Kanacrush

Noe Bourgeois, Morari Augustin-Constantin

22 August 2022

Table des matières

1 Introduction

2 Tâches

Indiquez les tâches que vous avez accomplies.

2.1 1. Fonctionnalité de base

2.2 4. Faire glisser le carré

2.3 10. Écran d'accueil

Lorsque vous démarrez le jeu, un écran d'accueil devrait apparaître pendant une seconde avant que l'écran principal du jeu ne soit chargé. L'écran d'accueil doit inclure votre nom.

2.4 14. Éditeur de tableau.

Ajoutez la possibilité de pouvoir modifier un niveau de manière interactive au lieu d'y jouer. Ce n'est évidemment pas dans le jeu original ! Vous devriez pouvoir enregistrer les modifications et ajouter de nouveaux niveaux.

3 Classes

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

3.1 Model

3.1.1 Mboard

Modèle de la board principal.

```
#include "src/parameters.h"
#include "src/1model/game/board/linked_list/Linked_list.h"
#include "src/1model/game/board/cell/Mcell.h"


class Board {
  std::string data_structure_;
  //cell_linked_list_array containers size
  static const int cells_containers_size_ = 9;
  //containers container size
  static const int cells_containers_container_size_ = 9;

  //crush
  std::array<std::shared_ptr<Linked_list>, cells_containers_size_>
  cell_linked_list_array;

  std::array<std::vector<std::array<int, 3>>, cells_containers_container_size_>
  crushable_cells_;
  bool crushable_ = false;
```

```
  //random access search & comparison
  std::array<std::array<std::shared_ptr<Cell>, cells_containers_container_size_>,
  cells_containers_size_>
  cell_array_array;

  std::string cells_value_distribution_;
  std::string cells_containers_head_orientation_;
  std::string state_;

 public:
  Board();
  static int get_cells_containers_size() ;
    static int get_cells_containers_container_size() ;
  bool crushFrom(int x, int y);
  void cellsInitRandom();
    void cellsInitValueAscent();

    //get_cells_
    std::shared_ptr<Linked_list> get_cells(int row);

  //new game
    void newGame();
    //crush
    bool crush();
    //swap
    void swap(int row1, int col1, int row2, int col2);

    // terminal print
    void print();

//  void cellsValueSetRandom();
  void setCellsValueDistribution(const std::string& value_distribution="random");
  void setCellsContainersHeadOrientation(const std::string&
  cells_containers_head_orientation="down");
  std::basic_string<char> getCellsContainersHeadOrientation();

  void replaceCellValue(int row
                        , int column);
  void rain(int row
            , int column);
  void setDataStructure(const std::string &data_structure="linked_list");
  void setCells(const std::string& mode="random",
                int value=1);

  void crushColumn(int column, std::vector<std::array<int, 3>>
  &origins_and_nodes_quantities);
  void crushable_cells_Print();
  void crushableCellsUpdate(int column
                            , int origin=-1 //-1 close column current serie or do nothing
                                );
  void crushableCellsAddNewSerie(int column
```

```
                                    , int origin
                                    );
  void searchCrushableCells();
  std::array<std::vector<std::array<int, 3>>, 9> getCrushableCells();
  void emptyCrushableCellsVectors();
  bool isCrushable() const;
  bool crushWhilePossible();
//  void crushWhilePossible();

};
```

3.1.2 Mcell

```
#include "srcCommon.h"

class Cell {
 protected:
  int value_;
  int type_;

 public:
    Cell();
    explicit Cell(int value);
    Cell(int value, int type);
    void setValue(int value);
    int getType() const;
    void setType(int type);
    void setRandom();
    void set(int value, int type);

  int getValue() const;
};
```

