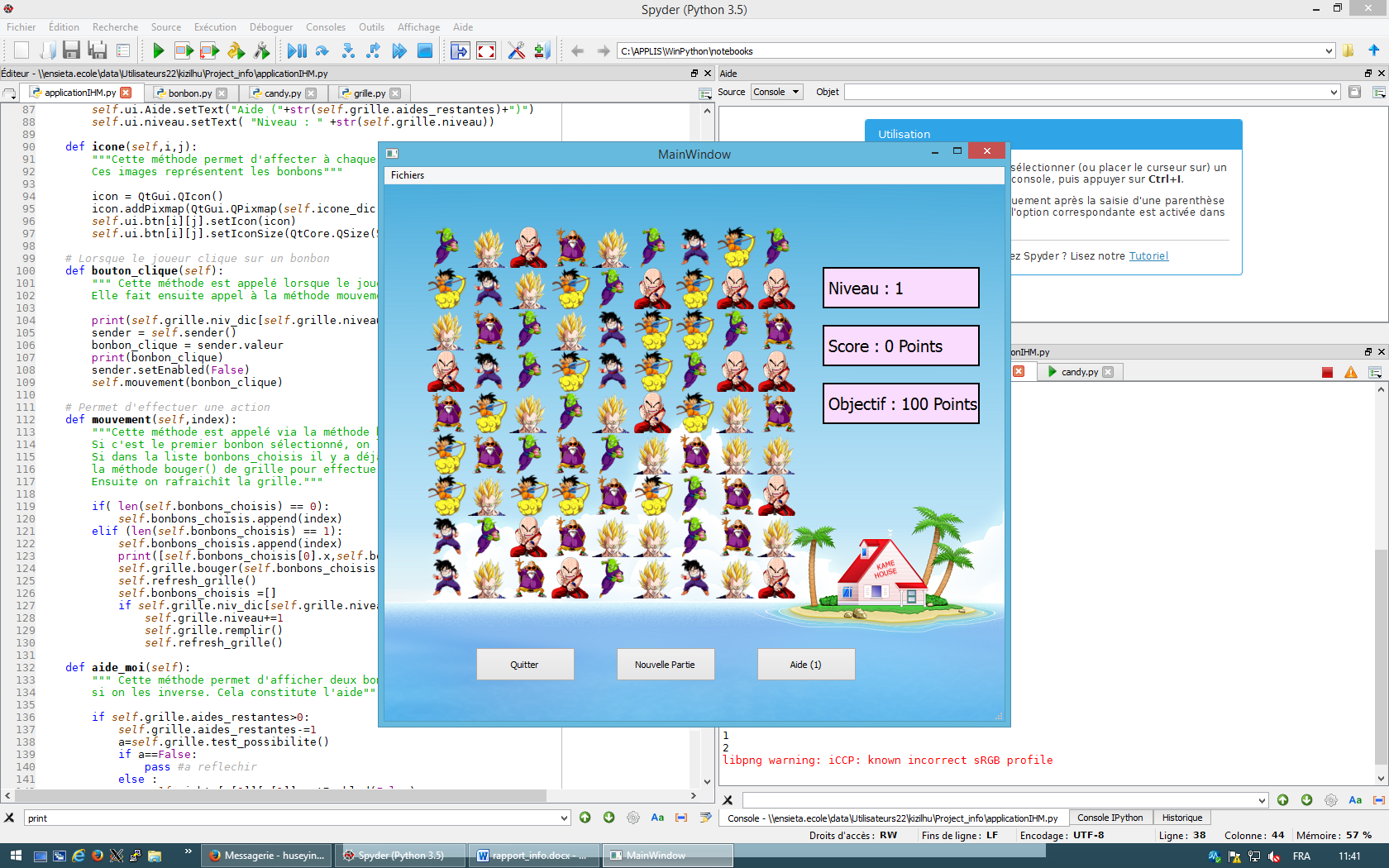


**UV 2.1 – Projet informatique**

**Année 2016/2017**

**Bejeweled**



**KHALILE Julien – KIZIL Huseyin**

**Groupe 4**

**Sommaire**

Introduction

1. Description générale
   1. Idées générales
   2. Pistes envisagées
   3. Hypothèse effectuée
2. L’application
   1. Présentation générale du programme
   2. Description des classes
   3. Description des méthodes
3. Réflexion sur le projet
   1. Tests unitaires
   2. Limitations du projet
   3. Apports personnels
   4. Perspectives et améliorations

