테트리스 게임 보고서

객체지향프로그래밍 및 실습 (나)

김연주 (20233077)

장동현 (20213140)

이준혁 (20213045)

목차

채점을 위한 실행 과정과 각 과정에 따른 메소드 구현 여부는 '4. 실행 화면' 부분 참고하시면 됩니다.

1. 프로젝트 설명

1-1. 프로젝트 개요

객체 지향 프로그래밍 언어인 자바의 속성 (상속, 인터페이스, 스레드 등) 을 이용하여 테트리스 게임을 구현한다.

1-2. 프로그램 진행 순서

테트리스 게임은 간단한 퍼즐 게임으로, 여러 조각들이 위에서 아래로 떨어지며 그 조각들을 조합하여 가로줄을 채우는 게임이다. 게임의 진행 순서는 다음과 같다.

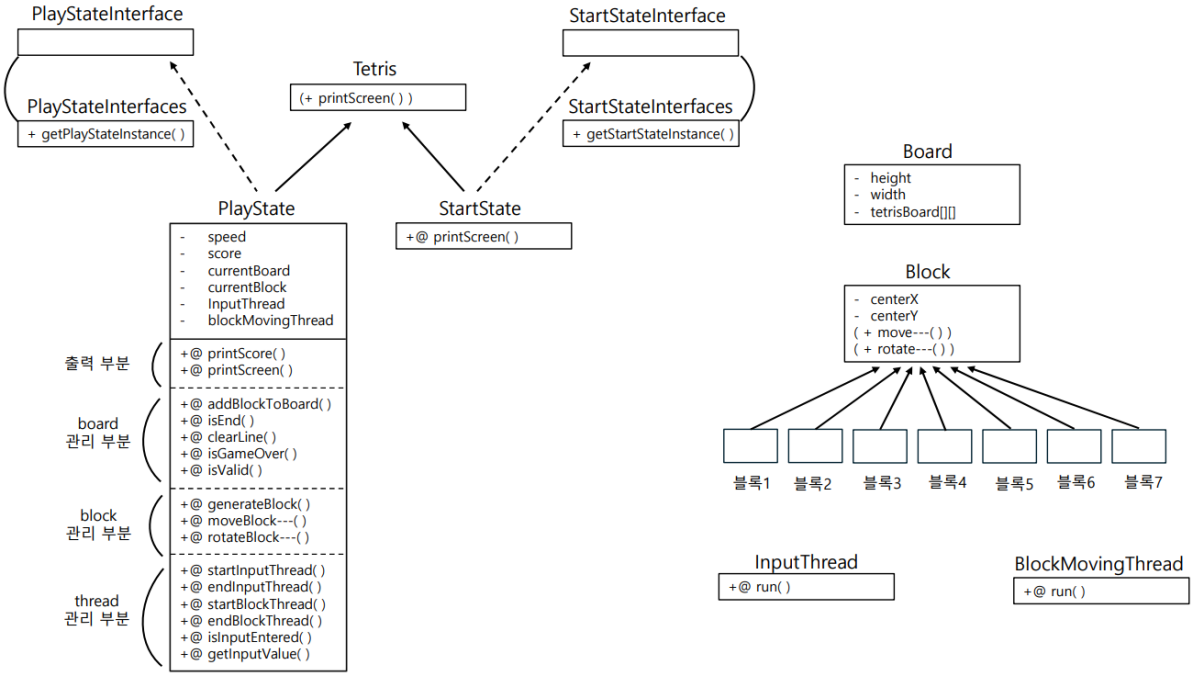
가) 프로그램을 실행한다.
나) 게임을 플레이한다.
다) 상단에서 블록이 내려오고, 특정 키를 누르며 블록을 이동 및 회전시키며 플레이한다.
라) 게임 종료 조건에 따라 게임이 종료되면, 최종 점수와 함께 게임 결과가 표시된다.