3.1.3 Linked_list

```
#include "src/1model/game/board/linked_list/node/Node.h"

class Linked_list {
//head and tail of the list
    std::shared_ptr<Node> head_;
    std::shared_ptr<Node> tail_;
    int size_{};

    //iterator_
    std::shared_ptr<Node> iterator_;

 public:
    Linked_list(const std::string& mode="random"
        , int value=1);
    ~Linked_list();
    void init(int size=9);
```

```
    void remove_all();
    void print();
  std::shared_ptr<Node> get_head();
  std::shared_ptr<Node> get_tail();

//iterator
  void begin();
  void end();
  void next();
  void previous();
//get_iterator
  std::shared_ptr<Node> get_iterator();
  bool is_begin();
  bool is_end();

  void remove(std::shared_ptr<Node> node);
  void set_next(const std::shared_ptr<Node>& node);
  void set_prev(const std::shared_ptr<Node>& node);
  void set_tail(std::shared_ptr<Node> node);
  void set_head(std::shared_ptr<Node> node);
  void add(const std::shared_ptr<Node>& node);
  void set_values(const std::string& mode="increment",
                  int head=0,
                  int tail=0);
  std::shared_ptr<Node> get_next(std::shared_ptr<Node> node);
  std::shared_ptr<Node> get_prev(std::shared_ptr<Node> node);
  std::shared_ptr<Node> get_node(int index);

  void move(std::shared_ptr<Node> origin,
            std::shared_ptr<Node> destination,
            int nodes_quantity,
            std::string mode="after");
  void setRandom(const std::shared_ptr<Node>& start,
                 int nodes_quantity,
                 int direction=-1);
  [[nodiscard]] int get_size() const;
  void debug();
//  void crush(std::vector<std::pair<int, int>> crush_vector);
  void crush(std::vector<std::array<int, 3>> crush_vector);
};
```

3.1.4 Node

```
#include "src/parameters.h"
#include "src/1model/game/board/cell/Mcell.h"

class Node : public Cell {
    std::shared_ptr<Cell> cell_;
    std::shared_ptr<Node> next_;
    std::shared_ptr<Node> prev_; //todo remove
```

```
public:
//     explicit Node(Node *cell);
  Node();
    ~Node();
    void set_next(std::shared_ptr<Node> node);
    void set_prev(std::shared_ptr<Node> node);
  std::shared_ptr<Cell> get_cell();
  std::shared_ptr<Node> get_next();
    std::shared_ptr<Node> get_prev();
    void print();
};
```

3.2 View

3.2.1 animation

Cette classe est supposée de faire les animations. Ce code est utile pour les classes Bounce et canvas qui l'utilisent.

```
#include "src/srcCommon.h"
#include "sketchable.h"
#include "srcCommon.h"
#include <math.h>
#include <time.h>
#include <array>
#include <memory>
#include <unistd.h>

#if __cplusplus >= 202002L
#include <numbers>
using std::numbers::pi;
#else
const double pi = 3.14159265358979323846264338327950288 4L;
#endif

using namespace std;

class Animation {
 protected:
  Animation();
  std::shared_ptr<Sketchable> to_animate_;
  bool animating_ = false;
  int duration_{};
  int time_{0};

  int direction_{};
  char direction_text_{} ;
//  Fl_Color color;
 public:
  virtual bool isComplete() =0;
  virtual void start(int direction, char directionText, Fl_Color newFillColor);
```

```
  virtual void animate(std::shared_ptr<Sketchable> to_animate, int duration, int direction,
                       Fl_Color newFillColor);
  void setState();
};
```

3.2.2 bounce

On utilise cette classe et la structure de Translation dans le canvas pour acceder faire tourner l'animation correctement.

```
#include "animation.h"
#include "ClickableCell.h"
#include <time.h>

class Bounce: public Animation {
  int bounce_height_;
    int duration;
    int bounceHeight;
    bool bouncing = false;
    int time{0};
    int direction = -1;
    char directionText;
    Fl_Color color;
//    Point currentTranslation();
 public:
  Bounce();
  Point currentTranslation();
  void draw();
  bool isComplete() override;
  void start(int direction, char directionText, Fl_Color newFillColor) override {
//    Animation::start();
    time = 0;
    bouncing = true;
    this->direction = direction;
    this->directionText = directionText;
    this->color = newFillColor;
  }
  void bounce(const shared_ptr<Sketchable>& sketchable, int duration, int bounceHeight, int
              Fl_Color newFillColor);
};

/*-------------------------------------------------------------
Translation Class
-------------------------------------------------------------*/
struct Translation {
    Translation(Point p, char directionText) {
        fl_push_matrix();
        if (directionText == 'V')
            fl_translate(p.x, p.y);
        else if (directionText == 'H')
            fl_translate(p.y, p.x);
    }
```

```
    ~Translation() {
        fl_pop_matrix();
    }
};
```

