Rapport de projet : Candy Crush Killer

IUT de Clermont-Ferrand - Département Informatique - Année 2014-2015

Boyer Alexandre, Bullat Théo, Ralite Jérome, Raymond Nicolas

Nous remercions M. Kauffmann et M. Laffont pour leur encadrement tout au long de ce projet.

Sommaire

Introduction

Dans le cadre de notre deuxième année au département informatique de l'IUT de Clermont-Ferrand, nous avons eu l'opportunité de réaliser un projet d'une durée de quatre mois en équipe. Cette dernière était composée de cinq étudiants: Alexandre Boyer, Théo Bullat, Jérôme Ralite, Nicolas Raymond et Laurent Ribière. Laurent étant retourné en première année, nous avons continué le projet à quatre.

Ce projet nommé « Candy Crush Killer », nous a été proposé par M. Kauffmann enseignant chercheur à l'IUT de Clermont Ferrand et M. Laffont enseignant à Polytech Clermont. Il avait pour intérêt de nous apprendre à travailler en équipe, en nous obligeant à nous répartir les différentes tâches, ainsi qu'en utilisant des outils tels que « Subversion », logiciel de gestion de versions, qui sont des outils régulièrement utilisés en entreprise. Un autre intérêt était de découvrir la programmation Android[1] et de tester les limites des smartphones sous l'OS.

Le sujet de ce projet était de réaliser une application Android capable de résoudre automatiquement des jeux informatiques chronophages tel que le célèbre Candy Crush Saga, cas étudié dans le cadre du projet, qui est un jeu de puzzle dans lequel le but est de regrouper des bonbons par couleur afin de les faire exploser.

Une problématique apparaît alors : comment concevoir une application permettant de résoudre les puzzles aléatoires du jeu Candy Crush Saga de manière autonome ?

Pour répondre à cette problématique nous avons dû créer une application Android qui filme un écran de pc affichant une grille de jeu. L'application analyse cette grille, en déduit les meilleurs coups à jouer et les envois au pc. Le pc déplace alors sa souris pour regrouper les bonbons.

Dans ce rapport nous présenterons dans un premier temps le projet commencé l'année dernière et ses objectifs puis dans un second temps, nous montrerons les principales étapes de la réalisation du projet allant de son organisation au développement des différentes tâches. Nous terminerons en établissant un bilan sur l'état du projet.

1) Présentation du projet

1.1) Objectifs

L'objectif principal de l'application était de résoudre des jeux informatiques simples tels que Tétris ou Candy Crush Saga automatiquement à l'aide d'un smartphone Android. Le smartphone devait filmer un écran de PC et être reconnu comme un périphérique standard par celui ci. Il devait alors déduire des actions à mener afin de jouer en utilisant une stratégie optimale. Les délais du projet étant relativement courts, il a été décidé qu'un seul jeu serait traité : Candy Crush Saga.

Candy Crush Saga est un jeu vidéo de réflexion développé par King en 2010 dont le but est de réaliser des combinaisons de bonbons colorés pour remplir différents objectifs dans les niveaux et de marquer un maximum de points. Ces bonbons sont positionnés sur une grille allant de 3 à 9 colonnes et lignes et peuvent être déplacés de haut en bas et de droite à gauche. Une combinaison consiste en un alignement d'au moins trois bonbons de même couleur sur une ligne ou colonne et permet de faire apparaître de nouveaux bonbons de façon aléatoire. L'objectif premier de notre application était donc de réaliser les meilleures combinaisons de bonbons possibles.

Figure n°1 : Grille de jeu de Candy Crush Saga

L'application devait être capable de capturer des images de la grille de bonbons à la suite et devait pouvoir les traiter pour en extraire une information cohérente et utilisable. En effet, de nombreuses contraintes environnementales extérieures peuvent dégrader les images et il était donc nécessaire de palier cela. L'application devait aussi comporter une phase d'analyse pour numériser les couleurs

des bonbons ainsi qu'une intelligence artificielle (IA[7]) pour déterminer les meilleures combinaisons de jeu. Enfin, elle devait être capable d'être reconnue comme un périphérique standard par le pc afin de pouvoir déplacer la souris.

1.2) Existant

« Candy Crush Killer » est un projet qui a débuté au cours de l'année 2013-2014 par cinq étudiants de deuxième année au sein de notre département. L'équipe avait déjà beaucoup réfléchi au fonctionnement de l'application, les différentes étapes nécessaires à la résolution et avait réalisé une IA fonctionnelle sur laquelle elle s'était concentrée principalement. Elle avait aussi débuté le travail sur l'application smartphone et avait réalisé un début de traitement d'image et d'analyse. La partie communication pc était totalement absente. Au final, la plupart des étapes qui devaient se dérouler sur le smartphone étaient implantées sur pc et étaient plus ou moins avancées.
A partir de ce constat et avec l'accord de nos encadrants, nous avons repris le projet à la base en recréant une application tout en conservant leurs idées sur le fonctionnement de l'application smartphone.

1.3) Environnement

Dans le but de réaliser ce projet, nous avons utilisé différents outils. Pour ce qui est de l'organisation du travail, nous nous sommes servis de « MSProject », logiciel de gestion de projets, qui nous a permis de créer notre diagramme de Gantt[3] et d'établir un suivi du projet. Nous avons aussi utilisé « TortoiseSVN », logiciel de gestion de versions, afin de pouvoir versionner nos travaux, de mettre notre code en commun aisément et avoir un travail organisé et efficace. Ce logiciel nous permettait en effet de travailler à plusieurs sur le projet et le tout était stocké sur la « Forge Clermont Université » qui nous a aussi offert la possibilité d'établir notre rapport.
Dans le cas de la programmation, nous avons utilisé Eclipse avec le SDK d'Android pour créer notre application smartphone ainsi que JavaFX pour réaliser notre légère application pc.

Concernant le matériel, un smartphone était nécessaire pour construire l'application rapidement et tester notre travail. Un support était impératif sur le long terme afin de tenir le smartphone debout durant le processus de résolution. Pour ces raisons, un Samsung Galaxy[2] nous a été fourni et nous avons fabriqué notre propre support, adapté à tout type de smartphone grâce à une aide extérieur. Nous avons aussi reçu une clé Bluetooth afin de faire communiquer le smartphone avec le pc.

2) Réalisation du projet

2.1) Gestion du projet

Afin d'être efficace au maximum sur la réalisation du projet, nous nous sommes partagés le travail de façon équitable selon les goûts et les compétences de chacun. Le projet ayant déjà démarré l'an passé, nous nous sommes concentrés sur les tâches critiques pour assurer son bon fonctionnement dans les temps impartis. Ces grandes tâches étaient :

- la prise photo et le traitement de l'image principalement réalisé par Alexandre Boyer
- l'analyse de l'image par Théo Bullat
- l'IA par Laurent Ribière puis par Théo Bullat
- la communication smartphone/pc par Jérome Ralite et Nicolas Raymond
- la construction de l'application par Alexandre Boyer

Malgré ces affectations, nous avons beaucoup communiqué entre nous et nous sommes souvent entraidés.

2.1.1) Diagramme de Gantt prévisionnel (cf annexe 1)

Nous devions débuter le projet le 17 novembre 2014 pour le finir le 26 mars 2015. Nous avions prévu d'étudier le rapport et la conception des étudiants durant une première semaine puis de commencer le développement sur la base de l'ancien :

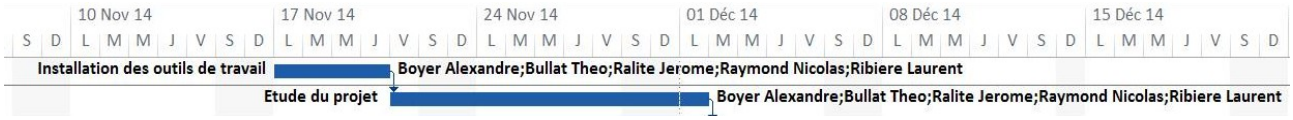


Figure n°2 : Gantt prévisionnel sur l'étude du projet existant

L'objectif était de fournir une application fonctionnelle capable de réaliser une combinaison sur un niveau simple et communiquant avec le pc via Wi-Fi[4] pour la première soutenance du 22 janvier 2015 :

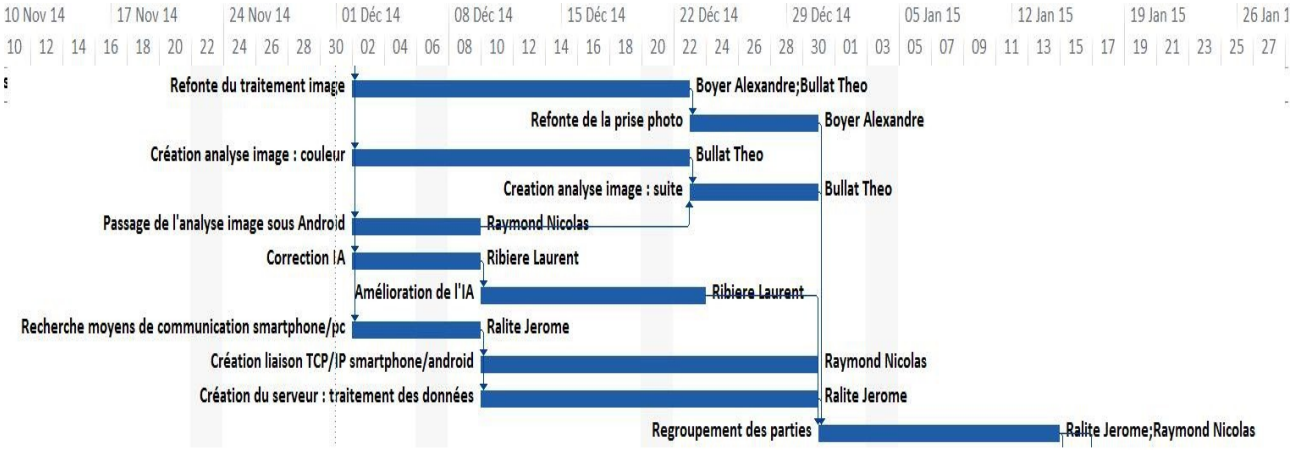


Figure n°3 : Gantt prévisionnel jusqu'au 22 janvier 2015

Pour la soutenance finale, « Candy Crush Killer » devait être capable de terminer un niveau simple du jeu. Tout devait se faire sur l'application smartphone et celle-ci devait communiquer via Bluetooth avec le pc. L'application devait aussi être construite de telle sorte qu'on puisse y ajouter un autre jeu facilement :

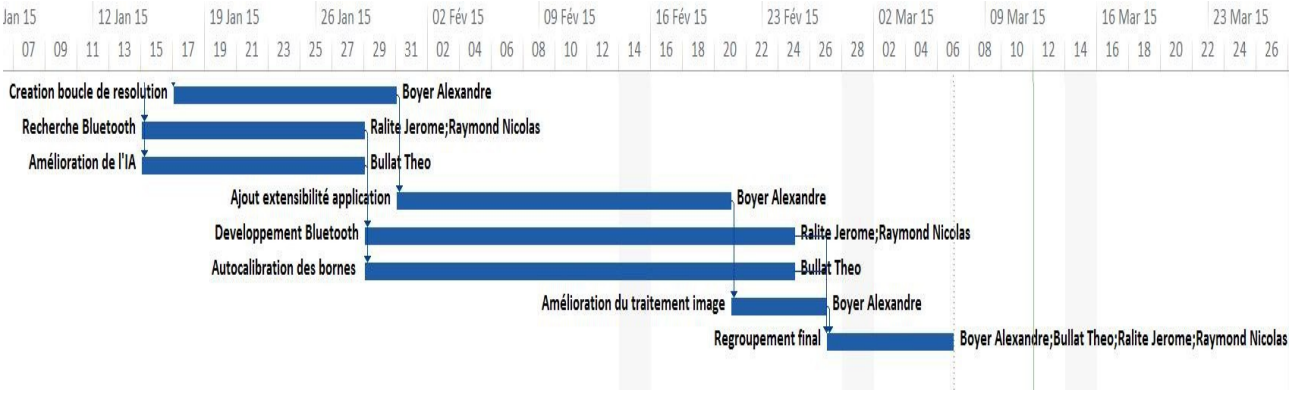


Figure n°4 : Gantt prévisionnel du 22 janvier 2015 au … mars 2015

2.1.2) Diagramme de Gantt réel (cf annexe 2)

Le projet a débuté le 17 novembre et a été terminé le 26 Mars comme prévu. L'étude du rapport et de la conception s'est fait dans les temps impartis. Malgré quelques changements mineurs, une première application fonctionnelle à pu être montrée en soutenance intermédiaire et les objectifs intermédiaires ont pu être atteints globalement :

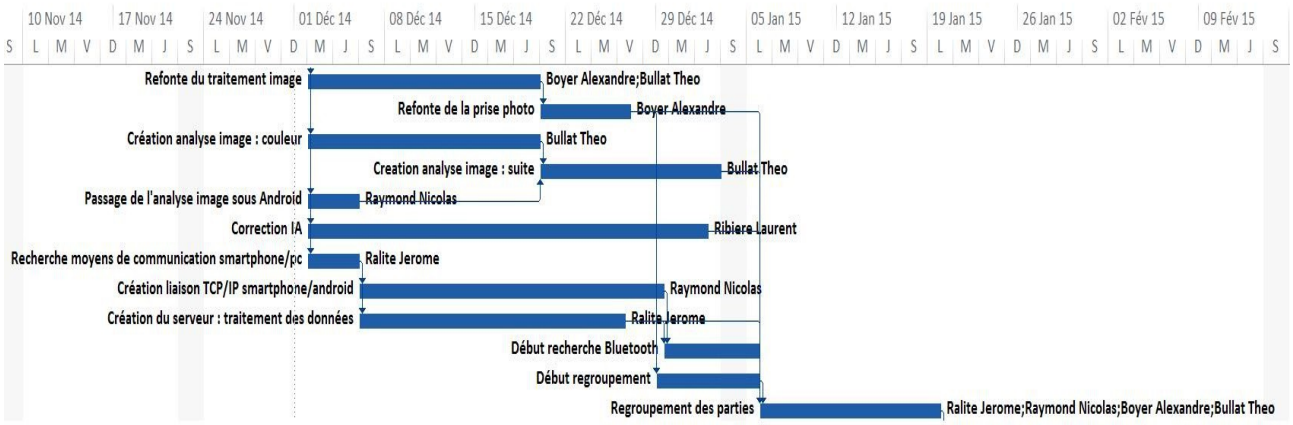


Figure n°5 : Gantt réel jusqu'au 22 janvier 2015

L'amélioration de l'intelligence artificielle n'a pas pu être faite, Laurent Ribière ayant préféré se concentrer avant tout sur la correction de l'IA existante et la simplification du code. Les recherches sur le Bluetooth furent avancées pour découvrir rapidement si la solution était envisageable compte tenu du temps restant et furent finalement abandonnées. La solution Bluetooth ne fut pas retenue ce qui changea considérablement l'organisation du travail à réaliser :

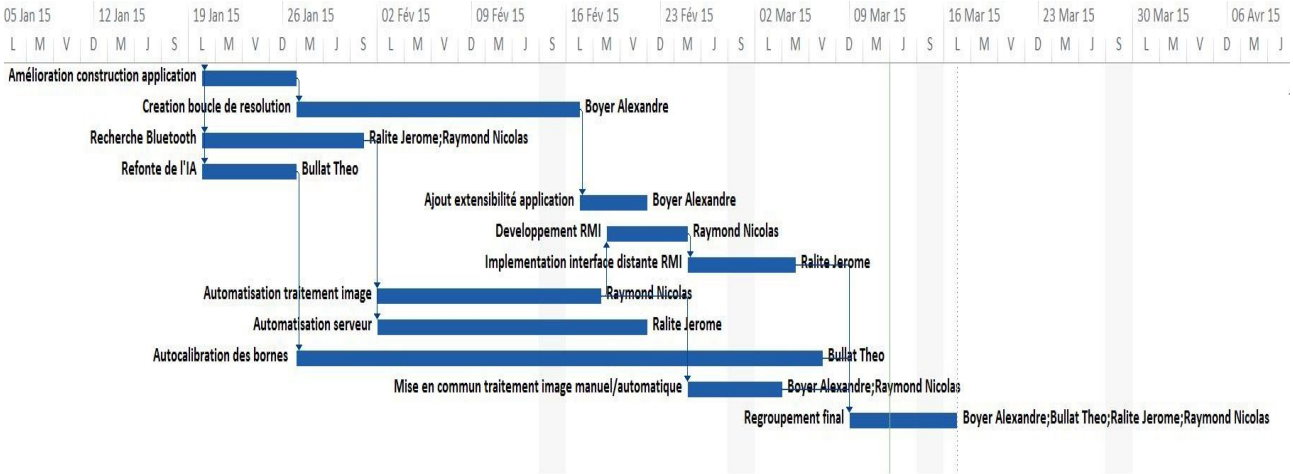


Figure n°6 : Gantt réel du 22 janvier 2015 au ... mars 2015

Nous avons donc décidé de nous concentrer sur l'automatisation de l'application smartphone, pour faire en sorte qu'elle soit le plus simple possible à utiliser. Jérome et Nicolas qui devaient se charger du bluetooth ont cherché et mis en place un moyen de communication plus simple. Nicolas s'est aussi chargé d'automatiser le traitement de l'image. Alexandre a amélioré l'application existante et a

passé plus de temps que prévu sur la mise en place d'une boucle pour permettre la résolution complète d'un niveau. Le résultat est donc assez éloigné de ce qu'il devait être.*(cf annexe 1)*

2.2) Diagramme de classes

Nous avons divisé le projet en deux parties bien distinctes qui sont l'application smartphone et l'application pc. Du côté du smartphone, on retrouve donc différents paquets :

- Le traitement de l'image *(cf annexe 3)*,
- L'analyse de l'image *(cf annexe 4)*,
- L'intelligence artificielle *(cf annexe 5)*,
- La communication smartphone *(cf annexe 6)*.

