LIFE GAME(OOP 팀플)

1조. 김남효(20163158) 박호연(20160769)

1. Contents

- What program we use for making Life Game program

- Our roles

- Our class files and their function.

- What we study new.

1. What program we use for making Life Game program.

- We use visual studio 2015 for running condition and we use Qt 5.9.2 for gui.

- Also we have to set stack size to 104857600.

1. Our Participation and Roles.

- We are two members team. So Nam Hyo did conditional part and Ho Yeon did file part.

So Nam Hyo made cell.cpp and lifegame.cpp, Ho Yeon made load.cpp and save() method. We also work together because we need each part’s code. Nam Hyo’s part need the input file’s data, and Ho Yeon’s part need to how to give data. So we worked almost together for working.

1. Now we will talk about Our codes.

First, we need cell(block) because printing of gui, clicked event, set color, and so on….

At first, this is our cell.h

|  |
| --- |
| enum cell\_type  {  DEAD,  LIVE  };  class Cell : public QFrame  {  Q\_OBJECT  private:  QPushButton\* button;  celltype type;  public slots:  void click();  public:  Cell(QWidget \*parent = NULL, int row = 1, int column = 1); //constructor  virtual ~Cell();  void setcolor();  cell\_type gettype() const;  void settype(cell\_type type);  private:  GlobalColor getcolortype();  }; |

We use enum to define Live and Dead easily. Also class Cell attributes QFrame for gui.

QPushButton is our cell.(We must recognize mouse click, so we made this cell pushbutton to recognize mouse click)

celltype type mean its cell’s type.

void click() method is mouse\_click event.

They are important methods, and don’t introduce get~, set~ method because it means get data, set data.

This is our cell.cpp

|  |
| --- |
| Cell::Cell(QWidget \*parent,int row, int column)  : QFrame(parent)  {  this->type = DEAD; //all is DEAD(initializing)  setFrameStyle(QFrame::Box);  this->button = new QPushButton(this); //make button  this->button->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Expanding);  this->button->setFixedSize(1280/column, 720/row);  QHBoxLayout \*layout = new QHBoxLayout(); //가로로저장하는 레이아웃을 만듬  layout->addWidget(this->button); //버튼을 레이아웃에 넣기  setLayout(layout);//  layout->setStretchFactor(this->button, 1); //버튼의 늘림인자를 1로 조정  layout->setContentsMargins(0, 0, 0, 0);//여백 0  layout->setSpacing(0);//space = 0  connect(this->button, SIGNAL(clicked()), this, SLOT(click())); // 버튼이 눌렸을 때 작업  set\_color();  } |

This is cell class constructor. We make button and its size is decided by its row counts and column counts. We decided cells layout size is 1280 \* 720 so we divide them. Also we have to recognize click\_event so this button is pushbutton. And make mouse\_click event and set color. This is constructor.

|  |
| --- |
| void Cell::click() //when button clicked, this event will occur  {  if (this->type == DEAD)  type = LIVE;  else  type = DEAD;  settype(type);  } |

This is out click method. It works when this button clicked. If button clicked, just change its type if its type was LIVE then DEAD, DEAD then LIVE. That’s all.

|  |
| --- |
| cell\_type Cell::gettype() const  {  return(this->type);  }  void Cell::settype(cell\_type type)  {  this->type = type;  set\_color();  } |

Gettype() method and settype method are made for lifegame.cpp. We have to control its cell type but type is in the private so we must use getmethod and setmethod. Settype() method put set\_color method because if sell decide its type then it must have color of its type.

|  |
| --- |
| GlobalColor Cell::get\_color()  {  switch (this->type)  {  default:  case DEAD:  return white;  case LIVE:  return black;  }  }  void Cell::set\_color() //click을 하면 type이 변한다. 그리고 그 타입이 변했으니 색깔도 그에 맞춰서 변해야 한다. 그 작업을 하는 것이다.  {  GlobalColor color = get\_color ();  this->button->setPalette(QPalette(color, color));  this->button->setAutoFillBackground(true);  this->button->setFlat(true);  } |

It is get\_color method and set\_color method. Get\_color method is get color if this type is DEAD then white, LIVE then black. It uses in set\_color method. Set\_color method is get\_color and make button’s color with get\_color. If this type is DEAD, so get\_color return white. And set\_color puts color(get\_color) in this button.

|  |
| --- |
| Cell::~Cell()  {  delete this->button;  } |

And this is destructor.

