2020 Spring OOP Assignment Report

과제 번호 : 3
학번 : 20190084
이름 : 권민재
Povis ID : mzg00

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

- □ 이 프로그램은 다형성을 이용하여 테트리스를 구현한 프로그램이다..
- □ 과제에서 미리 지정해 준 키들을 이용하여 게임을 플레이할 수 있다.
- □ 프로그램의 코드는 여러 파일로 구성되어 있으며, 별도의 디렉토리 구성은 하지 않았다.
- □ **winpty를 사용하지 않으며**, MSYS2 MinGW, macOS Clang, Linux GCC 등에서 컴파일 및 실행할 수 있다. 다만, IO는 MSYS2 환경에 최적화 되어있다.[1]

2. 프로그램의 구조 및 알고리즘[2]

A. 입출력[3]

- □ **void input(Queue<char> &inputQueue, mutex &m, bool &isNotPause, bool &isRun)**

  (추가 기능 구현 사항 – windows.h나 conio.h, ncurses 등의 헤더 없이 스레드와 stty로 입력 구현) input은 사용자 입력을 받기 위해 스레드에서 실행되는 함수이다. 여러 스레드에서 안정적으로 사용자의 입력을 처리하기 위해서 공용 인풋 큐를 사용한다. 함수 input의 스레드는 계속해서 입력을 받으며, 이것을 공용 인풋 큐에 push 한다. 그러면 다른 사용자 입력이 필요한 곳들에서 공용 인풋 큐에서 pop하여 사용자 입력을 가져온다. 이때, 스레드 간의 데드락 / 레이스 컨디션에 걸리지 않기 위해

---

[1] msys2 환경에서 system("clear")가 느리가 작동할 때가 있어서, 빈 줄을 다량 출력하는 것으로 대체하였으며, 입력 스레드 종료 이후에도 유저의 인풋을 기다리는 현상이 msys2에만 있어서 Press Any Key to Exit 안내문을 출력하였다.

[2] 각 클래스의 생성자, 소멸자 그리고 오버로딩된 연산자는 표에서 생략되어 있음.

[3] 유의미한 함수만 표시되어 있음.

mutex를 사용한다.

(추가 기능 구현사항 - DAS) sty cbreak -icanon -echo로 터미널의 모드를 변경한 후 스레드에서 getchar를 이용하여 입력을 받도록 구현하였기 때문에 자연스럽게 DAS가 되는 모습을 확인할 수 있다. 키를 꾹 누르고 있으면 처음에 한칸 움직였다가, 쭉 움직이게 된다.

□ **void logScore(int score)**

점수를 입력받아 점수 로그를 남기는 함수이다. 기존의 파일을 열어서 스트링 형식으로 저장한 이후, 점수를 문자열로 바꿔서 기존 로그의 앞에 붙인 후 파일에 기록한다.

□ **void getChar(Queue<char>& inputQueue, char& result, mutex& m)**

인풋 큐에서 입력을 가져오는 함수이다. 데드락을 피하기 위해서 뮤텍스를 사용하며, 큐에서 입력을 가져오는데 성공할 때 까지 인풋 큐에서 pop을 시도하여 result에 담아준다.

B. 게임

Game	
게임의 진행을 관리하는 클래스	
private	
멤버 변수	
int unitTick = 100000000	기본 게임 진행 틱
int inputTick = 100000	입력 틱
int score = 0	점수
int combo = 0	콤보
unsigned int gameProcess = 0	게임 진행 틱 카운터
unsigned int inputProcess = 0	유저 인풋 틱 카운터
bool isUsed[NUM_TETROMINO] {false, }	테트리미노 사용 여부를 저장하는 배열
double coefficient = 1	게임 속도 배수
BlockList* blockList	게임에서 사용할 블럭리스트 오브젝트의 주소
Board* board	게임에서 사용할 보드 오브젝트의 주소
Queue<char>* inputQueue	공유 인풋 큐 주소
Stack<Tetromino*> minoStack	사용한 테트리미노 스택
mutex* m	스레드 뮤텍스 주소
Tetromino* t	현재 게임 중인 테트리미노
Tetromino* next_t	다음 테트리미노
멤버 함수	
void initUsedArray()	테트리미노의 사용 여부 배열을 초기화 하는 메서드
bool isAllUsed()	테트리미노가 모두 사용되었는지 여부를 반환하는 메서드
Tetromino* getTetromino()	사용되지 않은 테트리미노를 만들어주는 메서드