Conclusion

**Introduction**

Notre projet consiste consistait au développement du célèbre jeu de plateforme Bejeweled. Le principe du jeu est simple. Il y a une grille de départ d’une certaine taille remplie de plusieurs bonbons de différentes couleurs. Il y a au total 6 bonbons différents. L'objectif est d'échanger un bonbon avec un bonbon adjacent pour former une chaîne horizontale ou verticale de trois bonbons ou plus de même couleur. Des points bonus peuvent être donnés lorsque des chaînes de plus de trois bonbons identiques sont formées et lorsque deux chaînes sont formées en un seul échange. Les bonbons explosent lorsque des chaînes sont formées et des gemmes tombent du haut pour combler les emplacements disponibles. Parfois, des réactions en chaîne sont déclenchées lorsqu’après explosion de nouveaux alignements apparaissent sans actions effectuées de la part du joueur. Des niveaux sont mis en place dans le but d’accroître la difficulté du jeu.

L’implémentation du code réside donc dans la maîtrise de deux choses : la gestion des traitements de données à travers une grille ainsi qu’une représentation graphique fluide de ladite grille. L’objectif était donc de faire interagir le code traitant les données avec l’interface homme-machine et inversement.

Le lancement du code s’effectue en exécutant le fichier python «applicationIHM.py».

**1. Description générale**

1.1 Idées générales

Nous allons donc décrire ici l’idée globale du code. Pour cela nous allons vous décrire exactement comment se déroule l’exécution du code lorsque le joueur sélectionne 2 bonbons adjacents. Deux classes majeures sont donc présentes : Bonbon() et Grille().

Il y a tout d’abord une grille de taille (n,m) contenant des objets de type ‘bonbon’ en donnée ainsi que (n,m) boutons représentant graphiquement cette grille. Les bonbons qui tombent sont générés de manière aléatoire.

Lorsque le joueur clique sur 2 bonbons adjacents, nous testons les alignements possibles. S’il y a alignement alors il y a échange et les bonbons concernés explosent. Les bonbons se situant au-dessus des bonbons explosés viennent les remplacer.

Lorsqu’un certain nombre de points est atteint il y a changement de niveau. Dans ce cas-là, soit certaines cases sont bloquées, ce qui restreint les mouvements possibles, soit le nombre de bonbons présent augmente de 1, ce qui accroît la difficulté.

1.2 Pistes envisagées

En premier lieu, nous avons consacrés plusieurs heures à la réflexion sur la structure générale et sur la manière dont le code devra fonctionner. Nous nous sommes focalisés sur le traitement des données dans une première partie puis sur l’interface homme machine.

Tout d’abord, nous avions pensé à utiliser des listes de listes (comme vue en TD). Cependant la structure la plus adéquate est la grille (array) via le module numpy.

Ensuite, concernant l’interface homme machine, l’idée initiale a été d’associer une image à chaque bonbon. L’image serait situé aux coordonnées (x,y) associées aux variables d’instances du bonbon. Mais la maitrise des coordonnées ainsi que la gestion des cliques s’annonçaient délicats. Nous avons donc eu l’idée de mettre des boutons pour chaque bonbon. Ainsi le nombre de boutons dépend de la taille de la grille. Une icône est associée à chaque type de bonbon.

1.3 Hypothèse effectuée

Dans le cadre de ce projet, nous avons fait une seule hypothèse simplificatrice.

Elle porte sur la génération des grilles.

