est dans un mode multi-joueurs, on commence par enlever le trait sur lequel un joueur vient d'appuyer ainsi que les carrés adjacents de leurs tableaux respectifs pour qu'on ne puisse plus appuyer dessus. Après cela, on exécute la méthode `fin` si la partie est terminée. On finit par enlever toutes les valeurs du tableau `tableauDistances`, pour pouvoir le réutiliser lors du prochain tour.

```
- (void) fin {

    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuBleu]];
    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuRouge]];
    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuVert]];
    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuJaune]];

    int valeurMax = -1;
    for (NSNumber * intMin in _scores) {

        int valeurActuelle = [intMin intValue];

        if (valeurActuelle > valeurMax) {

            valeurMax = valeurActuelle;
        }
    }

    if ([[_scores objectAtIndex:0]integerValue] == valeurMax) {
        [_podium addObject:[NSString stringWithFormat:@"Bleu"]];
    }
    if ([[_scores objectAtIndex:1]integerValue] == valeurMax) {
        [_podium addObject:[NSString stringWithFormat:@"Rouge"]];
    }
    if ([[_scores objectAtIndex:2]integerValue] == valeurMax) {
        [_podium addObject:[NSString stringWithFormat:@"Vert"]];
    }
    if ([[_scores objectAtIndex:3]integerValue] == valeurMax) {
```

```
        [_podium addObject:[NSString stringWithFormat:@"Jaune"]];
    }
```

Voici maintenant la méthode `fin`, qui ne se lance quand fin de partie. On commence par ajouter dans un tableau `scores` le nombre de point de chaque joueur. Après cela, la boucle `for` cherche le nombre de points le plus grand. Ensuite, on regarde pour chaque joueur s'il a le score maximum. Si c'est le cas, on ajoute son nom dans le tableau `podiumGrande`.

```
    if ([_podium count] == 1 ) {
            _nomDuMeilleur = [NSString stringWithFormat:
            @"%@ a gagné avec %d points",[_podium
            objectAtIndex:0],valeurMax ];
    }

    if ([_podium count] == 2 ) {
        _nomDuMeilleur = [NSString stringWithFormat:
            @"%@ et %@ sont en tête avec %d points",
            [_podium objectAtIndex:0],[_podium objectAtIndex:1],
            valeurMax];

    }
    if ([_podium count] == 3 ) {
        _nomDuMeilleur = [NSString stringWithFormat:
            @"%@, %@ et %@ sont en tête avec %d points",
            [_podium objectAtIndex:0],[_podium objectAtIndex:1],
            [_podium objectAtIndex:2], valeurMax];
    }
    if ([_podium count] == 4 ) {
        _nomDuMeilleur = [NSString stringWithFormat:
            @"%@, %@, %@ et %@ sont en tête avec %d points",
            [_podium objectAtIndex:0],[_podium objectAtIndex:1],
            [_podium objectAtIndex:2], [_podium objectAtIndex:3],
            valeurMax];
    }


    UIAlertView *fin = [[UIAlertView alloc] initWithTitle:
        @"Bravo !" message:[NSString stringWithFormat:
        @"%@",_nomDuMeilleur] delegate:nil cancelButtonTitle:
        @"OK" otherButtonTitles:nil, nil];

    // On définit un nouveau son

    AudioSessionInitialize (NULL, NULL, NULL,
    (__bridge void *)self);
    UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback;
    AudioSessionSetProperty (kAudioSessionProperty_AudioCategory,
```

```
        sizeof (sessionCategory), &sessionCategory);
    NSData *soundFileData;
    soundFileData = [NSData dataWithContentsOfURL:
        [NSURL fileURLWithPath:[[NSBundle mainBundle] pathForResource:
        @"fin.mp3" ofType:NULL]]];
    audioPlayer = [[AVAudioPlayer alloc] initWithData:soundFileData
        error:NULL];
    audioPlayer.delegate = self;
    [audioPlayer setVolume:1.0f];

    [audioPlayer play];

    [fin show];
    [_scores removeAllObjects];


}
- (void)viewWillDisappear:(BOOL)animated {
    nombreDePointsDuBleu = 0;
    nombreDePointsDuRouge = 0;
    nombreDePointsDuVert = 0;
    nombreDePointsDuJaune = 0;
    leTour = 0;
}

@end
```

Dans cette partie, on regarde le nombre de joueurs dans le tableau `podium`. Selon le nombre d'égalités en tête, on affiche un pop-up avec le nom du ou des vainqueurs, ainsi que le score. La fenêtre s'ouvre avec un son d'applaudissements. La dernière méthode `viewWillDisappear` remet les scores de tous les joueurs ainsi que la valeur `tour` à zéro lorsqu'on quitte la page. Cela permet de repartir sur une "base propre" lors d'une nouvelle partie. Ce qui clôt le commentaire de cette application.

4.2 Le compte-rendu de cette application

Ce que nous avons appris

Grâce à cette application, nous avons appris comment faire une intelligence artificielle, à faire une application contenant plusieurs pages et comment faire un timer. Nous avons également appris à mettre une icône pour l'application. C'est aussi la première application sur laquelle nous avons travaillé plus de 4 mois.

Problèmes rencontrés

Au cours du développement de notre application finale, nous avons rencontrés plusieurs problèmes. Un de ceux-ci, qui nous a pris plusieurs semaines à résoudre était de réussir

à mettre un délai juste après que l'IA ait joué, car sans cela elle jouait beaucoup trop rapidement ce qui ne nous laissait pas le temps de voir où elle jouait, ce qui rendait le jeu plus difficile. Un autre problème, qui nous a pris du temps mais que nous n'avons pas réussi à résoudre, était le son. En effet, la méthode que nous utilisons pour jouer un son dans notre application est obsolète sous iOS 7. Mais nous n'avons pas réussi à jouer un son autrement et de plus cette méthode fonctionne donc nous avons pris le choix de la laisser. Un dernier problème que nous avons eu était que l'IA ne fonctionnait pas bien sur la grille 3x3 et donc nous avons pris la décision d'enlever le mode 3x3 car ce n'était vraiment pas intéressant.

4.3 La démarche

Contrairement à ce qu'on pourrait penser en ayant lu ce travail de maturité, toutes les fonctionnalités de l'application finale n'ont pas été créées en même temps. Nous avons commencé par faire la page d'accueil et les différents menus. Puis nous avons codé la grille 3x3 en mode deux joueurs, puis la grande grille. Nous avons ensuite rajouté les modes trois et quatre joueurs et nous avons ajouté une IA qui se contentait de jouer de manière aléatoire. Après quelques semaines de réflexions, nous avons amélioré l'IA de manière à ce qu'elle ferme un carré lorsqu'il était à une valeur de trois et nous avons ajouté les scores et les sons. Et enfin nous avons créé une icône et nous avons encore amélioré l'IA de manière à ce qu'elle ne laisse pas de carré à une valeur de trois après avoir joué, de manière à ce que le joueur ne puisse pas marquer un point. Nous avons par la même occasion mis un délai entre chaque tour de l'IA.

Chapitre 5

Conclusion

Pour conclure ce travail de maturité, nous allons commencer par faire un récapitulatif des choses les plus importantes que nous avons apprises lors de ce travail de maturité. Nous avons appris en premier lieu le langage C qui était nouveau pour nous deux. Ce fut le premier grand sujet de ce travail de maturité. En second lieu, nous avons appris l'Objective-C, langage inconnu de nous deux et qui est utilisé pour notre application finale. Pendant ce travail, nous avons appris qu'Apple a introduit un nouveau langage de programmation, nommé Swift. Au début, nous avions pensé que tout ce que nous avons appris en Objective-C ne nous aurait servi à rien dans le futur, que ce langage était devenu obsolète. Mais, après coup, nous avons remarqué que ce travail nous a permis d'acquérir des bases en programmation qui sont utiles dans la plupart des langages. Nous avons par exemple compris le fonctionnement de la programmation orientée objet, que l'on retrouve par exemple en Java. Ce fut l'élément le plus important que nous avons appris de ce travail. Un troisième élément que nous avons appris en parallèle avec les deux langages est le langage latex, ce avec quoi nous rédigeons ce travail.