2. 프로그램 설계

2-1. 테트리스 매커니즘

1) 블록이 상단에서 하단으로 내려온다.
2) 블록은 총 7개가 있다.
3) 특정 키를 눌렀을 때 블록이 이동 및 회전한다.
4) 내려오는 블록이 바닥 또는 다른 블록에 닿으면 다음 블록이 내려온다.
5) 한 줄이 블록으로 완전히 채워지면 그 줄은 사라지고, 그 위에 있는 모든 블록들이 한 줄씩 아래로 내려온다. 이를 통해 점수를 얻는다.
6) 블록이 화면 상단까지 쌓이면 게임이 종료된다.

2-2. 프로그램 설계도



2-3. 클래스 및 메소드

- **인터페이스 : PlayStateInterface, StartStateInterface**

- **PlayStateInterface**의 메소드 :

  void printScreen(); *#화면 출력(보드 출력)*

  void printScore(); *#점수 출력*

  void addBlockToBoard(); *#block을 보드에다 반영*

  boolean isEnd(); *#block이 더 이상 내려갈 수 없음을 판단*

  void clearLine(); *#한 줄이 다 채워진 라인을 삭제 및 점수 증가*

  boolean isGameOver(); *#block이 더 이상 board에 들어갈 수 없음을 판단*

  boolean isValid(); *#block의 이동/회전 가능 여부 판단*

void generateBlock(); *#임의의 block 객체 생성*

void moveBlockLeft(); *#block을 좌로 이동*

void moveBlockRight(); *#block을 우로 이동*

void moveBlockDown(); *#block을 아래로 이동*

void moveBlockUp(); *#block을 위로 이동*

void rotateBlockRight(); *#block을 우로 회전*

void rotateBlockLeft(); *#block을 좌로 회전*

void setCurrentBlockNull(); *#한 줄을 채운 라인의 block들을 null값으로 지정하기 위함*

void startInputThread(); *#입력 스레드 시작*

void endInputThread(); *#입력 스레드 종료*

void startBlockThread(); *#block 이동 스레드 시작*

void endBlockThread(); *#block 이동 스레드 종료*

boolean isInputEntered(); *#사용자의 입력 여부 판단*

String getInputValue(); *#입력된 값 가져옴*

- **StartStateInterface**의 메소드:

    Void printScreen(); *#시작화면 출력*

- **Companion 클래스 : PlayStateInterfaces, StartStateInterfaces**

- **PlayStateInterfaces**의 메소드:

    getPlayStateInstance(); *#게임시작의 객체 생성*

- **StartStateInterfaces**의 메소드:

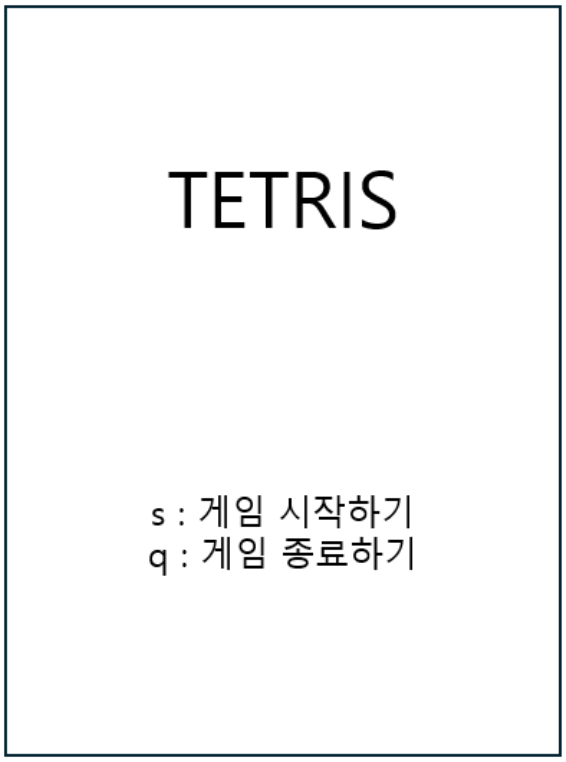
    getStartStateInstance(); *#시작화면의 객체 생성*

