2023 Spring OOP Assignment Report

과제 번호 : 5
학번 : 20190445
이름 : 허수범
Povis ID : sbh408

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

프로그램을 하다 보면 결정해야 할 세부 사항이 많은데, 이러한 세부 사항을 처리한 방법과 이유를 보고서에 쓰십시오.
독창적인 아이디어와 추가 기능은 보너스 점수를 받을 수 있으므로, 보고서에 명확히 기재하십시오.
문제가 여러 개인 경우, 각 문제별로 정리해서 작성합니다.

아래 문항별 설명은 편의를 위한 것으로, 삭제하고 제출한다.

1. 프로그램 개요

본 프로그램은 Qt creator를 이용해 2048게임을 구현한 것이다. 프로그램을 실행하면 4x4 grid의 랜덤한 위치에 2가 적혀 있고, grid를 방향키를 이용하여 이동하며 나타나는 랜덤한 숫자들을 합치거나 이동할 수 있다. 우측 상단의 Score: 옆에는 점수가 적혀있으며, 우측 하단에는 최대 세번 게임을 한 단계 되돌릴 수 있도록 해주는 restore버튼과 게임을 종료시키는 버튼인 exit버튼이 있다. Restore 버튼과 exit 버튼은 누르면 재차 결정을 확인하는 메세지가 뜨고, yes를 클릭하면 해당 기능이 진행된다.

프로그램 디렉토리에는 makefile의 역할을 하는 20190445.pro 파일과 main.cpp 파일이 있고, ui 디렉토리 내에는 게임의 전체적인 gui를 관리하는 gameui.h 와 gameui.cpp, block의 gui를 관리하는 block.h와 block.cpp가 있다. game directory 내에는 내부 로직을 구현하는 game.h, game.cpp, block.h, block.cpp board.h, board.cpp가 있다.

2. 프로그램의 구조 및 알고리즘

본 문제에서는 필수로 구현하도록 되어 있었던 class들만을 이용하여 프로그램을 구현하였다.

main.cpp:

GameUi 객체 gameUi를 생성하여 show하고, setFocus하여 입력이 제대로 되도록 하였으며, a.exec를 하여 프로그램이 실행되도록 하였다.

class Block

게임판 위의 블록이 정의되어 있는 클래스이다. Private 멤버 변수로 block에 할당되는 정수 값 blockVal이 있다. 생성자는 정수를 입력으로 받아 이를 blockVal에 대입하고, wrapping function getVal은 blockVal을 return하며, setVal은 정수를 입력 받아 blockVal을 해당 정수로 초기화한다.

class Board

block들로 이루어진 게임판을 관리하며, 게임판을 이동하는 로직이 구현되어 있다. 멤버 변수로는 16개의 Block 인스턴스의 포인터를 저장해 둔 Block* blocks[4][4], restore를 위해 move이전의 블록 값을 4 by 4 정수 array로 저장해 둔 int bufferBlock[4][4], 점수가 저장되는 int score, progress.txt에 저장하기 위해 만들어진 QString initState, buffer가 남아 있는지 없는지 확인하기 위해 선언한 bool noBuff가 있다.

생성자, 소멸자:

생성자는 아무 argument를 받지 않고, score를 0으로, noBuff를 true로 초기화한다. 이루 이중 for 문을 이용해 각 block을 동적 할당해주고, QRandomGenerator로 랜덤한 좌표 두개를 정해 해당되는 두 블럭을 2로 초기화해준 후, 좌표의 크기 관계에 따라 다르게 initState를 초기화해준다. 소멸자는 동적으로 할당된 Block들을 이중 for문을 활용해 delete해준다.

getBlock, setBlock:

두 함수 모두 x, y좌표를 입력으로 받고, setBlock은 Block 포인터를 추가로 받는다. 범위에서 벗어나지 않는다면, getBlock은 해당 좌표의 Block 포인터를 return하고, setBlock은 입력된 포인터로 해당 좌표의 포인터를 초기화해준다.