Voici à présent quelques améliorations que nous aurions pu rajouter. Une première, que nous avons cherché à faire mais pas réussi, est en fin de partie, lorsque le pop-up avec le gagnant apparait nous voulions que lorsqu'on on appuie sur le bouton "OK" cela nous ramène à la page d'accueil. Une deuxième amélioration à laquelle nous avons pensé est de pouvoir faire une taille de grille personnalisée mais après quelques essais nous nous sommes rendu compte que cela prendrait trop de temps.

Au début, nous avions l'intention de développer cette application sur iPad. Nous ne l'avons pas fait pour deux raisons. Premièrement, nous n'avions pas d'iPad, ce qui rendait moins pratique les tests de l'application. Deuxièmement, nous aurions dû faire une grille plus grande adaptée à la taille de l'écran. Vu le temps conséquent consacré à faire une petite grille, cela aurait été beaucoup trop long d'en faire une plus grande pour ce format.

Au terme de ce travail, plusieurs impressions communes ressortent, notamment le plaisir d'avoir mené à terme notre premier travail important, ainsi que d'avoir atteint nos objectifs initiaux. C'est la première fois qu'on a passé autant de temps sur un projet. Malgré le temps important consacré à ce travail, nous avons eu du plaisir à faire cela. Nous avions toujours envie de consacrer du temps à l'invention et au codage de l'application, c'était quelque chose de divertissant et passionnant. C'était la première fois qu'on avait vraiment envie de travailler autant sur un sujet. De plus, le travail en collaboration est

plus intéressant et motivant que tout seul car si une des deux personne à un peu de peine, l'autre peut l'aider, et inversement.

Au contraire, quelques détails nous ont posé quelques problèmes. La collaboration nous a posé quelques problèmes, malgré son côté motivant. En effet, il est parfois difficile de mettre en commun le travail, car chacun n'a pas forcement travaillé de la même manière. Par exemple, quand on essayait les deux de coder une même fonction, de manière à avancer plus vite, on n'employait pas les mêmes noms de variables, ce qui rendait la mise en commun difficile. D'autre part, il était parfois difficile de se voir pour mettre en commun, notamment pendant les vacances. Un autre problème aura été l'installation de l'application sur l'iPod. En effet il nous a fallu un moment pour que Xcode accepte l'iPod correctement. Le plus gros problème arriva récemment, à cause d'une perte de données d'un de nos ordinateurs, plus précisément celui configuré pour fonctionner avec l'iPod. Après cet événement, nous n'avons plus réussi à installer notre application sur cet appareil. Cependant, nous avions déjà pu tester le bon fonctionnement de l'application sur l'iPod, nous n'en avions plus vraiment besoin.

Au final, ce travail de maturité aura vraiment été une expérience aussi intéressante qu'enrichissante. Nous avons appris un grand nombre de choses, et une certaine compréhension dans le domaine de la programmation. De plus, cela nous a permis de nous donner une idée plus précise de ce qu'on voulait faire comme métier plus tard, ce qui nous facilite le choix de nos futures études.

Chapitre 6

Annexes

6.1 Programme de répétition de livrets en C

```
//
//  main.c
//  LivretsFinal
//
//  Created by Loïs Bilat and Joachim Droz on 27.02.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#include <time.h>
#include <stdio.h>
#include <stdlib.h>

////////////////////////////////////////////////////////
// Fonction qui crée un nombre aléatoire

int rand_a_b(int a, int b){
    int nbr;
    nbr = rand() % (b-a) + a;
    return nbr;
}

////////////////////////////////////////////////////////
// Mode afficher un livret

int livret(){
    printf("Vous avez choisis d'afficher un livret\n");
    printf("\n");
    printf("Quel livret voulez-vous afficher ?\n");
    printf("\n");
    int numeroLivret;
    scanf("%d",&numeroLivret);
    printf("Table des livrets de %d :\n",numeroLivret);
```

```
    printf("\n");
    int i = 1;
    for (i=1; i<21; i++) {
        printf("%2.d * %d = %d\n",i,numeroLivret,i*numeroLivret);
    }
    printf("\n");
    return 0;
}

////////////////////////////////////////////////////////
// Mode Exercice

int nbrEntreExercice;

int exercice(){
    printf("Mode exercice :\n");
    printf("\n");
    int a; // Borne min
    int b; // Borne max
    int nombreAleatoire1;
    int nombreAleatoire2;
    int difficulte;
    printf("Choisissez la difficulté (0 pour facile, 1 pour moyen,
        2 pour difficile et 3 pour personnalisé): ");
    scanf("%d",&difficulte);
    printf("\n");
    if (difficulte == 0) {
        a = 1;
        b = 13;
        printf("Mode facile\n");
    }
    if (difficulte == 1) {
        a = 2;
        b = 21;
        printf("Mode moyen\n");
    }
    if (difficulte == 2) {
        a = 13;
        b = 21;
        printf("Mode difficile\n");
    }
    if (difficulte == 3) {
        printf("Mode personnalisé\n");
        printf("Choisissez le plus petit livret de l'exercice : ");
        scanf("%d",&a);
        printf("Choisissez le plus grand livret de l'exercice : ");
        scanf("%d",&b);
```

```
        b++;
    }
    srand(time(NULL));
    printf("\n");
    printf("Entrez 0 pour revenir au menu principal\n");
    printf("\n");
    while (1) {
        nombreAleatoire1 = rand_a_b(a,b);
        nombreAleatoire2 = rand_a_b(a,b);
        int resultat = nombreAleatoire1 * nombreAleatoire2;
        printf("%d * %d = ",nombreAleatoire1,nombreAleatoire2);
        scanf("%d",&nbrEntreExercice);
        if (resultat==nbrEntreExercice) {
            printf("La réponse est juste\n");
        }
        else if (nbrEntreExercice == 0){
            printf("\n");
            printf("Vous avez quitté\n");
            printf("\n");
            break;
        }
        else{
            printf("Faux, la réponse est %d\n",resultat);
        }
    }
    return 0;
}

////////////////////////////////////////////////////////////////////
// Mode test

int nbrEntreTest;

int test(){ // Fonction pour faire le test
    printf("Mode Test\n");
    printf("\n");
    int a; // Borne min
    int b; // Borne max
    int difficulte;
    printf("Choisissez la difficulté (0 pour facile, 1 pour moyen, 2 pour
    difficile et 3 pour personnalisé): ");
    scanf("%d",&difficulte);
    if (difficulte == 0) {
        a = 1;
        b = 13;
        printf("Mode facile\n");
    }
```

```
if (difficulte == 1) {
    a = 2;
    b = 21;
    printf("Mode moyen\n");
}
if (difficulte == 2) {
    a = 13;
    b = 21;
    printf("Mode difficile\n");
}
if (difficulte == 3) {
    printf("Mode personnalisé\n");
    printf("Choisissez le plus petit livret du test : ");
    scanf("%d",&a);
    printf("Choisissez le plus grand livret du test : ");
    scanf("%d",&b);
    b++;
}
int x = 0;
int vrai = 0;
int faux = 0;
int nombreAleatoire1;
int nombreAleatoire2;
int n;
printf("Combien de calculs voulez-vous faire ?\n");
scanf("%d",&n);
srand(time(NULL));
for (x = 0; x<n; x++) {
    nombreAleatoire1 = rand_a_b(a,b);
    nombreAleatoire2 = rand_a_b(a,b);
    int resultat = nombreAleatoire1 * nombreAleatoire2;
    printf("%d * %d = ",nombreAleatoire1,nombreAleatoire2);
    scanf("%d",&nbrEntreTest);
    if (resultat==nbrEntreTest) {
        vrai++;
    }
    else{
        faux++;
    }
}
printf("\n");
printf("Il y a %d réponse(s) juste(s) et %d réponse(s) fausse(s)\n",
vrai,faux);
printf("\n");
printf("Fin du test\n");
printf("\n");
return 0;
```

```
}

////////////////////////////////////////////////////////////////////
// Menu principal

char programme;

int main(int argc, const char * argv[]){
    printf("Bienvenue dans le répétiteur de livrets !\n");
    printf("\n");
    printf("Menu principal :\n");
    printf("\n");
    printf("Appuyez sur 'l' pour afficher un livret\n");
    printf("Appuyez sur 't' pour lancer le mode test\n");
    printf("Appuyez sur 'e' pour lancer le mode exercice\n");
    printf("Appuyez sur 'q' pour quitter le programme\n");

    while (1) {

        scanf("%c",&programme);
        if (programme == 'q') {
            printf("Au revoir et à bientôt\n");
            break;
        }
        if (programme == 'l') {
            livret();
            continue;
        }
        if (programme == 't') {
            test();
            continue;
        }
        if (programme == 'e') {
            exercice();
            continue;
        }
        printf("Menu principal :\n");
        printf("\n");
        printf("Appuyez sur 'l' pour afficher un livret\n");
        printf("Appuyez sur 't' pour lancer le mode test\n");
        printf("Appuyez sur 'e' pour lancer le mode exercice\n");
        printf("Appuyez sur 'q' pour quitter le programme\n");
    }
    return 0;
}
```