void setTetromino()	현재 테트리미노와 다음 테트리미노를 세팅하는 메서드
char getInput()	인풋 큐로부터 입력 값을 가져오는 메서드
bool parseInput()	입력을 파싱하는 메서드
void checkScore(int num_lines)	점수를 계산하는 메서드
public	
멤버 함수	
int run()	게임을 실행하는 메서드
bool checkSaveLog()	로그를 남길 것인지 확인하는 메서드
bool checkRestart()	재시작 여부를 체크하는 메서드

- 게임 오브젝트의 **run, checkSaveLog, checkRestart**를 순서대로 수행하여 게임의 한 라운드를 진행할 수 있다. **run** 메서드에서는 입력 틱마다 공용 인풋 큐에서 입력을 가져오고, 게임 틱마다 게임을 진행시키는 역할을 한다. **checkSaveLog**에서는 유저에게 로그를 남길지 그 여부를 물어본 후, 로그를 남기는 함수를 호출하여 로그를 남길 수 있게 하였고, **checkRestart**에서는 재시작 여부를 물어본 후 재시작을 결정한다. **checkSaveLog**와 **checkRestart** 메서드에서, cout이 stty 설정과 충돌하기 때문에, 안내문을 안정적으로 출력하기 위해서 system("echo")를 이용했다.

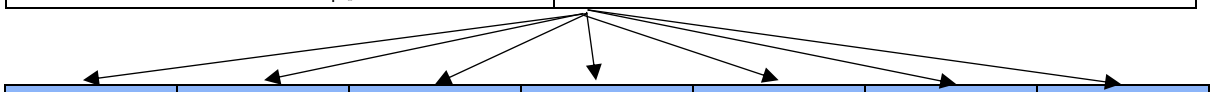
- **checkScore** 메서드에서 게임의 점수를 체크한다. 1 ~ 4 줄을 한번에 없앴을 때의 기본적인 점수 부여 시스템을 가지고 있으며, 점수에 따라서 변수 **coeffcient**를 조절하여 속도를 조절할 수 있도록 했다.

- **(추가 기능 구현사항 – 콤보) checkScore**에서 추가적으로 **콤보**를 구현하였다. 기본적으로 한 줄 이상을 지웠을 때 콤보가 1씩 증가하며, 현재 테트리미노가 정지되었는데 지운 줄 수가 0이라면 콤보를 0으로 초기화하게 구현하였다. 그에 따라 ((콤보 – 1) * 100) 만큼의 추가점수를 부여해준다.

C. 테트로미노

Tetromino	
테트리미노를 관리하는 기본 클래스	
private	
멤버 변수	
Position SRSDataSet[8][5]{}	SRS 오프셋 데이터셋
Block* blocks[4]{}	테트리미노의 블럭들 배열
Board* board{}	테트리미노가 사용할 보드
int color = RESET	테트리미노의 색상
int rotate_num = 1	테트리미노의 회전 상태
멤버 함수	
Position blockPosition[4]	블럭들의 상대 좌표 배열