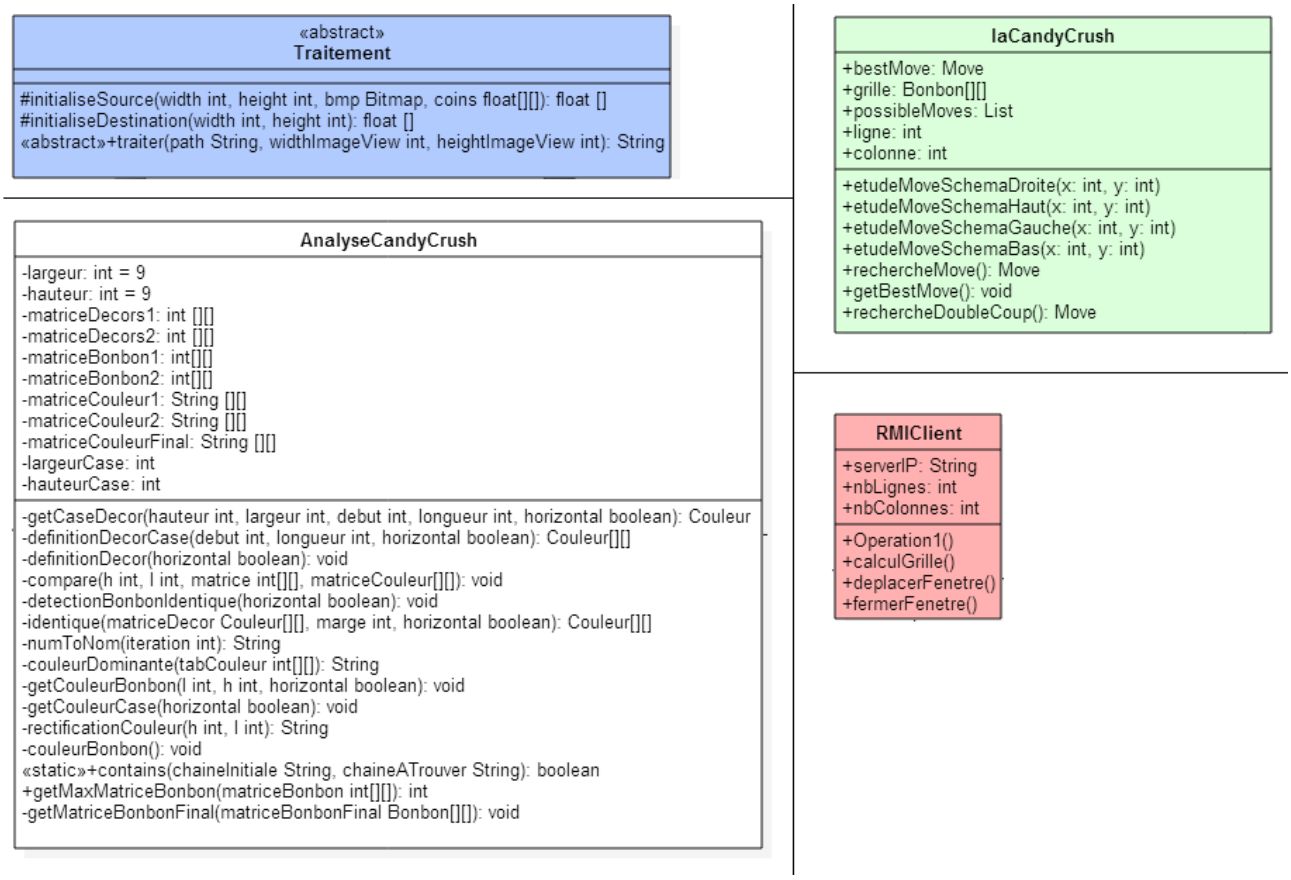


Figure n°7 : diagramme de classes simplifié

Quant au projet pc, il est composé d'un seul paquet concernant la communication *(cf annexe 7)*.

2.3) Travail effectué

2.3.1) Analyse d'image

Cette partie a été mise en place afin de permettre à l'application smartphone de reconnaître et de reproduire, sous la forme d'une matrice de chaînes de caractères, la grille du jeu Candy Crush Saga. Elle a été développée par Theo Bullat tout au long de l'année sur la base du code de Pierre Thomazon, élève travaillant sur le projet l'an passé. Elle représente la liaison entre l'image donnée par le traitement d'images et l'intelligence artificielle. Cette reconnaissance se fait à partir d'une image au format Bitmap[6], pré-découpé au niveau de la grille. La création de la matrice finale se découpe alors en trois étapes différentes à savoir :

- Le découpage de l'image
- La reconnaissance du décor
- La reconnaissance des bonbons

Chaque étape sera développée et présentée ci-après, cependant la reconnaissance des bonbons sera découpée en deux parties différentes. Une première où nous verrons son déroulement actuel, et une seconde où nous présenterons la deuxième solution que nous avons envisagée, ses avantages mais également les raisons pour lesquelles nous ne l'avons pas choisie. Toute la reconnaissance se faisant autour de la couleur, nous verrons également comment cette dernière se compose, et comment nous l'utilisons pour notre reconnaissance.

2.3.1.1) Découpage de l'image

Comme vous pouvez le voir sur la fig.8 une grille de jeu peut se découper en deux zones différentes à savoir une zone de décor et une grille de jeu composée de plusieurs cases. Chaque case contient un bonbon qui sera identifié lors de la reconnaissance que nous verrons plus tard. La première étape consiste donc à déterminer ces deux zones, sachant que ces dernières changent à chaque niveau. Sans cette étape la reconnaissance s'effectuerait sur toute l'image et pourrait trouver des bonbons grâce aux couleurs présentes dans les illustrations du décor. Cela offrirait alors la possibilité à l'intelligence artificielle de trouver un coup qui n'existe pas, ce qui entraînerait alors l'application dans une boucle infinie où elle cliquerait sans interruption sur la case du décor et un bonbon.

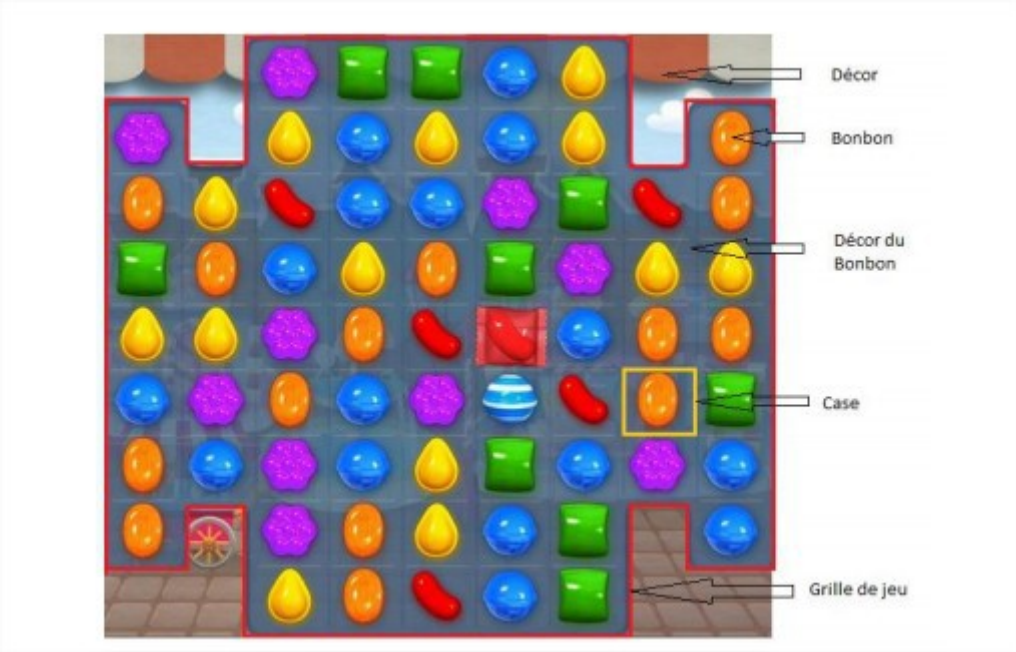


Figure n°8 : Découpage d'une grille de Candy Crush

Afin de s'assurer que la reconnaissance puisse fonctionner sur plusieurs grilles différentes, nous avons cherché et listé plusieurs règles permettant d'établir un protocole automatique qui fonctionne tout le temps. Voici ces règles :

1. Une grille peut toujours être décomposée sous forme de lignes et de colonnes. Il n'est donc pas possible de se retrouver avec des décalages d'une demi-case entre deux colonnes ou lignes consécutives.
2. Un décor peut toujours être décomposé comme un groupe de case, un décor ne peut donc pas être entouré de quatre bonbons.
3. Chaque ligne ou colonne peut au maximum être composé de 9 cases, qu'elle soit composée d'un bonbon ou non.
4. Chaque case comportant un bonbon possède un décor gris/bleu en fond.

Ainsi, en demandant à l'utilisateur de rentrer la taille de la grille qu'il veut résoudre, nous sommes capables de découper notre grille en plusieurs cases en divisant l'image par le nombre de lignes pour la hauteur, et le nombre de colonnes pour la largeur. Il suffit alors d'observer chaque case une par une afin de déterminer si cette dernière est un décor ou un bonbon.

2.3.1.2) Principe de la reconnaissance des couleurs

Que ce soit pour la reconnaissance du décor ou la reconnaissance des bonbons, nous avons utilisé la reconnaissance des couleurs. En effet il aurait était possible de reconnaître la forme de la grille ou la forme des bonbons. Mais le projet ayant était commencé l'an passé avec la couleur, nous avons préféré continuer sur cette même méthode.

Pour comprendre le fonctionnement de nos reconnaissances, il est important de bien comprendre comment se compose une couleur. Elle est tout d'abord constituée de trois composantes qui correspondent aux couleurs rouge, vert et bleu. Cette méthode de composition de couleurs est appelée découpage RGB, correspondant aux couleurs anglaise Red, Green, Blue. Chaque composante peut prendre une valeur entre 0 et 255. Plus la valeur augmente plus la teneur de la couleur de la composante sera importante dans la couleur final. Ainsi la couleur noir sera obtenu avec les valeurs (0,0,0) pour les composantes (R,G,B). A l'inverse, des valeurs maximales (255,255,255) donneront une couleur blanche.

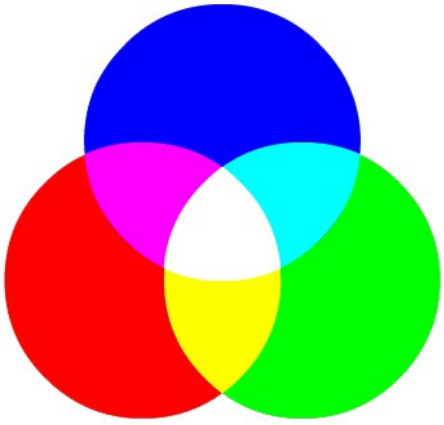


Figure n°9 : Combinaisons de composantes RGB

Par conséquent, chaque combinaison de valeurs amène à une couleur. Et de simples calculs sur les composantes permettent de traiter les couleurs. Pour exemple, faire le mélange de deux couleurs revient à faire la moyenne des composantes des deux couleurs pour chaque valeur de RGB.

En suivant le même principe, il est possible de reconnaître une couleur en comparant ses valeurs de RGB avec des bornes. Chaque borne correspondant à une plage de valeurs que peut prendre chaque composante. Pour exemple, une couleur sera violette si et seulement si ses composantes rouge et

bleu sont comprises entre 150 et 255, et que sa composante verte est presque nulle. Nous verrons l'utilisation de ce principe plus en détail lors de la reconnaissance des bonbons.

2.3.1.3) Reconnaissance du décor

Pour déterminer les cases représentant un décor nous avons utilisé la seconde et la quatrième règles que nous avons vues précédemment, à savoir qu'une case comportant un bonbon possède un décor gris/bleu en fond, et qu'une case décor ne peut pas être encerclée par quatre cases bonbon. Ainsi en observant les quatre côtés de chaque case, on peut déterminer si elles comportent un bonbon ou du décor.

Figure n°10 : Une case décor et une case bonbon

Deux passages sont effectués pour chaque case. Une première fois horizontalement pour observer les côtés haut et bas et une deuxième verticalement pour les côtés gauche et droite. Comme on peut le voir sur la figure ci-dessus, les couleurs des côtés sont bien plus unies pour une case bonbon plutôt que pour une case décor. C'est donc sur cette différence que la reconnaissance se base.

L'observation horizontale et verticale étant basée sur le même principe nous regarderons en détail la méthode horizontale. Nous cherchons donc à comparer le coté haut et le côté bas de la case. Pour cela, nous observons deux bandes de pixels[5] ayant pour longueur la taille de la case, et une largeur de trois pixels. Trois étant une taille assez grande pour pouvoir faire une moyenne sur un nombre de pixels important sans pour autant observer un pixel qui appartiendrait à un bonbon.

Figure n°11 : Bande de pixels observée pour la méthode horizontale

Nous formons ensuite une nouvelle couleur en sommant les composantes de chaque couleur pour chaque pixel, puis nous divisons chaque somme par le nombre de pixels. Nous obtenons ainsi trois valeurs correspondant à la moyenne de chaque composante. Ces trois valeurs donnant une couleur moyenne. De la même manière, nous calculons la couleur moyenne de la seconde bande de pixels. Enfin nous comparons les deux couleurs moyennes en se laissant une marge d'erreur de 10%. Cette marge permettant de limiter l'impact de la qualité de l'image sur la reconnaissance. Si les deux couleurs sont considérées comme identiques, de manière horizontale comme verticale, la case est considérée comme étant une case bonbon. A l'inverse, lorsque l'on trouve une différence entre les couleurs moyennes des deux bandes, la case est considérée comme étant du décor, et ne sera donc pas traitée lors de la reconnaissance des bonbons.

2.3.1.4) Reconnaissance des bonbons

Pour reconnaître la couleur ou le type d'un bonbon nous avons développé deux méthodes différentes. Chacune des méthodes utilise le même principe de parcours de l'image mais diffère dans la délimitation des bornes de couleurs. Nous allons donc dans un premier temps voir comment le parcours est effectué, puis nous verrons en quoi les deux méthodes diffèrent, leurs avantages et inconvénients.

A ce stade de l'analyse, la reconnaissance du décor ainsi que le découpage de l'image permet d'observer chaque case bonbon une par une. Le but est donc maintenant de reconnaître la couleur et le type de chaque bonbon. Pour cela nous parcourons chaque case horizontalement et verticalement afin d'apporter une certitude plus importante. Cependant observer chaque pixel de chaque case serait bien trop long. Pour éviter cela nous observons uniquement une bande de pixels située au centre de chaque case. Ces deux bandes de pixels sont représentées par les lignes en jaune fig. 12 :

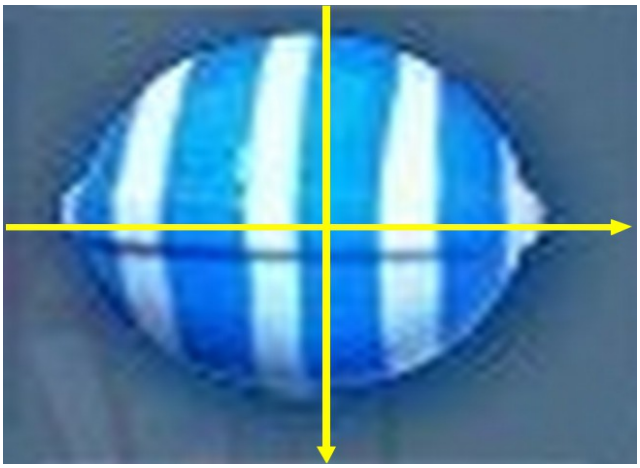


Figure n°12 : Parcours des pixels d'une case bonbon

Les parcours horizontal et vertical étant basés sur le même principe, nous allons voir comment fonctionne le parcours horizontal d'une case.

Pour reconnaître la couleur et le type du bonbon nous observons chaque pixel de la bande et nous comparons ses composantes avec les différentes bornes que nous avons délimitées et que nous verrons par la suite. Ainsi chaque pixel est reconnu comme une couleur. On réalise alors un histogramme qui compte le nombre d'apparitions de chaque couleur dans une bande. La figure ci-dessous montre un exemple d'histogramme que l'on peut obtenir en observant le bonbon de la figure précédente.

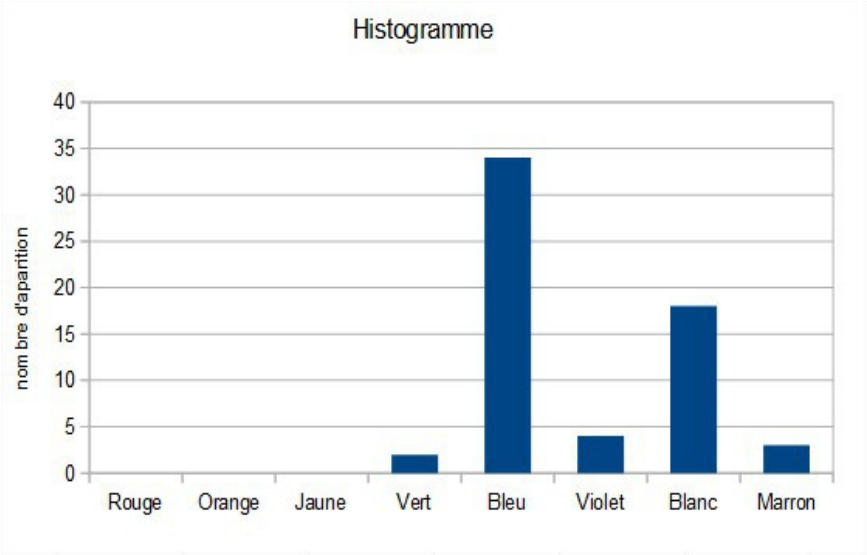


Figure n°13 : histogramme d'une analyse horizontale d'un bonbon bleu rayé

Comme on peut le voir, les couleurs bleu et blanc sont prédominantes. On en déduit alors que le bonbon est bleu et de type spécial. De plus il est possible de déterminer le sens des rayures selon le nombre d'apparitions de blanc en fonction de l'analyse horizontal ou vertical. Ainsi une forte valeur de blanc pour une coupe horizontale indique que le bonbon possède des rayures verticales, et vice versa. Cependant, cette reconnaissance du sens des rayures n'est pas réalisée dans le projet puisque l'intelligence artificielle gère les bonbons rayés de manière globale sans prendre leur orientation en compte.

Le nom de la couleur est alors stocké dans la matrice « matriceCouleur1 » pour la méthode horizontale et « matricecouleur2 » pour la méthode verticale. Une fois les deux parcours réalisés, on compare les deux matrices afin de former la matrice finale. En cas d'incohérence, on utilise la méthode « detectionBonbonIdentique » qui permet de déterminer si deux bonbons sont identiques et ainsi trancher entre la matricecouleur1, et matricecouleur2.

Une fois la matrice finale terminée, elle est retournée, et envoyée à l'intelligence artificielle. Nous allons maintenant voir les deux différentes méthodes que nous avons développées pour délimiter les bornes des couleurs.

2.3.1.5) Délimitation des bornes pour les couleurs : méthode manuelle

Dans un premier temps nous avons prédéfini nous même les bornes de reconnaissance des couleurs. Pour cela nous avons observé à la main les différentes valeurs que peuvent prendre les couleurs de chaque bonbon dans une grille. Afin de gérer indirectement l'impact de la dégradation obtenu lors de la photographie d'un écran, ces valeurs ont été calculées depuis une photo prise par un smartphone et non depuis une capture d'écran de l'ordinateur.

