Rapport Projet LDP Candy Crush

Louis Vanstappen, Francesco Nieri

January 2022

Contents

1 Introduction

Candy Crush is a particularly popular game that was unique to recreate. We all have a relative who is on level 1675 and this project gave us a chance to recreate the game to allow this relative of ours to potentially encounter an extra couple of hours of fun. We might not have gotten to the point where the game is actually fun and addictive, although it certainly looks and feels like Candy Crush.

In this project we managed to recreate the game with most of its core functionalities, such as special candies and icing.

2 Tasks

2.1 Accomplished tasks

2.1.1 Task 1: Base functionality

2.1.2 Task 2: Candy deletion animation

When a candy is deleted, an explosion sprite will replace the previous candy sprite for a short period of time. Afterwards the cell will be emptied to leave space for another candy.

2.1.3 Task 3: Walls

We have implemented static walls.

2.1.4 Task 4: Ability to drag candies

The player is able to use their mouse to drag a candy on the screen. This process is handled by first catching the cell located under the cursor upon the first click, which is then followed by an analysis of the cursor's direction.

2.1.5 Task 5: No more possible moves detection

After every move, the game will determine whether or not the player can still play and if not, the board is shuffled.

2.1.6 Task 6: Icing

We have implemented complete and half icing.

2.1.7 Task 7: Special candies

Our game implements special candies which act almost identically as the ones found in the original Candy Crush game. These candies can also interact with each other.

2.1.8 Task 8: Score

We have implemented a basic score calculation system which accounts for interactions, candy speciality and icing status.

2.1.9 Task 9: Best score

The best score is saved in a file. It is displayed when starting up a new game.

2.1.10 Task 10: Splash screen

Upon loading the game, the player is prompted with a *fun* splash screen displaying a descriptive image and the names of the almost talented creators of the game.

2.1.11 Task 11: Move suggestion

If the player has not interacted with the board for a couple of seconds, the game will suggest a move. This move suggestion is done in a thread to avoid blocking inputs from the user.

2.1.12 Task 13: Objective

2.2 Non accomplished tasks

Unfortunately, we were not able to finish tasks 12, 14, and 15.

3 Classes

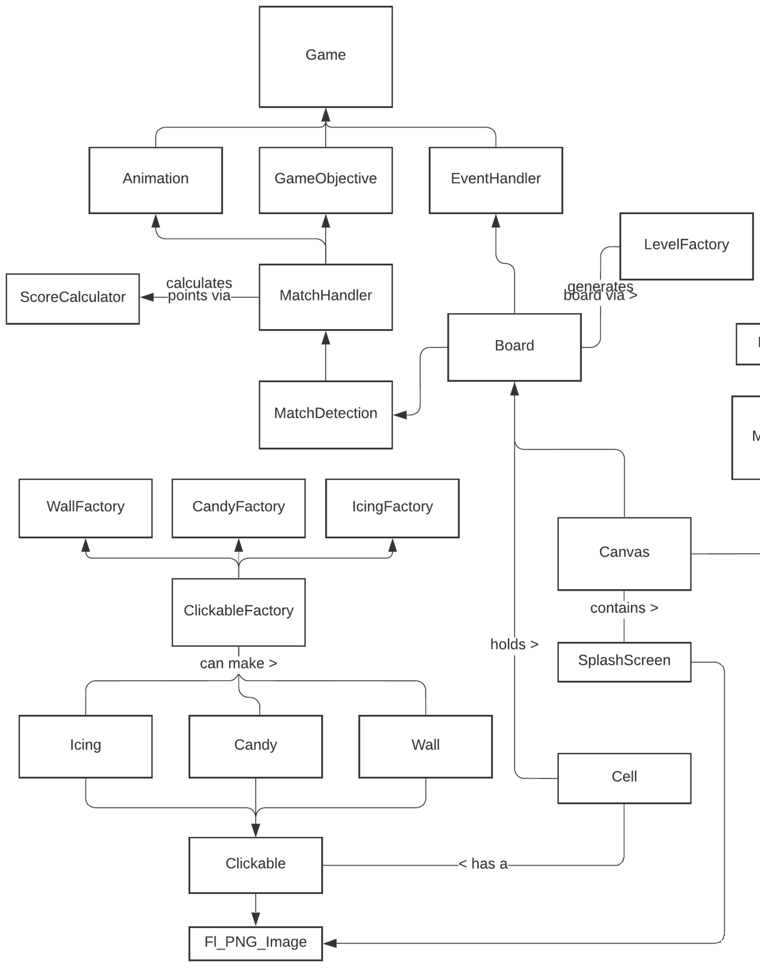

Figure 1: Class diagram

3.1 MainWindow

As the name suggests, this class handles the window and contains the main function. The class has a canvas object which contains everything related to the game. The *MainWindow* class also handles key and mouse event parsing to the canvas class.