Move 함수들:

Move 함수들은 각각 left, right, up, down이 있는데, moveLeft만 설명하도록 하겠다. 나머지는 방향만 바뀐 것일 뿐이기 때문이다. 먼저 이동을 progess.txt에 기록하기 위해 QFile 객체와 QTextStream 객체를 초기화한다. 이후 한 row를 저장할 수 있는 int 배열을 선언하고, 움직인 것이 있었는지 없었는지 표시할 수 있는 bool moved를 선언하고 false로 초기화한다. 각 row를 iterate 하기 위해 4번의 for loop을 돌리도록 하고, for loop 내에서 for loop을 하나 더 돌려 bofferBlock을 초기화하고, row array를 초기화해준다. 이후 해당 row에서 인접한 두 블럭의 값이 같고 0이 아닐때, 각 원소의 값을 초기화해 주고, moved = true, noBuff = false가 되도록 하고,

progress.txt에 변경 사항을 저장하도록 하였고, 이후에는 0을 제외한 나머지 원소를 모두 왼쪽으로 밀고 나머지를 0으로 채우는 작업을 하도록 하였다. 마지막으로 getblock->setVal을 통해 row 배열에 있던 값들을 모두 각 block에 다시 대입하도록 하였다. 마지막으로 moved bool 변수를 return하도록 하였다.

bool moveable():

이중 for 문을 이용해 각 원소가 인접한 원소와 같은 값을 가졌는지 혹은 0의 값을 가졌는지 진단하여 해당 board가 moveable한지 판정하도록 하였다.

void genRandBlock():

각 원소를 이중 for loop로 0인지 아닌지 검사하고 int empCount를 increment해주고, iteration이 끝나면 empCount는 비어 있는 블럭의 갯수가 되기 때문에, 이를 bound로 하여 random number를 generate한 후, 다시 이중 for loop를 이용해 해당 index에 도달하여 해당 블럭의 값을 instruction에 주어진 대로 random하게 초기화해주고, progress.txt에 기록해 주었다. 모든 난수는 QRandomGenerator::global()->bounded()로 생성해 주었다. 2 혹은 4를 결정하는 난수생성기에서는 삼항연산자를 사용해 보았다.

getScore(), getInit():

wrapping funtion으로, 각각 score과 initState를 return한다.

Void Board::restore():

이중 for loop을 이용해 각 블럭에 접근해 미리 저장해 둔 bufferBlock의 값으로 초기화한다.

class Game

Game class는 Board의 포인터를 private 멤버로 가지는, 게임을 전체적으로 관리하고, 게임의 종료 여부를 판단하는 메소드가 구현되어 있다.

생성자와 소멸자:

생성자는 아무 argument를 받지 않는다. 새로운 board를 동적 할당하고, 위에서 구한 initState를 getInit() 메소드를 이용해 얻어 progress.txt에 기록한다. 소멸자에서는 board를 delete한다.

getBoard:

wrapping funtion으로 board를 return한다.

int isGameOver():

이중 for loop를 이용해 2048의 값을 가진 block이 있는지 확인하고, 있다면 0을 return하고, 2048

짜리 block이 없다면 board가 moveable한지 확인하고 moveable하지 않다면 1을 return, moveable하다면 2를 return하도록 하였다.

class BlockUi

block의 ui를 관리하는 클래스이다. 멤버 변수를 가지지 않는다.

생성자와 소멸자:

소멸자는 아무 일을 하지 않도록 하였고, 생성자는 QLabel* parent와 Block* block을 argument로 받아 QLabel을 parent로 생성해주고, 사이즈를 정한 후 후술할 update 함수에 block을 입력하도록 하였다.

Void update(Block *block):

switch case문을 사용해 블럭이 value가 0부터 2048까지 각 경우일때 마다 다른 style을 정하도록 하였고, 0의 경우 blank한 text를 설정하고 return하도록 하였다. 나머지 경우에는 switch case문이 종료된 후 각 value로 setText하도록 하였다. 이후 pdf에 적힌 대로 widget들을 배치하였고, 마지막에 두 버튼을 후술할 funtion들에 connect하도록 하였다.

class GameUi

class GameUi는 게임의 전반적인 ui를 설정하도록 한 클래스이다. QWidget을 상속받아 Q_OBJECT 매크로를 먼저 설정해주었다. 멤버 변수로는 전체 layout인 QGridLayout *mainLayout, 게임판의 layout인 QGridLayout *board;, 스코어 label인 QLabel *scoreLabel, 두 버튼 QPushButton *exitButton과 QPushButton *restoreButton, Game *game, Block들에 gui룰 입힐 BlockUi*** gameboard, retore 횟수를 저장할 int restoreCount가 있다.

