2023 Spring OOP Assignment Report

과제 번호 : 5
학번 : 20220302
이름 : 김지현
Povis ID : jihyunk

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.
I completed this programming task without the improper help of others.

1. 프로그램 개요

- 이번 프로젝트에서는 2048 게임을 구현할 것이다. Visual Studio 가 아닌 QT Creater 프로그램을 사용해 인터페이스 창을 만들어 방향키를 이용해 2048 게임을 하고 2048을 만들게 되면 승리하는 게임을 구현할 것이다.

- 프로그램을 실행하기 위해서는 QT creater 프로그램에서 파일을 실행시킨 후, ctrl+R 을 이용해 인터페이스 창을 띄운다. 그 이후엔 방향키를 이용해 게임을 진행하면 되고 restore 또는 exit 를 하고 싶으면 마우스로 그 버튼을 클릭하면 된다.

- block.h 는 2048의 4 X 4 블락 하나에 저장된 정보를 가지고 있다. Board.h 는 앞의 블락을 총 16개 저장한 값을 가지고 있다. 이때, 동적할당을 자동적으로 해주는 shared pointer 를 이용해서 만들어주었다. 마지막으로 game.h 에는 restore 과 같은 게임 진행에 필요한 동작을 해주는 함수들이 포함되어있다.

- blockui.h 는 블락의 수에 맞는 텍스트와 색을 지정해 주는 유저 인터페이스 헤더파일이다. gameui.h 는 게임 진행을 인터페이스 창에 띄워주는 헤더파일이다.

- SharedPtr.h 는 앞서 말한 shared pointer 를 이용하기 위함이며 이는 assn 4 에서 구현한 헤더파일을 응용하였다.

2. 프로그램의 구조 및 알고리즘

block.h	
private	public
int value	Block() int get_value void set_value

- □ "value" saves the number of each block on the board.
- □ In default constructor, block(), we set the "value" equal to 0.
- □ get_value is a getter function which allows other functions to have use value saved in "value". set_value is a setter function which allows other functions to update the value saves in "value.

Board.h	
private	public

SharedArray<SharedArray<Block>> _board;
SharedArray<SharedArray<Block>> buffer;
int score;
int buffer_score;
	Board();
	SharedArray<SharedArray<Block>> get_block ()
	SharedArray<SharedArray<Block>> get_buffer ()
	SharedArray<Block>& operator[](int x)
	const SharedArray<Block>& operator[](int x) const
	void generate_random_number();
	void copy ();
	void restore();
	void moveUp();
	void MergeUp();
	void moveDown();
	void MergeDown();
	void moveRight();
	void MergeRight();
	void moveLeft();
	void MergeLeft();
	bool check_movement();

	void save_initial (int, int, int, int);

- _board and buffer are 2 dimensional shared array which has type of "block," which is introduced above. and score and buffer_score both contains the score of the game. As the blocks merge, the merged number is added to the score. _board and score is used normally during the game and buffer and buffer_score save the previous board and score which will be used when the user clicks restore button.
- In the default constructor, _board and buffer will be dynamically allocated and initialized. score and buffer_score are also initialized.
- operator[] is used for accessing each block in forms of "_board[][]" or "buffer[][]". This function also has const version.
- In <generate_random_number>, a new block that will be generated when the exiting blocks move will be created. the (x,y) coordinates and cost will be generated using srand which will give a random number based on time. while loop check for redundancy and QFile part will saves the x y coordinate and cost of the block into the txt file.
- copy is to save the previous _board condition to buffer before the movement and merge occur.
- restore is called when the user clicks the "restore" button. It restores the board condition by copying the values from the buffer back to the main board called "_board". This also restores the score.
- moveup/moveDown/moveRight/moveLeft are functions for moving the tiles upward/downward/right/left. Then MergeUp/MergeDown/MergeRight/MergeLeft are called to combine any adjacent tiles which have same value. Then, the "move" function is called to move the tiles again. During the merge, any merged tiles are recorded to the txt file.
- check_movement function checks whether any tile has moved during the "move" function. It compared the current board state with the buffer board state. If moved, this function returns true, if not, the function returns false.
- save_initial function saves the initial board state to a txt file.