6.2 Code du répétiteur de livrets en objective-C

6.2.1 ViewController.h

```
//
//  ViewController.h
//  livrets
//
//  Created by Loïs Bilat and Joachim Droz on 12.03.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController

@property (weak, nonatomic) IBOutlet UILabel *livret;
@property (weak, nonatomic) IBOutlet UITextField *entree;
@property (weak, nonatomic) IBOutlet UILabel *vraiFaux;
@property (weak, nonatomic) IBOutlet UISegmentedControl *onglets;
@property (weak, nonatomic) IBOutlet UISegmentedControl *mode;
@property (weak, nonatomic) IBOutlet UIButton *verifier;
@property (weak, nonatomic) IBOutlet UIButton *statistiques;

- (IBAction)suivant:(id)sender; // Bouton qui crée un nouveau livret

- (IBAction)verifier:(id)sender; // Bouton qui vérifie si la réponse est
   correcte

- (IBAction)niveau:(id)sender; // Capteur appui onglets

- (IBAction)statistiques:(id)sender; // Bouton qui affiche les statistiques

- (IBAction)mode:(id)sender; // Bouton mode de calculs

- (IBAction)entree:(id)sender; // Quand on écris dans le champ de texte

@end
```

6.2.2 ViewController.m

```
//
//  ViewController.m
//  livrets
//
//  Created by Loïs Bilat and Joachim Droz on 12.03.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
```

```
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController

// Variables importantes

int nombreAleatoire1;
int nombreAleatoire2;
int a = 2;
int b = 12;
int multiple;
int resultat;
int reinitialiser;
int zero = 0;
int provisoire;
float vrai = 0;
float faux = 0;
NSString *texteFaux = @"";
NSString *texteAjout;
NSString *listeLivretsFaux = @"";

// Fonction nombre aléatoire

int rand_a_b(int a, int b){

    int nbr;
    nbr = rand() % (b-a) + a;
    return nbr;
}

// Au démarrage

- (void)viewDidLoad{

    [super viewDidLoad];
    srand(time(NULL));

    UIAlertView *bienvenue = [[UIAlertView alloc] initWithTitle:
        @"Bonjour" message:@"Bienvenue dans notre application de
        répétition de calcul mental\n\nApplication crée par Loïs et
        Joachim dans le cadre de leur Travail de Maturité" delegate:
```

```
        nil cancelButtonTitle: @"Continuer" otherButtonTitles:nil];
    [bienvenue show];

    [self suivant:nil];
    self.statistiques.enabled = NO;
}

// Choix de la difficulté: Selon l'onglet sélectionné, on règle les
bornes des livrets

- (IBAction)niveau:(id)sender {
    self.onglets.enabled = YES;
    if (self.onglets.selectedSegmentIndex == 0){
        if (self.mode.selectedSegmentIndex == 0 ||
        self.mode.selectedSegmentIndex == 1){
            a = 2;
            b = 12;
        }
        else if (self.mode.selectedSegmentIndex == 2 ||
        self.mode.selectedSegmentIndex == 3){
            a = 10;
            b = 100;
        }
    }
    else if (self.onglets.selectedSegmentIndex == 1){
        if (self.mode.selectedSegmentIndex == 0 ||
        self.mode.selectedSegmentIndex == 1){
            a = 6;
            b = 16;
        }
        else if (self.mode.selectedSegmentIndex == 2 ||
        self.mode.selectedSegmentIndex == 3){
            a = 100;
            b = 1000;
        }
    }
    else if (self.onglets.selectedSegmentIndex == 2){
        if (self.mode.selectedSegmentIndex == 0 ||
        self.mode.selectedSegmentIndex == 1){
            a = 12;
            b = 20;
        }
        else if (self.mode.selectedSegmentIndex == 2 ||
        self.mode.selectedSegmentIndex == 3){
            a = 1000;
            b = 10000;
        }
```

```
    }
    [self suivant:nil]; // On crée un autre calcul
}

// Mode de calculs

- (IBAction)mode:(id)sender {
    [self niveau:nil]; // On fait comme quand on choisi le niveau
}

// Pop-up des statistiques

- (IBAction)statistiques:(id)sender {
    NSString *texte = @"";
    self.vraiFaux.text = @"";

    float pourcentage = vrai/(vrai+faux);
    if (pourcentage >= 0.95) {
        texte = @"Vous êtes un génie du calcul mental";
    }
    else if (pourcentage >= 0.8 && pourcentage < 0.95){
        texte = @"Vous êtes un pro du calcul mental";
    }
    else if (pourcentage >= 0.6 && pourcentage < 0.8){
        texte = @"Vous maîtrisez bien le calcul mental";
    }
    else if (pourcentage >= 0.4 && pourcentage < 0.6){
        texte = @"Il y a encore un peu de travail à faire";
    }
    else if (pourcentage >= 0.2 && pourcentage < 0.4){
        texte = @"C'est pas votre jour!";
    }
    else if (pourcentage >= 0.1 && pourcentage < 0.2){
        texte = @"ToDoList: réviser les maths";
    }
    else {
        texte = @"Le calcul mental c'est pas votre truc!";
    }

    UIAlertView *alert = [[UIAlertView alloc] initWithTitle:
        @"Statistiques" message:[NSString stringWithFormat:
        @"%@%.f%@%.f%@%@%@%@", @"Entrez 0 comme
        réponse pour effacer les statistiques\n\n Vous avez fait ",
        vrai,@" réponse(s) juste(s) et ",faux,@"  réponse(s) fausse(s)
        depuis le lancement de l'application\n\n", texte,listeLivretsFaux,
        texteFaux] delegate:self cancelButtonTitle:@"Fermer"
```

```
        otherButtonTitles:nil];
    [alert show];
}

// Pour vérifier si on entre zéro

- (IBAction)entree:(id)sender {

    self.verifier.enabled = YES;
    [self.verifier setTitle:@"Vérifier la réponse"
    forState:UIControlStateNormal];
    self.vraiFaux.text = @"";
    int n = [self.entree.text  integerValue];
    // Si on entre 0
    if (n == 0) {
        if (zero == 0) {
            [self.verifier setTitle:@"Rénitialiser les statistiques"
            forState:UIControlStateNormal];
            reinitialiser = 1;
        }
    }
    else {
        zero = 1;
        reinitialiser = 0;
    }
}

// Quand on appuye sur suivant

- (IBAction)suivant:(id)sender {

    // Désactive bouton vérifier

    self.entree.text = @"";
    self.verifier.enabled = NO;
    [self.verifier setTitle:@"" forState:UIControlStateNormal];

    self.vraiFaux.text = @""; // On efface le texte de réponse
    nombreAleatoire1 = rand_a_b(a,b+1);