bool isHit(int dx, int dy, bool checkCeiling)	테트리미노의 충돌 여부를 확인하는 메서드
bool checkStop()	블럭이 멈춰야 할 상태인지 체크하는 메서드
int stop()	블럭을 멈추는 메서드
protected	
멤버 함수	
void setSRSDataSet(Position set[][5])	SRS 오프셋 데이터셋을 저장하는 메서드
virtual void initTetromino(int x, int y)	테트로미노를 초기화하는 가상 메서드
virtual void SRSTestSetGenerator()	SRS 오프셋 데이터셋을 생성하는 가상 메서드
virtual void relativePositionGenerator(Position* _blockPosition)	블록의 상대좌표를 설정하는 가상 메서드
virtual int colorGenerator()	블록의 색상을 설정하는 가상 메서드
public	
멤버 함수	
Block& getBlockByIdx(int i)	인덱스로 블록 주소를 반환 하는 메서드
void setStr(string& target)	블럭들의 문자를 설정하는 메서드
void setShadowMino(bool isShadow)	블럭들을 그림자 블럭으로 설정하는 메서드
virtual void rotate(int direction)	테트리미노를 회전하는 메서드
void move(int dx, int dy, bool checkCeiling)	테트리미노를 움직이는 메서드
bool isStop()	테트리미노의 멈춤 여부를 반환하는 메서드
void onTick()	게임 틱마다 실행되는 메서드
void hardDrop()	테트리미노를 하드 드롭 하는 메서드



Mino_I	Mino_O	Mino_T	Mino_L	Mino_J	Mino_S	Mino_Z

void relativePositionGenerator(Position* _blockPosition) override;
 	
int colorGenerator() override;
 	
void SRSTestSetGenerator() override;
	void relativePositionGenerator(Position* _blockPosition) override;
	
	int colorGenerator() override;
	
	void rotate(int direction) override;
		void relativePositionGenerator(Position* _blockPosition) override;
		
		int colorGenerator() override;
			void relativePositionGenerator(Position* _blockPosition) override;
			
			int colorGenerator() override;
				void relativePositionGenerator(Position* _blockPosition) override;
				
				int colorGenerator() override;
					void relativePositionGenerator(Position* _blockPosition) override;
					
					int colorGenerator() override;
						void relativePositionGenerator(Position* _blockPosition) override;
						
						int colorGenerator() override;

□ 테트리미노를 기반으로, 여러 종류의 미노를 생성할 수 있도록, **initTetromino, SRSTestSetGenerator, relativePositionGenrator, colorGenerator** 메서드를 가상 메서드로 지정하여 다른 미노에서 override 할 수 있도록 하였다. 이 가상 메서드들의 반환 값은 미노마다 고유하며, **initTetromino** 메서드에서 이들의 반환값을 받아서 테트리미노를 세팅한다.

□ 메서드 rotate는 테트리미노를 회전시킨다. 이때 SRSDataSet을 참조하여 메서드 **ishit**을 통해 충돌 테스트를 시행함으로써 SRS에 기반한 회전이 일어날 수 있도록 하였다.

D. 블록

Block	
블록 클래스	
private	
멤버 변수	
Point p {0, 0, 0, 0}	블럭의 좌표 저장
int color	블럭 색상 지정
string str = "■"	표시 텍스트 지정
멤버 함수	
bool isStoppedBlock = false	블럭의 멈춤 여부
bool isShadowBlock = false	그림자 블럭 여부
public	
멤버 함수	
string toString()	블럭을 색상있는 문자열로 리턴하는 메서드
void setAxis(int x, int y)	블럭의 축 좌표 지정 메서드
void move(int dx, int dy)	블럭 이동 메서드
void setRelative(int x, int y)	블럭 상대 좌표를 정수형으로 지정하는 메서드
void setRelative(Position _p)	블럭 상대 좌표를 Position 으로 지정하는 메서드
void setColor(int _color)	블럭 색상 지정 메서드
int x()	블럭의 x 절대 좌표 반환 메서드
int y()	블럭의 y 절대 좌표 반환 메서드
int rel_x()	블럭의 x 상대 좌표 반환 메서드
int rel_y()	블럭의 y 상대 좌표 반환 메서드
void rotate(int direction)	블럭 회전 메서드
bool& isStop()	블럭의 멈춤 여부 반환하는 메서드
void setStr(string& s)	블럭의 문자열 지정
bool& isShadow()	블럭의 그림자 여부 반환

