INFO-F202 - Candy crush

Esteban Aguililla Klein
Vlad Marian Moruntale

December 2021

Table des matières

1 Introduction

Le but du ce projet est de créer notre propre version de Candy crush en utilisant la librarie FLTK en C++. Le mouvement, est fait par glissé déposé de la souris. Ensuite, les bonbons, bonbons spéciaux, murs et ingrédients sont dessinés en utilisant des figures géométriques grâce aux fonctions spéciales du FLTK.

2 Tâches

2.1 Fonctionnalité de base

Nous avons créé un array de taille 9X9 composé de smart pointers de type Item où les bonbons, ingrédients et murs sont stockés. Ainsi, ci-dessous se trouvent les dépendance de chaque type de .

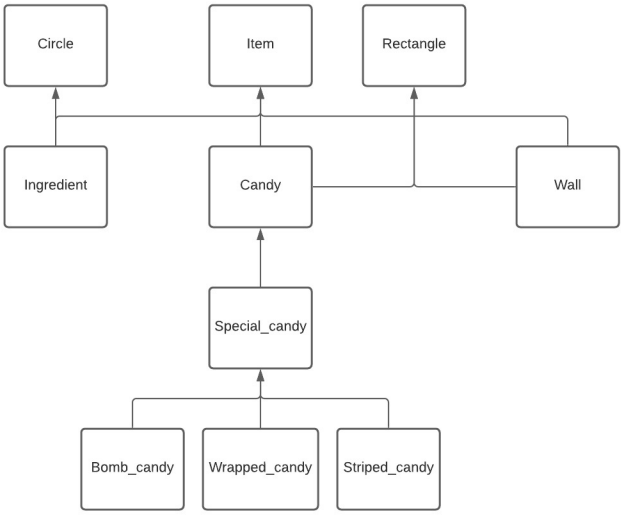

Figure 1 – Diagramme de classe simplifié spécifiant les dépendances entre objets

La classe Item possède un vecteur de Item stockant ses voisins permettant ainsi de vérifier les déplacement possibles. Alors, ce vecteur est utilisé en relation avec la méthode break_candies() qui vérifie d'abord s'il y a une suite de d'au moins 3 bonbons de la même couleur.Si c'est le cas, alors la méthode destroy_candies est appelée et marque les bonbons à faire tomber en noire. Plus tard, la méthode fall_candies() est appelée pour remplir les espaces vides avec les bonbons se trouvant au dessus. Mais, s'il n'y a aucune suite de bonbons à casser, alors les deux bonbons qui ont été changé durant le break_candies() seront remis à leur position initiale.

2.2 Animation des objets en cours de suppression

Nous avons implémenté 2 types d'animations :

— Animation de suppression : elle fait apparaître une bordure de rectangle qui grandit suivant la fonction

$$log(dure\_animation) * taille\_de\_la\_bordure$$

Ce qui lui permet de grandir rapidement au début et se de se stabiliser vers la fin donnant ainsi l'illusion de fluidité. Cette animation, est utilisé lorsqu'un bonbon est détruit ou lorsque le jeux donne un indice au joueur.

— Animation de glissage : permet à un bonbon de glisser progressivement par incréments égaux de sa position à une destination. Lors de de sa destruction le l'Item revient à sa position initiale ou non. Ainsi, cette animation est utilisée dans la méthode fall_candies() en plus du changement entre deux Item lorsque lors du choix du mouvement à effectuer

2.3 Murs

Le Mur est une sous-classe du Candy et de Rectangle qui sera toujours de couleur grise. Le joueur ne peut pas changer un bonbons avec un mur. Aussi, les espaces vides en dessous des murs seront remplis par le bonbons de gauche et sinon de droite. La classe mur est un classe simple qui a juste une méthode permettant son identification : is_wall().

2.4 Faire glisser le carré

L'utilisateur peut changer 2 bonbons en faisant glisser un carré avec son voisin. Cette méthode est gérer grâce à fltk.

