EIDIA
École d'Ingénierie Digitale et d'Intelligence Artificielle

UEMF
الجامعة الأورومتوسطية بفاس
EURO-MEDITERRANEAN UNIVERSITY OF FES
UNIVERSITÉ EURO-MÉDITERRANÉENNE DE FÈS

Final Project

2048 Game

CHAHDI GHITA
TAHIFA ZAHRA

Introduction:

2048 is an easy and fun puzzle game. Even if you don't love numbers you will love this game. It is played on a 4x4 grid using the arrows or W, A, S, D keys alternatively. Every time you press a key - all tiles slide. Tiles with the same value that bump into one-another are merged. Although there might be an optimal strategy to play, there is always some level of chance. If you beat the game and would like to master it, try to finish with a smaller score. That would mean that you finished with less moves.

The objective of the project:

The project follows the Model-View-Controller pattern used for Graphical Interfaces.
board.cpp is the Model of the project. game.cpp is the Controller of the project.

main.cpp

```
#include <QApplication>
#include "game.hpp"

int main(int argc, char *argv[])
{
   QApplication a(argc, argv);

   Game *game;
   game = new Game();
   game->show();
   game->start();

   return a.exec();
}
```

game.cpp

This file is the Controller of the project. It contains the Game class which encapsulates the four functions start keyPressEvent updateBoard()clearText.
The start function is the initialization of the game and is a View function. keyPressEvent is the event handler (controller) that manages the pressing of the Up Down Right Left keys. updateBoard allows the updating of the board and clearText allows the updating of the board text.

```
#include "game.hpp"
#include "board.hpp"
QBrush brush0(QColor(202, 192, 180));
QBrush brush1(QColor(237, 228, 219));
QBrush brush2(QColor(235, 222, 199));
QBrush brush3(QColor(233, 178, 128));
QBrush brush4(QColor(245, 149, 99));
QBrush brush5(QColor(230, 132, 105));
QBrush brush6(QColor(228, 107, 76));
QBrush brush7(QColor(234, 207, 115));
QBrush brush8(QColor(237, 204, 98));
QBrush brush9(QColor(237, 200, 80));
QBrush brush10(QColor(237, 197, 63));
QBrush brush11(QColor(237, 194, 45));
QBrush brush12(QColor(29, 90, 254));
QPen pen(Qt::black);
QList<QBrush> brush = {
    brush0,
    brush1,
    brush2,
    brush3,
    brush4,
    brush5,
    brush6,
    brush7,
    brush8,
    brush9,
    brush10,
    brush11,
    brush12
};
```

```
QFont font1;
std::map<int, int> map;
Game::Game(QWidget *parent): QGraphicsView(parent) {
   scene = new QGraphicsScene(0, 0, 800, 600);
   setScene(scene);
}

void Game::start() {

   map.insert(std::pair<int, int>(0, 0));
   for (int i = 1; i < 13; i++) {
      map.insert(std::pair<int, int>((int)std::pow(2, i), i));
   }
   map.insert(std::pair<int, int>(0, 0));
   map.insert(std::pair<int, int>(2, 1));
   map.insert(std::pair<int, int>(4, 2));
   map.insert(std::pair<int, int>(8, 3));
   font1.setBold(true);
   font1.setPixelSize(15);
   board = new Board();

   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE ; j++) {
         QGraphicsRectItem* rect = scene->addRect(100*i, 100*j, 90, 90,
pen, brush0);
         qboard.append(rect);

         QGraphicsTextItem* text = scene->addText("",font1);
         text->setX(100*i + 45);
         text->setY(100*j + 45);
         qtext.append(text);
      }
   }
```

```
updateBoard();
}
void Game::keyPressEvent(QKeyEvent* event) {
   if (event->key() == Qt::Key_Up) {
      std::cout << "Hello" << std::endl;
      board->upUpdate();
   } else if (event->key() == Qt::Key_Down) {
      board->downUpdate();
 } else if (event->key() == Qt::Key_Right) {
      board->rightUpdate();   } else if (event->key() == Qt::Key_Left) {
      board->leftUpdate();}
   board->showBoard();
   std::cout << std::endl;
   updateBoard();}
void Game::clearText() {
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE ; j++) {
         qtext[j * SIZE + i]->setPlainText("");  }}
}
void Game::updateBoard() {
   clearText();
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE ; j++) {
         int value = board->tiles[i][j];
         // std::cout << "value" << value << "i" << map[value] <<std::endl;
         qboard[j * SIZE + i]->setBrush(brush[map[value]]);
         // qtext[j * SIZE + i]->setPlainText(QString::number(value));
         if (value != 0) {
            std::cout << value << std::endl;
            qtext[j * SIZE + i]->setPlainText(QString::number(value));
         }
      }
   }
}
```