BlockList	
블록들을 관리하는 클래스	
private	
멤버 변수	
int size = 0	블럭 개수
멤버 함수	
Block* blockList[MAX_BLOCK] {}	블럭 주소 배열

void initRemoveLines(int simulateBoard[][COL], int line_size[ROW])	줄 삭제 메서드를 초기화하는 메서드
public	
멤버 함수	
Block* add(int axis_x = 0, int axis_y = 0, int rel_x = 0, int rel_y = 0, int color = RESET)	블록 리스트에 블록을 등록하고 반환하는 메서드
Block* append(Block* target)	임의의 블록을 리스트에 추가하는 메서드
Block* at(int idx)	특정 인덱스의 블록 주소 반환하는 메서드
void removeShadow()	그림자 블록을 삭제하는 메서드
int removeLines()	줄을 삭제하는 메서드
bool isGameOver()	게임 오버인지 확인하는 메서드

- 모든 **Block** 오브젝트들은 **BlockList** 오브젝트에 귀속되어 작동하도록 구현하였다.
- **removeLines** 메서드를 통해 꽉찬 줄을 삭제하고, 삭제한 줄 수를 반환한다. 해당 메서드에서는 시뮬레이션 보드를 세팅(**initRemoveLines**)하고, 한 줄의 블록 수가 가로칸 수와 같으면 해당 줄의 블록을 삭제하고 그 위의 블록들을 한 칸씩 아래로 움직이게 한 뒤, 다시 시뮬레이션 보드를 세팅하는 작업을 반복함으로써 구현하였다.
- **isGameOver** 메서드에서는 블록들 중에 보드의 높이 보다 높은 곳에서 멈춘 블록이 있다면, 게임 오버로 판정한다.

E. 보드 관리 및 렌더링

Board	
게임 보드를 렌더링하는 클래스	
private	
멤버 변수	
Block* gameBoard[ROW][COL]{}	게임 보드 배열
string infoBoard[INFO_ROW][INFO_COL]	정보 보드 배열
Tetromino* shadow	그림자 테트리미노
BlockList* blockList	이 보드에서 사용할 블록 리스트 오브젝트
bool isShadowOn = true	그림자 출력 여부
멤버 함수	
void initGameBoard()	게임 보드를 초기화 하는 메서드
void initInfoBoard()	정보 보드 배열 초기화
void setInfoBoard(Tetromino *nextMino, int score, int combo)	정보 보드를 출력하기 위해 세팅하는 메서드
void printLine(bool isEndl, int num)	줄을 출력하는 메서드
void makeShadow(Tetromino* currentMino)	그림자를 만드는 메서드
void deleteShadow()	그림자를 삭제하는 메서드
public	
멤버 함수	
void setGameBoard()	보드 배열을 출력하기 위해 세팅하는 메서드

void render(Tetromino *currentMino, Tetromino *nextMino, int score, int combo)	게임을 렌더링하는 메서드
Block* XY(int x, int y)	특정 좌표의 블럭 주소 반환
void shadowSwitch()	그림자 표시 모드를 지정하는 메서드

- 보드는 게임 보드를 관리하며, 화면에 렌더링해주는 역할을 가지고 있다. render 메서드에서 setInfoBoard 메서드를 호출하여 화면에 표시될 다음 테트리미노, 점수, 그리고 콤보를 세팅하고, setGameBoard 메서드로 화면에 표시될 블록들을 세팅한 후, 사용자가 보는 화면을 출력해준다.
- 그림자 기능은 현재 테트리미노를 복사한 후, 그 미노의 표시 문자를 바꾸고 하드드랍하여 보여주는 방식으로 구현하였다.
- **(추가 기능 구현사항 – 콤보 출력)** 콤보 기능을 구현해두었기 때문에, setInfoBoard를 통해 콤보를 출력할 수 있도록 설정하였다.
- **(추가 기능 구현사항 – 테트리미노 스폰 직후 화면 밖 움직임)** 블록들은 어떠한 좌표도 가질 수 있으며, 렌더링 될때는 게임 보드 범위 안의 블록들만 보여주게 구현되어 있기 때문에, 블록들은 블록이 채 내려오기 전인, 즉 스폰된 직후에도 움직이거나 회전할 수 있다.