2.5 Impossible

Nous avons utilisé notre méthode check_impossible() qui fait appel à la méthode forshadowing_over_9000() faisant servant à la partie **suggestion** qui a comme but de voir s'il y a au moins 3 bonbons qui peuvent être détruits. Ainsi, si il est impossible de faire un mouvement le plateau est rechargé.

2.6 Glaçage

Glaçage est un réalisé en utilisant un attribut appelé icing dans Candy avec layers pour savoir combien de layers l'icing a le bonbons.Quand des bonbons adjacents sont détruits à coté du bonbons glacé alors un layer serait soustrait jusque le layer arrive à 0 où le bonbons ne peux pas glacé.

2.7 Bonbons spéciaux

Les bonbons spéciaux sont les bomb,stripped candy et wrapped candy. Ils marchent juste comme dans candy crush.

2.8 Score

Voir séction Score

2.9 Meilleur score

Le meilleur score est sauvegardé dans un fichier .txt. Alors, le meilleur score commence à 0 pour un utilisateur qui vient de lancer le jeux pour la première fois. Ensuite, après chaque partie, si le score actuelle est plus grand que le meilleur score alors, le meilleur score sera actualisé dans le fichier .txt.

2.10 Écran d'accueil

L'écran d'accueil a 4 boutons :

— le créateur de niveau
— start the game qui lance la partie avec le niveau sélectionné par le joueur
— niveau précédent
— niveau suivant

En plus, des boutons, les auteurs, les matricules ULB, le meilleur score du joueur et le titre du jeux sont affichés.

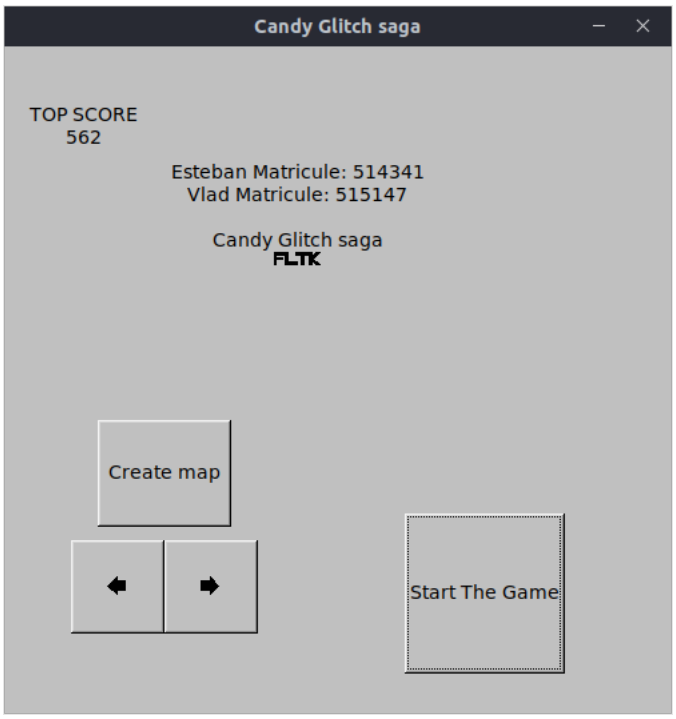

FIGURE 2 – Écran d'accueil d'un joueur ayant déjà joué

2.11 Suggestions

Ces dernières sont faites en utilisant la méthode forshadowing_over_9000() de la classe Hint qui prend chaque bonbons et ses voisins où il vérifie si il y a une combinaison de bonbons destructible si le bonbon était placé à leur place indiquant. Alors, il va utiliser l'animation de suppression pour faire pulser ces bonbons si le booléen can_vibrate est vrai.

2.12 Niveaux et sélection de niveau

Les niveaux sont sauvegardés sous forme de fichier texte dans le format suivant :

— C : un bonbon normal
— W : un mur
— I : un ingrédient
— w un bonbon emballé
— un chiffre n : un bonbon avec n couches de glaçages

La liste des niveaux est sauvegardée dans un vecteur. Ainsi, la sélection de niveaux se fait à partir de des boutons se situant sur l'écran d'accueil(-> et <-) qui utilisent un int pour parcourir le vecteur de niveaux sauvegardés dans les deux sens.