game.h (public Board)	
private	public

Board game
int restoreCnt
int gamescore
int bufferscore
	Game()
	void start_the_game()
	Board get_board()
	void set_gamescore (int)
	int get_gamescore();
	bool check_full ();
	bool check_2048();
	int get_restore_cnt();
	void set_restore_cnt()
	bool is_empty_buffer()
	void copy_buffer()
	void restore_buffer()

- game is a board class type object. restoreCnt is an integer variable to tract the number of restores available. gamescore are integer variables for the current score of the game.
- Game() is a default constructor which initializes restoreCnt to 3 and gamescore to 0.
- get_board is a getter function that returns a private variable game.
- set_gamescore updates the gamescore variable and get_gamescore returns the gamescore.
- start_the_game initializes the game board and place two random tiles with a value of 2 at random position on the board.
- check_full function checks whether the game board is full, meaning all tiles are occupied, or if there are any adjacent tiles that could be merged. if the board is not full, or tiles can be merged, the function returns true.
- check_2048 checks whether the game has reached 2048. It iterates through the game board and returns true if the 2048 tile exists.
- get_restore_cnt and set_restore_cnt are functions to update or return the restoreCnt variable.
- is_empty_buffer checks whether the buffer, which is used to restore the game state, is empty, If the buffer is empty, the function returns true.
- copy_buffer and restore_buffer are used to save the previous board to buffer and restore

it when the user clicks restore button.

blockUI (public QLabel)	
private	public
	BlockUi(QWidget* parent = nullptr) : QLabel(parent) void setBlockValue(int value)

- BlockUi's default constructor sets the size of the block and text alignment to center.
- setBlockValue applies appropriate style based on the value of the block.

GameUi (public QWidget)	
private	public
	GameUI(QWidget* parent = nullptr) updateUi(const Board& b) keyPressEvent(QKeyEvent* event) move_up() / move_down() / move_left() / move_right() showExitConfirmation() showRestoreConfirmation()

- GameUI's default constructor initializes the game user interface by setting up the UI components and connecting signals and slots
- updateUI updates the UI components based on the current state of the game board. It updates the block values, score label, and checks whether the user wins or loses the game.
- keyPressEvent handles the key press events and performs the corresponding game actions. Also, this function checks whether the game board is full and calls the appropriate move functions. Then it checks whether the user wins the game or the game is over.
- move_up() / move_down() / move_left() / move_right() functions are to move the blocks in the corresponding direction by calling the appropriate functions in the game board.
- showExitConfirmation() displays confirmation dialog asking the user if they want to quit the game.

- showRestoreConfirmation() displays a confirmation dialog asking the user if they wan to restore the game board to its previous state. It checks whether the restore count is greater than 0 (because the user can restore up to 3 times) and whether the user restored twice.

SharedPtr, typename ObjectType	
Private	Public
ObjectType* m_object int* m_ref_counter	SharedPtr (ObjectType* object = NULL) SharedPtr (const SharedPtr <ObjectType, Dealloc>& other) ~SharedPtr() SharedPtr& operator = (const SharedPtr& other) ObjectType* operator -> () const ObjectType* operator -> () const ObjectType& operator* () const ObjectType& operator* () const ObjectType & operator [] (int index) const ObjectType & operator [] (int index) const operator ObjectType const* () const operator ObjectType* ()

- m_object is a private member pointer with template of ObjectType.
- m_ref_counter is a integer type pointer which counts the number of pointer that counts the same memory.
- SharedPtr (ObjectType* object = NULL) is a default constructor. When constructing the object of class SharedPtr, and object is not declared, then the "object" will be a null pointer. Then assign object to m_object and m_ref_counter to null pointer. If object is not a null pointer, dynamically allocate m_ref_counter and set equal to 1.
- SharedPtr (const SharedPtr <ObjectType, Dealloc>& other) is a copy constructor. When declaring "SharedPtr<Object Type> ptr2 (ptr1);", in cpp file, ptr 2 will points the memory

ptr 1 pointed to. Therefore, in the function definition, each m_object and m_ref_counter will be "copied" and if the ptr 1 is not a null pointer (this is the if statement) the m_ref_counter will be added by one.