3.2 Canvas

The *Canvas* class inherits from the *Board* class. It holds the splash screen and displays it at the launch of the program for a couple of seconds. It also holds a message box to display messages to the player.

3.3 SplashScreen

The splashscreen class represents the image that is displayed upon startup. Like the Clickable class, it inherits from Fl_PNG_Image.

3.4 Game

The *Game* class is an abstract class containing mostly virtual methods that are then defined in child classes as well as the *CellsVector*, which is the most important variable of the game. It is used to allow for seamless calls between class methods. The class also holds some commonly used non virtual methods.

3.5 Board

The *Board* class is responsible for handling most actions on the cells vector other than animation, and match detection and handling. It handles method such as swapping cells, detecting whether or not a player can still play, and suggesting moves.

3.6 Match detection and handler

The *MatchDetection* class finds matches between 3 or more candies in the cells vector and checks for any potential special candies interaction. It then tells the *MatchHandler* class to handle the match or interaction accordingly. *MatchHandler* will then empty the cells it must empty and find the ones that must be dropped. These cells are then passed to the *Animation* class.

3.7 Animation

As stated, the *Animation class* receives cells that must be dropped. Its sole purpose is to cleanly animate their descent. It can also animate the diagonal cells descent if the *MatchHandler* class requests it.

3.8 Event handler

The small class *EventHandler* simply handles mouse events coming from the user, such as mouse click and mouse drag.

3.9 Cell, Candy, Icing, and Wall

Candy, Icing and Wall all derive from the Clickable class. They are essentially the objects displayed in the board that can be clicked by an user. Clicking on an icing or wall doesn't allow them to be moved. The cell is the one that contains these *Clickables*. We used a cell to logically represent the space where the clickable object will reside since it is easier to deal with an intermediate object. The user doesn't know that the cells are the ones containing the *Clickables* and that the candies are the one being swapped from a cell to another. They just have to worry about clicking the candies and swapping them.

3.10 Factories

We have implemented five factories: three for generating *Clickables* derived classes, one that groups them all, and one that generates a level starting from a .txt file.

3.10.1 Clickable Factories

Clickable factories have the role of generating a *Clickable*. The base factories are:

- *CandyFactory*
- *IcingFactory*
- *WallFactory*

They will first generate a path to the images based on the attributes these classes may have, specialty and color for candies and status for icings. Afterwards, generation methods take the paths and construct the objects requested. The construction of an image is done by Fltk using the constructor *Fl_PNG_Image*, which is the class that *Clickable* inherits from. *ClickableFactory* groups these three classes together to ease usage.

3.10.2 Level Factory

LevelFactory generates the vector of vector of cells *CellsVector*, which is usable by a Board class to display the level, from a path to a file. The structure of a level file is the following: each line is composed of 10 numbers that are not separated by any character. Each line is separated by a newline character. To each number corresponds a *Clickable*:

- 0: Candy
- 1 and 2: simple and double Icing
- 3: Wall

Level loading is done upon startup because there is just one level.

3.11 Game Objective

We have defined 3 main game objectives:

```
enum Objective {
    CLEAR_CANDIES,
    CLEAR_ICING,
    POINTS,
};
```

For the first objective, the user has to clear a set number of candies with a random speciality.

For the second one, they have to clear a random number of icings. For the icing to be counted towards the objective, it has to have a HALF_ICING status. If this objective is chosen, the board will generate a defined number of icings randomly in the level.

Lastly, the user has to get a random number of point for the level to be done.

3.12 Score calculation

Score calculation is done once a candy or icing is cleared or once an interaction between candy occurs. For interactions, we have defined an enum which contains all the possible interactions for special candies.

```
enum Interaction {
    NONE_MULTICOLOR = 5,
    DOUBLE_STRIPED = 10,
    STRIPED_WRAPPED = 15,
    STRIPED_MULTICOLOR = 30,
    DOUBLE_WRAPPED = 25,
    WRAPPED_MULTICOLOR = 45,
    DOUBLE_MULTICOLOR = 100
};
```

As we only need to use the interactions for calculating score, we can just define the score for each interaction in the enum. The score calculator will just have to return the int corresponding to the current interaction. As for the candy and icing score, we have defined their score based on their speciality or status.

- Normal Candies are worth 1 point.
- Striped Candies are worth 5 points.
- Wrapped Candies are worth 15 points.
- Multi-color Candies are worth 45 points.
- Full icing is worth 10 points and half icing is worth 5 points.

3.13 Enums

We have defined 5 enums:

```
    enum CandySpeciality {
        NONE,
        STRIPED_HORIZONTAL,
        STRIPED_VERTICAL,
        BOMB,
        MULTICOLOR,
        SPECIALITY_COUNT
    };
```

This enum represents the various specialties a candy can have. They are the same as the real game.

```
    enum class Color {
        RED,
        ORANGE,
        YELLOW,
        GREEN,
        BLUE,
        PURPLE,
        MULTICOLOR,
        NONE
    };
```

This enum represents the different colors a candy can have. NONE indicates that the candy is empty.

```

enum IcingStatus {
    HALF_ICING,
    COMPLETE_ICING,
    EMPTY
};
```

This enum represents the different states an icing will be in. An empty status represents an empty icing.