Now we show our lifegame.h and lifegame.cpp. They are run with lifegame’s conditions.

First lifegame.h

|  |
| --- |
| typedef pair<int, int> xy;  typedef tuple<int, int, bool> \_copy;  class LifeGame : public QWidget  {  Q\_OBJECT  private:  vector<vector<Cell\*> > cells;  QTimer \*timer;  unsigned int is\_live\_count = 0;  vector<xy> is\_live; // 살아있는 셀 저장  vector<xy> new\_live; // 살아있는 셀의 주변의 죽어있는 셀들을 저장  vector<\_copy> copy\_cells;//타입이 변할때 저장 3번째 값으로 인해 0이면 DIE 1이면 LIVE로 계산  public slots:  void run();  void next();  void lifegame();  void stop();  void save();  public:  LifeGame(QWidget \*parent = NULL, int row = 50, int column = 40);  virtual ~LifeGame();  void get\_cells\_copy(vector<vector<bool> > \*copy\_cells);  private:  QGridLayout\* setup(int row, int column);  QHBoxLayout\* setbutton();  void check\_live();  }; |

Method in public slots is use when events occur. get\_cells\_copy method is used in load class. It gets live cells from file and give them to lifegame class and lifegame’s cells variable push data in.

Setup method is push row\*column cell buttons. Setbutton method is push run, next, stop, save button.

Now we will show our lifegame.cpp

|  |
| --- |
| LifeGame::LifeGame(QWidget \*parent, int row, int column)  : QWidget(parent)  {  QGridLayout \*set = setup(row, column);  QHBoxLayout \*button = setbutton();  QVBoxLayout \*layout = new QVBoxLayout();//이 레이아웃에 다 저장  layout->addLayout(set);  layout->addLayout(button);  setLayout(layout);  }  LifeGame::~LifeGame()  {  } |

This is our lifegame’s construction and destructor. Constructor use setup method and setbutton method and push them all in layout.

|  |
| --- |
| QGridLayout\* LifeGame::setup(int rows, int column)  {  QGridLayout \*set = new QGridLayout();  set->setHorizontalSpacing(0);  set->setVerticalSpacing(0);//셀은 수직으로던 수평으로던 연속적으로 이어져야해서 이 작업을 하였다.  set->setSpacing(0);  set->setAlignment(Qt::AlignHCenter);    for (int i = 0; i < rows; i++)  {  vector<Cell\*> row;  this->cells.push\_back(row);  for (int j = 0; j < column; j++) //입력된 row만큼 col만큼 버튼을 넣는 작업을 할것이다.  {  Cell \*cell = new Cell(NULL, rows, column); // create new cell.  this->cells.at(i).push\_back(cell);  set->addWidget(cell, i, j); //add cell in (i,j)  set->setColumnStretch(j, 1);  }  set->setRowStretch(i, 1);  }  return set;  } |

This is setup method. We use gridlayout because we have to divide this layout with cells. We decided cells[rows][column]. And push cell in cells[rows][column]. Also use addWidget so push cell in this layout in(i,j).

|  |
| --- |
| QHBoxLayout\* LifeGame::setbutton()  {  QHBoxLayout \*button = new QHBoxLayout(); //create 수평적인 layout named button.  button->setAlignment(Qt::AlignHCenter);  QPushButton \*Run = new QPushButton("RUN"); //make pushbutton named "Run"  Run->setFixedSize(100, 25);  connect(Run, SIGNAL(clicked()), this, SLOT(run()));//if this button clicked run run function.  button->addWidget(Run);  QPushButton \*Next = new QPushButton("NEXT");  Next->setFixedSize(100, 25);  connect(Next, SIGNAL(clicked()), this, SLOT(next()));//if this clicked run next function.  button->addWidget(Next);  QPushButton \*Stop = new QPushButton("STOP");  Stop->setFixedSize(100, 25);  connect(Stop, SIGNAL(clicked()), this, SLOT(stop()));// if this clicked run stop function  button->addWidget(Stop);  QPushButton \*Save = new QPushButton("SAVE");  Save->setFixedSize(100, 25);  connect(Save, SIGNAL(clicked()), this, SLOT(save()));//if this clicked run save function  button->addWidget(Save);  return button;  } |

Setbutton method is push run, next, stop, save button and recognize it is clicked or not.