game.hpp

```
#ifndef GAME_HPP
#define GAME_HPP

#include <QGraphicsView>
#include <QGraphicsScene>
#include <QWidget>
#include <QList>
#include <QGraphicsRectItem>
#include <iostream>
#include <QBrush>
#include <QPen>
#include <QPainter>
#include <QColor>
#include <list>
#include <map>
#include <cmath>
#include <QFont>
#include <QString>
#include "board.hpp"
class Game: public QGraphicsView {
public:
    Game(QWidget* parent=NULL);
    void start();
    QGraphicsScene* scene;

    Board* board;
    QList<QGraphicsRectItem*> qboard;
    QList<QGraphicsTextItem*> qtext;
    void keyPressEvent(QKeyEvent* event);
    void updateBoard();
    void clearText();
private:
};
#endif
```

board.cpp

This file is the model of the project, it contains all the functions useful to the mechanics of the game (the functions moveTileLeft, moveTileRight, moveTileUp, moveTileDown allowing the management of the movements of the Tiles in the game. The functions leftUpdate, rightUpdate, upUpdate and downUpdate which are the wrappers of the previous functions on the whole Board.

The file also contains helpers (addTile, sumTiles, moveTile, clearTile, tileIn, numTilesInBoard) which are used in the previous functions. A last function (showBoard) allows to log the board in the console in order to debug the game if necessary.

```
#include "board.hpp"
using namespace std;
Board::Board() {
    tiles = new int*[SIZE];
    for (int i = 0; i < SIZE; i++) {
        tiles[i] = new int[SIZE];
        for (int j = 0; j < SIZE; j++) {
            tiles[i][j] = 0;
        };
    };

    addTile();
    addTile();
};

void Board::clearTile(int i, int j) {
    tiles[i][j] = 0;
}

void Board::sumTiles(int i1, int j1, int i2, int j2) {
    tiles[i2][j2] *= 2;
    clearTile(i1, j1);
}

void Board::moveTile(int i1, int j1, int i2, int j2) {
    tiles[i2][j2] = tiles[i1][j1];
    clearTile(i1, j1);
}
int Board::moveTileLeft(int i, int j, int* walls) {

    int prev_value = tiles[i][j];
    int new_j = walls[i];
    int cur_value = tiles[i][new_j];
    int adj_j = new_j + 1;
```

```
 if (TileIn(i, new_j)) {
      walls[i] ++;
      if (prev_value == cur_value) {
         sumTiles(i, j, i, new_j);
         return 1;
      } else {
         if (adj_j == j) {
            return 0;
         } else {
            moveTile(i, j, i, adj_j);
            return 1;    }     }
   } else {
      moveTile(i, j, i, new_j);
      return 1;
   }
}
int Board::moveTileRight(int i, int j, int* walls) {
   int prev_value = tiles[i][j];
   int new_j = walls[i];
   int cur_value = tiles[i][new_j];
   int adj_j = new_j - 1;

   if (TileIn(i, new_j)) {
      walls[i] --;
      if (prev_value == cur_value) {
         sumTiles(i, j, i, new_j);
         return 1;
      } else {
         if (adj_j == j) {
            return 0;
         } else {
            moveTile(i, j, i, adj_j);
            return 1;     }
      }
   } else {
      moveTile(i, j, i, new_j);
      return 1;
   }
}
```

```
int Board::moveTileLeft(int i, int j, int* walls) {

    int prev_value = tiles[i][j];
    int new_j = walls[i];
    int cur_value = tiles[i][new_j];
    int adj_j = new_j + 1;

    if (TileIn(i, new_j)) {
        walls[i] ++;
        if (prev_value == cur_value) {
            sumTiles(i, j, i, new_j);
            return 1;
        } else {
            if (adj_j == j) {
                return 0;
            } else {
                moveTile(i, j, i, adj_j);
                return 1;
            }
        }
    } else {
        moveTile(i, j, i, new_j);
        return 1;
    }
}
int Board::moveTileRight(int i, int j, int* walls) {

    int prev_value = tiles[i][j];
    int new_j = walls[i];
    int cur_value = tiles[i][new_j];
    int adj_j = new_j - 1;

    if (TileIn(i, new_j)) {
        walls[i] --;
        if (prev_value == cur_value) {
            sumTiles(i, j, i, new_j);
            return 1;
        } else {
            if (adj_j == j) {
                return 0;
            } else {
                moveTile(i, j, i, adj_j);
                return 1;
            }
        }
```

```
 } else {
      moveTile(i, j, i, new_j);
      return 1;
    }
}
int Board::moveTileUp(int i, int j, int* walls) {
    int prev_value = tiles[i][j];
    int new_i = walls[j];
    int cur_value = tiles[new_i][j];
    int adj_i = new_i + 1;
    if (TileIn(new_i, j)) {
      walls[j] ++;
      if (prev_value == cur_value) {
        sumTiles(i, j, new_i, j);
        return 1;
      } else {
        if (adj_i == i) {