3.2.3 canvas

Cette classe est utilisée par la GameWindow pour dessiner le tableau de jeu

```
#include "constants.h"
#include "ClickableCell.h"
#include "Bounce.h"
#include "src/1model/game/board/Mboard.h"

class Canvas {
  int width_{};
    int height_{};
  std::shared_ptr<Board> board_;
  std::shared_ptr<Bounce> bounce_;
  //todo
  // std::shared_ptr<Translate> translate_;

  static const int cells_containers_size_=9;
   static const int cells_containers_container_size_=9;

  std::array<
      std::array<
          shared_ptr<ClickableCell>,cells_containers_size_
          > ,cells_containers_container_size_> cells_;
//  std::vector< ClickableCell > cells_;

  std::string heads_orientation_;
  Point mouse_click{}, mouse_release{}, mouse_hover{};
  Fl_Color cellColor1{}, cellColor2{};
  int n_du_carre_1_x_{},nDuCarre1Y{}, nDuCarre2X{}, nDuCarre2Y{};
  bool Test1{}, Test2{}, Test3{}, Test4{};
  bool AdjacentX{}, AdjacentY{};
 public:
  explicit Canvas(std::shared_ptr<Board> board);
  void mouseClick(Point mouseLoc);
  void keyPressed(int keyCode);
  void mouseRelease(Point mouseLoc);
//  void changeColors(int concCarre1, int concCarre2);
//  std::vector<ClickableCell> getCells();
  void update();
  void print(const string &head_orientation="down");
  void draw(const string& head_orientation="down");
  void redraw();
  void debug();
  vector<shared_ptr<ClickableCell>> getCrushablesFromDirectionXY(int current_x,
                                                                 int current_y,
                                                                 int direction_x,
```

```
                                                                    int direction_y,
                                                                    Fl_Color color,
                                                                    vector<shared_ptr<Clickab
  vector<shared_ptr<ClickableCell>> getCrushables(int x,
                                                  int y,
                                                  Fl_Color color,
                                                  char direction,
                                                  vector<shared_ptr<ClickableCell>> crusha
  void translateCell(int x, int y, int dx, int dy);
};
```

3.2.4 ClickableCell

Ceci est utilisé dans les classes bounce et canvas pour mettre les carrés du jeu sur le canvas et effectuer des animations sur eux.

```
#include "srcCommon.h"
#include "constants.h"
#include "Rectangle.h"

class ClickableCell :
        public Rectangle
 {
 public:
  // Constructor
    explicit ClickableCell(Point center
                           , int w
                           , int h
                           , Fl_Color frameColor
                           , Fl_Color fillColor);
  // Methods that draw and handle events
//  void draw() override;
//  void animationV1(Point mouseLoc);
//  void animationV2(Point mouseLoc);
//  void animationH1(Point mouseLoc);
//  void animationH2(Point mouseLoc);
//  void bounce(Point mouseLoc,
//              int dir,
//              char direction,
//              Fl_Color newFillColor);
//  Fl_Color getColor(Point mouseLoc);
//  Fl_Color getFillColor();
//  void setFillColor(Fl_Color newFillColor);
//  bool isComplete();
//  void setFillColorFrom(int colorIndex);
//  void print();
};
```

3.2.5 GameWindow

Est utilisé dans la classe principale main pour afficher l'ecran du jeu.

```
#include <unistd.h>

#include "constants.h"
#include "canvas.h"
#include "1model/game/game.h"

class GameWindow : public Fl_Window
    {
  Game game_ = Game();
  Canvas canvas_ = Canvas(game_.getBoard());
 public:

    GameWindow() :Fl_Window(
        3000,
        300,
        windowWidth,
        windowHeight,
        "Game") {
        Fl::add_timeout(1.0 / refreshPerSecond, Timer_CB, this);
//        add(canvas_);
//        set_resizable((Fl_Widget &) this);

//      draw();
//      canvas_.redraw();
//      show();

      if (DEBUG_GAME_WINDOW) {
        std::cout << "SLEEP" << std::endl;
      }
      unsigned int microsecond = 1000000;
//      usleep(3 * microsecond);//sleeps for 3 second


//      return Fl::run();
//      run();
//      Fl::run();
//      run();

    };

  void draw() override;
  void run();
  int handle(int event) override;

  static void Timer_CB(void *userdata) {
    if (DEBUG_GAME_WINDOW) {
      std::cout << "Timer_CB" << std::endl;
    }
    auto *gameWindow = static_cast<GameWindow *>(userdata);
    gameWindow->redraw();
    gameWindow->canvas_.redraw();
```

```
    Fl::repeat_timeout(1.0 / refreshPerSecond, Timer_CB, userdata);
  }
};
```