Les bornes sont composées de deux couleurs, elles-mêmes composées de trois composantes RGB. Le principe est de trouver pour une couleur les valeurs les plus basses et les plus hautes possibles pour chaque composante. On obtient ainsi 16 couleurs correspondant aux bornes inférieures et supérieures des 8 couleurs que l'on doit reconnaître. Une couleur intermédiaire correspondant à la couleur moyenne a été introduite au centre de chaque borne. Voici la plage de couleur que l'on obtient :

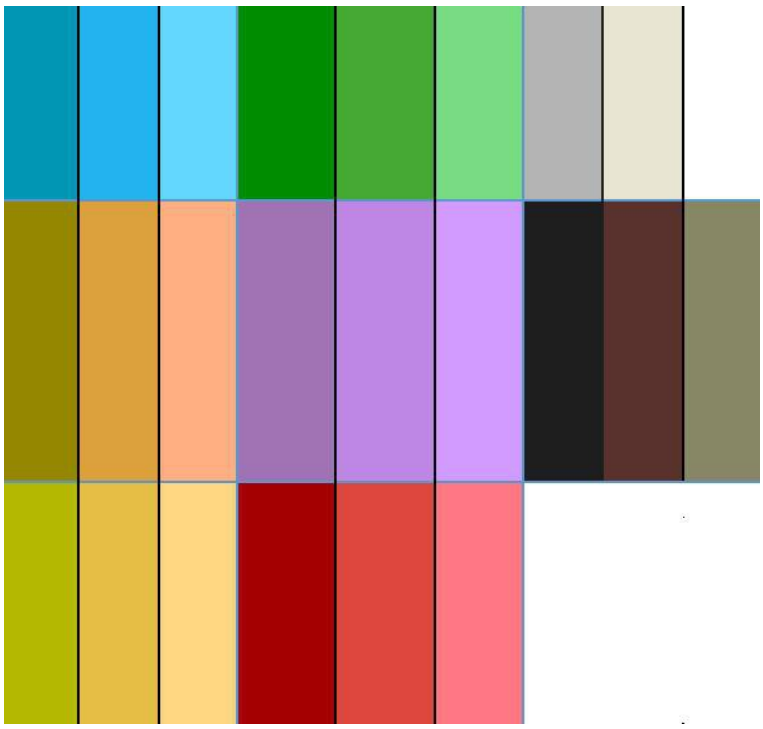


Figure n°14 : Plage de couleurs manuelle

Une première modification a été réalisée afin de sécuriser le choix des couleurs. C'est à dire que l'on s'assure qu'il existe toujours au moins une composante capable de différencier deux couleurs. Ainsi différencier la couleur orange du jaune revient à comparer la valeur de la composante verte. Les composantes rouge et bleu n'ayant pas d'impact sur le choix entre ces deux couleurs. On laisse également une zone creuse entre ces deux bornes afin de ne pas compter les pixels dont la couleur est douteuse.

Un lissage des valeurs a également été réalisé afin d'optimiser les chances de reconnaissance des couleurs. Nous avons, pour cela, augmenté les bornes pour les composantes à faible impact (composantes qui ne permettent pas directement de différencier deux couleurs) puis nous avons égalisé les bornes. C'est à dire que nous avons étalé les bornes pour englober un maximum de valeurs sans pour autant réduire la précision de la reconnaissance des couleurs. Voici au final les bornes que nous utilisons Fig.15 :

	R		G		B	
Rouge	140	255	0	115	0	130
Orange	150	255	115	145	0	130
Jaune	160	255	160	255	0	130
Vert	0	120	110	255	0	150
Bleu	0	80	125	255	180	255
Violet	150	210	110	155	180	255
Blanc	180	255	180	255	180	255
Marron	30	120	30	110	30	100

Figure n°15 : Bornes final utilisé pour la méthode manuelle

Ces bornes sont stockées en dur, directement dans le code, et se retrouvent sous différentes méthodes correspondant à une couleur. On assigne ainsi dans un tableau de taille 6, les différentes bornes pour chaque composante. Puis on appelle une méthode « validation » en lui passant le tableau de bornes. Cette méthode regarde si la couleur du pixel observé est comprise dans ces bornes et renvoie true si c'est le cas. La couleur est alors reconnue pour ce pixel et est comptabilisé dans l'histogramme :

```
public boolean validation(int tabBorne[]){
    int validation=0;

    if (tabBorne[0]<r && r<tabBorne[1])
        validation++;

    if (tabBorne[2]<g && g<tabBorne[3])
        validation++;

    if (tabBorne[4]<b && b<tabBorne[5])
        validation++;

    if (validation ==3)
        return true;
    return false;
}
```

Figure n°16 : méthode validant la couleur d'un pixel

Cependant cette méthode possède un problème majeur. Les bornes étant fixes, elles sont très sensibles au changement d'environnement. Ainsi les bornes peuvent se décaler par rapport aux valeurs que peuvent prendre les bonbons sur une photo. Sur une photo sombre par exemple, un bonbon jaune peut donner une majorité de pixels orange puisque la composante verte est plus faible. On se retrouve alors avec une reconnaissance erronée. Cette erreur entraîne un problème très

important puisque cela donnerait une surabondance de bonbons orange, augmentant ainsi les chances que l'IA trouve un coup non existant. Pour pallier ce problème nous avons tenté de mettre en place une deuxième méthode pour former les bornes de manière dynamique.

2.3.1.6) Délimitation des bornes pour les couleurs : méthode dynamique

Afin d'adapter les bornes à l'environnement dans lequel sont prises les photos, nous avons tenté de développer une méthode qui les calcule dynamiquement. Nous allons donc expliquer son fonctionnement puis nous montrerons pourquoi cette méthode ne pourra pas être utilisée dans le cadre de notre projet.

Dans un premier temps nous avons voulu utiliser le principe de l'étalonnage afin de mettre en place les bornes des couleurs. Pour cela nous avons réalisé une plage de couleur correspondant cette fois-ci non pas à une photo de l'écran, mais à une capture d'écran. Nous obtenons ainsi un découpage des couleurs en fonction de la couleur réelle et sans dégradation de l'image. Voici donc la plage composée des bornes de couleurs réelles :

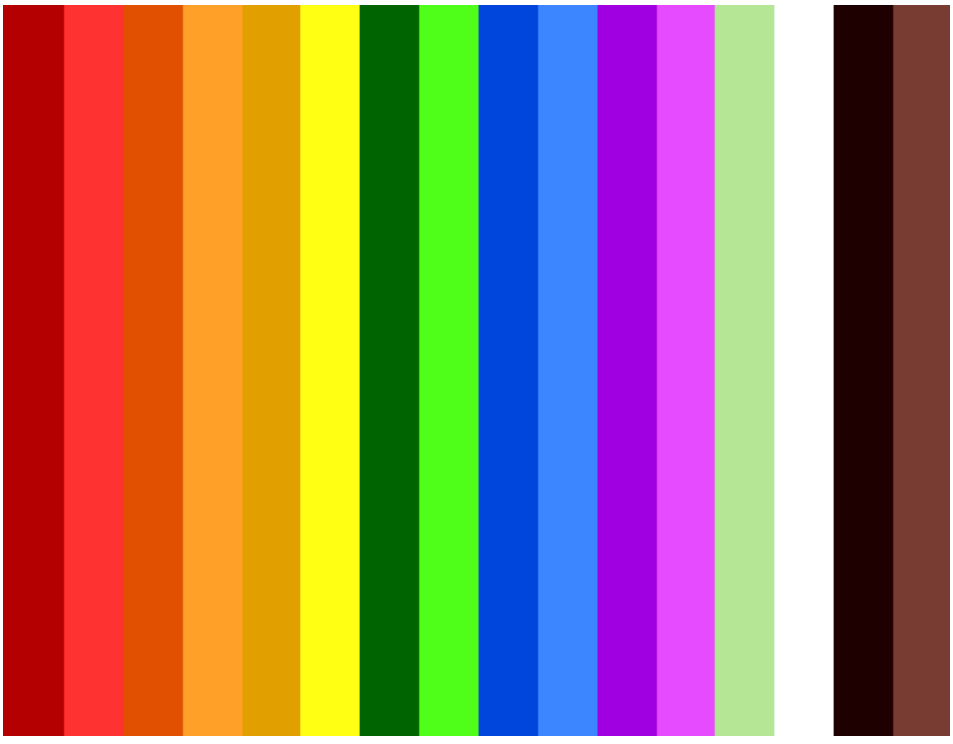


Figure n°17 : Plage de couleur pour méthode dynamique.

Cette plage est alors affichée à l'écran au niveau de la grille de jeu et prise en photo par le téléphone. La photo obtenue représente donc l'impact de la dégradation de l'image pour chaque couleur en fonction de la ligne de la grille. Ainsi lorsqu'une même couleur est perçue plus sombre sur le haut de l'image par rapport au bas, la borne sera également assombrie sur le haut, et s'adaptera ainsi aux couleurs de la grille. Nous avons cependant observé une large différence entre la théorie et la pratique lorsque nous avons réalisé nos premiers tests. En effet la dégradation de l'image est tellement importante que la photo des bornes est inutilisable. Les composantes des pixels observées varient énormément jusqu'à s'inverser entre les bornes inférieures et supérieures. Il est alors impossible d'établir une borne stable et précise afin de reconnaître les bonbons. De plus, la dégradation étant irrégulière le long de l'image, elle amène à des croisements dangereux entre les bornes de différentes couleurs. Si il était précédemment possible qu'un bonbon orange soit reconnu jaune lorsque la composante verte était trop dégradée, il est désormais possible qu'un bonbon orange soit reconnu à la fois comme un bonbon jaune, orange et rouge. Afin de diminuer cette dégradation de l'image et rendre la délimitation dynamique des bornes possible nous avons développé un traitement d'image qui atténue à la fois les reflets et les ombres, que nous verrons plus loin (cf page 33).

2.3.2) Intelligence Artificielle

Cette partie a pour but de déterminer le meilleur coup à jouer à partir d'une matrice de chaînes de caractères représentant la grille de jeu actuelle. Il faudra à la fois reconnaître les coups possibles, c'est à dire les mouvements qui permettront de détruire des bonbons, puis assigner un score à chaque coup afin d'en ressortir la meilleure combinaison à jouer. Afin de bien comprendre comment l'intelligence artificielle fonctionne, nous allons dans un premier temps observer le principe de déplacement dans le jeu Candy Crush, puis nous verrons comment peut se découper la recherche d'un coup. Enfin nous verrons sur quoi se base le score pour chaque coup.

2.3.2.1) Jouer un coup

Le jeu Candy Crush Saga est certes composé de niveaux tous différents les uns des autres, mais chacun des niveaux repose sur un unique et même principe : permuter deux bonbons. Il faut respecter deux règles très simples pour que la permutation soit effective et qu'elle soit reconnue comme un coup :

1. Seuls deux bonbons parfaitement côte à côte peuvent être permutés. Une permutation en diagonal est donc impossible.
2. La permutation des bonbons doit directement amener sur un alignement horizontal ou vertical d'au moins 3 bonbons identiques.

Lorsqu'une permutation amène à un alignement de plus de 3 bonbons, un nouveau bonbon spécial apparaît. Il en existe plusieurs, et chacun possède un bonus propre. Ces bonus sont déclenchés lorsque le bonbon spécial est détruit, et permettent de détruire de nombreux bonbons. Que ce soit la destruction d'une ligne, d'une colonne, des bonbons voisins, ou bien tous les bonbons de la même couleur dans la grille, les bonus permettent de marquer de nombreux points. Chercher le meilleur coup c'est donc chercher la permutation qui détruira ou formera le plus de bonbons spéciaux possibles. Pour cela il existe 3 formes différentes. Dans l'ordre d'importance :

- Un alignement de quatre bonbons identiques forme un bonbon rayé.
- Un double alignement de trois bonbons identiques forme un bonbon emballé
- Un alignement de cinq bonbons identiques forme un bonbon chocolat

En tenant compte de ces règles, et en regroupant les différentes formes d'alignements à rechercher nous avons pu mettre en place un algorithme qui recherche et assigne directement un score à chaque coup.

2.3.2.2) Recherche des coups

Un bonbon ne pouvant être permuté que dans quatre directions différentes, nous avons découpé la recherche en quatre. Chaque partie se base sur le même principe et diffère uniquement sur l'orientation de la recherche. Nous allons donc voir en détail comment se déroule la recherche d'un coup consistant à permuter un bonbon avec le bonbon à sa droite. Nous avons pour cela crée un patron correspondant à la position de tous les bonbons potentiellement intéressants lorsque l'on déplace un bonbon sur la droite. En voici une représentation graphique. Fig.18.

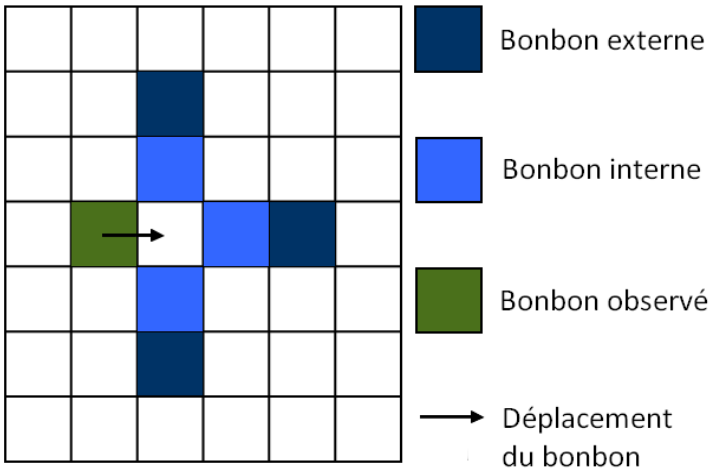


Figure n°18 : Représentation graphique du pattern.

Ce patron permet de reconnaître absolument tous les alignements possibles, depuis les alignements simples de trois bonbons, jusqu'aux alignements spéciaux de quatre et cinq bonbons. Nous parcourons alors la grille afin d'appliquer pour chaque bonbon le patron dans les quatre directions. Si un alignement est reconnu, le coup qui était étudié est ajouté dans une liste.

Une fois la grille entièrement observée, nous parcourons la liste de coups retenus afin de les comparer entre eux. Si nous trouvons deux coups avec les mêmes coordonnées, cela signifie que la permutation forme un alignement pour chacun des bonbons. Par exemple lorsque la permutation d'un bonbon bleu avec un vert permet de former un alignement de trois bleu, et un deuxième alignement de trois vert. On additionne donc les deux scores afin de créer un unique coup à jouer.

Après cette étape, la liste est parcourue puis renvoie le coup possédant le plus gros score. Ce dernier représentant le meilleur coup. Pour assurer cela, nous avons assigné une valeur spécifique à chaque bonbon du patron.

2.3.2.3) Création du score pour un coup

Le principe du score repose sur un procédé simple. Chaque position dans le patron possède une valeur. Cette valeur est ajoutée au score lorsque la position contient un bonbon de la même couleur que le bonbon observé à la base du patron. De plus les positions dites externes du patron sont testées et débloquées uniquement si la case interne correspondante a été reconnu comme identique.

Voici les différentes valeurs associées aux positions du pattern.

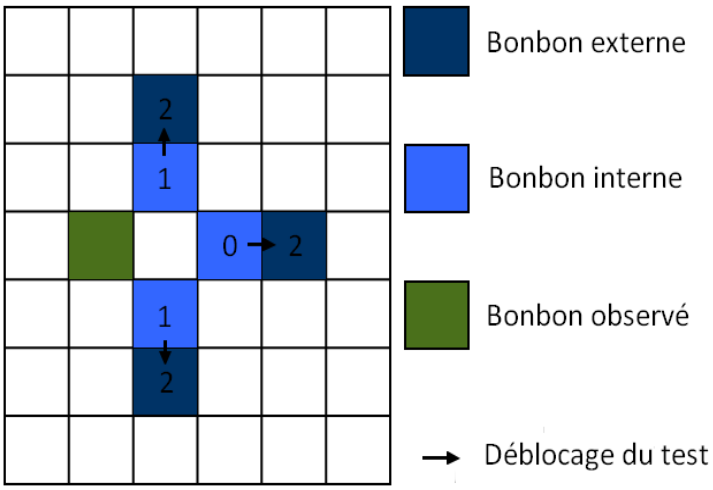


Figure n°19 : valeur associée aux positions dans un pattern

L'avantage de ce découpage est qu'il permet de reconnaître le type d'alignement en fonction de la valeur de score obtenu. Un coup est alors reconnu à partir d'un score supérieur ou égale à 2. En effet il est impossible d'obtenir un alignement en obtenant seulement un score de 0 ou 1, et à l'inverse, il est impossible d'obtenir un score supérieur ou égale à 2 sans avoir d'alignement. De plus le score augmente proportionnellement au nombre de bonbons identiques reconnus. Nous obtenons donc bien un score de plus en plus important, et correspondant au classement d'importance pour la création de bonbons spéciaux. Nous augmentons également le score d'un coup lorsque ce dernier est composé de bonbons spéciaux. Il est en effet souvent plus intéressant de jouer ces bonbons en premier.

2.3.3) Capture et traitement d'images

L'analyse ayant besoin d'une image propre, lisible et sans décor pour permettre à l'intelligence artificielle de trouver les meilleurs coups à jouer, une partie de traitement était nécessaire. Elle a été développé par chacun des membres de l'équipe. Nous observerons d'abord ce qui permet à l'application de capturer des images et nous montrerons ensuite ce que chacun a apporté à cette partie afin d'avoir un traitement opérationnelle et intelligent.

2.3.3.1) Capture d'images

Dans un premier temps, nous avons repris l'activité[8] permettant la capture de photos des membres du projet précédent qui fonctionnait pour le mieux, et nous y avons ajouté seulement une grille pour aider l'utilisateur à bien se positionner face à l'écran du pc. Cette activité fonctionne simplement et se contente de capturer une image et de l'enregistrer sur le téléphone. Les contraintes environnementales étant importantes, il a rapidement fallut y intégrer la partie traitement. En effet, malgré l'aide apporté par la grille sur la partie prise photo, les images données à l'analyse étaient toujours remplies par du décor et l'angle sous lequel était pris la photo étaient souvent mauvais et la dégradait considérablement, ce qui faussait complètement le résultat final. Nous avons pallié ce problème grâce à deux traitements différents sur le smartphone.

2.3.3.2)Traitement d'image manuel

Nous avons d'abord implémenté une activité permettant de redresser et redimensionner les images. Le redressement des images permet d'obtenir une grille de jeu toujours droite et carré, et ce, quel que soit l'angle sous lequel ont été prises les images. Cela permet à l'utilisateur de l'application de se placer n'importe où pour prendre la grille de jeu en photo. Pour réaliser ceci, nous avons utilisé le code des anciens membres du projet qui ne fonctionnait que partiellement : l'image obtenue après le traitement était toujours déformé. Nous avons crée une nouvelle activité suivant l'activité permettant la prise d'image. Celle-ci étant composée de la photo prise juste avant, de quatre points que l'utilisateur peut placer à sa guise sur l'image et d'un bouton permettant de débuter le traitement. Cf. Fig.19.

