Rapport Projet inspiré de Candy Crush

Nicolas Melaerts

Janvier 2022

Contents

1 Tâches

1.1 Tâches accomplies

1. Fonctionnalités de base
2. Animation des objets en cours de suppression
3. Murs
4. Faire glisser le carré
5. Impossible
6. Glaçages
7. Bonbons spéciaux
8. Score
9. Meilleur score
10. Ecran d'accueil
11. Niveaux et sélection de niveau
12. Niveaux à ingrédients

1.2 Tâches non-accomplies

1. Suggestion
2. Objectifs
3. Editeur de tableau

2 Classes

Pour chaque classe fournir l'interface (pas le corps des méthodes) et décrivez brievement le rôle de chaque classe et comment elle se rapporte aux autres classes.

2.1 Shapes.hpp

Classe Rectangle et Circle pour dessiner les éléments de jeu tels que toutes les sortes de bonbons ainsi que les murs, les glaçages et les ingrédients.

```
class Rectangle {
  Point center;
  int w, h;
  Fl_Color fillColor, frameColor;
 public:
  // constructeur
  Rectangle(Point center, int w, int h,
  Fl_Color frameColor = FL_BLACK Fl_Color fillColor = FL_WHITE);

  // getteur / setteur
  void setFillColor(Fl_Color newFillColor)
  Fl_Color getFillColor();
  void setFrameColor(Fl_Color newFrameColor);
  Fl_Color getFrameColor();
  void setWidth(int neww);
  void setHeight(int newh);
  int getWidth();
  int getHeight();
  Point getCenter();
  void setPoint(Point newC);
  Point getPoint();

  // methodes
  bool contains(Point p);
  void draw();
};
```

```
class Circle {
  Point center;
  int r;
  Fl_Color fillColor, frameColor;
  const double pi=3.14159265358979323846264338327950288 4L;
 public:
  Circle(Point center, int r,Fl_Color frameColor = FL_BLACK,
           Fl_Color fillColor = FL_WHITE);

  // getteur / setteur
  void setFillColor(Fl_Color newFillColor);
  Fl_Color getFillColor()
  void setFrameColor(Fl_Color newFrameColor);
  Fl_Color getFrameColor()
  Point getCenter();
  void setCenter(Point newCenter);

  // methodes
  void draw();
  bool contains(Point p);
};
```

2.2 Text.hpp

```
class Text {
  Point center;
  string s;
  int fontSize;
  Fl_Color color;
```

```
 public:
  //Constructor
  Text(string s, Point center, int fontSize = 10, Fl_Color color = FL_BLACK);

  //Setters and getters
  string getString();
  void setString(const string &newString);
  int getFontSize();
  void setFontSize(int newFontSize);
  Point getCenter();
  void setCenter(Point newCenter);

  // methode
  void draw();
};
```

La classe TextRectangle hérite de la classe Rectangle dans Shapes.hpp et de la classe Text. Avec cette classe on va pouvoir dessiner les ingrédients (élément de jeu).

```
class TextRectangle: public Text, public Rectangle {
 public:
  TextRectangle(Point center, int w, int h, string text, int fontSize = 10);

  // methode
  void draw();
};
```