3.2.6 MenuWindow

Est utilisé dans la classe principale main pour afficher le menu du jeu.

```
#include "Window.h"
#include "FL_includes.h"
#include "Fl/Fl_Button.H"
#include "Fl/Fl_Box.H"
#include "srcCommon.h"
#include "1model/Menu.h"

class MenuWindow : public Fl_Window {
//class MenuWindow : public GUIWindow {

  Menu menu_ = Menu();
//  Fl_Box *title;
//buttons
  int buttons_quantity = 5;
  int button_width = windowWidth / 2;
    int button_height = windowHeight / buttons_quantity;
    Fl_Button *button_start_ = new Fl_Button(10, 0, button_width, button_height, "Start");
    Fl_Button *button_exit_ = new Fl_Button(10, button_height * 1, button_width, button_he
    Fl_Button *button_help_ = new Fl_Button(10, button_height *2, button_width, button_heig
    Fl_Button *button_about_ = new Fl_Button(10, button_height *3, button_width, button_he
    Fl_Button *button_settings_ = new Fl_Button(10, button_height *4, button_width, button_
public:
    MenuWindow() :Fl_Window(3000, 300, windowWidth, windowHeight, "Kana Crush") {
    Fl::add_timeout(1.0 / refreshPerSecond, Timer_CB, this);

    add(button_start_);
    add(button_exit_);
    add(button_help_);
    add(button_about_);
    add(button_settings_);
//    set_resizable((Fl_Widget &) this);

    }
    void run();


  static void Timer_CB(void *userdata) {
    MenuWindow *o = (MenuWindow*) userdata;
    o->redraw();
    Fl::repeat_timeout(1.0/refreshPerSecond, Timer_CB, userdata);
  }

  void draw() override {
    Fl_Window::draw();
```

```
  }

//set buttons
//    void set_button_start(Fl_Button *button_start) {
//        button_start_ = button_start;
//    }
    void set_button_exit(Fl_Button *button_exit) {
        button_exit_ = button_exit;
    }
    void set_button_help(Fl_Button *button_help) {
        button_help_ = button_help;
    }
    void set_button_about(Fl_Button *button_about) {
        button_about_ = button_about;
    }
    void set_button_settings(Fl_Button *button_settings) {
        button_settings_ = button_settings;
    }
//
//    void mouseClick(Point p)  {
//        if (button_start_->inside(p)) {
//            button_start_->do_callback();
//        }
//        if (button_exit_->inside(p.x, p.y)) {
//            button_exit_->do_callback();
//        }
//        if (button_help_->inside(p.x, p.y)) {
//            button_help_->do_callback();
//        }
//        if (button_about_->inside(p.x, p.y)) {
//            button_about_->do_callback();
//        }
//        if (button_settings_->inside(p.x, p.y)) {
//            button_settings_->do_callback();
//        }
//    }



  static void close(void *w);
};
```

3.2.7 Rectangle

Ceci définit les données de ClickableCell.

```
#include "sketchable.h"

class Rectangle: public Sketchable {

 public:
  Rectangle(Point center_
```

```
            , int w_1, int h_1, Fl_Color frame_color, Fl_Color fill_color);
//  void draw() override;
  void draw() override;
  void drawAt(int x
      , int y
  );

  //print
  void print(const std::string& attribute) const;

};
```

3.2.8 sketchable

Contient des méthodes importantes pour gérer les rectangles. Utilisée par animation et Rectangle.

```
#include "srcCommon.h"
#include "constants.h"

class Sketchable {
  friend class Animation;
protected:
  Point center_{};
  int width_{};
  int height_{};
  Fl_Color fill_color_{};
  Fl_Color frame_color_{};

 public:
    Sketchable(Point center, int w, int h,
               Fl_Color frameColor,
               Fl_Color fillColor);
    void init(Point center, int w, int h,
               Fl_Color frameColor,
               Fl_Color fillColor);

    virtual void draw() = 0;
//  bool contains(Point p) const =0;
//  virtual bool contains(Point p) const =0;
//  Point getCenter() const;
//  [[nodiscard]] virtual Fl_Color PointgetColor(Point mouseLoc) const =0;
  [[nodiscard]] Fl_Color getFillColor() const;
  void setFillColor(Fl_Color newFillColor);
//  virtual void setFillColor(Fl_Color newFillColor) =0;
  virtual ~Sketchable() = default;;
  [[nodiscard]] Fl_Color getFrameColor() const;
  void setFrameColor(Fl_Color newFrameColor);
  void setFillColorFrom(int colorIndex);
  void setHeight(int new_height);
  [[nodiscard]] int getWidth() const;
  [[nodiscard]] int getHeight() const;
  void setWidth(int new_width);
```

```
  Point getCenter();
  [[nodiscard]] bool contains(Point p) const;
};
```