If Run button clicked then run method will be run. Also Stop button clicked then stop method will be run.

★ Run button must only one click. If we clicked two times before click stop. Then it is not stop.

|  |
| --- |
| void LifeGame::check\_live() //this function is check which cell is live.  {  while (!this->is\_live.empty()) {  this->is\_live.clear();  }  this->is\_live\_count = 0;  for (int i = 0; i < this->cells.size(); i++)  {  for (int j = 0; j < this->cells.at(i).size(); j++)  {  if (this->cells[i][j]->gettype() == LIVE) {  this->is\_live.push\_back(xy(i, j)); // if cell is livem then push its (x,y) to is\_live vector.  this->is\_live\_count++;  }  }  }  } |

check\_live method is used when run or next(lifegame method). It checked whole cells that type is LIVE. If type is LIVE then push it in is\_live vector with its (x, y). So if I want to check live\_cell then just look is\_live vector’s first and second data. Is\_live\_count is how many cells that type is LIVE.

|  |
| --- |
| void LifeGame::lifegame()//this function run with lifegame's condition.  {  check\_live(); //check only live cell  //int data\_count = is\_live.size();  int i = 0, j = 0, k = 0;  for (int l = 0; l<this->is\_live\_count; l++) { //check live cell's next is Live Or Dead  int neighbor = 0;  int a = this->is\_live[l].first; // a is x  int b = this->is\_live[l].second;// b is y  if (a >= 1 && b >= 1) {  if (this->cells[a - 1][b - 1]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a - 1, b - 1)); //near of this cell is DEAD cell then push it in new\_live vector.  i++;  }  }  if (a >= 1) {  if (this->cells[a - 1][b]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a - 1, b));  i++;  }  }  if (a >= 1 && b < this->cells.at(a).size() - 1) {  if (this->cells[a - 1][b + 1]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a - 1, b + 1));  i++;  }  }  if (b< this->cells.at(a).size() - 1) {  if (this->cells[a][b + 1]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a, b + 1));  i++;  }  }  if (b >= 1) {  if (this->cells[a][b - 1]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a, b - 1));  i++;  }  }  if (a < this->cells.size() - 1 && b >= 1) {  if (this->cells[a + 1][b - 1]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a + 1, b - 1));  i++;  }  }  if (a < this->cells.size() - 1) {  if (this->cells[a + 1][b]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a + 1, b));  i++;  }  }  if (a < this->cells.size() - 1 && b < this->cells.at(a).size() - 1) {  if (this->cells[a + 1][b + 1]->gettype() == LIVE)  neighbor += 1;  else {  this->new\_live.push\_back(xy(a + 1, b + 1));  i++;  }  }  if (neighbor < 2) { //if neighbor's count is less than 2 or more than 3 then this cell will be Dead.  this->copy\_cells.push\_back(\_copy(a, b, 0));// So I will push that its type will be change.  j++;  }  else if (neighbor > 3) {  this->copy\_cells.push\_back(\_copy(a, b, 0));  j++;  }  }  this->is\_live\_count = 0;//초기화  vector<xy>().swap(this->is\_live);  for (int m = 0; m<i; m++) {  int c = this->new\_live[m].first;  int d = this->new\_live[m].second;  int \_neighbor = 0;  if (c >= 1 && d >= 1)  if (this->cells[c - 1][d - 1]->gettype() == LIVE)  \_neighbor += 1;  if (c >= 1)  if (this->cells[c - 1][d]->gettype() == LIVE)  \_neighbor += 1;  if (c >= 1 && d < this->cells.at(c).size() - 1)  if (this->cells[c - 1][d + 1]->gettype() == LIVE)  \_neighbor += 1;  if (d < this->cells.at(c).size() - 1)  if (this->cells[c][d + 1]->gettype() == LIVE)  \_neighbor += 1;  if (d >= 1)  if (this->cells[c][d - 1]->gettype() == LIVE)  \_neighbor += 1;  if (c < this->cells.size() - 1 && d >= 1)  if (this->cells[c + 1][d - 1]->gettype() == LIVE)  \_neighbor += 1;  if (c < this->cells.size() - 1)  if (this->cells[c + 1][d]->gettype() == LIVE)  \_neighbor += 1;  if (c < this->cells.size() - 1 && d < this->cells.at(c).size() - 1)  if (this->cells[c + 1][d + 1]->gettype() == LIVE)  \_neighbor += 1;  if (\_neighbor == 3) { //if neighbor is 3 then its mean it can born. SO its type will be change so i push it in copy\_cells  this->copy\_cells.push\_back(\_copy(c, d, 1));  j++;  //this->is\_live.push\_back(xy(c, d));  //this->is\_live\_count++;  }  }  vector<xy>().swap(this->new\_live);  for (int o = 0; o < j; o++)  {  if (get<2>(this->copy\_cells[o]) == 0) // if last element is 0 then DEAD 1 is LIVE  this->cells[get<0>(this->copy\_cells[o])][get<1>(this->copy\_cells[o])]->settype(DEAD); //get x and y and set this type  else  this->cells[get<0>(this->copy\_cells[o])][get<1>(this->copy\_cells[o])]->settype(LIVE);  }  vector<\_copy>().swap(this->copy\_cells);  } |