The objective enum is described in the **Game Objective** subsection. The interaction enum is described in the **Score Calculation** subsection.

4 Game logic

When a player asks to swap two candies, the *Clickables* in the cells will be swapped if the candies are next to each other. If both candies are *special*, the interaction will be carried out by the *MatchHandler*. It will carry out the correct behaviour for the interaction, then a check for any match between candies will occur. If a match occurs, the candies will be cleared and if the match corresponds to a *special candy match*, a special candy will be generated. Candies will then drop down until no more matches have been found.

The Game will check if any empty cells remain. Empty cells are cells that contain an empty *Clickable*. A wall can never be empty because it is a fixed object. If there is any remaining empty cell, the game will try to drop candies diagonally. If an empty cell is below an icing or a wall and can't be reached from its upper diagonal, it won't be filled.

After each successful or unsuccessful move, a threaded function is launched and waits 4 seconds. Afterwards, it checks whether or not the player interacted with the board. If so, it returns, otherwise, it will suggest the first available move to the user. The suggestion is displayed by highlighting in red the background of the cells that can be swapped. The first move is found by trying to swap the cells without any animation and seeing if a match occurs. This process is also used to detect whether or not the player can still play and if not, the board will shuffle itself.

4.1 Candy matches

Candy matches are as defined in the real Candy Crush game. Whenever a candy swap occurs, the game will check whether or not three or more candies are side by side vertically or horizontally. It will then break the candies that are matching. If the match size is greater than 3 or if it forms a T, L, or + shape, it will create a stripped, a multi-color, or a wrapped candy.

4.2 Gravity

When candies break, the board must adapt properly and animate the candies dropping down. To achieve this, the game first finds empty cells and then searches for any potential cells that could take their spot. If a cell is found, it will either fall vertically along with the cells located over it, or it will fall diagonally with the cells located in its upper diagonal. The way this is formulated implies that the cells are moving in the *CellVector*, however this is not the case. Rather, the cells are sending their candy object to each other and never move in the matrix vector. After every drops, the game will check for any potential new matches before proceeding to the next drop.

4.3 Special candies interaction

Whenever a player makes two cells interact with one another with a swap attempt, the game will check whether or not both cells are special, and if so it will act accordingly. For example, if a cell is striped and the other is wrapped, it will clear a row and column with a height and width of three surrounding the cell just like in the real game.

4.4 Move suggestion

If the player has not moved for a few seconds, a function to search for the first match possible is launched. When the first match is found, the cells to be swapped will be highlighted. The player is free to decide whether or not they wish to swap them.

4.5 Key bindings

A player can quit by pressing q on their keyboard and can reset the current level by pressing r.

5 Conclusion

In conclusion, we enjoyed working on this project because it taught us a lot and pushed us to tailor our C++ programming skills as well as our group working skills, which will certainly prove useful in a nearby and farther future. It was a particularly arduous one that required us to properly use the knowledge gained not just in the *Programming Languages II* course, but also in the *Operating Systems* and *Analysis and Methods* courses. This proved to be challenging, therefore we are satisfied to have succeeded in accomplishing a working game.

We are a bit disappointed to not have been able to go all the way to where we would have liked to go and complete levels selection and creation. This is mainly due to a lack of time since we also had other projects and exams to prepare.

6 Headers

Animation.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*               515205          515694
* Header: Animation.h
* Date: 13/01/2022
*/

#ifndef __ANIMATION_H
#define __ANIMATION_H

#include "Game.h"
#include "Cell.h"

class Animation : virtual public Game {
public:

    /**
     * @brief Move down cells after explosion
     * and generate new ones on top
     *
     * @param cellsToReplace
     */
    void moveCellsDown(vector <vector<int>> cellsToReplace)
    override;

    /**
     * @brief Move cells diagonally when needed
     *
     * @param diagonalCells
     * @param lr
     */
    void moveCellsDiagonally(const vector<vector<int>> &
    diagonalCells, int lr) override;

    /**
     * @brief Animate cell destruction with png
     *
     * @param cell
     */
    static void destroyObject(Cell *cell);
};