Arrivée sur l'activité, l'image, dont le chemin est récupéré de l'activité « prise photo », est affichée à l'écran sous forme de Bitmap. L'utilisateur peut alors placer les quatre points sur les coins de la grille et appuyer sur le bouton « Traiter » pour procéder au redressement. La position de chacun des points est alors sauvegardée dans un tableau et correspond aux positions sources.

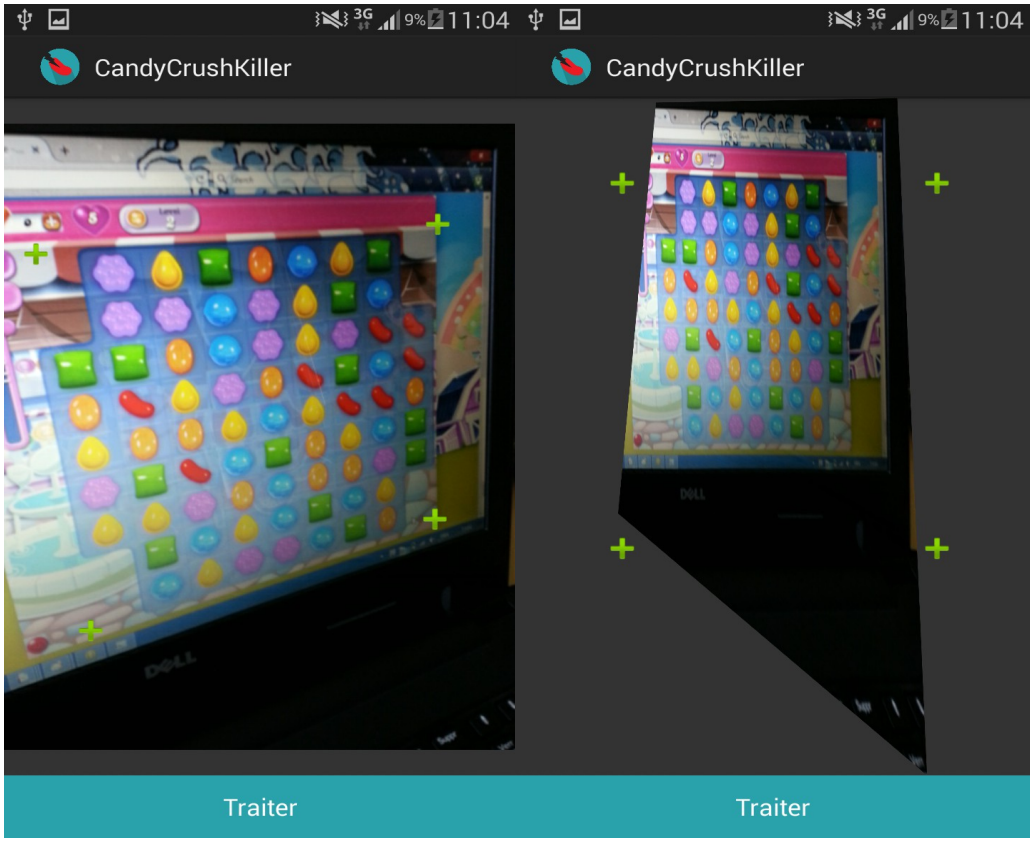


Figure n°19 : activité de redressement avant/après

Le tableau des positions finales est toujours le même et correspond à la taille de la surface utilisé pour afficher les images. On appelle ensuite une méthode présente initialement dans le SDK :

```
Matrix.setPolyToPoly(float[] src, int srcIndex, float[] dst, int dstIndex, int pointCount)
```

Figure n°20 : Code permettant le redressement d'un bitmap

Cette méthode prend en paramètre un tableau de positions sources, un premier indice pour le tableau source, un tableau de positions finales, un premier indice pour le tableau final et un nombre de points, et permet d'apposer les positions sources sur les positions finales. Une fois la transformation effectuée, on sauvegarde la nouvelle image dans un nouveau chemin, qu'on passe à l'activité permettant le redimensionnement.

Une fois redressées, les images comportent beaucoup de décors qui peuvent fausser l'analyse, il faut donc redimensionner l'image, ou plus précisément la couper pour ne garder que la grille de jeu. La vue utilisée est identique à celle du redressement avec quatre points que l'utilisateur peut déplacer aux coins de la grille et un bouton pour traiter.

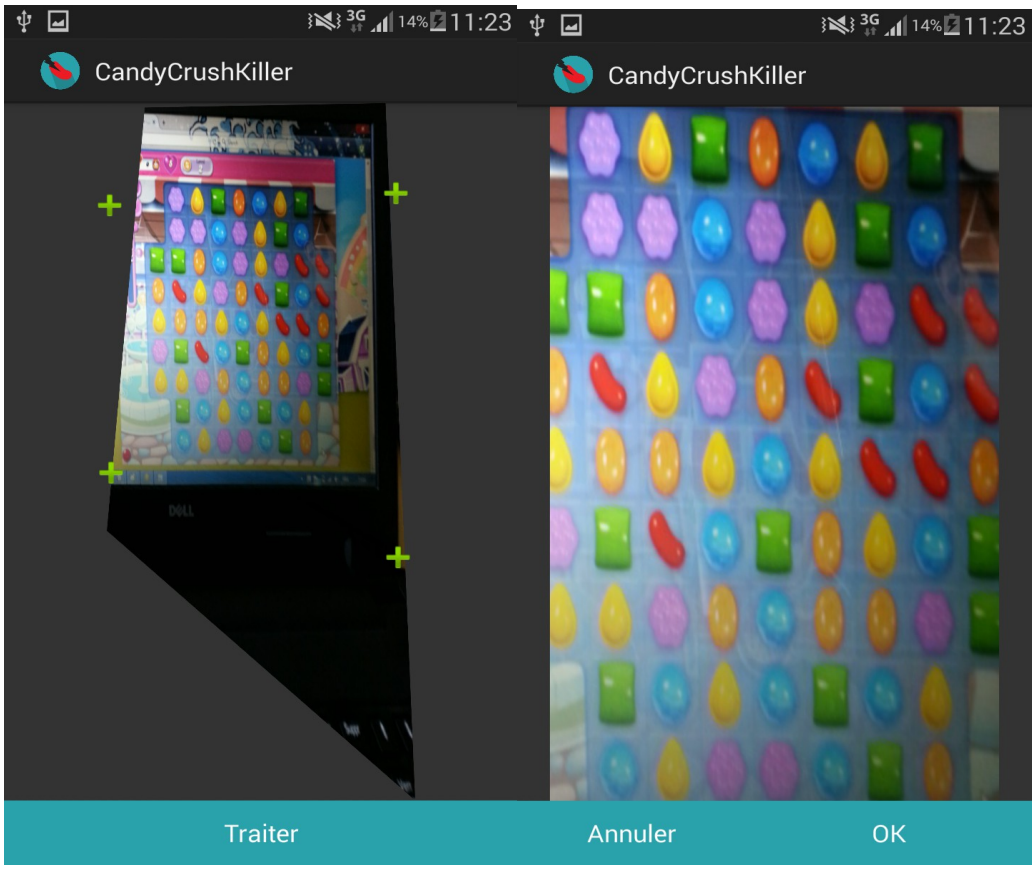


Figure n°21 :activité de redimensionnement avant/après

Le fonctionnement du redimensionnement fonctionne à peu près comme le redressement. Les positions des points sont sauvegardées dans un tableau qui correspond aux positions sources. On crée alors un nouveau Bitmap à partir du bitmap de l'image donnée. Ce bitmap a pour origine le premier point situé en haut à gauche, a pour hauteur la distance entre les points haut-gauche et bas-gauche et a pour largeur la distance entre les points haut-gauche et haut-droit. Le bitmap obtenu est ensuite enregistré dans un nouveau chemin.

Bien que le redimensionnement soit fonctionnel, il ne reste pas moins difficile à maîtriser. En effet, il persiste un problème dans la conversion des images en bitmaps qui fausse le positionnement des points sur l'aperçu de l'image et donc qui fausse le redimensionnement. Il est donc nécessaire de placer les points un peu à côté de la grille pour que cela fonctionne.

Finalement, on obtient une nouvelle image de la grille de jeu droite et sans décor avec des informations lisibles et utilisables par l'analyse. Cependant, ce traitement demande à l'utilisateur une action de sa part et la tâche est fastidieuse.

2.3.3.3) Traitement d'image automatique

La méthode précédente présentant plusieurs inconvénients : la présence et l'action de l'utilisateur, le placement des points peu précis ; nous avons décidé d'automatiser le redressement et le redimensionnement des images. Pour rendre l'application plus autonome nous avons décidé que le pc aurait un rôle à jouer et placerait quatre coins dans les angles de la grille pour que le smartphone arrive à les repérer par lui-même. L'utilisation du pc nous semblait intéressante, dans la mesure où celui-ci devait connaître la taille de la grille et la position de ces coins pour continuer le processus, il pouvait donc aider le smartphone à les trouver.

Au lancement de l'application la première image prise par le smartphone comprend quatre points représentant les coins de la grille de jeu. Ces points sont de couleurs rouge, vert, bleu et magenta et sont sur un fond totalement blanc. L'avantage du fond blanc est qu'il permet d'obtenir une photo sans reflet, ce qui pourrait dégrader l'image et les couleurs des points sont des couleurs primaires ou complémentaires ce qui nous permet d'avoir des bornes bien distinctes. Les points sont ainsi facilement repérables par le smartphone.

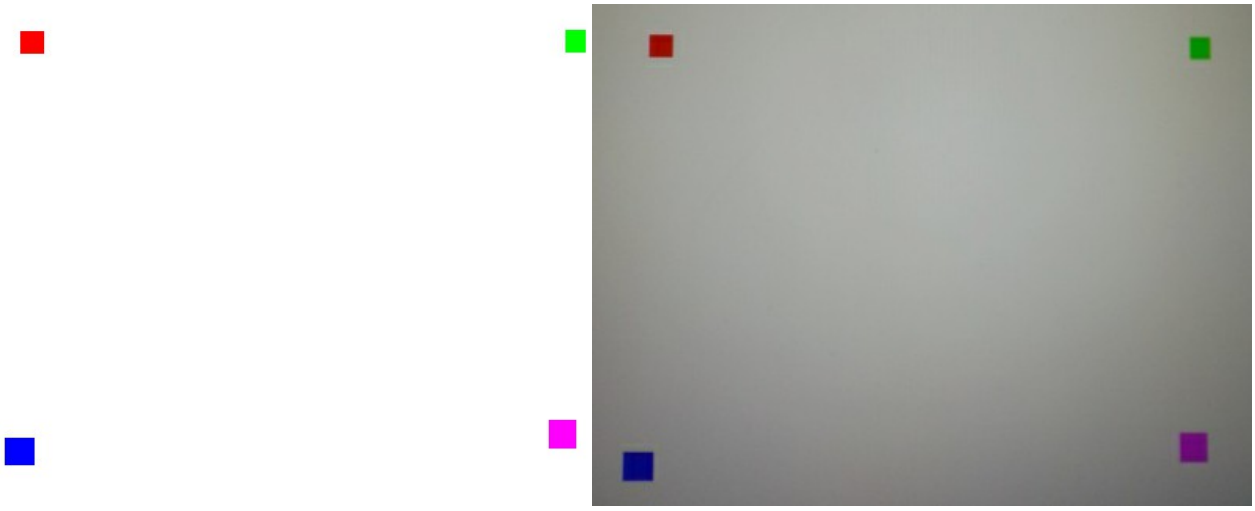


Figure n°22 : capture d'écran de l'ordinateur/photo prise par le smartphone

Une fois cette image prise, celle-ci est parcourue jusqu'à ce que les quatre coins soient trouvés. Pour cela nous avons dû créer des bornes comme pour l'analyse des bonbons(cf page 18). Celles-ci peuvent être très dégradées comme le montre le schéma ci-dessous :

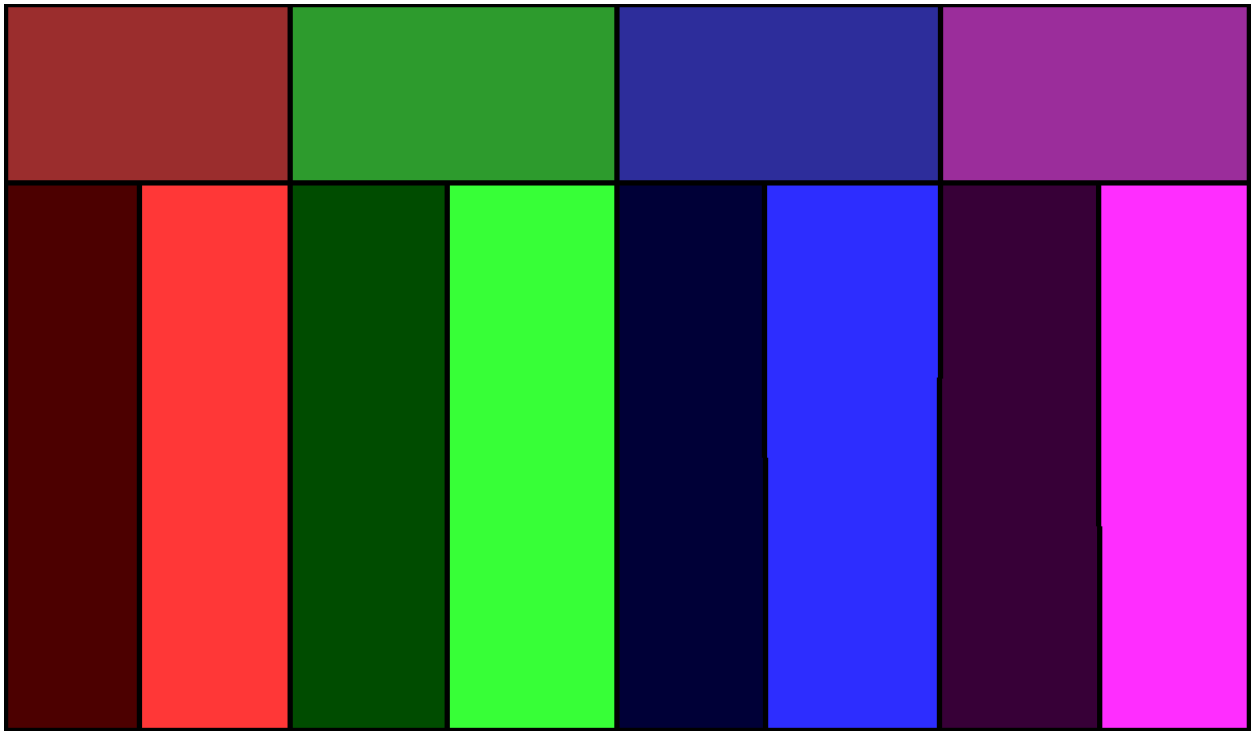


Figure n°23 : bornes de couleurs

A gauche se trouve la couleur la plus dégradée que nous acceptons, c'est à dire : 70R,0G,0B pour le rouge ; 0R,70G,0B pour le vert ; 0R,0G,50B pour le bleu ;50R,0G,50B pour le magenta ; ces valeurs correspondant aux composantes de chacune des couleurs. Le bleu et le magenta ont des valeurs plus basses que le rouge et le vert car ce sont les couleurs situées en bas de l'écran et elles ressortent plus foncées sur les photos prises par le smartphone. A droite nous avons la borne haute, c'est à dire : 255R,50G,50B pour le rouge; 50R,255G,50B pour le vert; 40R,40G,255B pour le bleu; 255R,40G,255B pour le magenta; enfin en haut se trouve une couleur moyenne c'est à dire entre la borne basse et la borne haute . Pour que le smartphone reconnaisse un point il faut donc qu'un pixel soit compris dans ces bornes.

Un premier redressement est ensuite effectué et le smartphone recherche une nouvelle fois les points sur cette nouvelle image. Cette deuxième recherche est obligatoire car le redressement à modifié l'image et les coordonnées des points sont modifiées.

Lors du premier redressement l'image devient très grande : environ 4000*3000 ; au lieu de la taille que nous fixons au début : 640*480. Pour des soucis de performances, nous avons choisi de parcourir l'image non pas pixel par pixel mais un pixel sur deux ce qui nous permet de rendre ce traitement plus rapide, le temps de traitement étant divisé par 4 et la précision perdue étant négligeable, sauf si le smartphone est trop éloigné de l'écran. En effet, il y a alors un risque de ne pas trouver les coins.

2.3.3.4) Aide au traitement automatique côté pc

En décidant d'automatiser le redimensionnement de l'image et le redressement, nous avons dû faire appel au pc puisque le smartphone avait besoin de connaître les coins de la grille. De plus, cela permettait au pc d'avoir un moyen de reconnaissance de la grille, utile dans le processus à venir, et qui enlevait la dépendance d'une action utilisateur pour définir la taille de la grille de jeu.

Pour trouver ces coins, l'application affiche une fenêtre blanche où quatre carrés de couleurs différentes sont placés à chacune des positions des coins de la grille. L'application pc affiche une page blanche en plein écran pour que le smartphone, qui doit prendre une photo, détecte bien les carrés de chaque couleur, et ne les confondent pas avec du décor.

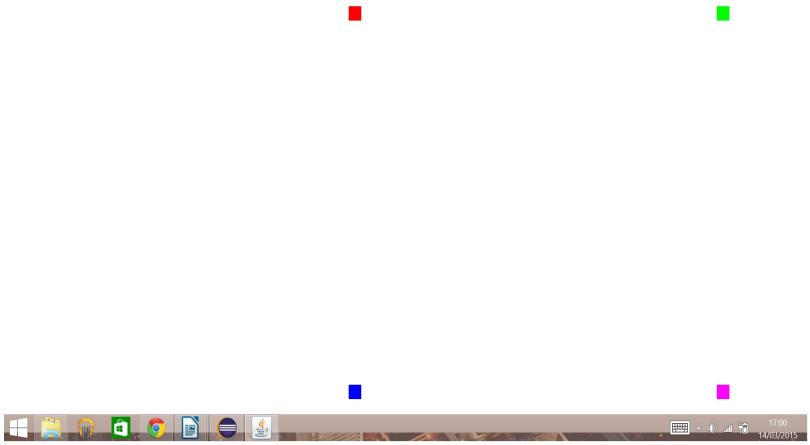


Figure n°23 : automatisation du traitement

L'application coté pc doit déterminer la position de ces carrés et pour cela, elle commence par prendre une capture d'écran et recherche la couleur du décor de fond de la grille de Candy Crush Saga. Puisque les grilles de ce jeu ne sont pas forcément carrées nous avons dû faire quatre parcours de l'image, en haut, en bas, à gauche et à droite en partant du milieu pour trouver les premiers pixels qui sont de la couleur du fond de la grille de jeu. Nous avons utilisé une plage de couleur pour détecter la couleur du décor, car le fond de la grille est translucide et les couleurs du fond peuvent changer :

Figure n°24: exemple contour de la grille

C'est à dire que pour comparer les couleurs, nous allons parcourir l'image en partant de chacun des bords et récupérer les valeurs rouge, vert et bleu de chaque pixel et le premier pixel qui aura les valeurs comprises dans la fourchette de valeurs rouge, vert, bleu prédéfinie sera considéré comme le contour de la grille. Cependant, afin d'être sûr que c'est bien la bordure de la grille et non pas un pixel parasite, nous comparons également les pixels qui l'entourent.