    nombreAleatoire2 = rand_a_b(a,b+1);
    multiple = nombreAleatoire2 * nombreAleatoire1;

    // Selon le mode choisi, on affiche le livret

    if (self.mode.selectedSegmentIndex == 0) {
        self.livret.text = [NSString stringWithFormat:
        @"%d%@%d",nombreAleatoire1,@" x ",nombreAleatoire2];
```

```
    }
    else if (self.mode.selectedSegmentIndex == 1){
            self.livret.text = [NSString stringWithFormat:
        @"%d%@%d",multiple,@" / ",nombreAleatoire1];
    }
    else if (self.mode.selectedSegmentIndex == 2){
        self.livret.text = [NSString stringWithFormat:
        @"%d%@%d",nombreAleatoire1,@" + ",nombreAleatoire2];
    }
    else if(self.mode.selectedSegmentIndex == 3){
        if (nombreAleatoire1 < nombreAleatoire2) {
            provisoire = nombreAleatoire2;
            nombreAleatoire2 = nombreAleatoire1;
            nombreAleatoire1 = provisoire;
        }
        else if(nombreAleatoire1 == nombreAleatoire2){
            [self suivant:nil];
        }
        self.livret.text = [NSString stringWithFormat:
        @"%d%@%d",nombreAleatoire1,@" - ",nombreAleatoire2];
    }
}

// Quand on vérifie la réponse

- (IBAction)verifier:(id)sender {

    zero = 0; // Pour désactiver le blocage du '0' qui affiche
     les statistiques

    self.statistiques.enabled = YES; // On permet d'afficher
     les statistiques

    if (reinitialiser == 1) { // Si on a entré 0 comme réponse,
     change le bouton pour reset les stats
        vrai = 0;
        faux = 0;
        self.vraiFaux.textColor = [UIColor darkGrayColor];
        self.statistiques.enabled = NO;
        self.verifier.enabled = NO;
        [self.verifier setTitle:@"" forState:UIControlStateNormal];
        self.entree.text = @"";
        reinitialiser = 0;
        texteFaux = @"";
        listeLivretsFaux = @"";
        [self suivant:nil];
        self.vraiFaux.text = @"Statistiques rénitialisées";
```

```
}
else { // Cas normal

    int m = [self.entree.text  integerValue]; // Ce qu'on entre

    // On définit le résultat selon le mode choisi

    if (self.mode.selectedSegmentIndex == 0) {
        resultat = nombreAleatoire2 * nombreAleatoire1;
    }
    else if (self.mode.selectedSegmentIndex == 1){
        resultat = multiple / nombreAleatoire1;
    }
    else if (self.mode.selectedSegmentIndex == 2){
        resultat = nombreAleatoire1 + nombreAleatoire2;
    }
    else if (self.mode.selectedSegmentIndex == 3){
        resultat = nombreAleatoire1 - nombreAleatoire2;
    }
    // On affiche un nouveau calcul


    if (m != resultat) {
        // On ajoute au texte des statistiques

        listeLivretsFaux = @"\n\nListe des calculs faux :\n\n";
        if (self.mode.selectedSegmentIndex == 0) {
            texteAjout=[NSString stringWithFormat:@"%d%@%d%@%d%@%d%@",
            nombreAleatoire1,@" x ",nombreAleatoire2,@" = ",resultat,
            @"\n(Vous avez mis ",m,@")\n\n"];
        }
        else if (self.mode.selectedSegmentIndex == 1) {
            texteAjout=[NSString stringWithFormat:@"%d%@%d%@%d%@%d%@",
            multiple,@" / ",nombreAleatoire1,@" = ",resultat,@"\n
            (Vous avez mis ",m,@")\n\n"];
        }
        else if (self.mode.selectedSegmentIndex == 2) {
            texteAjout=[NSString stringWithFormat:@"%d%@%d%@%d%@%d%@",
            nombreAleatoire1,@" + ",nombreAleatoire2,@" = ",resultat,
            @"\n(Vous avez mis ",m,@")\n\n"];
        }
        else if (self.mode.selectedSegmentIndex == 3) {
            texteAjout=[NSString stringWithFormat:@"%d%@%d%@%d%@%d%@",
            nombreAleatoire1,@" - ",nombreAleatoire2,@" = ",resultat,
            @"\n(Vous avez mis ",m,@")\n\n"];
        }
        texteFaux = [texteFaux stringByAppendingString:texteAjout];
```

```
            [self suivant:self];
            self.vraiFaux.text = [NSString stringWithFormat:@"%@%d",
            @"Faux,la réponse est ",resultat];
            self.vraiFaux.textColor = [UIColor redColor];
            faux++;



        }

        else {

            [self suivant:nil];
            self.vraiFaux.text = @"La réponse est correcte";
            self.vraiFaux.textColor = [UIColor greenColor];
            vrai ++;

        }
    }
}

@end
```

6.3 Code du modèle 2x2

6.3.1 ViewController.h

```
//
//  ViewController.h
//  pipopipette2x2
//
//  Created by Loïs Bilat and Joachim Droz on 01.04.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface ViewController : UIViewController
{
    int c1;
    int c2;
    int c3;
    int c4;
    int tour;

    int nombreDePointsBleu;
```

```
    int nombreDePointsRouge;
}
@property (weak, nonatomic) IBOutlet UIButton *btn1;
@property (weak, nonatomic) IBOutlet UIButton *btn2;
@property (weak, nonatomic) IBOutlet UIButton *btn3;
@property (weak, nonatomic) IBOutlet UIButton *btn4;
@property (weak, nonatomic) IBOutlet UIButton *btn5;
@property (weak, nonatomic) IBOutlet UIButton *btn6;

- (IBAction)actBtn1:(id)sender;
- (IBAction)actBtn2:(id)sender;
- (IBAction)actBtn3:(id)sender;
- (IBAction)actBtn4:(id)sender;
- (IBAction)actBtn5:(id)sender;
- (IBAction)actBtn6:(id)sender;

@property (weak, nonatomic) IBOutlet UIButton *btnA;
@property (weak, nonatomic) IBOutlet UIButton *btnB;
@property (weak, nonatomic) IBOutlet UIButton *btnC;
@property (weak, nonatomic) IBOutlet UIButton *btnD;
@property (weak, nonatomic) IBOutlet UIButton *btnE;
@property (weak, nonatomic) IBOutlet UIButton *btnF;

- (IBAction)actBtnA:(id)sender;
- (IBAction)actBtnB:(id)sender;
- (IBAction)actBtnC:(id)sender;
- (IBAction)actBtnD:(id)sender;
- (IBAction)actBtnE:(id)sender;
- (IBAction)actBtnF:(id)sender;

@property (weak, nonatomic) IBOutlet UIImageView *carre1;
@property (weak, nonatomic) IBOutlet UIImageView *carre2;
@property (weak, nonatomic) IBOutlet UIImageView *carre3;
@property (weak, nonatomic) IBOutlet UIImageView *carre4;

@property (weak, nonatomic) IBOutlet UILabel *texte;


@end
```