생성자와 소멸자:

입력받은 QWidget을 생성자에 넣어주고, restoreCount를 3으로 초기화해준다. 이후 각 멤버 변수들에 해당되는 객체들을 동적으로 할당해 주도록 하였다. gameBoard의 경우 이중 for loop을 이용해 2차원 포인터 array를 초기화하도록 하였고, 이를 바로 addWidget하여 게임판을 완성하였다. 소멸자에서는 반대로 생성자에서 동적으로 할당된 메모리를 모두 delete하도록 하였다.

void keyPressEvent(QKeyEvent* event):

이는 QWidget의 메소드를 override하는 함수로, key의 입력에 따라 다른 동작을 하도록 한다. 먼저 progress.txt에 기록하기 위해 필요한 객체를 초기화한다. 이후 switch case문을 이용해 각 key에 따른 동작을 하도록 한다. 각 key마다 동작은 대동소이하기에 이번에도 Key_Left의 경우만 설

명하도록 한다. 왼쪽 방향키가 눌리면 먼저 progress.txt에 LEFT를 기록하고, Board 클래스의 moveLeft를 호출하여 board를 이동시키는데, 움직인 것이 있다면 새로운 block을 Board 클래스의 genRandBlock을 이용해 생성햇고, 생성한 기록을 progress.txt에 저장하도록 한다. Switch case 문을 break한 이후에는 수정된 board를 ui에 update하기 위해 후술할 updateUi 함수를 호출한다.

void updateUi():

이중 for loop을 이용해 board의 각 block에 update()를 호출해 준다. 이후 score ui를 update된 값으로 수정해준다. 마지막으로 switch-case문으로 isGameOver메소드를 호출해 return값이 0이면 QMessage를 통해 축하 메세지를 출력하고 app을 종료하고, return값이 1이면 lose message를 호출하고 app을 종료하도록 하였다.

void showExit():

Exit button을 누르면 호출되는 함수로, QMessageBox를 이용해 확실히 종료할 것인지 재차 확인하는 메세지를 띄우고 선택지를 주며, yes를 선택하면 app이 종료되도록 하였다.

void showRestore();

restore 버튼을 누르면 호출되는 함수로, 여시 QMessageBox를 이용해 확실히 restore할 것인지 묻는 것과 동시에 남은 횟수를 보여주는 창을 띄우고, Yes를 선택한다면 각 경우에 따라 적절한 동작을 하도록 하였다. Restore 기회도 충분하고, 버퍼도 비어 있지 않다면, 성공했다는 메세지를 띠우고, restore method를 호출하고, restoreCount를 1 줄이며, noBuff를 true로 초기화해주고, updateUi를 진행해준다. 이후 진행된 restore를 progress.txt에 기록해준다. 가각 noBuff가 true인 경우와 restoreCount가 0인 경우 각 상황에 맞는 실패 메세지를 띠우도록 하였다.

3. 토론 및 개선

Qt는 처음 다루어보는데, 어느 정도 프레임워크에 익숙해지자, 기존에 배웠던 cpp 지식만으로도 유용한 프로그램을 만들 수 있다는 자신감을 가지게 되었습니다. 이제까지 배운 객체지향 프로그래밍의 개념들을 대부분 활용해 보려 노력했는데, 한 학기를 제대로 복습하고, 적용해본 것 같습니다. 또한, ui를 관리하는 클래스와 내부 로직을 관리하는 클래스를 분리하여 구현함으로서 각 객체들의 독립성과 구현의 편의성을 높이는 방법도 알게 되었습니다.

아쉬운 점은, 이제까지 사용할 수 있는 library에 제약이 있었기 때문에, 막상 제한이 풀리더라도 기존에 사용하던 기능만을 사용하게 되었다는 것입니다. STL을 적극 활용하고, 다양한 알고리즘을 적용해 보고 싶었지만, 경험이 없고, 프레임워크만을 익히는 데에도 시간이 부족해 아예 적용해보지 못했습니다.

또한, pdf에서 구현하라고 지시되어 있던 class들에 적절한 역할을 부여하고 적절하게 메소드를 분배하지 못한 것이 아쉽습니다. 다양한 클래스들 간의 상속과 friend를 적극적으로 사용하였더라면 redundancy가 적고, 간결한 코드를 작성할 수 있었을 것입니다.

하지만 이번 과제를 통해 어떤 과제가 주어져도, 혼자서 학습해 해결해나갈 능력이 생긴 것 같습니다. 한 학기 동안 감사했습니다!

4. 참고 문헌

https://doc.qt.io/ qt official document. Qt 문법 관련 전반적인 내용 참고.