Figure n°25 : Mécanisme de positionnement des points

Une fois ces quatre analyses faites, on récupère quatre données qui sont les coordonnées des bords trouvés de la grille, ainsi en connaissant la positon des côtés, on peut en déduire la valeur des positions des coins de la grille. Par exemple, « xhaut », « yhaut » sont les coordonnées du côté haut et « xgauche », « ygauche » sont les coordonnées du côté gauche, en gardant le « xgauche » et le « yhaut » on obtient le coin en haut à gauche.

2.3.3.5) Atténuation des reflets et des ombres

Le téléphone étant posé sur un support tout au long de la résolution, les photos sont toutes prises dans un même environnement. Ainsi, les reflets et les ombres perçues sur l'écran sont toujours les mêmes et entraînent toujours des dégradations aux mêmes endroits. Le but de ce traitement était donc de reconnaître ces dégradations afin de les supprimer et ainsi reconstituer une image plus nette et plus propre. Pour cela le téléphone doit prendre deux photos différentes, une première photographie d'un écran noir et une seconde d'un écran blanc. D'un point de vue théorique, si la photo était parfaite, sans aucune dégradation, les couleurs obtenues seraient totalement identiques à ce qu'affiche l'écran. Autrement dit, la photo sans dégradation devrait être composée uniquement de pixels noir ayant pour valeur RGB 0,0,0 ou bien totalement composée de pixels blanc avec pour valeur RGB 255,255,255. Ainsi chaque différence obtenue sur une photo réelle correspond à l'impact de la dégradation. Si un pixel de la photo possède les composantes RGB 15,13,4 cela signifie que la dégradation au niveau de ce pixel augmente de 15,13 et 4. Pour supprimer cette dégradation correspondant à un reflet, il suffit de soustraire l'image que l'on veut traiter par l'image de l'écran noir.

Figure n°26 : Suppression d'un reflet depuis une image noir

Le traitement de l'ombre repose globalement sur le même principe. Nous prenons en photo un écran totalement blanc composé uniquement de pixels au composantes 255,255,255. Puis nous obtenons

une image d'un écran blanc avec quelques ombres représentées par des pixels grisés. Pour supprimer cette ombre de l'image il faut diviser chaque composante par la valeur obtenue sur l'image de l'écran blanc puis la multiplier par 255. Ainsi, on réajuste les couleurs de l'image à traiter en utilisant les valeurs de dégradation observées sur l'image de l'écran blanc.

En effectuant ces deux traitements sur une image cela nous permet d'obtenir une image plus nette et avec des dégradations réduites. Cependant l'utilisation d'un Android pour prendre les photos apportent à cette méthode un obstacle important rendant le traitement inutilisable. En effet bien que le téléphone ne soit pas déplacé, l'environnement est modifié automatiquement lorsque l'écran passe de l'image noire à l'image blanche. Afin de donner des photographies adaptées, la camera du téléphone modifie automatiquement la luminosité de l'image. Ainsi, lorsque l'écran noir est pris en photo, le téléphone détecte un environnement sombre, et augmente la luminosité. A l'inverse lorsque l'on prend un écran blanc, il va détecter un environnement lumineux, et va diminuer la luminosité.

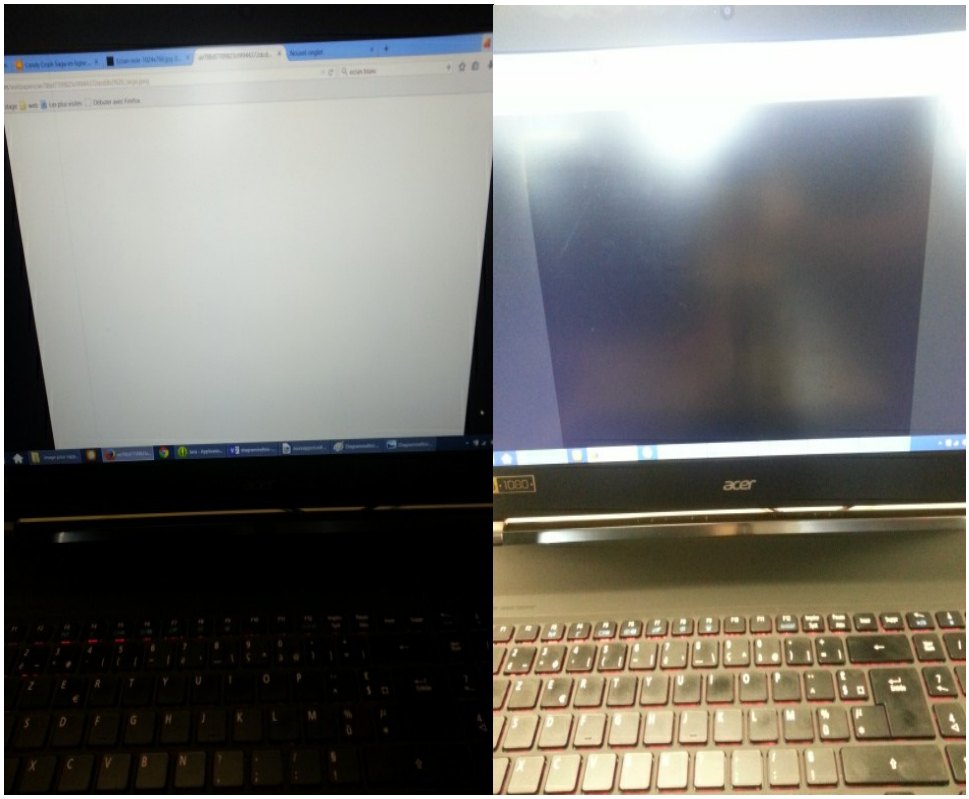


Figure n°27 : Différence de luminosité en fonction du contenu de l'écran

On obtient donc des dégradations bien trop importantes. De plus le changement d'environnement amène à des différences de dégradations entre la prise de photo de l'écran noir et la prise de photo de la grille. On effectue alors des modifications trop importantes sur des pixels étant à la base

faiblement touchés sur la photo de la grille. Le résultat du traitement donne donc au final une image plus difficile à utiliser et plus dégradée que l'original.

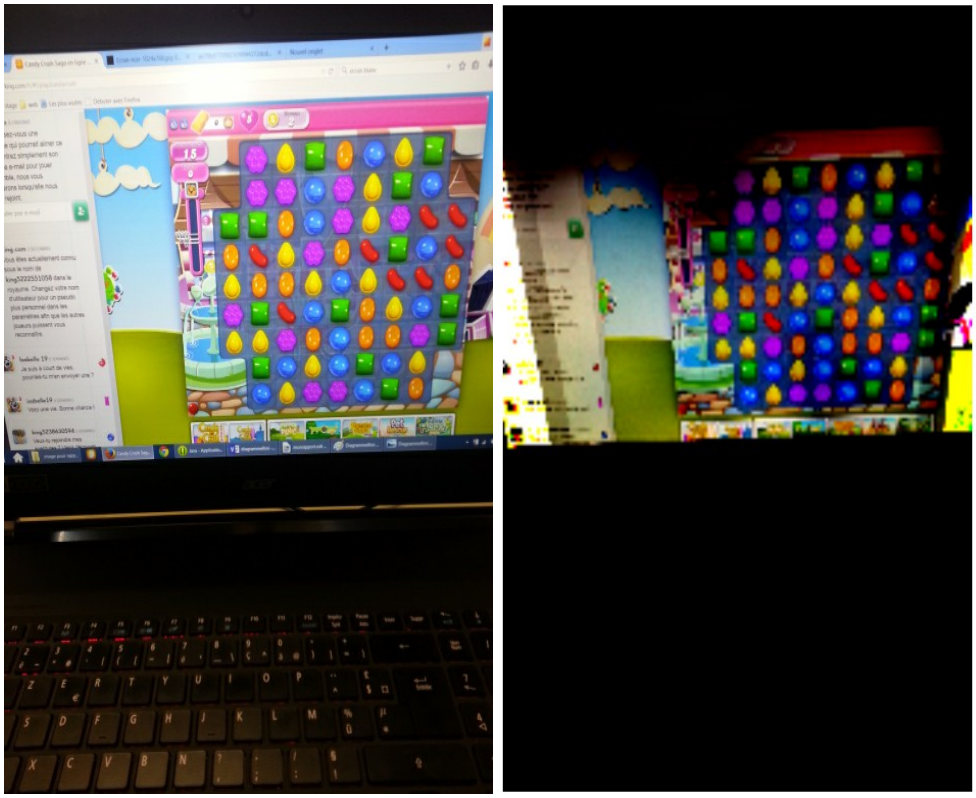


Figure n°28 : Résultat du traitement

2.3.4) Communication Smartphone/PC

Afin de compléter l'application et de permettre la résolution d'un niveau Candy Crush Saga, il a fallu mettre en place un moyen de communication entre le smartphone et le PC. Nicolas Raymond et Jérome Ralite se sont concentrés dessus. Nous avons donc entrepris une analyse des différents protocoles de communication, afin de trouver le plus adapté à notre projet.

2.3.4.1) Recherche d'un moyen de communication

La première solution retenue était d'utiliser le protocole Bluetooth HID (Human Interface Device), qui aurait fait en sorte que le smartphone soit reconnu comme un périphérique externe, ainsi il n'aurait pas été nécessaire d'avoir un serveur qui tourne sur l'ordinateur pour recevoir et traiter les données, l'avantage de cette solution était qu'aucune application ne serait lancée sur le pc.

Cependant cette solution présentait plusieurs désavantages. Premièrement celle-ci impliquait de par sa difficulté une occupation de deux personnes jusqu'a la fin du projet sans certitude de résultat. En effet cette solution utilisant le langage C, nous aurions dû utiliser le NDK[9] afin de travailler au plus proche de la plate-forme matérielle. De plus toutes les documentations trouvées lors de nos recherches utilisait cette solution. Autres contraintes, il était nécessaire d'avoir un pc ayant le Bluetooth et d'avoir un smartphone root, c'est à dire un smartphone autorisant tous les droits à son propriétaire et permettant d'obtenir le contrôle total du téléphone, pour avoir accès au matériel du bluetooth, ce qui n'était pas le cas du S3 prêté pour le projet.

Afin d'être sûr de pouvoir rendre une application fonctionnelle à la fin du temps imparti, nous avons donc décidé de communiquer par wifi en utilisant le protocole TCP/IP[10]. L'inconvénient de cette solution est qu'il est nécessaire d'avoir un programme coté PC pour recevoir et traiter le données, cela nécessite donc l'intervention de l'utilisateur coté PC, alors que nous voulions que l'application soit la plus automatisée possible.

Au cours du projet nous avons donc changé plusieurs fois la méthode de fonctionnement de la communication entre le smartphone et l'ordinateur afin d'accroitre l'automatisation.

2.3.4.2) Un premier moyen de communication

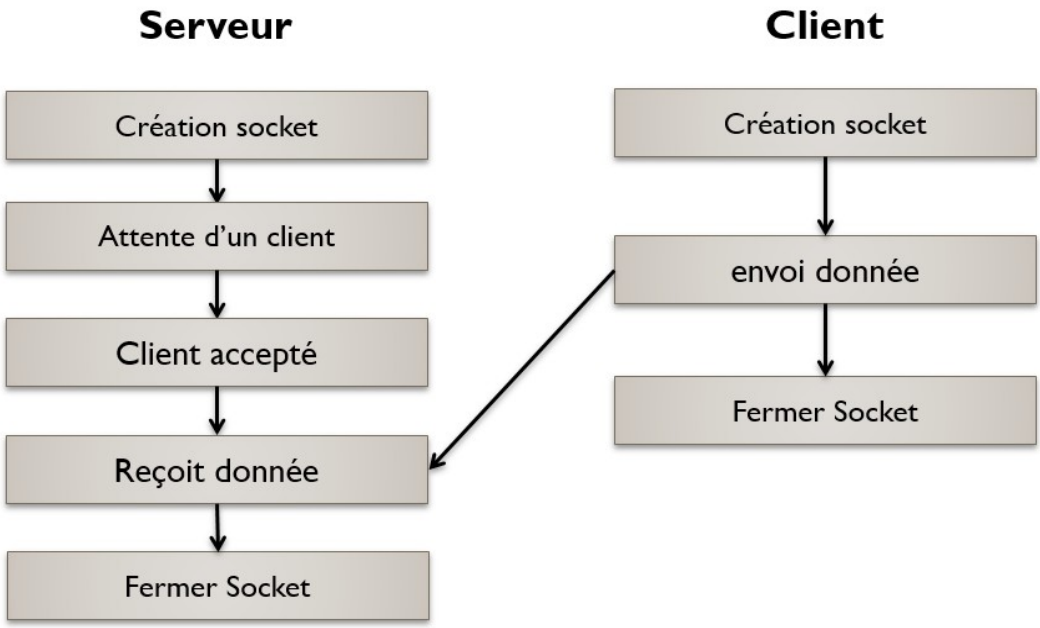


Figure n°29 : fonctionnement du TCP/IP

La première étape, consiste à lancer l'application qui tourne coté pc qui correspond au serveur sur le schéma et qui recevra les messages du smartphone, le client. Dans un premier temps, le serveur crée une socket[11] et attend qu'un client se connecte sur cette socket. Le smartphone crée une socket à son tour et pour cela une adresse IP[12] et un port sont nécessaires. L'adresse IP est celle du serveur et le port sera le port sur lequel le serveur écoute.

```
InetAddress serverAddr = InetAddress.getByName(SERVER_IP);
socket = new Socket(serverAddr, SERVERPORT);
```

Figure n°30 : création d'une socket

Une fois le client connecté, le serveur attend les messages envoyés par le smartphone puis les traite. Pour cela, l'ordinateur a besoin de posséder la position de chaque bonbon sur l'écran pour pouvoir les déplacer. L'utilisateur doit donc placer une grille transparente au niveau la grille et derrière la page de jeu, pour permettre au pc d'accéder et stocker les coordonnées de chacun des bonbons sur la grille.

Figure n°31 : grille placée par l'utilisateur

Une fois cette étape effectuée, le programme reçoit une chaîne de caractères contenant quatre numéros, par exemple: 1,2,2,2. Deux numéros forment une coordonnée sur la grille placée par l'utilisateur. Ici nous aurions les coordonnées des bonbons qui sont à la première ligne, deuxième colonne et deuxième ligne, deuxième colonne. Une fois l'acquisition de ces coordonnées faite, on se sert de la méthode « bouger » qui effectue un calcul sur les numéros des coordonnées pour obtenir deux indices. La méthode obtient, grâce à ces indices, la position où le curseur de souris doit se rendre. Pour déplacer la souris, on utilise la classe robot de java, qui permet de simuler les actions

d'une souris. La méthode peut ainsi déplacer le curseur et simuler un clic de souris et peut donc permuter les deux bonbons et continuer le jeu.

```
public void bouger(int caseX1, int caseY1, int caseX2, int caseY2) {
    Coordonee c1, c2;
    Robot robot = null;
    try {
        robot = new Robot();
    } catch (AWTException e) {
        e.printStackTrace();
    }


    //renvoie la position qui correspond aux coordonnees passés en argument
    c1=tabCoordonnee[((caseX1-1)*9+ caseY1)-1];
    c2=tabCoordonnee[((caseX2-1)*9+ caseY2)-1];

    //déplace la souris à la position (c1.X,c2.Y)
    robot.mouseMove(c1.getX(),c1.getY());
    //effectue un clic
    robot.mousePress(InputEvent.BUTTON1_MASK);
    robot.mouseRelease(InputEvent.BUTTON1_MASK);

    try {
        Thread.sleep(1000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    //déplace la souris à la position (c2.X,c1.Y)
    robot.mouseMove(c2.getX(),c2.getY());
    //effectue un clic
    robot.mousePress(InputEvent.BUTTON1_MASK);
    robot.mouseRelease(InputEvent.BUTTON1_MASK);
}
```

Figure n°32 : corps de la méthode permettant la permutation de deux bonbons

Cette solution était efficace mais présentait cependant quelques inconvénients. En effet, cela fonctionnait uniquement sur des grilles de neuf bonbons par neuf et l'utilisateur était obligé d'intervenir pour placer la grille sur le jeu puis pour la placer en arrière-plan, alors que nous souhaitions avoir l'application la plus automatisée possible.

2.3.4.3) Un second moyen de communication

Nous avons donc développé une seconde solution où l'ordinateur prend une capture d'écran et l'analyse afin de trouver les contours de la grille, pour ne plus à avoir à placer la grille manuellement comme nous l'avons vu précédemment (cf page 37). Une fois les coins de la grille trouvés, nous nous en servons pour trouver les positions de chaque bonbon sur l'écran afin de pouvoir les bouger par la suite.

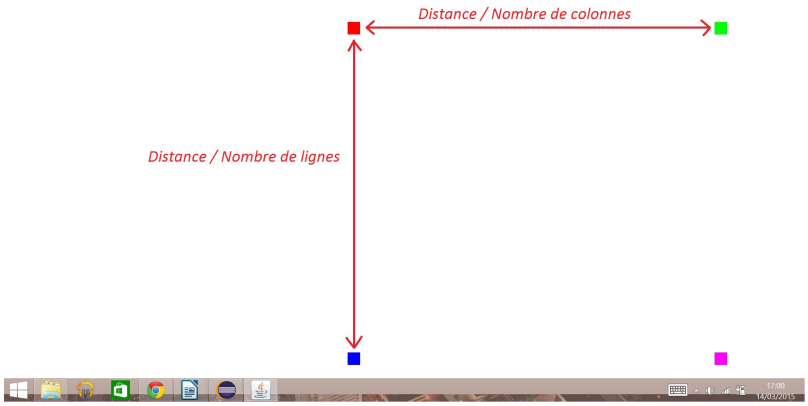


Figure n°33 : grille placé automatiquement

Nous utilisons les coordonnées des coins, et nous divisons la distance comprise entre le coin de gauche et le coin de droite puis la distance comprise entre le coin du haut et celui du bas par un nombre de colonnes et un nombre de ligne qui sera envoyer par l'utilisateur depuis le smartphone. Avec cette opération, nous pouvons déduire le nombre de cases entre chaque point qui correspondra à une position de bonbon. Ces positions sont toujours stockées dans un tableau de coordonnées, puis traitées de la même façon par la classe robot de java.