6.3.2 ViewController.m

```
//
//  ViewController.m
//  pipopipette2x2
//
//  Created by Loïs Bilat and Joachim Droz on 01.04.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import "ViewController.h"

@interface ViewController ()

@end

@implementation ViewController



- (void)viewDidLoad
{
    [super viewDidLoad];


}

- (void)didReceiveMemoryWarning
{
    [super didReceiveMemoryWarning];
    c1 = 0;
    c2 = 0;
    c3 = 0;
    c4 = 0;
    tour = 0;

    nombreDePointsBleu = 0;
    nombreDePointsRouge = 0;
    // Dispose of any resources that can be recreated.
}
////////////////////////////////////////////////////////////////////

- (void) fonctionBords:(int)cX andOther: (UIButton *) btnX
 andOther: (UIImageView *) carreX andOther: (UILabel *) texte{
    if (tour == 0) {
        btnX.enabled = NO;
```

```
        btnX.backgroundColor = [UIColor blueColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"bleu.jpg"];
            nombreDePointsBleu ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge,
            @" Pour Rouge"];

        }
        else if (cX != 4) {
            tour = 1;
        }
    }
    else if (tour == 1) {
        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor redColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"rouge.jpg"];
            nombreDePointsRouge ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu, @" points pour Bleu et ",
            nombreDePointsRouge,
            @" Pour Rouge"];
        }
        else if (cX != 4) {
            tour = 0;
        }
    }
}

//////////////////////////////////////////////////////////////////

- (void) fonctionInterieur:(int)cX andOther: (int)cY andOther: (UIButton *)
btnX andOther: (UIImageView *)carreX andOther: (UIImageView *)carreY
andOther:(UILabel *)texte{
    if (tour == 0) {
        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor blueColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"bleu.jpg"];
            nombreDePointsBleu ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu,@" points pour Bleu et ",nombreDePointsRouge,
             @" Pour Rouge"];
        }
        if (cY == 4) {
```

```
            carreY.image = [UIImage imageNamed:@"bleu.jpg"];
            nombreDePointsBleu ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu,@" points pour Bleu et ",nombreDePointsRouge,
            @" Pour Rouge"];
        }
        if (cX != 4 && cY != 4){
            tour = 1;
        }
    }
    else if (tour == 1) {
        btnX.enabled = NO;
        btnX.backgroundColor = [UIColor redColor];
        if (cX == 4) {
            carreX.image = [UIImage imageNamed:@"rouge.jpg"];
            nombreDePointsRouge ++;
            texte.text = [NSString stringWithFormat:@"%d%@%d%@",
            nombreDePointsBleu,@" points pour Bleu et ",nombreDePointsRouge,
            @" Pour Rouge"];
        }
        if (cY == 4) {
            carreY.image = [UIImage imageNamed:@"rouge.jpg"];
            nombreDePointsRouge ++;
            texte.text = [NSString stringWithFormat:
            @"%d%@%d%@",nombreDePointsBleu,
            @" points pour Bleu et ",nombreDePointsRouge,@" Pour Rouge"];
        }
        if (cX != 4 && cY != 4){
            tour = 0;
        }
    }
}

//////////////////////////////////////////////////////////////////////

- (IBAction)actBtn1:(id)sender {
    c1++;
    [self fonctionBords:c1 andOther:_btn1 andOther:_carre1 andOther:_texte];
    }
- (IBAction)actBtn2:(id)sender {
    c2++;
    [self fonctionBords:c2 andOther:_btn2 andOther:_carre2 andOther:_texte];
}
- (IBAction)actBtn3:(id)sender {
    c1++;
    c3++;
    [self fonctionInterieur: c1 andOther: c3 andOther: _btn3
```

```
    andOther: _carre1 andOther: _carre3 andOther: _texte ];
}
- (IBAction)actBtn4:(id)sender {
    c2++;
    c4++;
    [self fonctionInterieur: c2 andOther: c4 andOther: _btn4
    andOther: _carre2 andOther: _carre4 andOther: _texte ];
}
- (IBAction)actBtn5:(id)sender {
    c3++;
    [self fonctionBords:c3 andOther:_btn5 andOther:_carre3 andOther:_texte];
}
- (IBAction)actBtn6:(id)sender {
    c4++;
    [self fonctionBords:c4 andOther:_btn6 andOther:_carre4 andOther:_texte];
}
- (IBAction)actBtnA:(id)sender {
    c1++;
    [self fonctionBords:c1 andOther:_btnA andOther:_carre1 andOther:_texte];
}
- (IBAction)actBtnB:(id)sender {
    c1++;
    c2++;
    [self fonctionInterieur: c1 andOther: c2 andOther: _btnB
    andOther: _carre1 andOther: _carre2 andOther: _texte ];
}
- (IBAction)actBtnC:(id)sender {
    c2++;
    [self fonctionBords:c2 andOther:_btnC andOther:_carre2 andOther:_texte];
}
- (IBAction)actBtnD:(id)sender {
    c3++;
   [self fonctionBords:c3 andOther:_btnD andOther:_carre3 andOther:_texte];
}
- (IBAction)actBtnE:(id)sender {
    c3++;
    c4++;
    [self fonctionInterieur: c3 andOther: c4 andOther: _btnE
    andOther: _carre3 andOther: _carre4 andOther: _texte ];
}
- (IBAction)actBtnF:(id)sender {
    c4++;
    [self fonctionBords:c4 andOther:_btnF andOther:_carre4 andOther:_texte];
}
@end
```

6.4 Code de l'application finale

6.4.1 ViewController.h

```
//
//  ViewController.h
//  laPipopipette
//
//  Created by Loïs Bilat and Joachim Droz on 20.05.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>


@interface ViewController : UIViewController


- (IBAction)aPropos:(id)sender;
- (IBAction)commentJouer:(id)sender;


@end
```

6.4.2 ViewController.m

```
//
//  ViewController.m
//  laPipopipette
//
//  Created by Loïs Bilat and Joachim Droz on 20.05.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import "ViewController.h"
@interface ViewController ()
@end
@implementation ViewController

// On active l'auto-rotation

- (BOOL) shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
toInterfaceOrientation {
    return NO;
}
```

```
// On désactive la barre en haut uniquement sur la première page

- (void)viewWillAppear:(BOOL)animated
{
    [self.navigationController setNavigationBarHidden:YES animated:YES];
}

- (void)viewWillDisappear:(BOOL)animated
{
    [self.navigationController setNavigationBarHidden:NO animated:YES];
}

// On définit le bouton retour de la page d'après

- (void)viewDidLoad
{
     [self.navigationItem setBackBarButtonItem:[[UIBarButtonItem alloc]
     initWithTitle:@"Retour" style:UIBarButtonItemStyleBordered target:nil
     action:nil]];

}
// Boutons vers l'aide et vers l'à propos

- (IBAction)aPropos:(id)sender
{
    UIAlertView *aPropos = [[UIAlertView alloc] initWithTitle:@"A propos"
    message:@"Bienvenue dans notre jeu de la Pipopipette\n\nApplication
    crée par Loïs et Joachim dans le cadre de leur Travail de Maturité
    au Gymnase de Burier" delegate:nil cancelButtonTitle:@"Fermer"
    otherButtonTitles:nil];
    [aPropos show];
}

- (IBAction)commentJouer:(id)sender
{
    UIAlertView *commentJouer = [[UIAlertView alloc] initWithTitle:
    @"Règles du jeu" message:@"Les règles: \n- Chaque joueur joue à
    tour de rôle en appuyant sur le trait désiré. \n- Le but est de
    réussir à fermer un carré. Si c'est le cas, vous pourrez rejouer.
    \n- Le joueur ayant fermé le plus de carré gagnera." delegate:nil
    cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [commentJouer show];
}

@end
```