3.2.9 WelcomeWindow

Est utilisé dans la classe principale main pour afficher la fenêtre de bienvenue.

```
#include "Window.h"
#include "FL_includes.h"
#include "Fl/Fl_Button.H"
#include "Fl/Fl_Box.H"
#include "srcCommon.h"
#include "src/1model/Welcome.h"

class WelcomeWindow : public Fl_Window {
  Welcome welcome_ = Welcome();
  //title
    Fl_Box *title_ = new Fl_Box(0
        , 0
        , windowWidth
        , windowHeight / 10
        , "Kana Crush"
        );
    //authors
//    std::char author1 = "Bourgeois Noe";
    std::string author1 = "Bourgeois Noe";
    std::string author2 = "Morari Augustin";
//    char *authors = new char[author1.length() + author2.length() + 1];
//    strcpy(authors, author1.c_str());

    Fl_Box *authors_ = new Fl_Box(0, windowHeight / 10, windowWidth, windowHeight / 10,
                                  "Created by Morari Augustin and Bourgeois Noe"
//        , "Created by : \n" + author2 + "\n & \n" + author1
        );
public:
    WelcomeWindow() :Fl_Window(3000, 300, windowWidth, windowHeight, "Kana Crush") {
//        Fl::add_timeout(1.0 / refreshPerSecond, Timer_CB, this);

        add(title_);
        add(authors_);
//        set_resizable((Fl_Widget &) this);

    };
//    ~WelcomeWindow();
    void run();

  static void close(void *w);
};
```

3.2.10 Window

Est utilisé dans MenuWindow et WelcomeWindow comme une fenêtre principale pour définir ces deux énoncés plutôt.

```
#include "constants.h"
#include "FL_includes.h"

class GUIWindow : public Fl_Window{
public:
  GUIWindow(int x, int y, int w, int h, const char *l) : Fl_Window(x, y, w, h, l) {
//     Fl::add_timeout(1.0 / refreshPerSecond
//                     ,Timer_CB(this)
//                     ,this
//                     );
  }
  void run();

  static void Timer_CB(void *userdata
//                          ,class *window
                            ) {
//     this *o = (this*) userdata;
//     o->redraw();
//     Fl::repeat_timeout(1.0/refreshPerSecond, Timer_CB(), userdata);
  }

};
```

4 Logique du jeu

Supposons que le jeu démarre, le premier niveau est sélectionné (si la tâche 11 est terminée), attente de 10 secondes, puis un coup est joué.

Décrivez en détail ce qui se passe dans votre code.

Control : l'endroit des clic et release est enregistré sous forme de coordonnée.

Model : Board échange les bonbons correspondants via void swap(int row1, int col1, int row2, int col2) ; puis cherche les cellules crushables via Board : :searchCrushableCells() si il n'y a pas de cellule crushable : les bonbons sont reéchangés , si il y en a, Board les crush puis recherche à nouveau, etc jusqu'à ce qu'il n'y en aie plus. View : Canvas met à jour sont plateau via update() puis s'affiche.via show

5 Modèle-Vue-Contrôleur

Nous avons utilisé ce modèle de conception car il a les avantages de modularité de l'orienté objet portés à l'échelle d'un projet multi-plateforme. Nos classes correspondent à ce modèle de conception de la manière suivante :

5.1 Model

Partie autonome du fonctionnement interne du programme.

5.2 Control-View

Les parties Control et View sont fusionnées pour une execution plus rapide. Elles se greffent au Model pour un changement d'interface possible en ne modifiant que le fichier main.cpp.

5.2.1 Control

5.2.2 View

6 Score

Nous calculons le score

score observer met à jour le score affiché

7 ressources

L'écriture du code a été accélérée à l'aide du plugin "Github Copilot". Certains bouts de code, modifiés, des sessions des TPs ont été utilisés dans notre programme. https://copilot.github.com/