#endif
```

headers/Animation.h

Board.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: Board.h
* Date: 13/01/2022
*/

#ifndef __BOARD_H
#define __BOARD_H


#include <thread>
#include <mutex>

#include "EventHandler.h"
#include "MatchDetection.h"
#include "LevelFactory.h"
#include "GameObjective.h"
#include "Enums/Interaction.h"

using std::thread;

class Board : public EventHandler, public MatchDetection {
    int cellSize;
    int numberOfCells;
    Cell *selectedCell = nullptr;
    Point selectedCellCenter{0, 0};
    Point selectedCellPosition{0, 0};
    Cell *toSwapCell = nullptr;
    Point toSwapCellCenter{0, 0};
    Point toSwapCellPosition{0, 0};

    vector<Cell *> suggestedCells{};
    bool runSuggestionThread = false;
public:
    Board(int cellSize, int margin, int numberOfCells);

    /**
     * @brief Generate new candies upon pressing r
     */
    void reset() override;

    //Getters

    //Get cell from fltk point
    Cell *cellAt(Point p) override;
    //Get cell from i,j point
    Cell *getCellFromPosition(Point p);
    //Get i,j of point
    Point getPositionOfCell(Point p) override;

    //Setters
    void setSelectedCell(Cell *newCell) override;
    void setSelectedCellPosition(Point p) override;
    void setSwapCell(Cell *newCell) override;
    void setSwapCellPosition(Point p) override;

    /*
    * @brief Change CellsVector content from one clickable
    * to another
    */
    void setCellAt(CandySpeciality speciality, Color newColor, int
    i, int j) override;
    /*
    * @brief Swap cells in vector based on position
    */
    void swapCells(Cell *swapCell, Point swapCellPosition) override
    ;

    /*
    * @brief Swap cells without animation
    */
    void swapCellsNoAnim(Cell *cell1, Cell *cell2) override;

    bool checkIfShuffleIsNeeded();

    bool isMoveAllowed(Point cell1Position, Point cell2Position)
    override;

    //Mouse events handlers
    bool handleBoardContains(Point p) override;
    void handleBoardDrag(Point p1, Point p2) override;

    virtual void draw();

    void shuffle();
    /**
     * @brief Exchange two cells when swapping
     *
     * @param cell1
     * @param cell2
     */
    void exchangeCells(Cell *cell1, Cell *cell2);

    /**
     * @brief Create icing cells when objective
     * is of type CLEAR_ICING,
     */
    void matchIcingObjective();

    void initializeLevel();

    void handleSuggestionThread();

    void resetHighlighting();

};


#endif //__BOARD_H
```

headers/Board.h

Candy.h

```cpp
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*             515205          515694
* Header: Candy.h
* Date: 13/01/2022
*/

#ifndef __CANDY_H
#define __CANDY_H

#include <Fl/Fl_PNG_Image.H>
#include "Common.h"
#include "Enums/Color.h"
#include "Enums/CandySpeciality.h"
#include "Clickable.h"

class Candy :  public Clickable {
    const char *filename;
    Color color;
    CandySpeciality speciality;
public:
    Candy(const char *filename, Color color, CandySpeciality
    speciality=CandySpeciality::NONE);
    Candy(const Candy &c);
    ~Candy();
    [[nodiscard]] Color getColor() const;
    [[nodiscard]] CandySpeciality getSpeciality() const;

    bool isEmpty() const override;
    bool visitCandy() override {return true;};
};

#endif
```

headers/Candy.h

CandyFactory.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*              515205          515694
* Header: CandyFactory.h
* Date: 13/01/2022
*/

#ifndef _CANDYFACTORY_H
#define _CANDYFACTORY_H

#include <string>
#include <cstdlib>
#include <ctime>
#include <cstring>
#include <utility>

#include "Candy.h"
#include "Enums/CandySpeciality.h"
#include "Common.h"
#include "Enums/Color.h"

class CandyFactory {
private:

    static std::string generateSpecialityPath(CandySpeciality
    speciality);

    static std::string
    generateFullPath(CandySpeciality speciality, const std::string
    &colorPrefix, const std::string &specialityPath);

    static std::string generateImageName(Color color,
    CandySpeciality speciality);

public:
    static Color generateColor();

    static std::string generateColorPrefix(Color color);

    static Candy generateBoomCandy();

    static Candy generateCandy( CandySpeciality speciality);

    static Candy generateCandy(CandySpeciality speciality, Color
    color);

    static Candy generateEmptyCandy();

};

#endif
```

headers/CandyFactory.h

Canvas.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: Canvas.h
* Date: 13/01/2022
*/

#ifndef __CANVAS_H
#define __CANVAS_H

#include <vector>
#include <cmath>
#include <iostream>
#include <FL/fl_draw.H>

#include "Cell.h"
#include "Common.h"
#include "Board.h"
#include "Splashscreen.h"


class Canvas: public Board {
    bool drawAchievement = true;
    bool keyInputAllowed = true;
    bool showTopInfo = true;
    bool showSplashscreen = true;
    bool showBoard = true;
    unique_ptr<Splashscreen> splashscreen;
    int messageX = 175, messageY = 300;
public:
    Canvas(int cellSize, int margin, int numberOfCells);
    void draw() override;
    void keyPressed(int keyCode);


    /**
     * @brief Show splashScreen upon start
     */

    void showSplashScreen();

    /**
     * @brief Show current informations about:
     * Score, high score, moves left and current objective
     *
     */
    void drawCurrentObjective();

    /**
     * @brief Check if game is over
     *  If true, print message to screen
     */
    void checkLevelDone();

    void showShuffle();
    void shuffleCurrentLevel();
    void hideAll();
    void showAll();
    void showCenterMessage(std::string message) const;
    void showReset();



    /**
     * @brief Reset current level if game is over
     *  or upon press of r key
     */
    void resetCurrentLevel();

    //Setters
    bool isKeyInputAllowed() const;
    void setDrawAchievement(bool);
    void setKeyInputAllowed(bool);
    void setShowTopInfo(bool);
    void setShowBoard(bool);

    //Getters
    bool getShowBoard() const;
    bool getShowTopInfo() const;
    bool getShowSplashscreen() const;
};


#endif
```