6.4.3 choixModeJeu.h

```
//
//  choixModeJeu.h
//  laPipopipette
//
//  Created by Loïs Bilat and Joachim Droz on 21.05.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import <UIKit/UIKit.h>

@interface choixModeJeu : UIViewController
<UIPickerViewDataSource,UIPickerViewDelegate>

@property (weak, nonatomic) IBOutlet UIPickerView *picker;
@property (strong, nonatomic) NSArray *tabModes;
@property (nonatomic) int modeDeJeu;

@end
```

6.4.4 choixModeJeu.m

```
//
//  choixModeJeu.m
//  laPipopipette
//
//  Created by Loïs Bilat and Joachim Droz on 21.05.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import "choixModeJeu.h"

#import "grille.h"

@interface choixModeJeu ()

@end

@implementation choixModeJeu

//On définit le titre de la page, le bouton retour d'après et on initialise
  le tableau pour le picker

- (void)viewDidLoad
{
    self.navigationItem.title = @"Selection du nombre de joueurs";
```

```
    self.tabModes=[[NSArray alloc] initWithObjects:@"1 joueur",@"2 joueurs",
        @"3 joueurs", @"4 joueurs", nil];

    self.modeDeJeu = 1;
}

// On configure le picker

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView;
{
    return 1;
}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRowsInComponent:
    (NSInteger)component;
{
    return 4;
}

-(NSString *)pickerView:(UIPickerView *)pickerView titleForRow:
    (NSInteger)row forComponent:(NSInteger)component
{
    return [self.tabModes objectAtIndex:row];
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(NSInteger)row
    inComponent:(NSInteger)component
{
    switch(row)

    {
        case 0:
            _modeDeJeu = 1;
            break;
        case 1:
            _modeDeJeu = 2;
            break;
        case 2:
            _modeDeJeu = 3;
            break;
        case 3:
            _modeDeJeu = 4;
    }
}

// On envoie la variable du nombre de joueurs dans la page d'après
```

```
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    grille *destination = [segue destinationViewController];
    destination.nombreDeJoueurs = _modeDeJeu;
}
@end
```

6.4.5 grille.h

```
//
//  grilleGrande.h
//  laPipopipette
//
//  Created by Loïs Bilat and Joachim Droz on 24.05.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <AVFoundation/AVFoundation.h>
#import <AudioToolbox/AudioToolbox.h>


@interface grille : UIViewController <AVAudioPlayerDelegate>

{
    AVAudioPlayer * audioPlayer;
}


@property (nonatomic) int nombreDeJoueurs;

@property (strong, nonatomic) IBOutletCollection(UIImageView)
   NSMutableArray *traits;

@property (strong, nonatomic) IBOutletCollection(UIImageView)
   NSMutableArray *premierCarre;

@property (strong, nonatomic) IBOutletCollection(UIImageView)
   NSMutableArray *deuxiemeCarre;

@property NSMutableArray *tableauDistances;

@property NSMutableArray *scores;

@property (weak, nonatomic) IBOutlet UILabel *scoreBleu;
```

```
@property (weak, nonatomic) IBOutlet UILabel *scoreRouge;
@property (weak, nonatomic) IBOutlet UILabel *scoreVert;
@property (weak, nonatomic) IBOutlet UILabel *scoreJaune;
@property (weak, nonatomic) IBOutlet UILabel *texte;

@property NSMutableArray *podium;

@property NSString *nomDuMeilleur;

@property UIImageView *traitUnJoueur;

@property UIImageView *traitUnIA;
@end
```