Cependant il y a toujours quelques inconvénients avec cette solution, par exemple l'utilisateur doit encore intervenir pour fermer la fenêtre de redimensionnement une fois que ce dernier est terminé coté smartphone. On aurait pu fermer la fenêtre en envoyant depuis le smartphone une chaîne de caractères mais cela imposait un serveur plus complexe qui aurait dû gérer les différentes données pour enchaîner les étapes alors que notre but était de faire le minimum d'actions du côté du pc pour respecter au mieux les objectifs.

2.3.4.4) La touche finale : l'ajout du RMI

Au cour du projet, nous avons découvert la méthode RMI[13] (Remote Method Invocation), qui permet l'appel d'une méthode exécutée dans une machine différente de celle de l'objet l'appelant. C'est à dire que l'on peut appeler les méthodes du serveur coté PC depuis le smartphone comme si l'objet serveur était sur celui-ci, ce qui a pu nous aider pour l'automatisation de l'application. Par conséquent, la seule action utilisateur requise sur le pc est le lancement du serveur.

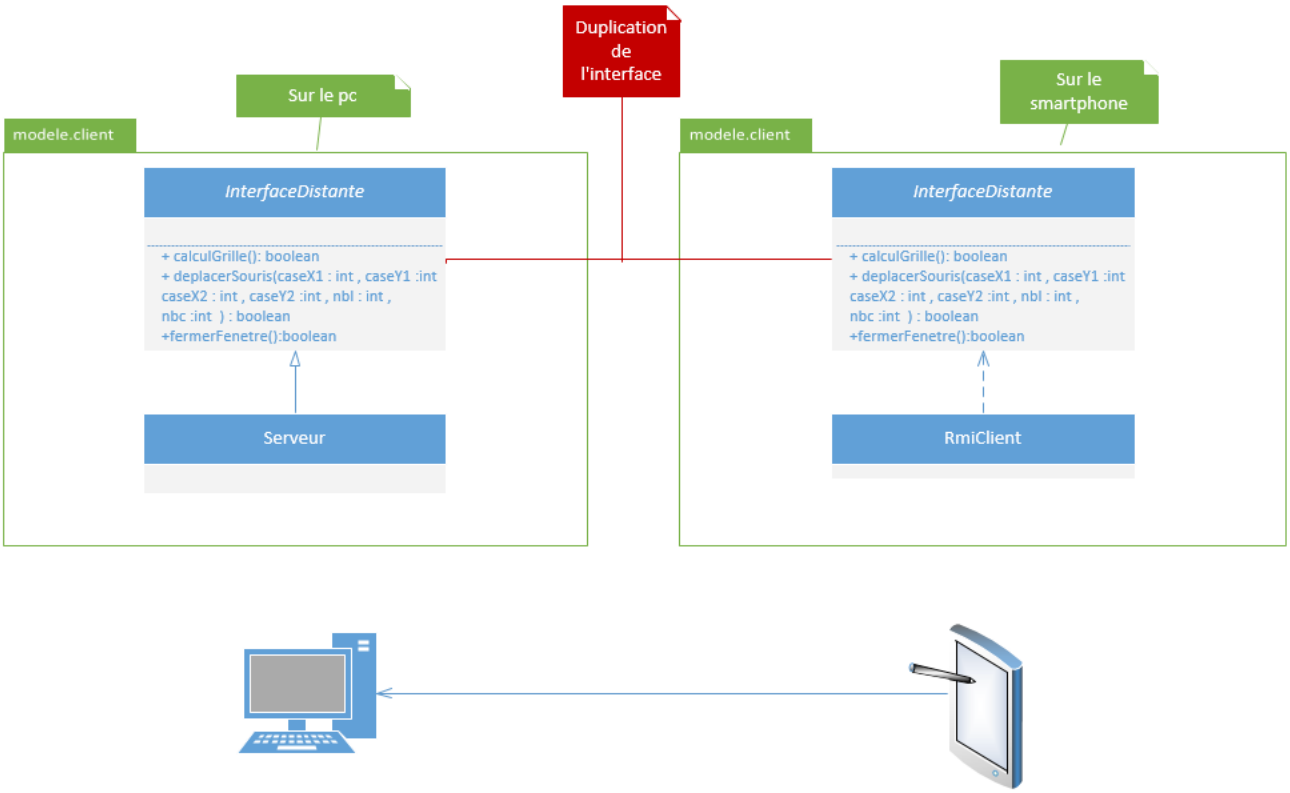


Figure n°34 : fonctionnement de RMI

Les fonctions que le smartphone va exécuter sont:

1 -calculGrille : cette méthode permet de détecter des coins de la grille et afficher les quatre coins permettant le redimensionnement , cette méthode est appelée par le smartphone lorsque la configuration de l'IP ainsi que le nombre de lignes et de colonnes de la grille sont établis .

2 – fermerFenetre : cette méthode permet de fermer l'affichage des coins, cette fonction est appelée coté smartphone une fois que les quatre coins sont trouvés et que le redressement et le redimensionnement sont effectués avec succès.

3- deplacerSouris : cette méthode permet de transmettre au serveur les coordonnées du déplacement ainsi que les nombres de lignes et de colonnes de la grille de jeu. Elle déplace les bonbons à l'aide de la souris et elle est appelée au cours de la résolution une fois que l'IA a trouvé un coup à jouer .

```
public void calculGrille(){
            if( interfaceDistante == null)
                  return ;
            interfaceDistante.CalculGrille();
            try {
                  client.close();
            } catch (IOException e) {
                  e.printStackTrace();
            }
}
```

Figure n°35 : une des méthodes du smartphone appelant les méthodes sur PC

Sur l'exemple ci-dessus, on montre l'appelle de la méthode calculGrille sur le smartphone. Nous pouvons voir que l'appel se fait via l'interface distante comme si celle-ci était instanciée sur le smartphone. L'avantage de cette solution est une mise en œuvre facile et permet l'abstraction des sockets cependant le port et l'adresse IP du serveur sont toujours nécessaires.

Nous avons rencontré un problème pour mettre en œuvre la méthode RMI, la JVM(machine virtuelle java) d'Android étant plus réduite que celle proposée sur un pc, les classes permettant la mise en œuvre de RMI ne sont pas implémentées et il a donc fallu utiliser une librairie externe (lipermi) permettant l'utilisation de RMI.

2.3.4.5) Rendu final :

L'utilisateur commence par lancer l'application, cette dernière attend que le smartphone se connecte. Une fois cette opération effectué, le smartphone appelle la méthode qui affiche la fenêtre de traitement composé des quatre coins de la grille.

Figure n°36 : lancement du serveur

Une fois que le smartphone a fini le redimensionnement et le redressement, il appelle la méthode pour fermer la fenêtre. Le smartphone appelle ensuite la méthode permettant de trouver les positions des bonbons, avec en paramètre, le nombre de lignes et de colonnes saisis par l'utilisateur lors du lancement de l'application mobile. Ensuite à chaque fois que l'intelligence artificielle trouve un coup à jouer, le smartphone appelle la fonction pour déplacer la souris, avec les valeurs retournées par l'IA en paramètre.

2.3.5) Application smartphone : organisation de la résolution

L'application smartphone permet de lier les différentes parties du projet vues précédemment. Celle-ci a été conçue par Alexandre Boyer, elle a pour but d'être un maximum réutilisable et a pour finalité la résolution complète d'un niveau de « Candy Crush Saga ».

2.3.5.1) Résolution des niveaux

L'application est composée de cinq activités permettant d'arriver au but final. Tout d'abord, l'application démarre sur une activité de paramétrage qui nous propose entre autres de choisir les nombres de lignes et de colonnes de la grille et d'entrer l'adresse ip du pc.

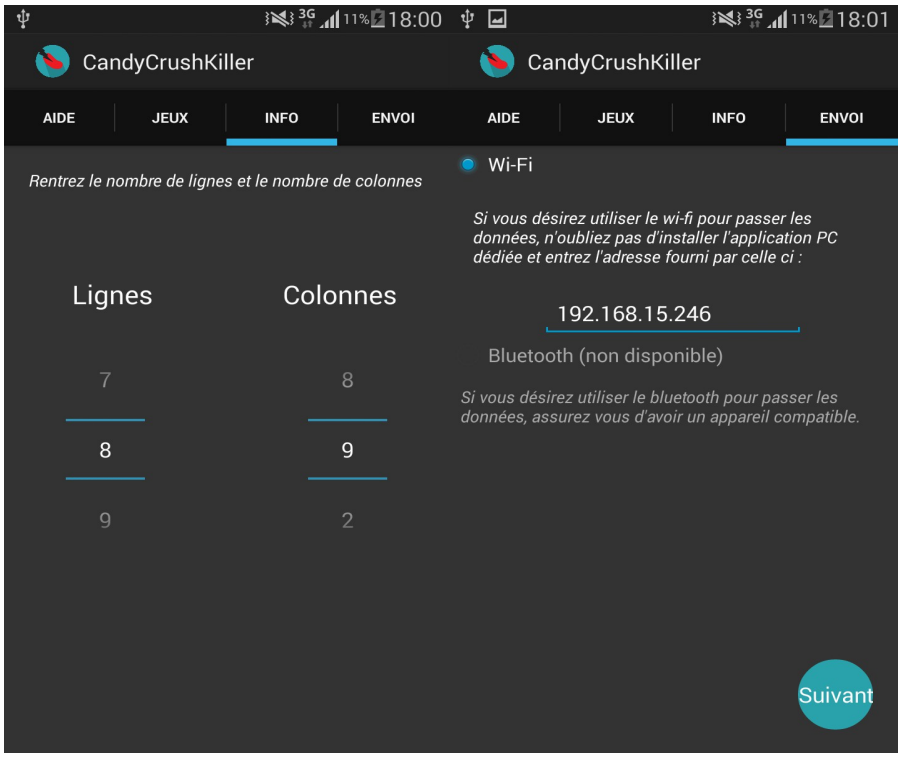


Figure n°37 : activité de paramétrage

Les nombres de lignes et de colonnes de la grille sont utiles à l'analyse de l'image car l'application n'est pas capable de déterminer seule ces informations et celles ci sont nécessaires dans le processus de résolution, mais aussi pour le pc. Elles sont donc envoyées au pc pour que celui ci puisse découper la grille afin de pouvoir positionner la souris sur les bonbons de façon précise et ainsi éviter des erreurs. Ces informations sont transmises grâce à une adresse ip que l'utilisateur entre dans l'application. Celle-ci permet donc l'envoi d'informations au pc telles que les nombres de colonnes et de grilles du niveau, les différents coups à jouer et permet aussi l'avancée du serveur, puisque le smartphone contrôle l'application pc. Tout a été conçu afin que l'utilisateur n'ait pas à toucher à l'application coté pc.

Une fois l'image capturée, celle-ci est traitée soit de façon automatique grâce aux points placés sur l'écran du pc, soit de façon manuelle grâce à des points disposés sur l'écran du smartphone à placer soi-même. Quel que soit le choix de l'utilisateur, les positions de ces points sur l'image prise en photo sont conservées dans une liste de points dite statique, ce qui veut dire qu'elle est facilement accessible. Ces positions peuvent donc être réutilisées plus tard, ce qui signifie que l'utilisateur n'a besoin de les rentrer qu'une seule fois pour que l'application tourne. A partir de là, toutes les informations nécessaires à la résolution d'un niveau sont connues par l'application et elle peut donc lancer son algorithme de résolution.

L'algorithme est placé dans la dernière activité de l'application et exécute chacune des parties du projet en boucle. L'arrêt de la boucle est causé par la pression d'un bouton, n'ayant pas eu le temps de développer un moyen plus avancé pour stopper l'algorithme à la fin du niveau. Il commence par prendre une première photo de la grille de jeu qu'il traite en redressant et en redimensionnant l'image et ce, à l'aide des positions des points enregistrées précédemment. La nouvelle image est analysée et la grille numérique est donnée à l'intelligence artificielle qui détermine la meilleure combinaison à jouer. Ce coup est transmis au pc qui peut alors déplacer la souris et faire évoluer la grille de jeu. Le processus est répété en boucle jusqu'à l'arrêt par l'utilisateur ou bien par un problème technique interne à l'application.

```
while(estEnCours) {
    //prend une photo de la grille
    gestion.prendrePhoto("/TMPphoto.jpg" );
    path=gestion.getPath();

    //traite l'image prise
    path = redressement.traiter(path, width, height);
    path = redimensionnement.traiter(path, width, height);

    //analyse la nouvelle image
    Bitmap grille = ParametreBitmap.convertBitmap(path,0,0);
    AnalyseCandyCrush a = new AnalyseCandyCrush(grille,RmiClient.nbColonnes,RmiClient.nbLigne);
    Bonbon [][] matrice = a.getMatriceBonbon ();

    //détermine le meilleur coup grâce à l'image
    IaCandyCrush superIa = new IaCandyCrush(matrice,RmiClient.nbColonnes,RmiClient.nbLigne);
    Move mouvement = superIa.rechercheMove();

    //envoie le coup à jouer au pc
    Intent serv = new Intent(getBaseContext(), DeplacerSouris.class);
    serv.putExtra("x1", mouvement.getX1());
    serv.putExtra("x2", mouvement.getX2());
    serv.putExtra("y1", mouvement.getY1());
    serv.putExtra("y2", mouvement.getY2());
    startService(serv);
}
```

Figure n°38 : code permettant la résolution d'un niveau

Une fois l'arrêt de l'application enclenché, l'application ramène l'utilisateur à l'activité de paramétrage pour lui permettre de résoudre le niveau suivant. L'adresse IP du pc est enregistrée dans l'application et n'a plus à être entré pour une utilisation sur un même ordinateur.

2.3.5.2) Extensibilité de l'application

Un des objectifs initiaux du projet était de pouvoir faire fonctionner l'application smartphone avec d'autres jeux informatiques chronophages simples. Nous avons donc, dans les derniers moments du projet, établit une conception permettant une implémentation facile de résolution de nouveaux jeux. Nous avons donc ajouté une liste à l'activité de paramétrage permettant à l'utilisateur de choisir le jeu à résoudre. Cf. fig.39. Celle-ci est, pour le moment, composée de « Candy Crush Saga » mais pourra être complété facilement.

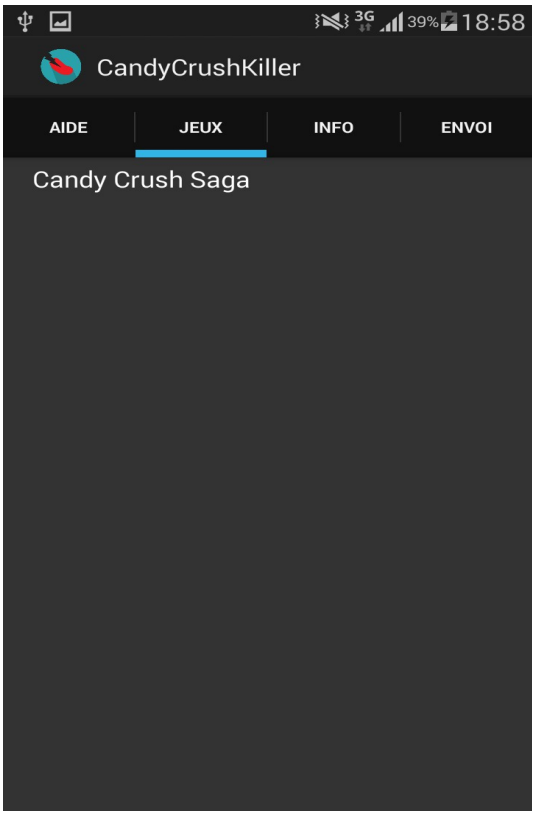


Figure n°39 : liste de jeux présente dans l'activité de paramétrage

Chaque item présent dans cette liste est une instance de la classe « Jeu ». Cette classe est composée d'un nom, d'une analyse et d'une intelligence artificielle. En effet, l'analyse et l'intelligence artificielle sont les parties qui diffèrent le plus entre chaque jeu. Les classes « Analyse » et « IA » sont abstraites, ce qui signifie que ce sont des classes générales, non instanciables et chaque jeu a donc une analyse et une intelligence propres. On est ici en présence d'un patron stratégie.

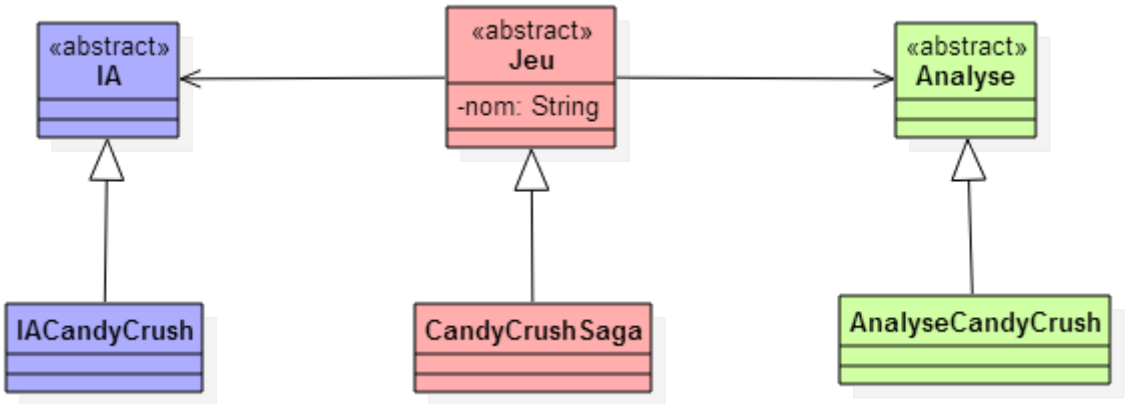


Figure n°40 : patron stratégie pour implémenter de nouveaux jeux facilement

Le jeu choisi par l'utilisateur est enregistré pour être réutilisé dans l'algorithme de résolution. Manquant de temps, nous n'avons pas pu aller au bout de cette idée, mais l'analyse et l'intelligence artificielle utilisés dans l'algorithme auraient donc été propres au jeu joué et ainsi il aurait été possible d'ajouter un autre jeu à la liste simplement en créant une nouvelle classe dérivant de « Jeu » et deux classes d'analyse et d'intelligence artificielle propres au nouveau jeu.

3) Bilan Technique

Ces quatre mois de travail sur le projet nous ont permis de réaliser une application smartphone fonctionnelle capable, théoriquement, de résoudre un niveau simple de « Candy Crush Saga » et ce quelle que soit la taille de la grille de jeu. Deux modes de traitement d'images, manuel et automatique, permettent à l'utilisateur d'avoir une certaine liberté sur la façon dont les photos sont prises et traitées, c'est à dire que l'application gère au mieux les contraintes environnementales. De plus, celle-ci est relativement extensible et l'ajout de nouveaux jeux pourrait se faire aisément par d'autres programmeurs.