3. main함수

3-1. main함수 단락 별 설명

<main함수에서 시작화면 구현>

- Companion Class인 *StartStateInterfaces*를 통해 *StartStateInterface*타입의 객체를 생성
- Scanner로 사용자 입력값을 String타입으로 받아 "s"일 경우 게임시작(doGame()호출), "q"일 경우 종료하도록 구현.
- 실행 화면 :



<doGame() 함수로 게임 시작>

- Companion Class인 *PlayStateInterfaces*를 통해 *PlayStateInterface*타입의 객체를 생성
- non-blocking input을 구현하기 위해 입력 스레드(*startInputThread()*)를, 사용자로부터 값을 입력받는 동시에 Block이 움직이도록 하기 위해 block이동 스레드(*startBlockThread()*)호출.
- loopOut으로 중첩된 while(true)문을 한번에 빠져나가도록 설정.

<첫 번째 while(true)문>

- *generateBlock()*으로 임의의 블록(객체)을 생성
- 블록이 처음부터 들어갈 수 없는 경우 : *isValid()*로 판단하여 만약 들어갈 수 없다면 입력Thread

(endBlockThread()) 와 block이동Thread(*endInputThread*)를 종료시키고, 점수출력(*printScore()*) 및 반복문을 loopOut으로 빠져나간다.
- 블록이 들어갈 수 있는 경우 : addBlockToBoard()로 블록을 보드에 반영

<두 번째 while(true)문>

- 입력 thread 처리 구체화. *isInputEntered()*로 입력 여부를 판단하고, *getInputValue()*로 입력 값을 받아오는 것으로 구현함. 입력이 들어온 상태인 경우, *getInputValue()*로 받아온 입력 값을 문자열 타입인 inputString에 저장하여 블록의 이동 및 회전 구현.

- "a"를 입력한 경우 : 왼쪽으로 이동하도록 구현함.

  만약 왼쪽으로 갈 수 없는 경우를 *isValid()*로 판단하여, 갈 수 있다면 *addBlockToBoard()*로 보드에 반영, 갈 수 없다면 다시 원상복구 시키기 위해 오른쪽으로 이동(*moveBlockRight()*)시킴.

- "d"를 입력한 경우 : 오른쪽으로 이동하도록 구현함.

  만약 오른쪽으로 갈 수 없는 경우를 *isValid()*로 판단하여, 갈 수 있다면 *addBlockToBoard()*로 보드에 반영, 갈 수 없다면 다시 원상복구 시키기 위해 왼쪽으로 이동(*moveBlockLeft()*)시킴.

- "s"를 입력한 경우 : 아래로 이동하도록 구현함.

  만약 아래쪽으로 갈 수 없는 경우를 *isValid()*로 판단하여, 갈 수 있다면 *addBlockToBoard()*로 보드에 반영, 갈 수 없다면 다시 원상복구 시키기 위해 위쪽으로 이동(*moveBlockUp()*)시킴.