headers/Canvas.h

Cell.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*               515205          515694
* Header: Cell.h
* Date: 13/01/2022
*/

#ifndef __CELL_H
#define __CELL_H

#include <FL/Fl.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_PNG_Image.H>
#include <FL/Fl_Box.H>
#include <utility>

#include "Common.h"
#include "Candy.h"
#include "Enums/Color.h"
#include "Clickable.h"
#include "Enums/IcingStatus.h"
#include "Icing.h"
#include "Wall.h"

using std::make_shared;

class Cell {
    Point center;
    int cellSize;
    shared_ptr<Clickable> cellClickable;
    int margin;
    bool drawBox = false;
    bool suggesting = false;
    Fl_Color highlightColor = FL_LIGHT3;
public:
    Cell(Point, int, Clickable*, int);
    Cell(const Cell &);

    //Cast arriving clickable from constructor
    void castClickable(shared_ptr<Clickable>);
    void castClickable(Clickable*);

    //Check if point is in cell
    bool contains(Point p) const;

    //Check if cell has type of clickable
    bool hasCandy();
    bool hasIcing();
    bool hasWall();

    //Setters
    void setCenter(Point);
    void setHighlighted(bool val);
    //Set clickable of cell
    void setClickable(const Clickable&);
    void setClickable(Clickable*);

    void setHighlightColor(Fl_Color);

    //Animation

    //Animate candy, only candy can be animated
    //Because icing and wall can't be animated
    void animateCandy(Cell*);
    //Animate gravity of cell
    void animateGravity(Point destination);
    void draw();
    void resetHighlight();
    void setSuggestion(bool suggestion);

    //Getters
    Color getColor();
    CandySpeciality getSpeciality();
    Point getCenter();
    bool isEmpty();
    Candy* getCandy();
    Clickable* getClickable();
    IcingStatus getStatus();

    //Return casted object from Clickable
    template <class cellObject>
    shared_ptr<cellObject> returnCasted();


};

#endif
```

headers/Cell.h

Clickable.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: Clickable.h
* Date: 13/01/2022
*/

#ifndef __CLICKABLE_H
#define __CLICKABLE_H

#include <Fl/Fl_PNG_Image.H>


class Clickable : public Fl_PNG_Image {
    private:
        const char* filename;
    public:
        Clickable(const char* filename);
        Clickable(const Clickable &c);
        virtual ~Clickable()=default;
        virtual bool isEmpty() const=0;

        /*
        * @brief Visitor methods to check if this method is being
        * called on a certain derived class
        * Corresponding method is reimplemented in derived classes
        */

        virtual bool visitCandy() {return false;};
        virtual bool visitIcing() {return false;};
        virtual bool visitWall() {return false;};

};

#endif
```

headers/Clickable.h

ClickableFactory.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: ClickableFactory.h
* Date: 13/01/2022
*/

#ifndef _CLICKABLEFACTORY_H
#define _CLICKABLEFACTORY_H

#include "Enums/CandySpeciality.h"
#include "Enums/Color.h"
#include "Candy.h"
#include "Wall.h"
#include "Icing.h"
#include "Enums/IcingStatus.h"
#include "CandyFactory.h"
#include "IcingFactory.h"
#include "WallFactory.h"


class ClickableFactory : public CandyFactory, public IcingFactory,
                         public WallFactory {
    public:

        static Candy makeCandy(CandySpeciality speciality);

        static Candy makeCandy(CandySpeciality speciality, Color
    color);

        static Candy makeEmptyCandy();

        static Icing makeIcing(IcingStatus status);

        static Wall makeWall();
};


#endif
```

headers/ClickableFactory.h

Common.h

```cpp
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: Common.h
* Date: 13/01/2022
*/


#ifndef _COMMON_H
#define _COMMON_H
#include <iostream>
#include <experimental/filesystem>


namespace fs = std::experimental::filesystem;


using std::shared_ptr; using std::make_shared;
using std::unique_ptr; using std::make_unique;
using std::dynamic_pointer_cast;
using std::vector;
using std::out_of_range;
using std::array;
using std::to_string;

struct Point {
    int x, y;

    bool operator ==(const Point&p) const{
        return (x == p.x && y == p.y);
    }

    bool operator !=(const Point&p) const{
        return (x != p.x || y != p.y);
    }
};