2.13 Objectifs

L'objectif est généré aléatoirement s'il n'y a pas d'ingrédients sur le plateau. Ainsi, s'il y a des ingrédients l'objectif sera toujours de les faire tomber en dehors du plateau en les faisant descendre à la dernière rangée de ce dernier. Dans le cas normal, il y a 2 types d'objectif :

— détruire un nombre aléatoire de bonbons sans tenir compte de leur couleur
— détruire un nombre aléatoire de bonbons d'une couleur spécifique

Pour chaque objectif le nombre de coups maximum que le joueur peut faire est aléatoire entre 25 inclus et 45 inclus. Si le joueur réussis l'objectif avant que son nombre de coup arrive à 0 alors la partie est gagnée et le prochain niveau est chargé (s'il y a un). Sinon, le joueur perd la partie et alors le niveau sera rechargé.

2.14 Éditeur de tableau

L'éditeur de tableau est fait d'un plateau de 9x9 bonbons blancs où l'utilisateur appuie sur un bonbon pour le changer autant qu'il veut. Ainsi, le changemenmt se fait dans l'ordre suivant bonbon normal, bombe, bonbon rayé, bonbon emballé, mur et ingrédient. Une fois qu'il n'y a plus de bonbons blancs sur l'écran, le niveau est sauvegardé dans le fichier edited_level.txt et le joueur peut accéder à son niveau en utilisant le bouton next sur l'écran d'acceuil.

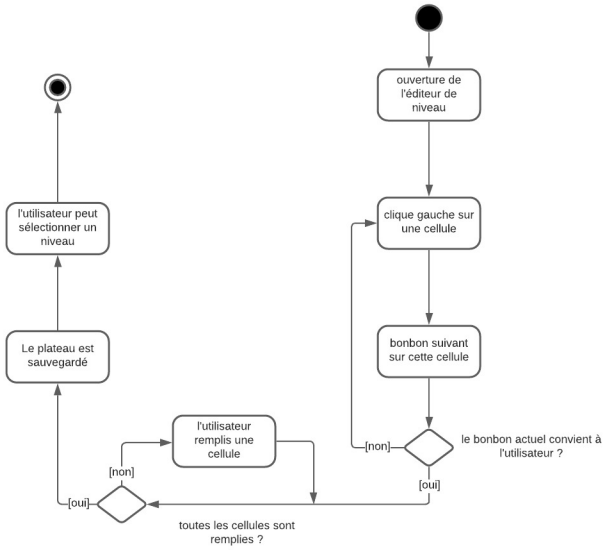

FIGURE 3 – Diagramme d'activité du fonctionnement de l'éditeur de niveau

2.15 Niveaux à ingrédients

Les ingrédients sont des cercles de couleur rouge foncé. Ainsi, un niveau avec des ingrédients aura toujours comme objectif de faire tomber tout les ingrédients présent comme indiqué dans la section **objectif**.

3 Classes

3.1 Rectangle

Classe recopiée du TP modifié pour pouvoir changer le type de rectangle pour différencier les différents bonbons. Ainsi, cette classe permet d'afficher à l'écran un rectangle. De plus, tout les objets héritant de Candy l'utilisent.

- Rectangle(Point,int,int,Fl_Color,Fl_Color)
- Fl_Color getFillColor()
- Fl_Color getFillColor()
- Fl_Color getFrameColor()
- void setCenter(Point)
- void setWidth(int)
- void setHeight(int newh)
- int getWidth()
- int getHeight()
- Point getCenter()
- void setCode(Fl_Color)
- void draw()
- void setFillColor(Fl_Color)
- void setFrameColor(Fl_Color)
- bool contains(Point)

3.2 Circle

Classe recopiée du TP utilisée seulement par les ingrédients permettant d'afficher à l'écran un cercle.

- Circle(Point center, int r,Fl_Color frameColor = FL_BLACK,Fl _Color fillColor = FL_WHITE)
- void draw()
- void setFillColor(Fl_Color newFillColor)
- Fl_Color getFillColor()
- void setFrameColor(Fl_Color)
- bool contains(Point)
- Point getCenter() Introduction : Comme la classe Rectangle, la classe Circle est utilisé pour dessiner les fruits qui héritent de cette classe.