Dans un jeu, il faut s’assurer qu’il existe un moyen de remporter la partie. Dans notre cas, nous avons considérés que la génération aléatoire des bonbons au sein de la grille amenait avec une probabilité très élevée à une solution concluant sur la victoire du joueur. Lorsque nous avons joué, nous ne nous sommes jamais confrontés à une situation dans laquelle nous n’avions plus de possibilité.

**2 L’application**

2.1 Présentation générale du programme

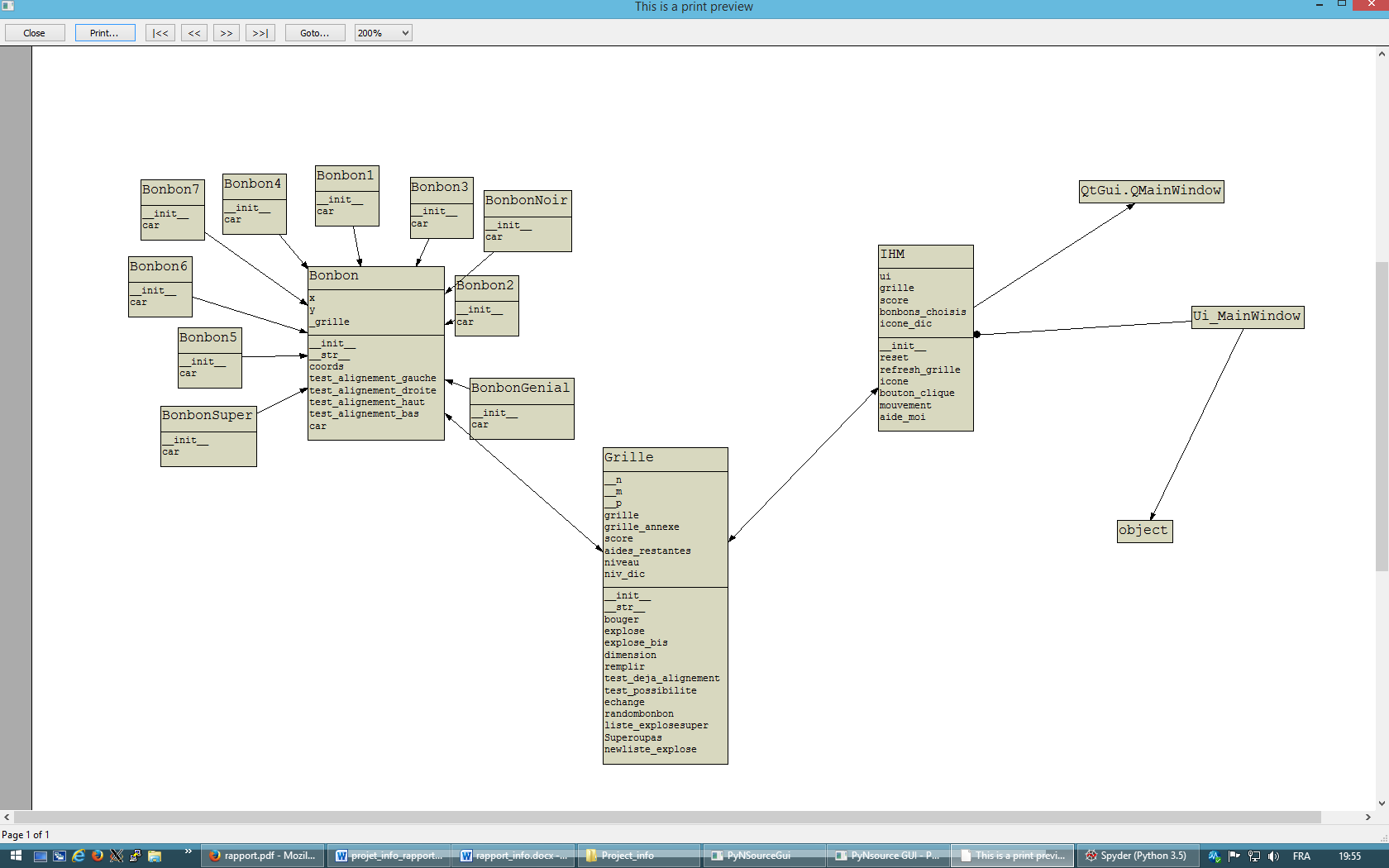


Figure - Diagramme de classe

2.2 Descriptions des classes

2.2.1 Classe Bonbon()

Cette classe représente les variables « globales » du jeu, à savoir les différents bonbons classiques puis les bonbons bonus (super et génial) ainsi les bonbons noirs. Cela est effectué via un polymorphisme. Ces bonbons ont comme variables d’instances leurs coordonnées dans la grille de jeu ainsi que la grille à laquelle elle appartient. Cette classe contient quatre principales méthodes : test\_alignement\_droit(), test\_alignement\_gauche (), test\_alignement\_haut() et test\_alignement\_bas(). Ces méthodes seront revues en détail dans la suite de ce rapport.

2.2.2 Classe Grille()

Cette classe est le cœur de notre programme. Comme son nom l’indique, cette classe est responsable de la gestion de la grille de bonbons. Les variables d’instances utiles au fonctionnement du code sont : sa taille (n et m), le score, le niveau, un dictionnaire définissant les scores à atteindre ainsi que la grille (array du module numpy) contenant les différents bonbons. La méthode bouger() est la pièce maîtresse de l’application puisque c’est elle qui assure la gestion des mouvements dans la grille.

2.3 Description des méthodes

2.3.1 Méthode Bouger()

Cette méthode permet d’échanger deux bonbons adjacents si un ou plusieurs alignement(s) en découle(nt). Alors elle explose l’ensemble des bonbons concernés, fait tomber les bonbons situés au-dessus et génère aléatoirement des bonbons dans les cases vides.

2.3.2 Méthode test\_alignement\_X()