#define NUMBER_OF_COLORS 6
#define WORKING_DIRECTORY std::string(fs::current_path())
#define BEST_SCORE_FILE std::string("best_score/best_score.txt")


#endif
```

headers/Common.h

EventHandler.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: EventHandler.h
* Date: 13/01/2022
*/

#ifndef _EVENTHANDLER_H
#define _EVENTHANDLER_H

#include "Common.h"
#include "Game.h"


class EventHandler : virtual public Game {
private:
    Point firstPosition{};
public:
    void resetEvent();
    void setFirstPosition(Point p);
    void handleMouseDrag(Point p);

    void handleMouseEvent(Point p);

};


#endif //TEMP_CLASSES_EVENTHANDLER_H
```

headers/EventHandler.h

Game.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*               515205          515694
* Header: Game.h
* Date: 13/01/2022
*/

#ifndef __2122_GAME_H
#define __2122_GAME_H

#include <cstdio>
#include <fstream>
#include <iostream>
#include <memory>
#include <vector>
#include <algorithm>
#include "unistd.h"

#include "Clickable.h"
#include "Candy.h"
#include "Cell.h"
#include "CandyFactory.h"
#include "Enums/CandySpeciality.h"
#include "Common.h"
#include "ClickableFactory.h"
#include "Icing.h"
#include "Wall.h"
#include "Enums/Objective.h"


class Game{
protected:
    int margin;
    vector <vector<Cell>> CellsVertex;
    int score = 0;
    int hiScore;
    bool acceptInput = true;
    bool gameOver = false;
    bool shuffling = false;
    bool resetting = false;
    int movesLeft = 15;

public:

   //Setters
    void setMargin(int m);
    void setShuffling(bool);
    void setAcceptInput(bool newState);
    void resetScore() {score = 0;};
    void setMovesLeft(int);
    void setGameState(bool);
    void decreaseMovesLeft();
    void addToScore(int scoreToAdd);
    void setResetting(bool);
    //----//
    void gameWait(useconds_t time);

    //Save highscore in file
    void saveHighscore();

    //Cell highlighting
    void highlight(Point p);
    void unHighlightAll();



    //Getters
    void getInitialHighScore();

    //Check if cell is of clickable derived class
    bool isCandy(Cell *cell);
    bool isCandy(Cell cell);
    bool isIcing(Cell cell);

    //Get board cells
    vector <vector<Cell>> getCells();

    //Reset moves and state
    void resetGame();

    bool isShuffling();

    //Check if cell contains points
    bool contains(Point p);
    bool isInputAllowed();
    int getScore();
    int getMovesLeft();
    bool gameIsOver();
    bool isResetting();
    bool remainingEmptyCells();

    vector<vector<int>> findEmptyCells();

    //---//

    void createSpecialCandy(int, int, CandySpeciality);

    virtual void moveCellsDiagonally(const vector<vector<int>> &
    diagonalCells, int lr) = 0;
    virtual void moveCellsDown(vector <vector<int>>) = 0;

    virtual void setCellAt(CandySpeciality newSpeciality, Color
    newColor, int i, int j) = 0;

    virtual void reset() = 0;

    virtual bool handleBoardContains(Point p) = 0;

    virtual void handleBoardDrag(Point p1, Point p2) = 0;

    virtual bool isMoveAllowed(Point cell1Position, Point
    cell2Position) = 0;
```

```cpp

    virtual void swapCellsNoAnim(Cell *cell1, Cell *cell2) = 0;

    virtual bool checkMatches() = 0;

    virtual Point getPositionOfCell(Point p) = 0;

    virtual void setSelectedCell(Cell *) = 0;

    virtual Cell *cellAt(Point p) = 0;

    virtual void setSwapCell(Cell *) = 0;

    virtual void setSelectedCellPosition(Point p) = 0;

    virtual void swapCells(Cell *, Point) = 0;

    virtual void setSwapCellPosition(Point p) = 0;

    virtual void emptyCells(vector <vector<int>> cellsToEmpty) = 0;


};



#endif //PROJET_LDP_2122_GAME_H
```
headers/Game.h

GameObjective.h

```cpp
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*              515205          515694
* Header: GameObjective.h
* Date: 13/01/2022
*/

#ifndef _GAMEOBJECTIVE_H
#define _GAMEOBJECTIVE_H

#include "Game.h"
#include <Fl/Fl_PNG_Image.H>


class GameObjective :virtual public Game {
    int initialNumberToAchieve;
    int currentNumberToAchieve;
    Objective currentObjective;
    unique_ptr<Candy> candyToRemove = nullptr;
    bool isComplete = false;

        //Create currentObjective
        void createObjectiveType();

        void makeNumberToAchieve();

        //Make number to achieve based on speciality
        void makeCandyNoToAchieve(CandySpeciality);

        void setNumberToAchieve(int min, int max);