- ~SharedPtr() is a destructor. If the pointer exists and the reference counter equals to 1 (meaning only one pointer points to that certain memory), then run the deallocation function, called "Dealloc", for m_object and delete m_ref_counter.
- SharedPtr& operator = (const SharedPtr& other) is an assignment operator. If the "this" pointer, which is a reference to an invoking object, equals to the "other" shared pointer, than the function returns nothing. If not, the check the m_ref_counter and if exists and equals to 1, Dealloc m_object and delete m_ref_counter. Then, set m_object and m_ref_counter each equals to "other" pointer's m_object and m_ref_counter. Then, if the m_ref_counter is not a null pointer, add 1 to m_ref_counter.
- ObjectType* operator -> () is a pointer operator for "->". This function will return m_object.
- const ObjectType* operator -> () const does the same function as above but this is a constant function, meaning that the function will not modify inside the function and returns a constant value / pointer.
- ObjectType& operator* () is a pointer operator for "*". This function will return *m_object.
- const ObjectType& operator* () const does the same function as above but this is a constant function.
- ObjectType & operator [] (int index) accesses an array element. It returns m_object[index].
- const ObjectType & operator [] (int index) const does the same function as above but this is a constant function.
- operator ObjectType* () is a type casting operator. It is used when converting the Shared pointer to regular pointer. For Instance, it is used when assigning shared pointer ptr1 of type MyClass to a regular pointer MyClass* ptr2.
- operator ObjectType const* () const does the same function but for constant regular pointer.

3. 토론 및 개선

- 이번 과제를 통해 객체 지향 프로그래밍 수업에서 배운 내용을 응용했을 뿐만아니라 QT 사용법도 익힐 수 있었다. 코드의 구조화와 재사용성을 향상시키며 move function 과 merge function 을 작성하였고 Block, Board, Game class 로 나눠 게임의 주요 요소와 BlockUI 와 Game UI class 인 인터페이스를 구현하는 요소로 나눠서 작성하였다. 또한 상속을 통해 각 클래스 간의 연결을 하였고, 저번 과제에서 작성한 Shared pointer/array 를 이용해 동적할당과 동적할당해제를 해주었다.

- 그러나 이번 과제에서는 QT 사용법을 익히는데에 많은 시간을 할애하였다. 로직을 구현하기 전에 인터페이스 창을 띄우고 버튼을 만든뒤, 각 버튼마다 function 을 구현하는 것이 익숙치 않아 많은 시행착오를 겪기도 하였다. 그러나 한번 작동법을 익히고 나니 다른 게임도 만들어 보고 싶다는 호기심이 생기기도 하였다.

- QT 작성법과 관련하여 프로젝트를 하면서 질문이 생기기도 했다. 과제 설명 pdf 에 보면 인터페이스 창의 크기를 가로 1300, 세로 1000 로 설정하라고 되어있는데 노트북을 사용하다 보니, 위의 크기로 설정하면 모든 화면이 다 보이지 않아 가로 650, 세로 500 으로 고친 뒤, 블락의 크기도 창의 크기 축소에 맞게 줄여서 테스트를 진행하였다.

- 제작된 프로그램은 2048이 만들어 지면 자동적으로 종료되게 설정되어있지만 2048보다 더 큰 수를 만들도록 진행해도 재밌는 게임이 완성될 것 같다. 또한, move 와 merge function 이 각각 for 문을 2개씩 사용하여 complexity 가 증가하였는데 각각의 i 와 j 에 해당하는 값을 넘기는 방식으로 공통된 함수 하나를 사용한다면 더욱 빠른 프로그램을 작성할 수 있을 것이라고 생각한다.

4. 참고 문헌

- QT 작성법을 위해서 아래와 같은 소스를 참고하였습니다.
- https://doc.qt.io/qt-6/qkeyevent.html
- https://doc.qt.io/qt-6/qlabel.html
- https://doc.qt.io/qt-6/qpushbutton.html
- https://doc.qt.io/qt-6/qvboxlayout.html
- https://doc.qt.io/qt-6/signalsandslots.html
- https://doc.qt.io/qt-6/qstring.html
- https://doc.qt.io/qt-6/qtextstream.html
- https://doc.qt.io/qt-6/stylesheet-reference.html