          return 0;
        } else {
          moveTile(i, j, adj_i, j);
          return 1;
        }
      }
    } else {
      moveTile(i, j, new_i, j);
      return 1;
    }
}
int Board::moveTileDown(int i, int j, int* walls) {
    int prev_value = tiles[i][j];
    int new_i = walls[j];
    int cur_value = tiles[new_i][j];
    int adj_i = new_i - 1;
    if (TileIn(new_i, j)) {
      walls[j] --;
      if (prev_value == cur_value) {
        sumTiles(i, j, new_i, j);
        return 1;
      } else {
        if (adj_i == i) {
          return 0;
        } else {
          moveTile(i, j, adj_i, j);
          return 1;
        }
      }
```

```
 } else {
      moveTile(i, j, new_i, j);
      return 1;
    }
}
void Board::leftUpdate() {
   int walls[SIZE];
   for (int k = 0; k < SIZE; k++) {
      walls[k] = 0;
   };
   int moved = false;
   for (int j = 1; j < SIZE; j++) {
      for (int i = 0; i < SIZE; i++) {
         if (TileIn(i, j)) {
            // cout << i << " " << j << endl;
            int cur_move = moveTileLeft(i, j, walls);
            moved = moved || cur_move;
         }
      }
   };
   if (moved == true) {
      addTile();
   }
}

void Board::rightUpdate() {

   int walls[SIZE];

   for (int k = 0; k < SIZE; k++) {
      walls[k] = SIZE - 1;
   };
 int moved = false;
   for (int j = SIZE - 2; j > -1; j--) {
      for (int i = 0; i < SIZE; i++) {
         if (TileIn(i, j)) {
            // cout << i << " " << j << endl;
            int cur_move = moveTileRight(i, j, walls);
            moved = moved || cur_move;
         }

      }
   };
   if (moved == true) {
      addTile();
   }
}
```

```
void Board::upUpdate() {
   int walls[SIZE];

   for (int k = 0; k < SIZE; k++) {
      walls[k] = 0;
   };
   int moved = false;
   for (int i = 1; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         if (TileIn(i, j)) {
            // cout << i << " " << j << endl;
            int cur_move = moveTileUp(i, j, walls);
            moved = moved || cur_move;
         }
      }
   };
   if (moved == true) {
      addTile();
   }
}

void Board::downUpdate() {

   int walls[SIZE];

   for (int k = 0; k < SIZE; k++) {
      walls[k] = SIZE - 1;
   };

   int moved = false;
   for (int i = SIZE - 2; i > -1; i--) {
      for (int j = 0; j < SIZE; j++) {
         if (TileIn(i, j)) {
            // cout << i << " " << j << endl;
            int cur_move = moveTileDown(i, j, walls);
            moved = moved || cur_move;
         }
      }
   };
  if (moved == true) {
      addTile();
   }
}

void Board::showBoard() const {

   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         cout << tiles[i][j];
         cout << "    ";
      };
```

```
 cout << endl;
   };
};

bool Board::TileIn(int i, int j) {
   if (tiles[i][j] != 0) {
      return true;
   }
   return false;
}

int Board::numTilesInBoard() {
   int k = 0;
   for (int i = 0; i < SIZE; i++) {
      for (int j = 0; j < SIZE; j++) {
         if (TileIn(i, j)) {
            k++;
         }
      };
   };
   return k;
}

int Board::addTile() {

   int a = (int) ( (float)rand() * 10 / RAND_MAX );

   // cout << a << endl;

   if (a > 0) {
      a = 2;
   } else {
      a = 4;
   }

   if (numTilesInBoard() == SIZE * SIZE) {
      return 0;
   }


   int ai = (int) ( (float)rand() * SIZE / RAND_MAX );
   int aj = (int) ( (float)rand() * SIZE / RAND_MAX );

   while (TileIn(ai, aj)) {
      ai = (int) ( (float)rand() * SIZE / RAND_MAX );
      aj = (int) ( (float)rand() * SIZE / RAND_MAX );
   }

   tiles[ai][aj] = a;

   return 1;
}
```

board.hpp

```
#ifndef BOARD_HPP
#define BOARD_HPP

#include <iostream>
#include <cstdlib>
#include <random>
#include <list>
#include <iterator>

#include <QGraphicsItem>
#include <QKeyEvent>
#include <QList>

#define SIZE 4

int init_board();

class Board {

  public:

    int size;
    int** tiles;

    Board();

    void showBoard() const;

    void sumTiles(int i1, int j1, int i2, int j2);
    void moveTile(int i1, int j1, int i2, int j2);
    void clearTile(int i, int j);

    int addTile();
    bool TileIn(int i, int j);
    int numTilesInBoard();

    int moveTileLeft(int i, int j, int* walls);
    int moveTileRight(int i, int j, int* walls);
    int moveTileUp(int i, int j, int* walls);
    int moveTileDown(int i, int j, int* walls);

    void leftUpdate();
    void rightUpdate();
    void upUpdate();
    void downUpdate();
};
#endif
```

Result:


#include <QList>
projet_tutus2
4
2
8
2
4
2
2
2
4
Sortie de l'a
projet_tut
2
2