6.4.6 grille.m

```
//
//  grille.m
//  laPipopipette
//
//  Created by Loïs Bilat and Joachim Droz on 24.05.14.
//  Copyright (c) 2014 TM2014. All rights reserved.
//

#import "grille.h"

@interface grille ()

@end

@implementation grille

int indexMin;
int leTour = 0;
int nombreDePointsDuBleu = 0;
int nombreDePointsDuRouge = 0;
int nombreDePointsDuVert = 0;
int nombreDePointsDuJaune = 0;
int tailleImage;
int placeDuMeilleur;
int placeLibre = 0;
int positionIA;
int tourDeLIA = 1;
int dejaFait;
bool timerAllume;
int compteur;
```

```
// On masque la barre

- (void)viewWillAppear:(BOOL)animated
{
    [self.navigationController setNavigationBarHidden:YES animated:YES];
}

// Fonction pricipale

- (void) fonction2Joueurs:(int)cX andOther: (int)cY andOther:
    (UIImageView *)image andOther: (UIImageView *)carreX
    andOther: (UIImageView *)carreY{

    if (leTour == 0) {

        image.image = [UIImage imageNamed:@"bleu.jpg"];

        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 1;
            self.texte.text = @"C'est au tour du Joueur Rouge de jouer";
        }
    }
    else if (leTour == 1) {
        image.image = [UIImage imageNamed:@"rouge.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 0;
```

```
            self.texte.text = @"C'est au tour du Joueur Bleu de jouer";
        }
    }
    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];
}

- (void) fonction3Joueurs:(int)cX andOther: (int)cY andOther:
    (UIImageView *)image andOther: (UIImageView *)carreX andOther:
    (UIImageView *)carreY{
    if (leTour == 0) {
        image.image = [UIImage imageNamed:@"bleu.jpg"];

        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 1;
            self.texte.text = @"C'est au tour du Joueur Rouge de jouer";
        }
    }
    else if (leTour == 1) {
        image.image = [UIImage imageNamed:@"rouge.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 2;
            self.texte.text = @"C'est au tour du Joueur Vert de jouer";
        }
    }
```

```
    else if (leTour == 2) {
        image.image = [UIImage imageNamed:@"vert.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 0;
            self.texte.text = @"C'est au tour du Joueur Bleu de jouer";
        }
    }
    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];
    self.scoreVert.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuVert];
}
- (void) fonction4Joueurs:(int)cX andOther: (int)cY andOther:
    (UIImageView *)image andOther: (UIImageView *)carreX andOther:
    (UIImageView *)carreY{
    if (leTour == 0) {
        image.image = [UIImage imageNamed:@"bleu.jpg"];

        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreBleu.jpg"];
            nombreDePointsDuBleu ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 1;
            self.texte.text = @"C'est au tour du Joueur Rouge de jouer";
        }
    }
    else if (leTour == 1) {
        image.image = [UIImage imageNamed:@"rouge.jpg"];
```

```
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreRouge.jpg"];
            nombreDePointsDuRouge ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 2;
            self.texte.text = @"C'est au tour du Joueur Vert de jouer";
        }
    }
    else if (leTour == 2) {
        image.image = [UIImage imageNamed:@"vert.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreVert.jpg"];
            nombreDePointsDuVert ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 3;
            self.texte.text = @"C'est au tour du Joueur Jaune de jouer";
        }
    }
    else if (leTour == 3) {
        image.image = [UIImage imageNamed:@"jaune.jpg"];
        if (cX == 4) {
            [audioPlayer play];
            carreX.image = [UIImage imageNamed:@"CarreJaune.jpg"];
            nombreDePointsDuJaune ++;
        }
        if (cY == 4) {
            [audioPlayer play];
            carreY.image = [UIImage imageNamed:@"CarreJaune.jpg"];
            nombreDePointsDuJaune ++;
        }
        if (cX != 4 && cY != 4){
            leTour = 0;
            self.texte.text = @"C'est au tour du Joueur Bleu de jouer";
```

```
        }
    }
    self.scoreBleu.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuBleu];
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];
    self.scoreVert.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuVert];
    self.scoreJaune.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuJaune];
}

- (void)viewDidLoad {

    [super viewDidLoad];


    // On initlialise le son de crayon

    AudioSessionInitialize (NULL, NULL, NULL, (__bridge void *)self);
    UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback;
    AudioSessionSetProperty (kAudioSessionProperty_AudioCategory, sizeof
        (sessionCategory), &sessionCategory);
    NSData *soundFileData;
    soundFileData = [NSData dataWithContentsOfURL:[NSURL fileURLWithPath:
        [[NSBundle mainBundle] pathForResource:@"ding.mp3" ofType:NULL]]];
    audioPlayer = [[AVAudioPlayer alloc] initWithData:soundFileData
        error:NULL];
    audioPlayer.delegate = self;
    [audioPlayer setVolume:1.0f];

    // On initlialise les tableaux

    self.podium = [[NSMutableArray alloc] init];
    self.scores = [[NSMutableArray alloc] init];
    self.tableauDistances = [[NSMutableArray alloc] init];

    // On masque des éléements de l'interface en fonction du
        nombre de joueurs

    if (self.nombreDeJoueurs == 2) {
        self.scoreVert.text = @"";
        self.scoreJaune.text = @"";
    }
    else if (self.nombreDeJoueurs == 3) {
        self.scoreJaune.text = @"";
    }
```

```
    else if (self.nombreDeJoueurs == 1){
        self.scoreVert.text = @"";
        self.scoreJaune.text = @"";
    }

    // On fait un random

    srand(time(NULL));

    // Info pour 1 joueur

    if (self.nombreDeJoueurs == 1) {
        UIAlertView *infoCouleurs = [[UIAlertView alloc] initWithTitle:
            @"Information" message:[NSString stringWithFormat:
            @"Vous êtes le joueur Bleu, le joueur Rouge est geré
            automatiquement."] delegate:nil cancelButtonTitle:@"OK"
            otherButtonTitles:nil, nil];
        [infoCouleurs show];
    }
}

// Fonction IA

- (void) onTick:(NSTimer *)timer {

    // On regarde si on peux finir un carré dans le premier tableau


    placeLibre = 0;

    for (UIImageView *traitVide in _premierCarre) {
        if ([traitVide tag] == 3) {
            placeLibre = 1;
            break;
        }
    }

    // Si pas dans le premier tableau, on regarde dans le deuxieme

    if (placeLibre == 0) {
        for (UIImageView *autreTraitVide in _deuxiemeCarre) {
            if ([autreTraitVide tag] == 3) {
                placeLibre = 2;
                break;
            }
        }
    }
```

```
// Si celui qu'on peut finir est dans le premier tableau, on trouve
    sa position

if (placeLibre == 1) {
    for (UIImageView *celleATrois in _premierCarre) {
        if ([celleATrois tag] == 3) {
            positionIA = [_premierCarre indexOfObject:celleATrois];
            break;
        }
    }
}

// Pareil si dans le deuxième tableau

else if (placeLibre == 2) {
    for (UIImageView *celleATrois in _deuxiemeCarre) {
        if ([celleATrois tag] == 3) {
            positionIA = [_deuxiemeCarre indexOfObject:celleATrois];
            break;
        }
    }
}

// Sinon, position aléatoire

else {
    compteur = 0;

    // SI l'IA risque de donner un point ( tag à deux ), on la fait
        rejouer ailleur. Mais après 100 essais, tant pis.
    positionIA = (arc4random() % [_traits count]);
    while ([[_premierCarre objectAtIndex:positionIA] tag] == 2 ||
        [[_deuxiemeCarre objectAtIndex:positionIA] tag] == 2) {
        positionIA = (arc4random() % [_traits count]);
        compteur ++;
        if (compteur == 300) {
            break;
        }
    }
}

// On ajoute 1 au tag d'ou l'IA à joué

[[_premierCarre objectAtIndex:positionIA] setTag:([[_premierCarre
    objectAtIndex:positionIA] tag ] + 1)];
```

```
[[_deuxiemeCarre objectAtIndex:positionIA] setTag:([[_deuxiemeCarre
    objectAtIndex:positionIA] tag ] + 1)];

// On colorie le trait

_traitUnIA = [_traits objectAtIndex:positionIA];
_traitUnIA.image = [UIImage imageNamed:@"rouge.jpg"];

// SI elle ferme un carré, on le rempli est c'est de nouveau à
    elle de jouer

if ([[_premierCarre objectAtIndex:positionIA] tag] == 4) {
    UIImageView *carreUnIA = [_premierCarre objectAtIndex:positionIA];
    carreUnIA.image = [UIImage imageNamed:@"CarreRouge.jpg"];
    nombreDePointsDuRouge ++;
    [audioPlayer play];
    tourDeLIA = 1;

}

if ([[_deuxiemeCarre objectAtIndex:positionIA]tag] == 4) {
    UIImageView *carreDeuxIA = [_deuxiemeCarre objectAtIndex:
        positionIA];
    carreDeuxIA.image = [UIImage imageNamed:@"CarreRouge.jpg"];
    nombreDePointsDuRouge ++;
    [audioPlayer play];
    tourDeLIA = 1;

}

// Sinon, ce n'est plus à elle de jouer

else if  ([[_premierCarre objectAtIndex:positionIA] tag] != 4 &&
    [[_deuxiemeCarre objectAtIndex:positionIA]tag] != 4){
    tourDeLIA = 0; //////
}

// On enlève les objets des tableaux

[_premierCarre removeObjectAtIndex:positionIA];
[_deuxiemeCarre removeObjectAtIndex:positionIA];
[_traits removeObjectAtIndex:positionIA];

// On actualise les scores

self.scoreBleu.text = [NSString stringWithFormat: @"%d",
