**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.
* 이 프로그램은 여러 파일로 구성되어 있으며, 별도의 디렉토리 구성은 하지 않았다.

1. **프로그램의 구조 및 알고리즘[[1]](#footnote-1)**
   1. **입출력[[2]](#footnote-2)**
   * **void input(Queue<char> &inputQueue, mutex &m, bool &isNotPause, bool &isRun)**

사용자 입력을 받기 위해 스레드에서 실행되는 함수이다. 스레드에서 안정적으로 사용자의 입력을 받기 위해서 공용 인풋 큐를 사용한다. 함수 input의 스레드는 계속해서 입력을 받으며, 이것을 공용 인풋 큐에 push한다. 그러면 다른 사용자 입력이 필요한 곳들에서 공용 인풋 큐에서 pop하여 사용자 입력을 가져온다. 이때, 스레드 간의 데드락 / 레이스 컨디션에 걸리지 않기 위해 mutex를 사용한다.

* + **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.

* 1. **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() | 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 |

* + 프로그램은 다형성을
  1. **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 |

* + 프로그램은 다형성을
  1. **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 Position으로 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 |

* + 프로그램은 다형성을
  1. **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 |

* + 프로그램은 다형성을
  1. **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) | Position으로 좌표를 설정하는 메서드 |
| void swap() | x와 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) | Position으로 축을 설정하는 메서드 |
| void setRelative(int x, int y) | 상대 좌표를 설정하는 메서드 |
| void setRelative(Position p) | Position으로 상대 좌표를 설정하는 메서드 |
| void rotate(int direction) | 좌표를 회전하는 메서드 |

* + 프로그램은 다형성을
  1. **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 |

* + 프로그램은 다형성을

1. **토론 및 개선**
   * + 비밀번호를 그냥 저장하지 않고 **해시를 이용**하여 저장함으로서 gdb와 같이 메모리를
2. **참고 문헌**
   * + 파일을 스트

1. 각 클래스의 생성자, 소멸자 그리고 오버로딩된 연산자는 표에서 생략되어 있음. [↑](#footnote-ref-1)
2. 유의미한 함수만 표시되어 있음. [↑](#footnote-ref-2)