F. 좌표계

Position	
기본적인 좌표를 관리하는 클래스	
private	
멤버 변수	
int pos_x = 0	x 좌표
int pos_y = 0	y 좌표
public	
멤버 함수	
int& x()	x 좌표를 반환하는 메서드
int& y()	y 좌표를 반환하는 메서드
void setPosition(int x, int y)	정수형으로 좌표를 설정하는 메서드
void setPosition(Position p)	Position 으로 좌표를 설정하는 메서드
void swap()	x 와 y 좌표 값을 바꿔주는 메서드

Point
블럭의 좌표를 관리하는 클래스
private
멤버 변수

Position relative	축으로부터의 상대 좌표
Position axis	축의 좌표
public	
멤버 함수	
int x()	x 절대 좌표를 반환하는 메서드
int y()	y 절대 좌표를 반환하는 메서드
int rel_x()	x 상대 좌표를 반환하는 메서드
int rel_y()	y 상대 좌표를 반환하는 메서드
void setAxis(int x, int y)	축 좌표를 설정하는 메서드
void setAxis(Position p)	Position 으로 축을 설정하는 메서드
void setRelative(int x, int y)	상대 좌표를 설정하는 메서드
void setRelative(Position p)	Position 으로 상대 좌표를 설정하는 메서드
void rotate(int direction)	좌표를 회전하는 메서드

- 모든 블록은 보드의 제일 왼쪽 아래를 (0, 0)으로 가지며, 왼쪽으로 갈수록 x 좌표 값이 커지고, 위로 갈수록 y 좌표값이 커지는 좌표계를 가지고 있다. **클래스 Position**은 이러한 좌표계의 가장 작은 단위이며, **클래스 Point**는 축 Position과 상대 좌표 Position을 가지는 클래스로, 축과 상대 위치가 필요한 블록에게 알맞은 좌표계를 제공한다.
- Point::rotate 메서드를 통해 좌표를 직접적으로 회전시킬 수 있으며, 회전 행렬을 응용하여 x, y 좌표를 swap하고 알맞게 부호를 바꾸는 식으로 구현하였다.

G. 자료구조

Node	
스택과 큐에 사용되는 노드 클래스	
private	
멤버 변수	
T node_data	노드에 저장될 데이터
Node* node_next	다음 노드 주소
Node* node_prev	이전 노드 주소
public	
멤버 함수	
Node*& next()	다음 노드 주소를 참조형으로 반환하는 메서드
T& data()	데이터를 참조형으로 반환하는 메서드
Node*& prev()	이전 노드 주소를 참조형으로 반환하는 메서드

Queue
큐 구현 클래스
private

멤버 변수	
Node<T>* begin = nullptr	큐의 시작 노드
Node<T>* end = nullptr	큐의 마지막 노드
int queue_size = 0	큐 크기
public	
멤버 함수	
int size()	큐의 크기를 반환하는 메서드
void push(T data)	큐에 데이터를 추가하는 메서드
bool pop(T& input)	큐에서 데이터를 제거하는 메서드

Stack	
스택 구현 클래스	
private	
멤버 변수	
Node<T>* top = nullptr	스택의 가장 위 노드 주소
int stack_size = 0	스택의 크기
public	
멤버 함수	
int size()	스택의 크기를 반환하는 메서드
void push(T data)	스택에 데이터를 추가하는 메서드
bool pop(T& input)	스택에서 데이터를 빼는 메서드

3. 토론 및 개선

- 최대한 시스템 디펜던트하지 않게 입력을 구현하기 위해 노력하였지만, 현재 사용하고 있는 방법도 stty를 지원하지 않는 시스템에서는 돌아갈 수 없다. 하지만, 이에 대해 찾아보면서 <unistd.h>에 있는 select를 이용하여 우리가 원하는 방식의 입력을 구현할 수 있음을 알 수 있었다.
- stty를 통해 직접적으로 시스템 설정을 만지기 때문에, 프로그램이 비정상적으로 종료되면 터미널이 이상하게 보일 수 있는 점을 개선할 필요가 있다.
- doxygen 문법으로 주석을 작성하여 추후 재사용성을 높였다.

4. 참고 문헌

- stty man page
  http://man7.org/linux/man-pages/man1/stty.1p.html