2.3 ElementDeJeu.hpp

Classe abstraite ElementDeJeu, dont vont hériter les classes Bonbon, BonbonSpecialRond, Mur, Glacage, BonbonSpecialHorizontal, BonbonSpecialVertical, BonbonSpecialRondCookies (qui hérite d'abord de BonbonSpecialRond) et Ingredient.

```
class ElementDeJeu{
    private:
        Point posPlat;
        Fl_Color couleur;

    public:
        Animation *animation;

        // constructeur
        ElementDeJeu(Point posPlat, Fl_Color couleur);

        // getter and setter
        Point getPoint();
        Point getposPlat();
        void setPosPlat(Point newPosPlat);
        void setCouleur(Fl_Color newc);

        // methodes
        virtual void draw()=0;
        virtual void drawWithoutAnimate()=0;
        virtual int getMyId()=0;
```

```
        virtual Fl_Color getcouleur()=0;
        virtual void mouseClick(Point mouseLoc)=0;
        virtual void mouseMove(Point mouseLoc)=0;
        virtual Point getPosPlatifcontain(Point coord)=0;
        virtual void DoExplosion()=0;
        virtual void ElementMove(int sens)=0;
        virtual bool animation_is_complete()=0;
};
```

```
class Bonbon: public ElementDeJeu{
    private:
        Rectangle r;

    public:
        Bonbon(Point posPlat, Fl_Color couleur, int w, int h);

        // getter
        Rectangle getR();
};
```

```
class Mur: public ElementDeJeu{
    private:
        Rectangle r;
    public:
        Mur(Point posPlat, int w, int h);
};
```

```
class Glacage: public ElementDeJeu{
    private:
        Rectangle r;
        int vie;        // Glacage de Niv 1 ou Niv 2
    public:
        Glacage(Point posPlat, int w, int h);

        // methodes
        int getVie();
        void perdVie();
};
```

```
class Ingredient: public ElementDeJeu{
    private:
    TextRectangle r;
    public:
        Ingredient(Point posPlat, string s, int w,
        int h):ElementDeJeu(posPlat, FL_WHITE),
        r{{50*posPlat.y+48, 50*posPlat.x+70}, w, h, s, 20}{}

};
```

```
class BonbonSpecialRond: public ElementDeJeu{   // bonbon emballe Bombe
    private:
        Circle c;
    public:
        BonbonSpecialRond(Point posPlat, Fl_Color couleur, int r);
};
```

```
class BonbonSpecialRondCookies: public BonbonSpecialRond{
    public:
    BonbonSpecialRondCookies(Point posPlat, Fl_Color couleur,
    int r):BonbonSpecialRond(posPlat, couleur, r){}
};
```

```
class BonbonSpecialVertical: public Bonbon{
    public:
    BonbonSpecialVertical(Point posPlat, Fl_Color couleur, int w,
    int h):Bonbon(posPlat, couleur, w, h){}
};
```

```
class BonbonSpecialHorizontal: public Bonbon{
    public:
    BonbonSpecialHorizontal(Point posPlat, Fl_Color couleur, int w,
    int h):Bonbon(posPlat, couleur, w, h){}
};
```

2.4 Animation.hpp

Les animations se font sur des ElementDeJeu. Par exemple les bonbons peuvent subir une translation dans six sens différents et peuvent exploser (il vont dans ce cas subir une rotation). Tandis que les murs par exemple ne bougeront et n'exploseront jamais par les règles du jeu.

```
struct Translation {
  Translation(Point p) {
    fl_push_matrix();
    fl_translate(p.x, p.y);
  }
  ~Translation() {
    fl_pop_matrix();
  }
};

struct Rotation {
  Rotation(Point center, double angle) {
    fl_push_matrix();
    fl_translate(center.x, center.y);
    fl_rotate(angle);
    fl_translate(-1*center.x, -1*center.y);
  }
  ~Rotation() {
    fl_pop_matrix();
  }
};
```

```
class Animation{
public:
    enum AnimationType {explosion, move};
private:
    const int animationTime = 60;
    const int descente = 100;
    int sens;
    ElementDeJeu *b;
```

```
    AnimationType animationType;
    int time{0};
    Point MoveElementDeJeu();
    double ExploseElementDeJeu();
public:
    Animation(ElementDeJeu *BonbonToAnimate, AnimationType animationType, int sens):b{BonbonTo

    //methodes
    void draw();
    bool isComplete();
};
```

2.5 Jeu.hpp

Gère le fonctionnement du jeu. Utilise la classe ElementDeJeu pour savoir quel élément se trouve sur le plateau (Mur, Bonbon, Ingrédient,...). Utilise aussi la classe ScoreAndNbcoups pour gérer le score, les coups restants et le meilleur score durant une partie.

```
class jeu{
private:
    vector<vector<int> > plateau;
    vector< vector<shared_ptr<ElementDeJeu>> > E;
    vector<Fl_Color > Colors{FL_RED, FL_BLUE,FL_YELLOW, FL_GREEN, FL_MAGENTA, fl_rgb_color(251

    ScoreAndNbcoups c;

    int taille_plateau = 9;
    int nb_couleurs_bonbon = 6;
    int niv;
    bool jeu_en_cours = 0;
    bool aucun_coups_poss = 0;

    void ouvertureNiv(int niv);
    void ouvertureInfo(int niv);

public:

    jeu(int niv);

    void setGame(int niveau);

    // getteur et setteur
    shared_ptr<vector< vector<shared_ptr<ElementDeJeu>> >> getPtrPlateau()
    shared_ptr<ScoreAndNbcoups> getPtrScoreAndNb_coups()
    vector<vector<int> > get_plateau()
    bool get_aucun_coup_poss()
    void set_aucun_coup_poss(bool coup_poss);
    bool jeu_possible();
    bool get_jeu_en_cours();
    void set_jeu_est_en_cours(bool en_cours);
    int get_taille_plateau();
    void set_plateau(vector<vector<int>> newplat);
    int get_nb_couleur();
    int getelemplateau(int a, int b);
    void reset_meilleur_score();
```

```
    void init_plateau(vector< vector<shared_ptr<ElementDeJeu>> > E);
    void search_combinaison();
    vector<vector<int> > check_3bonbons(vector<vector<int> > plat);

    // methodes
    void check_5bonbons();
    void coup_cookie(int id, Point cookie_to_delete);
    void check_4bonbons();
    void check_L();

    void echange(Point a, Point b);
    bool coup_possible(Point a, Point b);   // swap autorise
    bool coup_possible();       // verif si coup possible sur le plateau si non -> melanger()
    void melanger();    // si pas la poss de faire une combi de 3 bonbons

    vector<Point> setToExplose();
    void DoExplose(vector<Point> to_explose);
    void Explose(vector<Point> to_explose);
    void ExploseLigne(int l);
    void ExploseColonne(int c);
    void ExploseBBspecialRond(Point empl);
    void DoExploseGlacage();
    void check_ingredient();

    void fall();
    void fall_mur_diagonale();
    bool get_finish_fall();

    void afficher_plateau_de_jeu();

    void wait_anim();
};
```