Ces quatre méthodes permettent à un bonbon donné si, lorsqu’il effectue un mouvement vers X (droite, gauche, haut ou bas), il y a un alignement horizontal et/ou vertical et renvoie la liste des bonbons à exploser.

2.3.3 Méthode Explose() **(Récursive)**

Cette méthode récursive explose les bonbons concernés par un alignement. De plus, elle gère les alignements causés par la chute des bonbons. S’il y a alignement, la fonction est rappelée.

2.3.4 Méthode Explose\_bis()

Cette méthode représente la « gravité » puisqu’elle fait tomber les bonbons. Elle permet également d’actualiser les coordonnées des bonbons déplacés. Enfin, elle fait apparaitre les nouveaux bonbons en haut de la grille.

2.3.5 Méthode Remplir()

Cette méthode remplit en début de chaque partie et de chaque niveau la grille aléatoirement. De plus, nous avons pris en compte le fait qu’il soit possible qu’il y ait des alignements à la génération de la grille. Alors nous les explosons sans les prendre en compte pour le score.

2.3.6 Méthode test\_possibilite()

Cette méthode représente l’aide puisqu’elle parcourt la grille à la recherche de mouvements possibles. Elle renvoie le premier couple de bonbons rencontrés qui engendrerait une explosion, s’il y en a.

**3 Réflexion sur le projet**

3.2 Les limitations

Lors du commencement de notre projet, nous avons réalisé que générer une grille aléatoirement pouvait créer des problèmes que la grille obtenu n’assurait pas forcément au moins une possibilité au joueur de gagner (c’est-à-dire passer au niveau suivant).Ce problème se pose à deux moments distincts.

Lors de la génération de la grille, rien ne nous garantit que le joueur puisse créer un alignement en inversant deux bonbons. Faisons une analogie avec les graphes colorés connexe et notre grille. Nous possédons 6 bonbons qui sont équivalents à six couleurs et chaque case est adjacente à quatre cases au maximum. Ainsi il est facilement possible de colorier un graphe qui respecte la structure de notre grille (chaque case ou sommet est relié à ses cases adjacentes).