Now, this is lifegame method and it run with condition. So this method is very important. First, it calls check\_live method because we must know where cell is LIVE. After check\_live method then is\_live\_count and is\_live must be set. So I will check this LIVE cell with lifegame’s condition.

Every cell get up to 8 neighbors, which are the cells type is LIVE that are horizontally, vertically, or diagonally adjacent.

1. If cell type is LIVE and neighbors are less than 2 then DIE
2. If cell type is LIVE and neighbors are more than 3 then DIE
3. If cell type is LIVE and neighbors are 2 or 3 then LIVE
4. If cell type is DEAD and neighbors are three then type is changed to LIVE

I just check only LIVE cell so I don’t need 4th rules not now. Also I don’t need 3rd because this cell is already LIVE so type doesn’t have to change. Just stay this cell. So I just check only 1st and 2nd rules in first loop. Check if neighbor cell is LIVE then neighbor += 1 if not, then push neighbor’s (x, y) in new\_live vector because DEAD cell can be LIVE cell when neighbors are 3. Neighbors are just LIVE cells counts so if DEAD cell change to LIVE cell, we just check only DEAD cell which near LIVE cell. So push it in new\_live vector and check condition 1st and 2nd. If neighbors are less than 2 or more than 3 then LIVE cell change to DEAD cell. So push this LIVE cell’s (x, y) in copy\_cells vector. And push 0 in copy\_cells. When this method check copy\_cells, just check third element so if 0 then settype DEAD if not, then LIVE. When first loop finish then check second loop. Second loop is check new\_live vector. This loop is almost same as first loop but different is it use only 3rd. rules. Only check this DEAD cell can be LIVE cell. If this DEAD cell can be LIVE cell then push its (x, y) in copy\_cells vector with 1 in 3rd element. Final loop is for copy\_cells. In final loop, we check copy\_cells’ 3rd  element is 1 or 0. If 0 then cells[x][y] type is DEAD, 1 then cells[x][y] type is LIVE.

This method reduce time because it only check row\*column, LIVE cells and their neighbor cells.

|  |
| --- |
| void LifeGame::run()  {  this->timer = new QTimer(this);  connect(this->timer, SIGNAL(timeout()), this, SLOT(lifegame()));  this->timer->start(100);//run after 100ms.  }  void LifeGame::next()  {  lifegame();  }  void LifeGame::stop()  {  this->timer->stop(); //stop this timer. so don't run anymore  } |

They run when their button clicked. Run method work until stop button clicked. We used Qtimer(it is almost same as sleep that delay 100ms) to run lifegame method. If we click run button, lifegame will run continue with delay 100ms. If we click stop button, timer is stop so don’t run lifegame anymore.