2.6 Score.hpp

Gère le score et le meilleur score (tient à jour un fichier contenant le meilleur score pour le niveau sélectionné), le nombre de coups restants pour que la classe jeu sache si on peut continuer à jouer.

```
class ScoreAndNbcoups{
    private:
        int meilleur_score;
        int score;
        int nb_coups_restants;
        int nb_coups_joue = 0;

    public:
        ScoreAndNbcoups(int meilleur_score, int nb_coups_restants, int score=0):
        meilleur_score{meilleur_score},nb_coups_restants{nb_coups_restants} ,score{score}{};

        // getter and setter
        void set_nb_coups_restants(int new_nb_coups);
        int get_nb_coups_restants();
        void set_score(int new_score);
        int get_score();
        void set_meilleur_score(int new_m_score);
```

```
        int get_meilleur_score();

        // methodes
        void do_coup(){nb_coups_restants--; nb_coups_joue++;}
        void augmente_score(int niv, int point);
        void write_meilleur_score(int niv, int s);
        void reset_nb_coup_joue();
};
```

2.7 Afficher.cpp

Vue de la classe jeu, affiche le plateau d'éléments de jeu avec la classe Canvas, les informations relatives au score et le nombre de coups restants avec la classe AfficherScoreAndNb_coups.
La classe EcranAccueil est liée aux classes d'éléments de jeu, pour afficher comment sont représentés les différents éléments de jeu avant de pouvoir jouer.

```
class Canvas{
    private:
        shared_ptr<jeu> j;
        shared_ptr<vector< vector<shared_ptr<ElementDeJeu>> >> ptrPlateauE;

    public:
        Canvas(shared_ptr<jeu> j):j(j){};

                //methode
        void draw();
};

class AfficherScoreAndNb_coups{
    private:
        shared_ptr<jeu> j;
        shared_ptr<ScoreAndNbcoups> ptrScoreAndNb_coups;

    public:
        AfficherScoreAndNb_coups(shared_ptr<jeu> j):j(j){};

                //methode
        void draw();
};

class Menu{
    private:
        shared_ptr<jeu> j;
        Text info_niv{"1", {200, 15}, 20, FL_BLACK};
        int time=0;

    public:
        Menu(shared_ptr<jeu> j):j{j}{};

                //methodes
        void draw();
        void changeNiv();
        void AffichagePendantPartie();
        int getniv();
```

```
};

class EcranAccueil{
    private:
        Glacage g{{3,5}, 45, 45};// pour lui faire perdre une vie
        et afficher un glacage de rectangle_info_niv 1
    public:
        EcranAccueil(){g.DoExplosion();};

        //methode
        void draw();
};
```

2.8 Control.cpp

Contrôleur, classe qui permet successivement de récupérer les différents événements provoqués par le joueur, de pouvoir interagir avec la classe jeu et d'effectuer les changements adéquats sur le jeu.

```
class ControlJeu{
    private:
        shared_ptr<jeu> j;
        vector<Point> ForDrag; // coordonnee souris lors d'un swapping
        shared_ptr<vector< vector<shared_ptr<ElementDeJeu>> >> ptrPlateauE;
    public:
        ControlJeu(shared_ptr<jeu> j);

                //methodes
        void mouseMove(Point mouseLoc);
        void drag(Point mouseLoc);
        void mouseClick(Point mouseLoc);

        void tentativeSwap();
        DoublePoint findBonbonToSwap(int idxForDrag);
        void SwapIfCoupPossible(Point p1, Point p2, Point idx1, Point idx2);
        void SwapBonbonAnim(Point p1, Point p2, Point idx1, Point idx2);

        void changeNiv(int niv);
        bool get_jeu_en_cours();
        void reset_meilleur_score();
        vector<Point> getForDrag();
};
```

3 Logique de jeu

Supposons que je démarre le jeu, sélectionne le premier niveau (si la tâche 11 est terminée), attends 10 secondes, puis fais un coup. Décrivez en détail ce qui se passe dans votre code.