    nombreDePointsDuBleu];
```

```
    self.scoreRouge.text = [NSString stringWithFormat: @"%d",
        nombreDePointsDuRouge];

    if ([_traits count] ==  0) {
        [timer invalidate];
        timerAllume = FALSE;
        [self fin];
    }
    else if (tourDeLIA == 0) {
        [timer invalidate];
        timerAllume = FALSE;
        self.texte.text = @"C'est au joueur Bleu de jouer";
    }

}


// Quand on appuye

- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event {
    if (timerAllume == FALSE) {
        dejaFait = 0;

        // On capte la position du clic

        UITouch *touch = [touches anyObject];
        CGPoint point = [touch locationInView:self.view];
        CGFloat pointX = point.x;
        CGFloat pointY = point.y;
        int x = pointX;
        int y = pointY;
        int positionXImage;
        int positionYImage;

        for (UIImageView *image in _traits) {

            // On définit la position et la taille de chaque image

            CGPoint positionBase = image.frame.origin;
            positionXImage = positionBase.x + image.frame.size.width/2;
            positionYImage = positionBase.y + image.frame.size.height/2;
            tailleImage = image.frame.size.width/2 +
                image.frame.size.height/2;

            // On crée un tableau avec les distances entre où on
                clique et chaque trait
```

```
        float distance = sqrtf(pow(positionXImage - x, 2) +
            (pow(positionYImage - y, 2)));
        NSNumber *distanceNombre = [NSNumber numberWithFloat:distance];
        [_tableauDistances addObject:distanceNombre];
    }

    // On cherche la valeur minimum du tableau

    int valeurMin = 1000;
    for (NSNumber * intMin in _tableauDistances) {

        int valeurActuelle = [intMin intValue];

        if (valeurActuelle < valeurMin) {

            valeurMin = valeurActuelle;
        }
    }

    // On trouve la position dans le tableau du trait vers lequel
        on appuye

    for (NSNumber *laDistance in _tableauDistances){

        if (dejaFait == 0) {

            if (valeurMin == [laDistance intValue] & valeurMin <
                tailleImage){
                dejaFait = 1;

                indexMin = [_tableauDistances indexOfObject:laDistance];

                // On ajoute 1 au tag

                [[_premierCarre objectAtIndex:indexMin] setTag:
                    ([[_premierCarre objectAtIndex:indexMin] tag ]
                    + 1)];
                [[_deuxiemeCarre objectAtIndex:indexMin] setTag:
                    ([[_deuxiemeCarre objectAtIndex:indexMin] tag ]
                    + 1)];

                NSInteger tag1 = [[_premierCarre objectAtIndex:
                    indexMin] tag ];
                int numeroTag1 = tag1;
                NSInteger tag2 = [[_deuxiemeCarre objectAtIndex:
                    indexMin]  tag ];
                int numeroTag2 = tag2;
```

```
// Pour 2, 3 ou 4 joueurs, on execute la fonction
    correspondante

if (_nombreDeJoueurs == 2) {
    [self fonction2Joueurs:numeroTag1 andOther:
        numeroTag2 andOther:[_traits objectAtIndex:
        indexMin] andOther:[_premierCarre
        objectAtIndex: indexMin] andOther:
        [_deuxiemeCarre objectAtIndex: indexMin]];
}
else if (_nombreDeJoueurs == 3) {
    [self fonction3Joueurs:numeroTag1 andOther:
        numeroTag2 andOther:[_traits objectAtIndex:
        indexMin] andOther:[_premierCarre
        objectAtIndex: indexMin] andOther:
        [_deuxiemeCarre objectAtIndex: indexMin]];
}
else if (_nombreDeJoueurs == 4) {
    [self fonction4Joueurs:numeroTag1 andOther:
        numeroTag2 andOther:[_traits objectAtIndex:
        indexMin]andOther:[_premierCarre
        objectAtIndex: indexMin]andOther:
        [_deuxiemeCarre objectAtIndex: indexMin]];
}

else if (_nombreDeJoueurs == 1){ // 1 joueur

    // On colore le trait

    _traitUnJoueur = [_traits objectAtIndex:indexMin];
    _traitUnJoueur.image = [UIImage imageNamed:
        @"bleu.jpg"];

    // Si il fini un des carré, on le rempli

    if ([[_premierCarre objectAtIndex:indexMin] tag]
        == 4) {
        UIImageView *carreUnJoueur = [_premierCarre
            objectAtIndex:indexMin];
        carreUnJoueur.image = [UIImage imageNamed:
            @"CarreBleu.jpg"];
        nombreDePointsDuBleu ++;
        [audioPlayer play];
    }
    if ([[_deuxiemeCarre objectAtIndex:indexMin]tag]
        == 4) {
```

```
    UIImageView *carreDeuxJoueur = [_deuxiemeCarre
        objectAtIndex:indexMin];
    carreDeuxJoueur.image = [UIImage imageNamed:
        @"CarreBleu.jpg"];
    nombreDePointsDuBleu ++;
    [audioPlayer play];;
}

// Si il n'en rempli pas, c'est au tour de l'IA

if ([[_premierCarre objectAtIndex:indexMin] tag]
    != 4 && [[_deuxiemeCarre objectAtIndex:
    indexMin] tag] != 4) {
    tourDeLIA = 1;

}

// On enleve le trait et les carrés adjacents
    des tableaux

[_premierCarre removeObjectAtIndex:indexMin];
[_deuxiemeCarre removeObjectAtIndex:indexMin];
[_traits removeObjectAtIndex:indexMin];

// On actualise le score

self.scoreBleu.text = [NSString stringWithFormat:
    @"%d", nombreDePointsDuBleu];
self.scoreRouge.text = [NSString stringWithFormat:
    @"%d", nombreDePointsDuRouge];

// Si c'est au tour de l'IA

if (tourDeLIA == 1) {

    self.texte.text = @"C'est au tour de l'IA
        de jouer";

    timerAllume = TRUE;

    NSTimer *timer = [NSTimer
        scheduledTimerWithTimeInterval: 1 target:
        self selector:@selector(onTick:) userInfo:
        nil repeats:YES];
}
else if ([_traits count] == 0) {
    [self fin];
```

```
                        }
                    }

                    // Pour les modes multi, on enlève des tableaux

                    if (_nombreDeJoueurs != 1) {
                        [_premierCarre removeObjectAtIndex:indexMin];
                        [_deuxiemeCarre removeObjectAtIndex:indexMin];
                        [_traits removeObjectAtIndex:indexMin];
                        if ([_traits count] == 0) {
                            [self fin];
                        }
                    }
                }

            }

        }

        // On remet les distances à 0

        [_tableauDistances removeAllObjects];
    }
}

- (void) fin {

    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuBleu]];
    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuRouge]];
    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuVert]];
    [_scores addObject:[NSNumber numberWithInt:nombreDePointsDuJaune]];


    int valeurMax = -1;
    for (NSNumber * intMin in _scores) {

        int valeurActuelle = [intMin intValue];

        if (valeurActuelle > valeurMax) {

            valeurMax = valeurActuelle;
        }
    }

    if ([[_scores objectAtIndex:0]integerValue] == valeurMax) {
        [_podium addObject:[NSString stringWithFormat:@"Bleu"]];
    }
```

```
if ([[_scores objectAtIndex:1]integerValue] == valeurMax) {
    [_podium addObject:[NSString stringWithFormat:@"Rouge"]];
}
if ([[_scores objectAtIndex:2]integerValue] == valeurMax) {
    [_podium addObject:[NSString stringWithFormat:@"Vert"]];
}
if ([[_scores objectAtIndex:3]integerValue] == valeurMax) {
    [_podium addObject:[NSString stringWithFormat:@"Jaune"]];
}

// Selon le nombre de gens dans le podium, on écrit le texte de fin

if ([_podium count] == 1 ) {
    _nomDuMeilleur = [NSString stringWithFormat:@"%@ a gagné avec
        %d points",[_podium objectAtIndex:0],valeurMax ];
}

if ([_podium count] == 2 ) {
    _nomDuMeilleur = [NSString stringWithFormat:@"%@ et %@ sont en
        tête avec %d points",[_podium objectAtIndex:0],[_podium
        objectAtIndex:1],  valeurMax];

}
if ([_podium count] == 3 ) {
    _nomDuMeilleur = [NSString stringWithFormat:@"%@, %@ et %@
        sont en tête avec %d points",[_podium objectAtIndex:0],
        [_podium objectAtIndex:1], [_podium objectAtIndex:2],
        valeurMax];
}
if ([_podium count] == 4 ) {
    _nomDuMeilleur = [NSString stringWithFormat:@"%@, %@, %@ et %@
        sont en tête avec %d points",[_podium objectAtIndex:0],
        [_podium objectAtIndex:1],[_podium objectAtIndex:2],
        [_podium objectAtIndex:3], valeurMax];
}

UIAlertView *fin = [[UIAlertView alloc] initWithTitle:@"Bravo !"
    message:[NSString stringWithFormat:@"%@",_nomDuMeilleur]
    delegate:nil cancelButtonTitle:@"OK" otherButtonTitles:nil, nil];

// On définit un nouveau son

AudioSessionInitialize (NULL, NULL, NULL, (__bridge void *)self);
UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback;
AudioSessionSetProperty (kAudioSessionProperty_AudioCategory,
    sizeof (sessionCategory), &sessionCategory);
NSData *soundFileData;
```

```
    soundFileData = [NSData dataWithContentsOfURL:[NSURL fileURLWithPath:
        [[NSBundle mainBundle] pathForResource:@"fin.mp3" ofType:NULL]]];
    audioPlayer = [[AVAudioPlayer alloc] initWithData:soundFileData
        error:NULL];
    audioPlayer.delegate = self;
    [audioPlayer setVolume:1.0f];

    [audioPlayer play];

    [fin show];
    [_scores removeAllObjects];


}

- (void)viewWillDisappear:(BOOL)animated {
    nombreDePointsDuBleu = 0;
    nombreDePointsDuRouge = 0;
    nombreDePointsDuVert = 0;
    nombreDePointsDuJaune = 0;
    leTour = 0;
    [_scores removeAllObjects];

}
@end
```