|  |
| --- |
| void LifeGame::get\_cells\_copy(vector<vector<bool> > \*copy\_cells)//use in load.cpp for check only live cell in file.  {  for (int i = 0; i < this->cells.size(); i++)  {  for (int j = 0; j < this->cells.at(i).size(); j++) //if this part is live cell.  {  if (copy\_cells->at(i)[j])  this->cells[i][j]->settype(LIVE); //settype LIVE  }  }  } |

get\_cells\_copy method is connected with load class. this method’s copy\_cells is get data from load class. So lifegame class’ cells get data on load class’ copy\_cells data. This procces is reading vector<vector<bool>> data, and translate this on cells, which is on this class.

|  |
| --- |
| void LifeGame::save() //function for saving this information to file  {  QString filename = QFileDialog::getSaveFileName(  this,  tr("save File"),  "C://",  "Text Files (\*.txt)"  );  if (filename == NULL)  {  return;  }  QFile \*fout = new QFile(filename);  char \*temp;  char line[] = { '\r','\n' };  fout->open(QIODevice::WriteOnly);  temp = (char\*)malloc(sizeof(char) \* 5);  ltoa(this->cells.size(), temp, 10);  fout->write(temp);  fout->write(line);  ltoa(this->cells[0].size(), temp, 10);  fout->write(temp);  fout->write(line);  free(temp);  temp = (char\*)malloc(sizeof(char) \* this->cells[0].size());  for (int i = 0; i < this->cells.size(); i++)  {  for (int k = 0; k < this->cells[i].size(); k++)  {  if (this->cells[i][k]->gettype() == LIVE)  {  fout->write("X");  }  else  {  fout->write("-");  }  }  fout->write(line);  }  fout->close();  QMessageBox::information(0, tr("SAVE"), "SAVE COMPLETE!");  return;  } |

This is save function, which is save button event function. We open new dialog, which is setting save file on which folder we want to save, and when we set this, we parsing our data and write our save file. All processes is over, information dialog is open that save is complete.

At least, we will show our load.h

|  |
| --- |
| #ifndef LOAD\_H  #define LOAD\_H\_  #include <QtWidgets>  #include <QtGui/QGuiApplication>  #include "lifegame.h"  #include <vector>  #include <QApplication>  using namespace std;  class Load :public QWidget  {  Q\_OBJECT  public:  Load();  virtual ~Load();  public slots:  bool importFile();//start to import  void defaultRun();//run default set  private:  vector<vector<bool> > \*copy\_cells;//data point which record input file data  QHBoxLayout\* setupButtonRow();//button set up function  bool loadflag;//flag that load is success of not  LifeGame\* program;//class point that call main game  void initiate();//show program(main game) to play  int row, column;//data which decide cell size  };  #endif |

In fact, we are already write all code except this, and write this part at least.

This code show that 2 button dialog, Import and New.

Import, when click this button, new dialog show that select txt file which we want to import.

New, when click this button, empty default map is open.

Public slots: functions are for button events.

Now we show our function on load.cpp

|  |
| --- |
| Load::Load()//initiate  {  QVBoxLayout \*layout = new QVBoxLayout();  QHBoxLayout \*buttonRow = setupButtonRow();  layout->addLayout(buttonRow);  setLayout(layout);  loadflag = false;  copy\_cells = NULL;  row = 50, column = 40;  }  Load::~Load()  {  } |

This is constructor and destructor, that setting buttons and default setting

In this case, default row number is 50, and column number is 40.

//

Follow funciotns are button events function.

|  |
| --- |
| void Load::defaultRun()  {  initiate();  return;  } |

This is function that call initiate function directly.

We divided this and initiate, for button event.

Follow function is importFile()

|  |
| --- |
| bool Load::importFile()  {  row = -1, column = -1;  int temprow = 0;  int i;  bool datainput\_flag = false;  QString filename = QFileDialog::getOpenFileName(  0,  tr("Open File"),  "C://",  "Text Files (\*.txt)"  );  //call dialog that can find file, to set file  if (filename == NULL)  {  row = 50, column = 40;  return false;  }  //Exception handling, if not set any file, return this function and go to first statement  //reading input data and doing Exception handling, when exception occured, show error message, return this function and go to first statement  QFile \*fin = new QFile(filename);  fin->open(QIODevice::ReadOnly); |

This part show that we open new dialog to set data file which we want to import.

filename: QString has file data.