Quand je démarre le jeu, la classe EcranAccueil est initialisée. Un pointeur vers la classe jeu qui a comme paramètre l'entier 1 pour le premier niveau (par défaut le niveau sélectionné quand on démarre le jeu) est initialisé. Enfin, les classes

Canvas, AfficherScoreAndNb_coups, Menu et ControlJeu avec en paramètre un le pointeur vers la classe jeu sont initialisées.

L'écran d'accueil ne va s'afficher que pendant quelques secondes et puis les méthodes draw de Canvas, AfficherScoreAndNb_coups et Menu vont dessiner les éléments adéquats. Pour faire un coup, je déplace ma souris sur un bonbon, via FL_MOVE, je récupère les coordonnées de ma souris et avec les méthodes mouseMove de la classe ControlJeu et ensuite de la classe ElementDeJeu, les contours du bonbon où se trouve ma souris vont changer de couleur (noir ou blanc en fonction de la couleur du bonbon pour une meilleure visibilité). Je clique sur le bonbon, le déplace vers un bonbon adjacent et relâche le clic. Les coordonnées de la souris lors du déplacement pendant le clic sont envoyées dans un vecteur appelé ForDrag dans ControlJeu, et au relachement du clic (FL_RELAESE) on appelle la méthode tentativeSwap de ControlJeu pour réagir. Avec les méthodes de la classe jeu, on vérifie si le coup est possible ou non. Si le coup est possible, on effectue l'animation pour interchanger les bonbons d'emplacement et on vérifie sur le plateau de jeu si le coup est valide (s'il apporte une combinaison). On fait ensuite appel à la fonction de la classe jeu search_combinaison qui va vérifier sur le plateau de jeu s'il y a eu des combinaisons de 5, 4, 3 bonbons, exploser les glaçages si nécessaire, faire tomber, les bonbons et en mettre des nouveaux, vérifier l'emplacement des ingrédients, augmenter le score dans la classe Score et enfin vérifier si un coup est jouable sur le plateau qui a subi des modifications. Si ce n'est pas le cas, on appelle la méthode de jeu, melanger, pour remplacer les bonbons normaux. Pour finir, la fonction search_combinaison appelle la fonction do_coup de la classe Score pour diminuer d'une unité le nombre de coups restants. Quand tous les changements dans le jeu ont été effectués, on vide le vecteur ForDrag et on attend un nouvel événement du joueur (un nouveau coup ou 'q' pour quitter le programme par exemple).

4 Modéle-Vue-Contrôleur

Avez-vous utilisé ce modèle de conception ? Si c'est le cas, expliquez comment vos classes correspondent à ce modèle de conception.
Oui j'ai utilisé le modèle de conception MVC. J'ai une classe ControlJeu qui va récupérer les différents événements tels que les mouvements de souris et les appuis sur des touches, et donc traiter les entrées de l'utilisateur et apporter des changements au modèle. Cette classe étant liée à la classe jeu, elle va pouvoir apporter des changements au jeu tels que les swap de bonbons par exemple car celle-ci connait l'état et la logique du jeu. Ensuite il y a les classes Canvas, AfficherScoreAndNb_coups et menu qui sont la vue, et qui vont afficher à l'utilisateur les données du modéle telles que le plateau de jeu et ce qui est lié au score.

Lorsque j'interchange deux bonbons, le clic et le mouvement de souris vont être interprétés dans la classe ControlJeu. L'utilisateur manipule le contrôleur, si le coup est possible on effectue un changement dans la classe jeu, on change le plateau, on fait tomber les bonbons et on en fait apparaitre de nouveaux.

Le contrôleur apporte donc des modifications au modèle. La classe Canvas et AfficherScoreAndNb_coups sont des représentations visuelles du modèle, et l'utilisateur voit la vue (le plateau de jeu et les infos liées au score).

5 Score

Le score peut augmenter en fonction de divers événements. Lorsqu'un bonbon normal explose, le score augmente de 100; il augmente de 500 quand un bonbon spécial explose, de 1000 lorsqu'un ingrédient atteint le bas du plateau et de 250 pour une explosion adjacente à un glaçage (500 au total pour l'explosion des deux niveaux de glaçage)