Cependant, bien que de nombreux objectifs soient remplis, il reste encore quelques problèmes. En effet, en pratique, l'application n'est pas encore complètement opérationnelle concernant l'analyse d'image puisqu'elle a du mal à faire la différence entre des bonbons jaunes et oranges, ce qui fausse de façon non négligeable la résolution des niveaux. Bien que certaines contraintes environnementales soient traitées par l'application, il en reste néanmoins à gérer pour avoir une application infaillible car les reflets d'écrans peuvent dégrader considérablement les images prises par le smartphone. De plus, l'extensibilité de l'application, bien qu'avancée, n'a pas pu être totalement terminée. Enfin, un des objectifs principaux du projet n'a pas été abordé par manque de temps et de moyens techniques. Il s'agit de la communication du smartphone au pc via bluetooth sans utilisation de serveur côté pc. Nous avons effectué de longues recherches sur le sujet avant d'abandonner l'idée et nous sommes concentrés sur l'automatisation de l'application. Nous avons fait en sorte d'avoir un minimum de code sur le pc pour respecter au mieux une des principales idées du projet et minimiser le nombre de contacts avec l'utilisateur, la plupart des informations étant sur le smartphone.

Il reste donc encore quelques points à améliorer, notamment la calibration dynamique des bornes de couleurs pour que l'application fonctionne parfaitement, puis l'implémentation d'une communication bluetooth, objectif que nous n'avons pas pu remplir. Continuer sur l'automatisation de l'application, comme réaliser un arrêt du processus de résolution à la fin du niveau, et terminer l'extensibilité de l'application pourrait être intéressant et permettrait au projet de tendre vers l'infaillibilité.

Conclusion

Ce projet nous a apporté de multiples enseignements tant sur le plan technique que sur l'expérience d'un travail en équipe sur une longue durée.

Tout d'abord, sachant que le projet devait se réaliser principalement sur smartphone, nous nous sommes initiés au développement sur Android en travaillant notamment sur l'IHM de l'application. Nous avons du apprendre à gérer les changements entres les différentes activités et contrôler la caméra du smartphone qui était un des gros points difficiles du projet. Cela nous à permis de découvrir quelques spécificités d'Android. Nous avons acquis des connaissances concernant l'analyse et le traitement d'images qui représentaient des parties importantes du projet. En effet, nous avons appris à reconnaître des couleurs sur le smartphone, et à calibrer des images bien que cette dernière partie ne fonctionne pas encore à l'heure actuelle. Nous avons aussi appris à redresser et redimensionner des bitmaps. L'implémentation d'une intelligence artificielle nous a permis de également d'aborder ce domaine. Notre travail sur la communication entre le smartphone et le pc nous a permis de développer nos connaissances en réseau, puisque nous avons effectué de nombreuses recherches sur différents protocoles comme TCP/IP et Bluetooth.

De plus, au delà de la programmation Android, ce projet à été le travail le plus long et le plus complexe de nos deux années de DUT. Pour le gérer nous avons appris à travailler en équipe afin de nous répartir les tâches de façon équilibrées et en fonction des compétences et des goûts de chacun. Nous avons mis en application nos cours de gestion de projet étudiés au cours de nos années à l'IUT afin de gérer au mieux notre temps de travail. Nous nous sommes fixés des objectifs afin de prévoir et planifier notre travail tout au long du projet pour le finir à temps et nous avons gagné en autonomie.

Durant ces quatre mois, nous nous sommes impliqués de façon conséquente dans ce projet, nous avons communiqué de façon accrue entres nous et cela fut et restera une très bonne expérience, tant au niveau humain que technique.

English summary

This projet is part of our two-year degree in Computer Science at the Clermont-Ferrand University Institute of Technology. We worked on a four-month practical project in a group of five on "Candy Crush Killer". The final goal of this project was to develop an Android application which automaticaly solve a time-consuming game like "Candy Crush Saga".

We did not start from scratch as the application had already been developed by students the previous year. For this project, the task assigned to us was to create an application which able us to take photos, then modify and analyze them, find the best move to make and move the mouse of the computer to solve the grid step by step.

In the first step the main objective was to analysis the goal of the project with our tutors. Next, we spend a long time analysing the code which the previous team had done last year in order to understand it and to find hard points of the program.

Firstly, our tutors M. Kauffmann and M. Laffont went through the project description with us so that we could begin the analysis stage. Then, we spend a long time analysing the code which the previous team had done last year in order to understand it and to find hard points of the program. For the next stage, we divided the work among us so as to be more efficient.

In conclusion, the application we designed is efficient but there are still minimal problems. Lots of main objectives are done and if we had had more time, we would have created a foolproof application. Overall, the project was a good learning experience, it enabled us to develop our interpersonnal skills and we gained in programming skills especially in Android.

Lexique

Android[1]: Android, est un système d'exploitation mobile pour smartphones, tablettes tactiles, PDA, smartwatches et terminaux mobiles. C'est un système open source utilisant le noyau Linux. Il a été lancé par unes tartup du même nom rachetée par Google4 en 2005.

Samsung Galaxy[2]: Le Samsung Galaxy est un smartphone haut de gamme de Samsung

Diagramme de Gantt[3]: Le diagramme de Gantt est un outil utilisé en ordonnancement et en gestion de projetet permettant de visualiser dans le temps les diverses tâches composant un projet. Il s'agit d'une représentation d'un graphe connexe, valué et orienté, qui permet de représenter graphiquement l'avancement du projet.

Wi-Fi[4]: c'est un ensemble de protocoles de communication sans fil . Un réseau Wi-Fi permet de relier par ondes radio plusieurs appareils informatiques (ordinateur, routeur, smartphone, décodeur Internet, etc.) au sein d'un réseau informatique afin de permettre la transmission de données entre eux.

Pixel[5]: Le pixel est l'unité minimale adressable par le contrôleur vidéo. C'est aussi l'unité utilisée pour spécifier les définitions d'affichage (largeur × hauteur). À chaque pixel est associée une couleur, usuellement décomposée en trois composantes primaires (Rouge vert bleu).

Bitmap[6]: c'est une image matricielle, (de l'anglais « bitmap »), une image constituée d'une matrice de points colorés. C'est-à-dire, constituée d'un tableau, d'une grille, où chaque case possède une couleur qui lui est propre et est considérée comme un point. Il s'agit donc d'une juxtaposition de points de couleurs formant, dans leur ensemble, une image

IA[7]: L'intelligence artificielle est l'intelligence des machines et des logiciels. C'est une discipline scientifique qui recherche des méthodes de création ou simulation de l'intelligence.

Activité[8]: Une activité est une interface graphique sur android, avec laquelle l'utilisateur va intérargir.

NDK[9]: (Native Developpement Kit) est un outil complémentaire du SDK qui permet de construire des portions critiques du code afin de mieux gérer les performances de votre application. Il utilise le JNI (Java Native Interface) pour interfacer du code écrit en C/C++ avec du code écrit en Java, il fournit des entêtes natives et des bibliothèques qui vous permettent de travailler de plus en plus proches de la plateforme matériel .

TCP/IP[10]: C'est l'ensemble des protocoles utilisés pour le transfert des données sur Internet.

Socket[11]: Les sockets servent à communiquer entre deux hôtes appelés Client / Serveur à l'aide d'une adresse IP et d'un port que j'appelle prise ; ces sockets permettront de gérer des flux entrant et sortant afin d'assurer une communication entre les deux (le client et le serveur) de manière fiable à l'aide du protocole TCP/IP.

Adresse IP[12]: C'est un numéro d'identification qui est attribué de façon permanente ou provisoire à chaque appareil connecté à unréseau informatique utilisant l'Internet Protocol. L'adresse IP est à la base du système d'acheminement (le routage) des messages sur Internet.

RMI[13]: C'est une API[14] Java permettant de manipuler des objets distants (c'est-à-dire un objet instancié sur une autre machine virtuelle, ou sur une autre machine du réseau) de manière transparente pour l'utilisateur, c'est-à-dire de la même façon que si l'objet était sur la machine virtuelle (JVM) de la machine locale.

API[14]: interface de programmation (Application Programming Interface) est un ensemble normalisé de classes et de méthodes qui sert de façade par laquelle un logiciel offre des services à d'autres logiciels.

Bibliographie

Documentation Android officielle :
http://developer.android.com/guide/index.html

Forum et autres sites utiles de développement Android :
http://android.developpez.com/
http://stackoverflow.com/
https://github.com/

Tutoriel pour créer une listView :
https://www.youtube.com/watch?v=eAPFgC9URqc

Tutoriel pour créer un fragment :
https://www.youtube.com/watch?v=w4Qyhh9KKlI

Tutoriel pour créer un dialog :
https://www.youtube.com/watch?v=Oz8KG4wwBzA

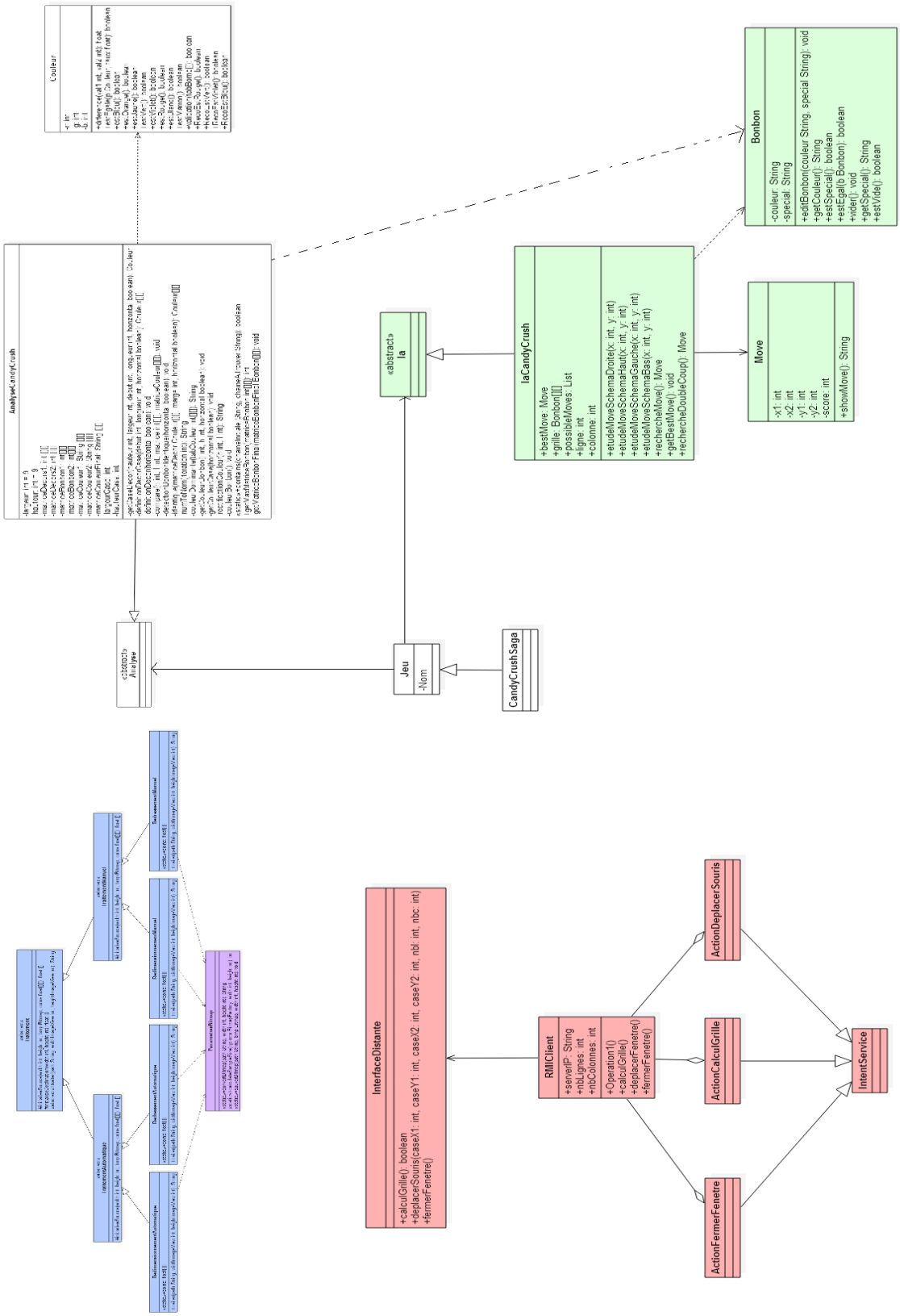
Tutoriel pour faire marcher la caméra Android sans preview :
http://www.41post.com/3794/programming/android-take-a-picture-without-displaying-a-preview

RMI :
http://lipermi.sourceforge.net/
http://stackoverflow.com/questions/13788738/using-java-rmi-in-android-application

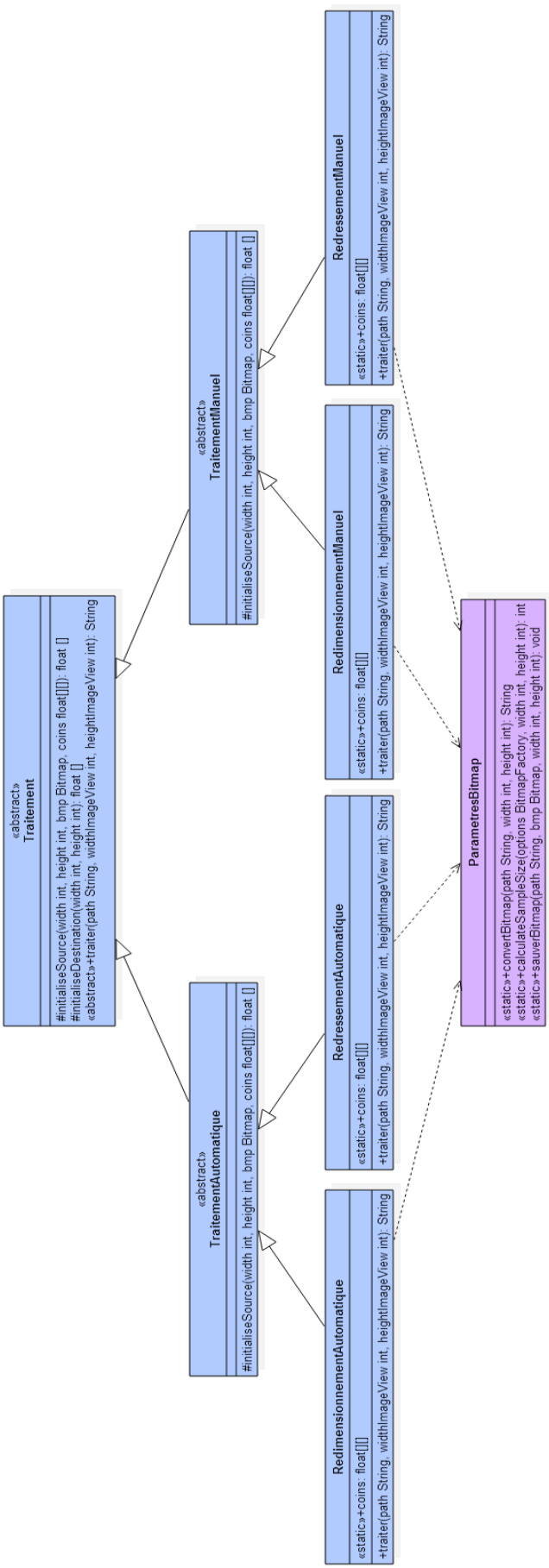
Recherche sur le Bluetooth :
https://code.google.com/p/androhid/
https://github.com/Arseny-N/Android2Mouse
https://www.mistra.fr/utiliser-ndk-dans-un-projet-android.html
https://github.com/RonsDev/BlueCtrl