        //Create candyToRemove
        Candy makeObjectiveCandy(CandySpeciality, Color);

        std::string getSpecialityString(CandySpeciality);
        std::string buildCandyString();
    public:
        //Create new objective
        void objectiveInit();
        //Decrease objective based on cell sent
        void decreaseObjective(Cell);
        //Helper functions
        void decreaseCandyObjective(Cell cell);
        void decreaseIcingObjective();
        void decreasePointObjective();

        //Getters
        bool isObjective(Objective);
        int getRemainingObjective();
        std::string getObjectiveString();
        Objective getObjective();
        bool objectiveIsComplete();

        //Check if objective is over and update
        void updateObjective();


};

#endif
```

headers/GameObjective.h

Icing.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*               515205          515694
* Header: Icing.h
* Date: 13/01/2022
*/

#ifndef __ICING_H
#define __ICING_H

#include <Fl/Fl_PNG_Image.H>
#include "Common.h"
#include "Enums/IcingStatus.h"
#include "Clickable.h"

class Icing : public Clickable {
    private:
        const char* filename;
        IcingStatus status;

    public:
        ~Icing()=default;
        Icing(const char* filename, IcingStatus status);
        Icing(const Icing &i);
        IcingStatus getStatus() const;
        bool isEmpty() const override;
        bool visitIcing() override {return true;};
};

#endif
```

headers/Icing.h

IcingFactory.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header:
* Date: 13/01/2022
*/

#ifndef __ICINGFACTORY_H
#define __ICINGFACTORY_H


#include "Enums/IcingStatus.h"
#include "Icing.h"
#include <cstdlib>
#include <cstring>


class IcingFactory {
    private:

        static std::string
            generateIcingPath(IcingStatus status);

    public:
        static Icing generateIcing(IcingStatus status);
};
#endif
```

headers/IcingFactory.h

LevelFactory.h

```
/* LDP INFO–F–202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: LevelFactory.h
* Date: 13/01/2022
*/

#ifndef _LEVELFACTORY_H
#define _LEVELFACTORY_H


#include "Candy.h"
#include "ClickableFactory.h"
#include "Enums/IcingStatus.h"
#include "Cell.h"
#include <cstdlib>
#include <iostream>
#include <fstream>
#include <string>
#include <vector>
#include <math.h>

class LevelFactory {
    private:
         static Cell buildCell(int id, Point center, int cellSize,
    int margin);
    public:
        static
        vector<vector<Cell>> buildCellVector(std::string filename,
    int margin, int cellSize, int noOfCells);
};

#endif
```

headers/LevelFactory.h

MainWindow.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*              515205          515694
* Header: MainWindow.h
* Date: 13/01/2022
*/

#ifndef _MAINWINDOW_H
#define _MAINWINDOW_H

#include <FL/Fl.H>
#include <FL/Fl_Box.H>
#include <FL/Fl_Double_Window.H>
#include <FL/fl_draw.H>
#include <FL/Fl_Window.H>
#include <FL/Fl_Shared_Image.H>
#include <FL/Fl_PNG_Image.H>
#include <time.h>
#include <chrono>
#include <iostream>
#include <random>
#include <string>
#include <vector>

#include "Canvas.h"

class MainWindow : public Fl_Window {
    const int windowWidth = 700;
    const int windowHeight = 700;
    const double refreshPerSecond = 60;

    Canvas canvas{50,60,100};
    public:
        MainWindow();
        void draw() override;
        int handle(int event);
        static void Timer_CB(void *userdata);
};

#endif
```

headers/MainWindow.h

MatchDetection.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: MatchDetection.h
* Date: 13/01/2022
*/

#ifndef _MATCHDETECTION_H
#define _MATCHDETECTION_H

#include <memory>
#include <vector>
#include <cmath>
#include <iostream>
#include <memory>

#include "MatchHandler.h"


class MatchDetection : public MatchHandler {
    Color currentCellColor;
    Bool handleMatch;
public:

    Color getCellColor(int x, int y);

    bool cellsColorMatch(int i, int j);

    void setHandleMatch(bool handleM);

    bool checkMatches();

    bool checkMatch(vector <array<int, 2>> match, int i, int j,
    CandySpeciality speciality);

    bool checkMatchFive(int i, int j);

    bool checkWrappedCandy(int i, int j);

    bool checkHorizontalMatchFour(int i, int j);

    bool checkVerticalMatchFour(int i, int j);

    bool checkMatchThree(int i, int j);

    bool
    checkForCandiesInteraction(Cell *firstCell, Point
    firstCellPosition, Cell *secondCell, Point secondCellPosition);

    bool canStillPlay();
};