If you not select ant file in this dialog, filename has NULL data, and this program return this function and come back first dialog

|  |
| --- |
| QTextStream read(fin);  do{  QString tmp = read.readLine();  int tmpSize = tmp.length();  for (i = 0; i < tmpSize; i++){  if (tmp[i] == '#' || tmp[i] == '\n'){  break;  }  else if (row == -1){  row = (int)tmp.toDouble();  datainput\_flag = false;  break;  }  else if (column == -1){  column = (int)tmp.toDouble();  copy\_cells = new std::vector<vector<bool> >(row);  for (int k = 0; k < row; k++){  copy\_cells->at(k).resize(column);  }  datainput\_flag = false;  break;  }  else if (tmp[i] == 'X'){  copy\_cells->at(temprow)[i] = true;  datainput\_flag = true;  }  else if (tmp[i] == '-'){  copy\_cells->at(temprow)[i] = false;  datainput\_flag = true;  }  else{//exception handling: other char input  QMessageBox::warning(0, tr("ERROR"), "FILE HAS CRIT ERROR!");  row = 50, column = 40;  return loadflag=false;  }  }  if (datainput\_flag){  if (copy\_cells->at(temprow).size() != column){  //exception handling: column error  QMessageBox::warning(0, tr("ERROR"), "FILE HAS CRIT ERROR!");  row = 50, column = 40;  return loadflag=false;  }  temprow++;  }  } while (!read.atEnd());  if (copy\_cells->size() != row){//exception handling: row error  QMessageBox::warning(0, tr("ERROR"), "FILE HAS CRIT ERROR!");  row = 50, column = 40;  return loadflag=false;  }  loadflag = true;  initiate();  return true;  } |

This part is parsing process.

First, we must skip if # is detected.

And we may get row data and column data first(when input file is not error file,) so we take row and column data first.

And we get two data, now setting map data and input on copy\_cells: vector<vector<bool>>.

And when every processes are not problem, setting loadflag true, and we going to initiate().

If any process occurred error, open error dialog that show which is error, and return this function and come back first dialog.

//

|  |
| --- |
| QHBoxLayout\* Load::setupButtonRow()  {  QHBoxLayout \*buttonRow = new QHBoxLayout(); // Creates horizontal box for buttons.  buttonRow->setAlignment(Qt::AlignHCenter);  QPushButton \*Import = new QPushButton("Import");  Import->setFixedSize(100, 25);  connect(Import, SIGNAL(clicked()), this, SLOT(importFile()));  buttonRow->addWidget(Import);  // Import button: get importdata  QPushButton \*New = new QPushButton("NEW");  New->setFixedSize(100, 25);  connect(New, SIGNAL(clicked()), this, SLOT(defaultRun()));  buttonRow->addWidget(New);  // NEW button: run default data  return buttonRow; // Returns bottom of layout.  } |

This function is setting buttons and put this buttons on button layer.

We setting buttons event on importFile() and defaultRun(), which are already introduce.

|  |
| --- |
| void Load::initiate()  {  if (loadflag)//check that reading is no problem  {  program = new LifeGame(NULL, row, column);  program->get\_cells\_copy(copy\_cells);  program->setFixedSize(1280, 760);  program->show();  // program->setFixedSize(1080, 900);  this->close();  }  else//run new mode  {  program = new LifeGame(NULL, row, column);  program->setFixedSize(1280, 760);  program->show();  //program->setFixedSize(1080, 900);  this->close();  }  } |

This function is initiating game, depending on setting.

If load flag is true, it means that we select Import. So this function initiate program(lifegame) and put copy\_cells data on program object. And show this program.

If copy\_cells is NULL, it means that we select NEW. So this function initiate program(lifegame) and And show this program.

Finally, this is how to run this program(main function)

|  |
| --- |
| #include <QtGui/QGuiApplication>  #include "load.h"  #include <stdexcept>  #include <string>  #include <fstream>  #include <sstream>  using namespace std;  int main(int argc, char \*argv[])  {  QApplication app(argc, argv);  Load \*start = new Load();  start->show();  return app.exec();    } |

In this project, we can know how to run GUI library on cpp, and how to use vector class efficiently.

But, in this project, we think that most important thing that we gain this project is divide working.

We select each class which we write code, and it make working efficiently. For example, NamHyo writes game logic, and main frame. And HoYeon writes file input, output, and initiating part. We learn that opp is useful to do team working.