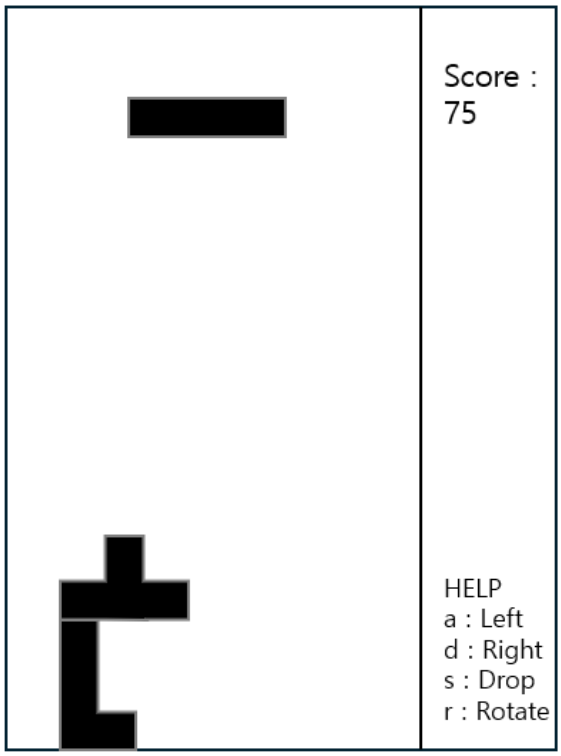
- "r"을 입력한 경우 : 오른쪽으로 회전하도록 구현함.(한 방향으로만 회전할 수 있게 구현)

  만약 오른쪽으로 회전할 수 없는 경우를 *isValid()*로 판단하여, 회전할 수 있다면 *addBlockToBoard()*로 보드에 반영, 회전할 수 없다면 다시 원상복구 시키기 위해 왼쪽으로 회전(*rotateBlockLeft()*)시킴.

여기서 *rotateBlockLeft(), moveBlockUp()*는 입력 값에 따라 실행시키기 위함이 아니라 이동/회전이 불가한 경우를 대비해 그 전 상태로 복구 시키기 위한 메소드임.

이후 이동/회전이 적용되었을 수 있으므로 화면을 새로 출력함.

- 실행 화면 :



<if(isEnd)>

블록이 더 이상 내려갈 수 없는 경우.

- *clearLLine()*으로 보드를 검사하여 채워진 라인을 삭제 및 점수 증가

- *printScreen()*으로 보드가 수정되었을 수 있으니 보드를 다시 출력

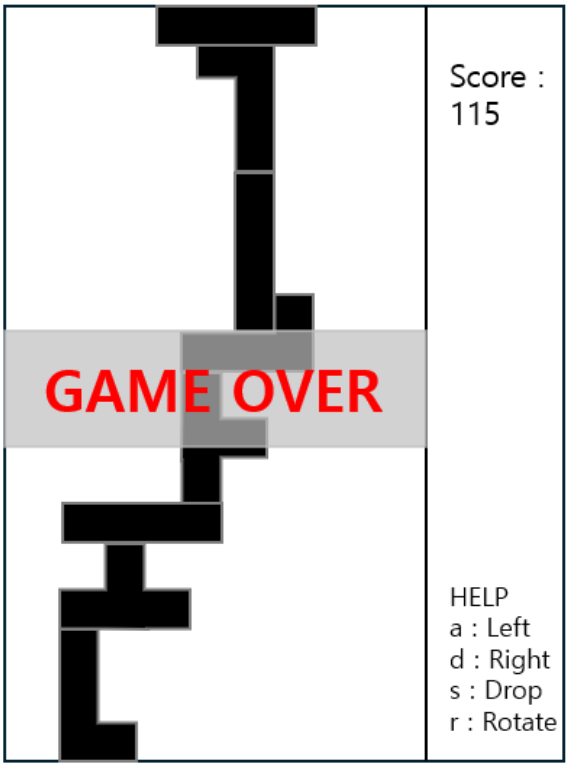
- *setCurrentBlockNull()*으로 현재 이동 중인 block이 존재하지 않는 것으로 취급하여 null로 지정