Documentation Java :
http://openclassrooms.com/

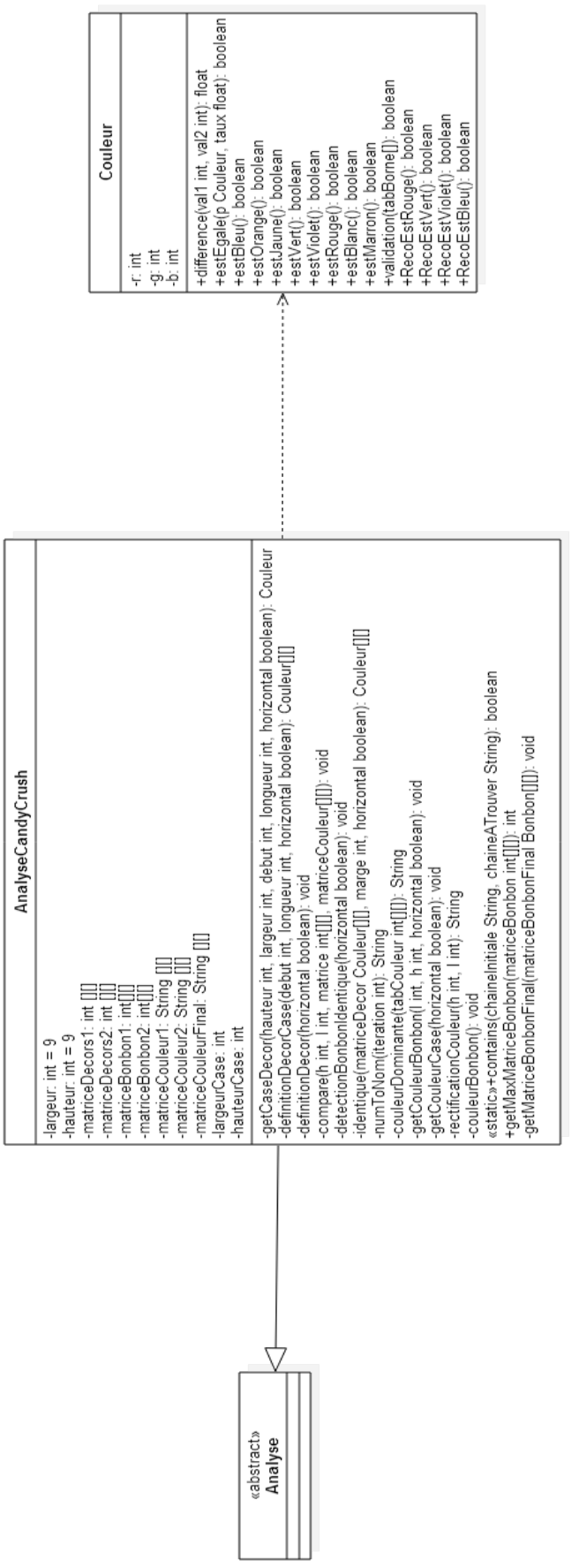
Annexes



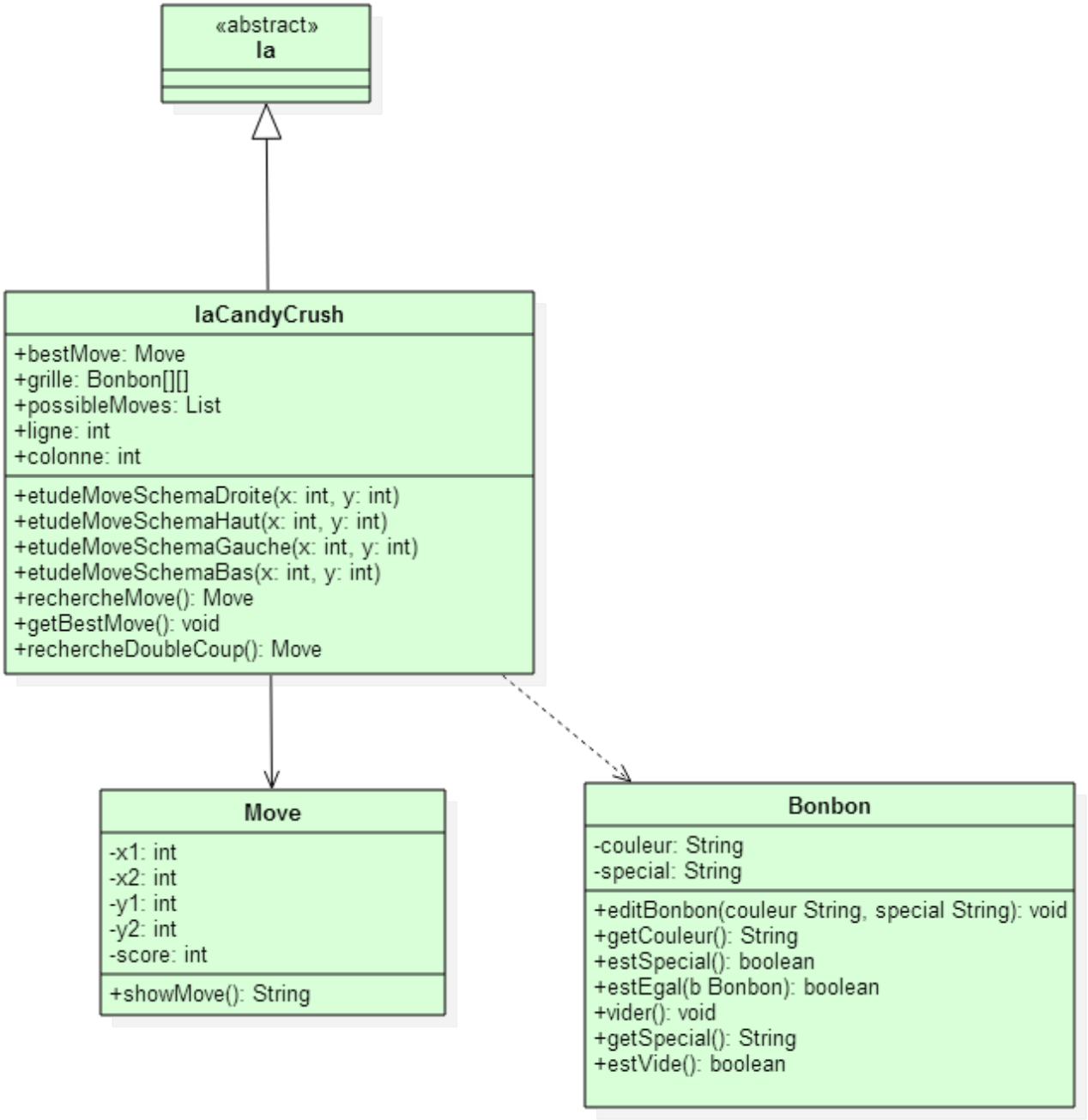
Annexe n°1 : diagramme de classes du modèle côté smartphone



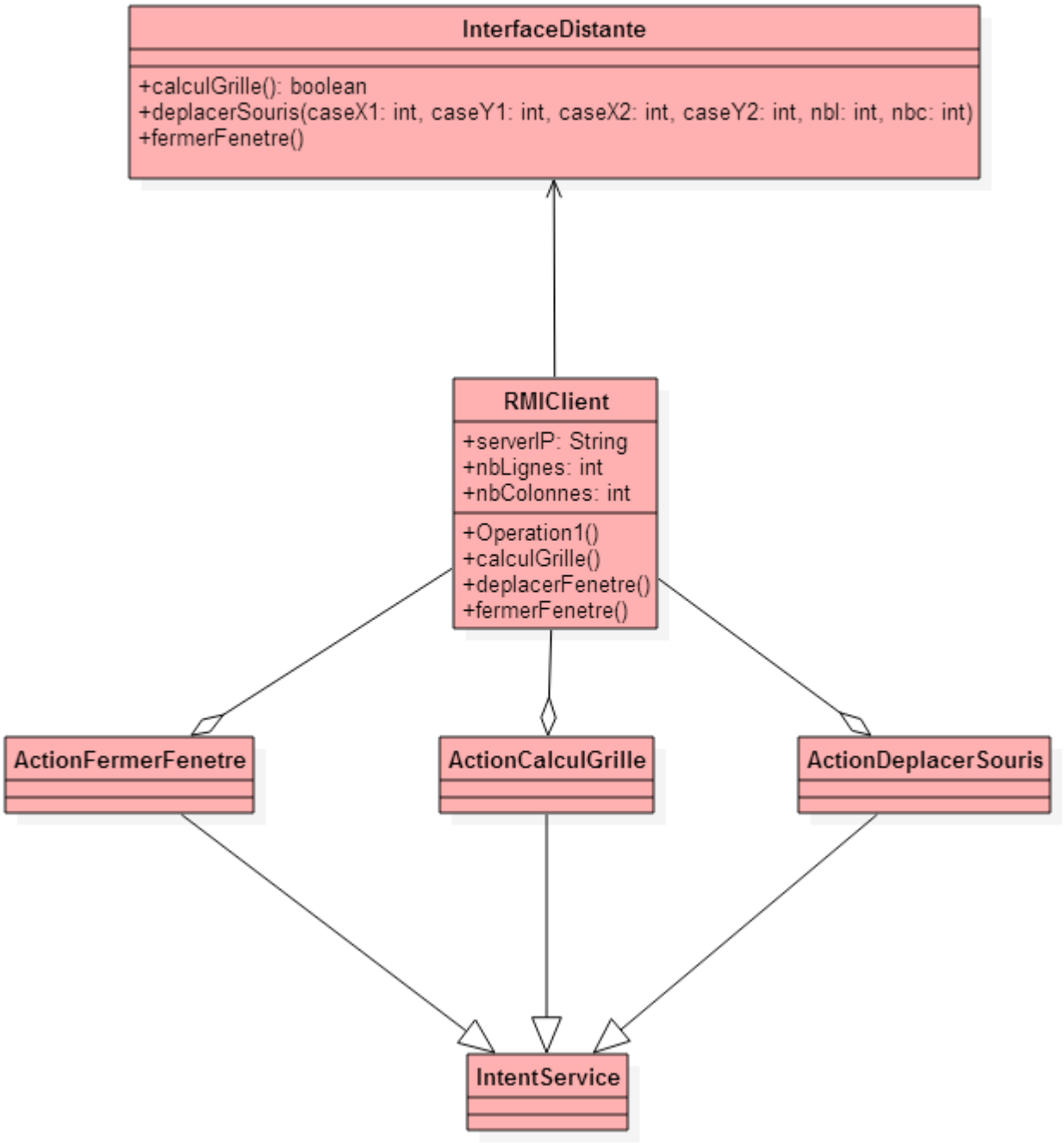
Annexe n° 2 : modele.traitement



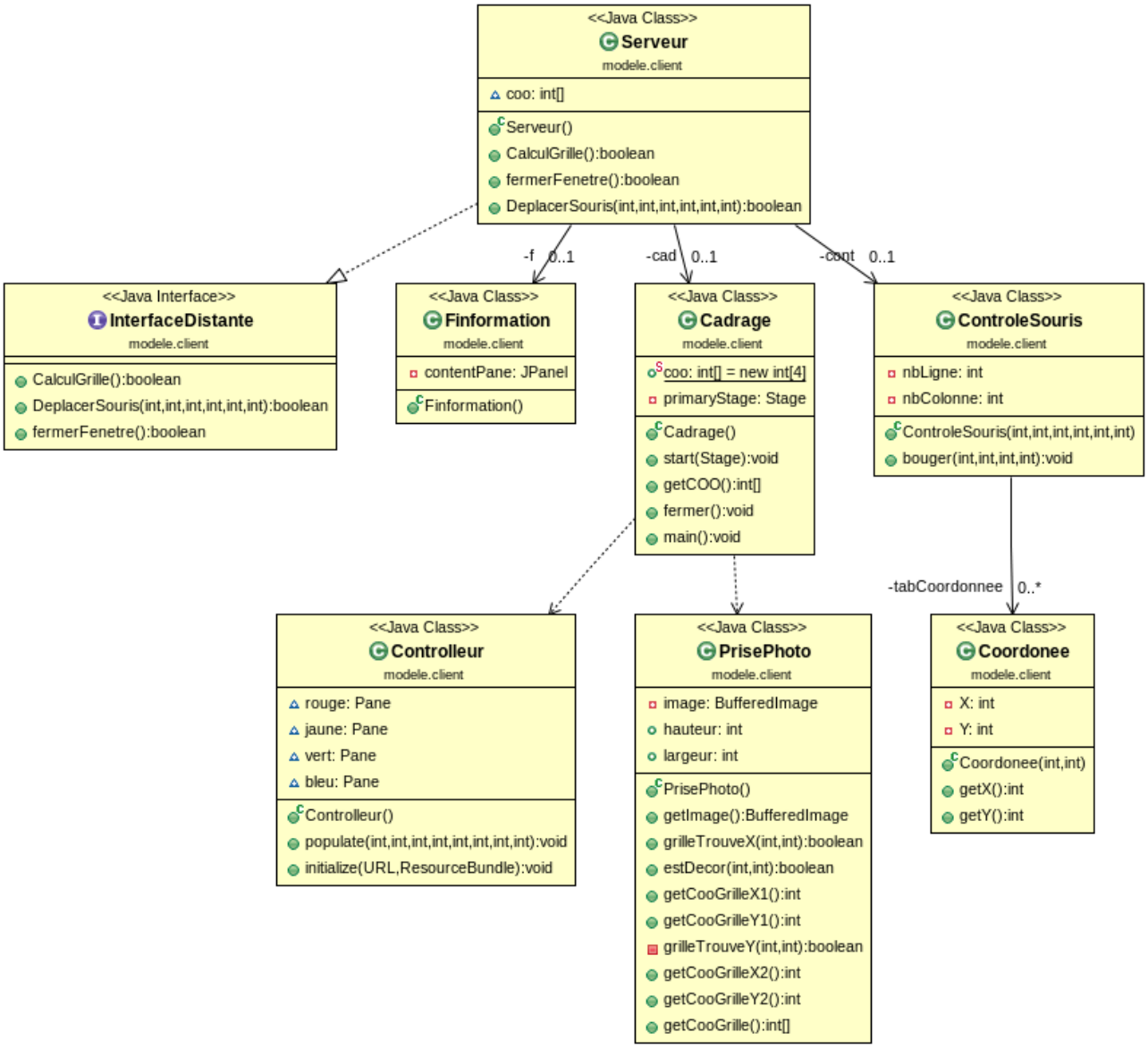
Annexe n°3 : modele.analyse



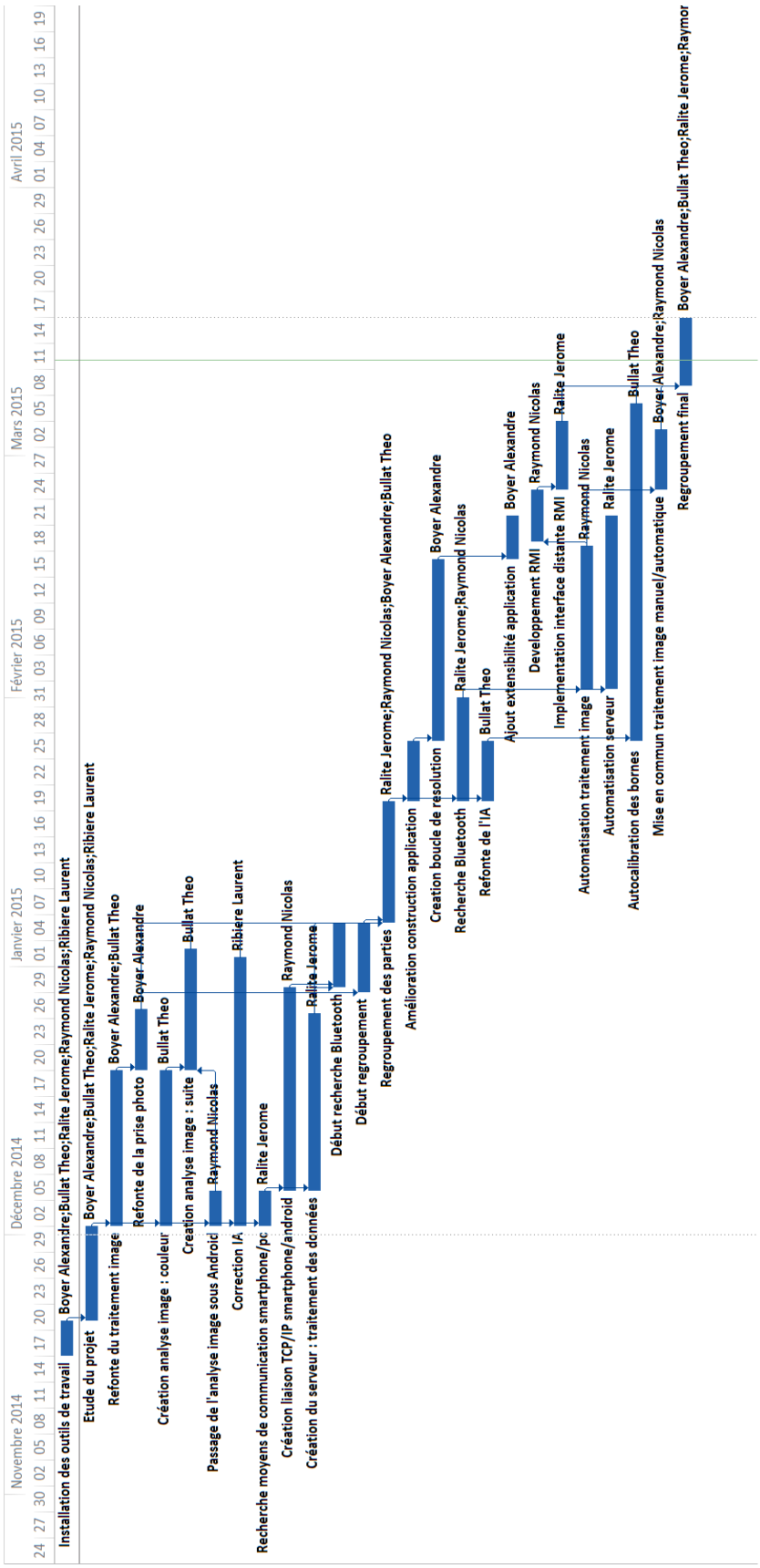
Annexe n°4 : modele.ia



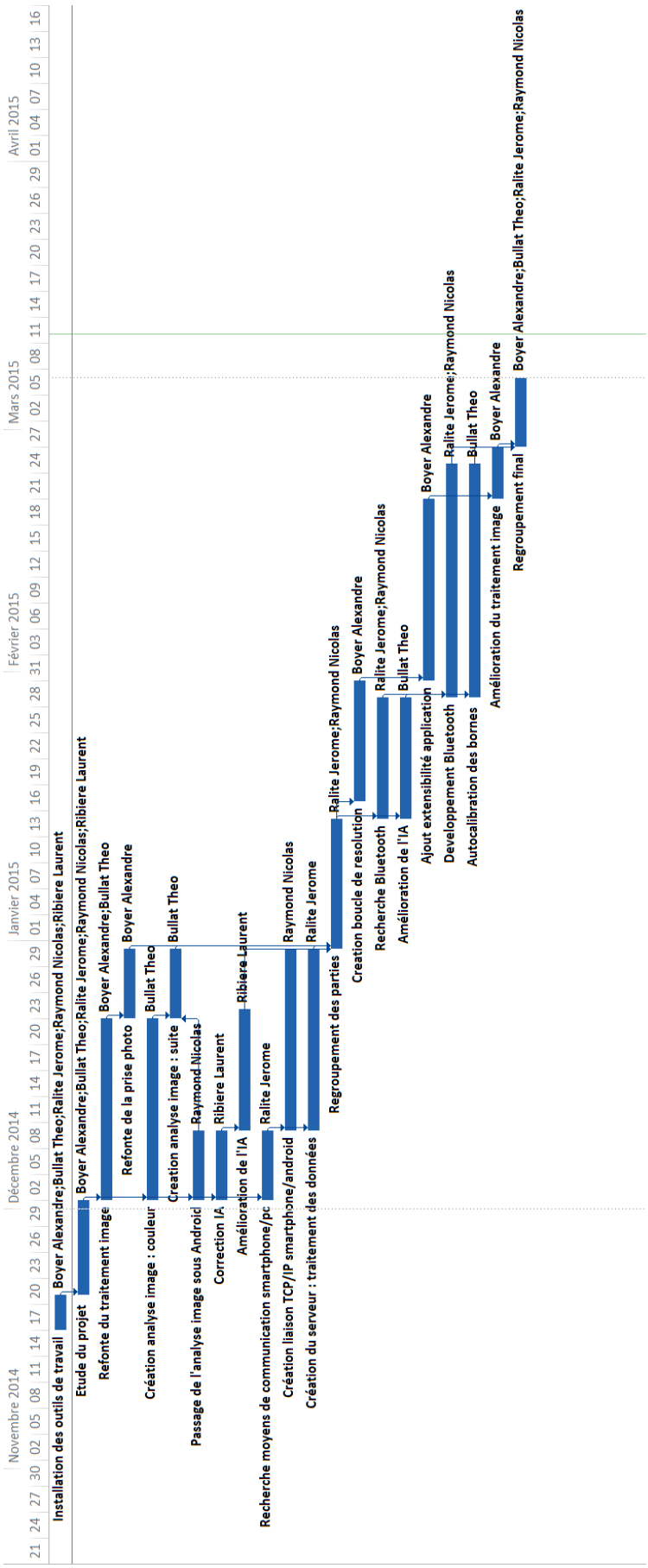
Annexe n°5 : modele.client



Annexe n°6 : modele.serveur



Annexe n°7 : diagramme de Gantt réel



Annexe n°8 : diagramme de Gantt prévisionnel