Figure 2 : un graphe coloré représentant une grille de bonbons

Donc il est possible de créer de manière aléatoire une grille qui ne permette pas au joueur de gagner.

Le second problème survient lors de l’explosion des bonbons causée par l’inversion de deux bonbons. On génère alors des bonbons aléatoirement qu’on vient placer au sommet de notre grille. Cependant, ces bonbons ne dépendent pas du chemin suivi par le joueur au cours du jeu. Heureusement, il semble que les cas où le joueur ne peut créer d’alignement soit extrêmement rare.

De plus, l’aléa ne permet pas de concevoir un niveau qui possède un degré de difficulté constant. L’alignement de bonbon peut soit juste entrainer la destruction des bonbons de l’alignement soit créer une réaction en chaine d’explosion de bonbons par l’arrivée des nouveaux bonbons crées de manière aléatoire.

3.3 Les apports personnels

Afin d’assurer un minimum d’intérêt au joueur pour le jeu, celui-ci comporte différents niveaux d’une difficulté croissante. A partir du niveau 5, un septième type de bonbon est ajouté à notre grille ce qui permet de diminuer les alignements possibles et ainsi d'augmenter la difficulté du niveau.

Pour créer de la diversité dans les niveaux, on a également inventé la classe BonbonNoir qui hérite de Bonbon. Ces bonbons ne peuvent ni être déplacés ni être explosés. Ils permettent de bloquer une case. En fonction du niveau, on les place dans la grille de manière à garantir un certain niveau de difficulté.

On a également créé deux classes de bonbon bonus qui hérite de Bonbon.

Le premier BonbonSuper apparait lorsque l’on aligne 4 bonbons de même type. La difficulté est d’arrivé à exploser se bonbon, pour cela il doit être dans un alignement qui comporte au minimum 3 bonbons de même type. L’explosion de ce BonbonSuper entraine la destruction de tous les bonbons qui sont sur sa ligne et sa colonne.

Le second, BonbonGenial apparait lorsque 5 bonbons de même type sont dans les configurations suivantes.

Configuration 1 Configuration 2

Pour exploser ce bonbon, il suffit de double cliquer dessus ce qui entraine la destruction de tous les bonbons qui sont sur sa ligne et sa colonne.

De plus, dans le cas où le joueur se trouve en difficulté, une aide est mise à disposition qui permet de montrer les bonbons que l’on peut échanger. Le nombre d’aides présent dépend du niveau dans lequel le joueur se trouve.

Une interface agréable a été mise en place.

3.4 Perspectives et améliorations

La perspective, qui semble essentiel pour tout jeu qui se respecte, est de donner la possibilité au joueur de gagner à chaque début de niveau. C'est l'aléa dans ce code qui ne permet pas de le garantir.

Une des solutions pour résoudre ce problème serait de générer deux grilles. La première serait celle avec laquelle le joueur interagirait et la seconde contiendrait les bonbons qui tomberont au moment des explosions. Ainsi on connaîtrait à l'avance l'ordre des bonbons qui tomberont. La difficulté réside dans le fait qu'elles doivent permettre au joueur de gagner une partie en empruntant différent chemin sinon la difficulté serait trop grande.

Lorsque l’on joue CandyCrush ou Bejeweled, on comprend bien que c'est l’esthétique et la simplicité de l'IHM qui rend le jeu plaisant plus que son principe. Les améliorations du jeu doivent se focaliser sur ce point comme par exemple animer la chute des bonbons dans la grille et leurs explosions.

**Conclusion**

Ce projet informatique, le premier de ce genre s’est avéré être un réel challenge pour nous. Nous avons tout au long de ce projet développé une réelle attirance pour la programmation puisque l’objectif était ici de coder un jeu. Nous pensons y être parvenus bien qu’une part d’aléa intervienne. Plusieurs niveaux sont présents ainsi qu’une interface homme-machine agréable. Nous remercions notre encadrant M.COATANHAY ainsi que M.TOUMI pour leurs aides.