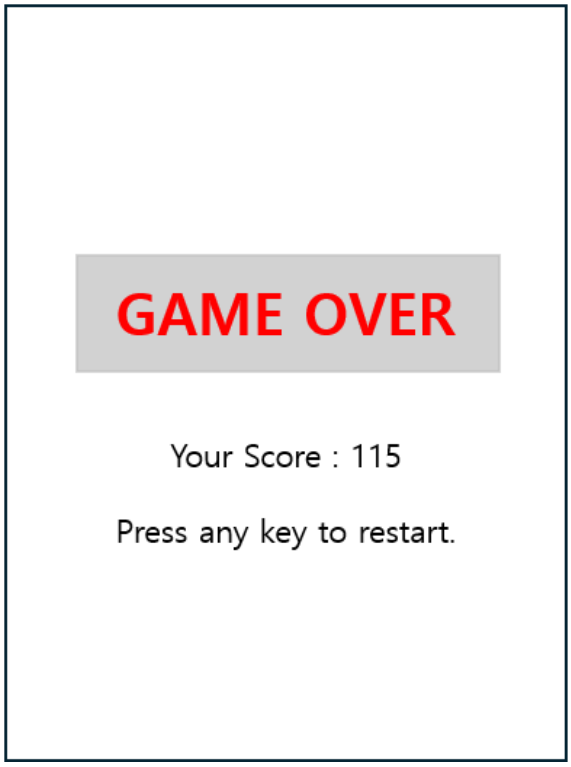
- if(*isGameOver()*)로 게임 오버인지 검사함. 블록이 더 이상 내려 올 수 없는 경우 게임 오버.

    입력 스레드(endInputThread())와 블록 이동 스레드(*endBlockThread()*)를 종료 시키고 총 점수를 출력(*printScore()*).

- 지정된 loopOut으로 반복문을 빠져나감.

- 실행 화면 :





3-2. main함수 구현 코드

```
import tetris.*;

import java.util.Scanner;


public class MainClass {

    public static void main(String[] args) {

        StartStateInterface startInstance = StartStateInterfaces.getStartStateInstance();  //시작화면 객체 생성


        Scanner inputStream = new Scanner(System.in);

        String inputString;

        while(true)
```

```
        {
            startInstance.printScreen();      //시작화면 출력
            inputString = inputStream.next();      //사용자 입력값 입력
            if(inputString.equals("s"))
            {
                doGame();      //테트리스 게임 시작
            }
            else if(inputString.equals("q"))
            {
                return;      //게임 종료
            }
            else
            {
                continue;
            }
        }
    }

    public static void doGame()      //테트리스 게임 구현
    {
        PlayStateInterface playInstance = PlayStateInterfaces.getPlayStateInstance();
        playInstance.startInputThread();      //입력 스레드 시작
        playInstance.startBlockThread();      //블록 이동 스레드 시작
```

```
loopOut:    //중첩된 while문을 한번에 빠져나가기 위한 label 설정
while(true)
{
    playInstance.generateBlock();    //임의의 블록 객체 생성
    if(!playInstance.isValid())    //block을 애초에 넣을 수 없는 경우
    {
        playInstance.endBlockThread();
        playInstance.endInputThread();
        playInstance.printScore();
        break loopOut;    //지정한 label로 빠져나감
    }
    playInstance.addBlockToBoard();    //block을 넣을 수 있는 경우 block을 보드로 반영

    String inputString;
    while(true)
    {
        if(playInstance.isInputEntered())    //입력이 들어온 상태인 경우
        {
            inputString = playInstance.getInputValue();    //입력 값을 가져옴
            if(inputString.equals("a")) // 왼쪽으로 이동(move)
            {
                playInstance.moveBlockLeft();
                if(playInstance.isValid()) playInstance.addBlockToBoard();    //보드 반영
                else playInstance.moveBlockRight();    //불가한 경우 반대쪽으로 이동
```

```
}
else if(inputString.equals("d")) // 오른쪽으로 이동(move)
{
    playInstance.moveBlockRight();
    if(playInstance.isValid()) playInstance.addBlockToBoard();  //보드 반영
    else playInstance.moveBlockLeft();      //불가한 경우 반대쪽으로 이동
}
else if(inputString.equals("s")) // 아래쪽으로 이동(move)
{
    playInstance.moveBlockDown();
    if(playInstance.isValid()) playInstance.addBlockToBoard();      //보드 반영
    else playInstance.moveBlockUp();      //불가한 경우 반대쪽으로 이동
}
else if(inputString.equals("r")) // 오른쪽으로 회전(rotate)
{
    playInstance.rotateBlockRight();
    if(playInstance.isValid()) playInstance.addBlockToBoard();      //보드 반영
    else playInstance.rotateBlockLeft();      //불가한 경우 반대쪽으로 회전
}
else
{
    continue;
}
playInstance.printScreen();
```