#endif //_MATCHDETECTION_H
```

headers/MatchDetection.h

MatchHandler.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: MatchHandler.h
* Date: 13/01/2022
*/

#ifndef _MATCHHANDLER_H
#define _MATCHHANDLER_H

#include "Animation.h"
#include "ClickableFactory.h"
#include "ScoreCalculator.h"
#include "Enums/Interaction.h"
#include "GameObjective.h"

class MatchHandler : public Animation, public GameObjective {
    bool isInteracting = false;
    Color interactionColor = Color::NONE;
    CandySpeciality interactionSpeciality = CandySpeciality::NONE;

public:
    void handleCellsToReplace(vector<vector<int>> cellsToReplace);

    vector<vector<int>> getDiagonalCells(int col, int row, int lr);

    bool handleDiagonalCells();

    void handleGravity();

    void handleStripedHorizontal(int i, int j, vector<vector<int>>
    cellsToMove);

    void handleStripedVertical(int i, int j, vector<vector<int>>
    cellsToMove);

    void handleWrapped(int i, int j, vector<vector<int>>
    cellsToMove, int leftDownMargin, int rightUpMargin);

    bool wrappedInRange(int i, int j,
                        int partialVerticalOffset, int
    partialHorizontalOffset,
                        int verticalLimit, int horizontalLimit);

    void emptyCell(int i, int j);

    void emptyCells(vector<vector<int>> cellsToEmpty) override;

    void doubleStriped(Point firstCellPosition, Point
    secondCellPosition, const vector<vector<int>> &cellsToMove);

    void doubleWrapped(Point firstCellPosition, Point
    secondCellPosition, vector<vector<int>> cellsToMove);

    void multiColorSpecial(Point firstCellPosition, Point
    secondCellPosition,
                            CandySpeciality speciality, Color color)
    ;

    bool doubleSpecialCandyInteraction(Point firstCellPosition,
    Point secondCellPosition, Color firstCellColor,
                                       Color secondCellColor,
    CandySpeciality firstCellSpeciality,
                                       CandySpeciality
    secondCellSpeciality);

    void normalCandyAndMulticolorInteraction(Color colorToRemove,
    Point multicolorPosition);

    void doubleStripedCandyInteraction(Point firstCellPosition,
    Point secondCellPosition);

    void doubleMulticolorInteraction();

    void handleWrappedStriped(Point firstCellPosition, Point
    secondCellPosition,
                              vector<vector<int>> cellsToMove, bool
     isHorizontal);


    //Functions that use score calculator

    void
    setInteraction(bool interacting, Color color = Color::NONE,
    CandySpeciality speciality = CandySpeciality::NONE);

    void sendInteractionScore(Interaction);

    void sendScoreMulticolor(CandySpeciality);

    void sendIcingScore(IcingStatus);

    void sendSpecialityScore(CandySpeciality);

    void clearIcing(int i, int j);
};



#endif //TEMP_CLASSES_MATCHHANDLER_H
```

headers/MatchHandler.h

ScoreCalculator.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                515205          515694
* Header: ScoreCalculator.h
* Date: 13/01/2022
*/

#ifndef _SCORECALCULATOR_H
#define _SCORECALCULATOR_H

#include "Enums/CandySpeciality.h"
#include "Enums/IcingStatus.h"
#include "Enums/Interaction.h"

class ScoreCalculator {

    public:
        static int returnInteractionScore(Interaction);
        static int returnCandyScore(CandySpeciality);
        static int returnIcingScore(IcingStatus);

};

#endif
```

headers/ScoreCalculator.h

SplashScreen.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                 515205           515694
* Header: Splashscreen.h
* Date: 13/01/2022
*/

#ifndef __SPLASHSCREEN_H
#define __SPLASHSCREEN_H

#include <Fl/Fl_PNG_Image.H>
#include <Fl/Fl_Shared_Image.H>
#include <iostream>

class Splashscreen : public Fl_PNG_Image {
public:
    Splashscreen(const char *filename);
};

#endif
```

headers/SplashScreen.h

Wall.h

```
/* LDP INFO—F—202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                 515205          515694
* Header: Wall.h
* Date: 13/01/2022
*/

#ifndef __WALL_H
#define __WALL_H

#include <FL/Fl_PNG_Image.H>
#include "Common.h"
#include "Clickable.h"

class Wall : public Clickable {
    private:
        const char* filename;
    public:
        ~Wall()=default;
        Wall(const char* filename);
        Wall(const Wall &w);
        bool isEmpty() const override;
        bool visitCandy() override {return false;};
        bool visitIcing() override {return false;};
        bool visitWall() override {return true;};

};

#endif
```

headers/Wall.h

WallFactory.h

```
/* LDP INFO-F-202 First Session project.
* Authors: Louis Vanstappen, Francesco Nieri
*                 515205          515694
* Header: WallFactory.h
* Date: 13/01/2022
*/

#ifndef _WALLFACTORY_H
#define _WALLFACTORY_H


#include "Common.h"
#include <cstring>
#include "Wall.h"


class WallFactory {
    public:
        static Wall generateWall();
};
#endif
```

headers/WallFactory.h