3.3 MainWindow

Classe qui va construire la fenêtre principale du jeux.

- void draw() override
- int handle(int) override
- static void Timer_Cb(void*)

3.4 Intro Window

Classe principale du programme gérant le menu d'accueil permettant ainsi la sélection de niveau et l'ouverture d'une nouvelle fenêtre de jeu.

- static void create(Fl_Widget*,void*)
- void create_the_map()
- static void prev_map(Fl_Widget*,void*)
- void select_prev_map()
- static void next_map(Fl_Widget*,void*)
- void select_next_map()

— static void make_score_board
— static void start_game_candy(Fl_Widget*,void*)
— bool get_game()
— set_mc(MainWindow*)

3.5 Candy

Classe qui va construire les bonbons normaux dont tout les Special_candy héritent permettant de gérer les animations, l'affichage du rectangle et de vérifier les voisins du bonbon.
— bool verify_neighbours(shared_ptr<Item>)
— void start_pop_animation()
— void start_slide_animation(Point, bool=true)
— bool is_slide_complete()
— bool get_fruit()
— void set_fruit(bool)
— void set_neigh(shared_ptr<Item>)
— void draw()

3.6 Special_candy

Classe héritant de Candy servant de base à tout les special_candy
— bool is_special_candy()

3.7 Striped_candy

Classe représentant les bonbons rayés, elle permet d'avoir la direction des rayures en plus d'agir comme un Special_candy
— void set_direction(int) ;
— int get_direction() ;
— bool is_striped()

3.8 Wrapped$_c$*andy*

Classe représentant les bonbons emaballés.
— bool is_wrapped()

3.9 Bomb_candy

Classe représentant les bombes, elle permet de connaître la couleur à casser en plus d'agir comme un Special_candy
— void set_color_to_break(Fl_Color) ;
— Fl_Color get_color_to_break() ;
— bool is_bomb()

3.10 Wall

Classe représentant un murbool is_wall()

3.11 Score

Voir section **score**
— void set_score(int)
— int get_score()

— int get_best_score()

3.12 Score Board

Classe affichant le widget du meilleur scores sur le menu d'accueil arpès l'avoir lu dans son fichier texte
— Score_board(int, int , int , int)

3.13 Animation

Classe parent de toutes les animations permettant le templating en fonction de la forme rectangle ou de cercle
— bool is_complete()

3.14 Animation Slide

Classe implémentant le glissage d'un objet Form qui est templaté en cercle ou rectangle appelée par candy dans la méthode start_fall_animation()
— void draw()

3.15 Animation Pop

Classe implémentant l'animation de destruction d'un candy lorsque ce dernier fait appel à la méthode start_pop_animation()
— void draw()

3.16 Canvas

Classe dessinant les bonbons, interprétant les input de l'utilisateur en les transmettant au à GameManager et faisant aux indices.
— void make_board(string)
— void read_file(string)
— void draw()
— void mouseMove(Point)
— void mouseClick(Point)
— void keyPressed(int)
— void set_the_neighbours()
— bool is_board_moving() Introduction : Classe qui gérer toutes les functionnalité importants de jeux. Elle va permettre au player de changer les maps, elle dessine les bonbons et s'occupe des signaux reçu de mouse qui sont convertie dans la structure Point pour faire les mouvements.

3.17 GameManager

Classe qui vérifier la validité des mouvements, détruit les bonbons, effectue les combos, crée les bonbons spéciaux et fait tomber les bonbons.
— void set_up(shared_ptr<Candy>**,Score,Objective)
— bool break_candies(int ,int ,int ,int ,bool=false)
— void destroy_candies(int ,int ,int,int,bool=false)
— int fall_candies(int,int, bool)
— void fall_walls(int,int)
— void set_the_neighbours()

3.18 Hint

- void set_up(shared_ptr<Item>**,bool, bool)
- bool forshadowing_over_9000(int,int,Fl_Color,bool)
- void check_impossible(int,int,bool) Introduction : Classe qui fait que après quelques seconds où le player n'a pas joué aucun coup va faire 3 bonbons qui peux s'alligner et se détuire de pulser.

3.19 Objective

Classe qui va créer un objective aléatoire voir section **Objectif** pour plus de détails

- void setUp()
- void fruits_on(int)
- void inc_fruits()
- void dec_fruits()
- bool constant_check()
- void mv_done(int,int,Fl_Color)
- void objCompleted()
- int nr_breaks()
- int nr_tries() Introduction : Classe qui va creer un objective random que le joeur doit le terminer dans un nombre random du coup ça s'il n'y a pas des fruits sur la mape sinon l'objective serai toujours de faire ces fruits de tomber.

3.20 Item

Classe Parent de tout les objets se trouvant sur le plateau introduisant des méthodes polymorphiques à surcharger.

- virtual Point getCenter()
- virtual void setCenter(Point)
- virtual Fl_Color getFillColor()
- virtual void setFillColor(Fl_Color)
- virtual void setCode(Fl_Color)
- virtual void setFrameColor(Fl_Color)
- virtual bool contains(Point)
- virtual bool verify_neighbours(shared_ptr<Item>)
- virtual void update_frosted_neighbours()
- virtual void start_pop_animation()
- virtual void start_fall_animation(Point, bool=true)
- virtual bool is_fall_complete()
- virtual bool is_wall()
- virtual bool is_ingredient()
- virtual void set_fruit(bool)
- virtual void set_neigh(shared_ptr<Item>)
- virtual void draw()
- virtual bool has_frosting()
- virtual int get_layers_of_frosting()
- virtual void set_layers_of_frosting(int)
- virtual void set_box_type(Fl_Boxtype)
- virtual Fl_Boxtype get_box_type()
- virtual void set_direction(int)
- virtual int get_direction()
- virtual bool is_striped()
- virtual bool is_special_candy()

— virtual bool is_wrapped()
— virtual bool is_bomb()
— virtual void set_color_to_break(Fl_Color)
— virtual Fl_Color get_color_to_break()
— virtual int current_box()
— virtual void inc_box()

4 Logique du jeu

Nous commençons le jeu par sélectionner le premier niveau et nous attendons 10 seconds avant de jouer le première coup. Tout d'abord, le plateau casse tout les bonbons qui font un coup sans augmenter le score jusqu'à qu'il n'y ai plus aucune suite d'au moins 3 bonbons. Après 7 seconds où le joueur n'a pas joué un coup alors comme expliqué dans la partie **suggestions** il va y avoir une animation sur une suite d'au moins 3 bonbons qui peuvent être détruits qui sera calculée en passant par la méthode check_impossible() de la classe Hint. Ensuite, le mouvement des bonbons se fait par glissé déposé alors, on stocke l'Item ou l'on a commencé le mouvement dans current. Puis, on vérifie que l'Item se trouvant sous la souris lors du déposage se trouve dans le voisinnage de current. Si c'est le cas, une fois que les bonbons sont alignés dans break_candies() du GameManager alors leurs couleurs seront mise en noir et il va y avoir une animation de suppression. Après, les bonbons se trouvant au dessus des bonbons noirs (à remplacer) vont descendre avec la méhtode fall_candies() pour remplacer ces derniers en utilisant une animation de glissage Animation_fall via start_fall_animation(). Ainsi, s'il y a une animation de glissage en cours le joueur ne peut pas bouger de bonbons. Quand une partie commence un objectif aléatoire sera généré parmi les objectif possibles voir section **objectif**. Si l'objectif est accomplis alors le prochain niveau sera chargé s'il y a un sinon le même niveau sera rechargé avec un différent objectif. Pendant le jeu le joueur peut créer des bonbons spéciaux et faire des combos comme dans candy crush saga.

5 Score

Le score est calculé dans la méthode destroy_candies. Chaque bonbons répresente 3 points du coup si un mouvement détruits 3 bonbons alors le score deviendra 9. Le score sera affiché sur l'écran du jeux ensemble avec le score meilleur de l'utilisateur qui est à 0 au début car l'utilisateur n'a jamais joué au jeu.