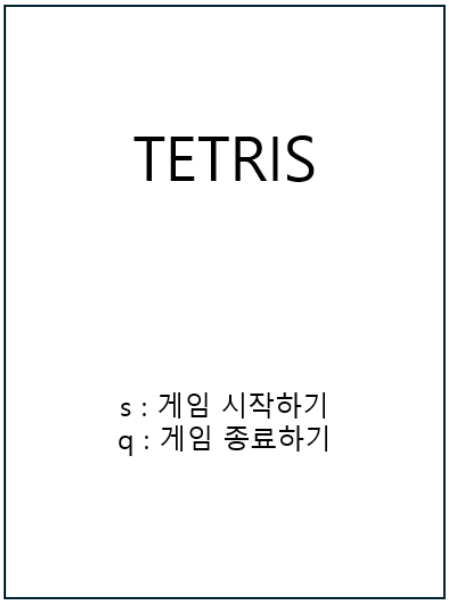
```
                }

                if(playInstance.isEnd())      //block이 더 내려갈 수 없는 경우
                {
                    playInstance.clearLine();      //라인 삭제 및 점수 증가
                    playInstance.printScreen();      //board가 수정되었을 수 있으므로 다시 출력
                    playInstance.setCurrentBlockNull();      //현재 block을 존재하지 않는 것으로 지정
                    if(playInstance.isGameOver())      //게임이 종료되었는지 검사 (block이 더 이상 보드에 들어갈 수 없을 때)
                    {
                        playInstance.endInputThread();      //입력 스레드 종료
                        playInstance.endBlockThread();      //block 이동 스레드 종료
                        playInstance.printScore();      //점수 출력
                        break loopOut;      //지정된 label로 반복문 빠져나감
                    }
                    break;      //첫 번째(바깥)while문 종료
                }
            }
        }
    }
}
```

4. 실행 화면

1) 시작 화면

1-1) 시작 화면에서 s를 누르면 게임이 시작된다.

1-2) 시작 화면에서 q를 누르면 게임이 종료된다.



정상적으로 동작하는 경우, 아래의 메소드들이 구현된 것.

- StartStateInterfaces

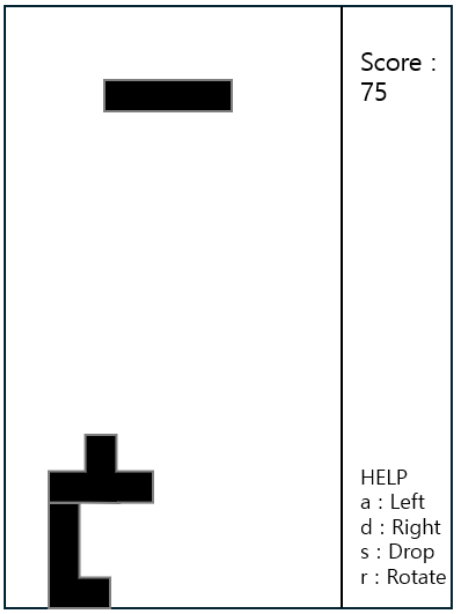
getStartStateInstance()

- StartStateInterface

printScreen()

2) 게임 시작

2-1) 상단에서 블록이 내려오고, 특정 키(a, d, s, r)로 블록을 이동/회전시키며 플레이한다.



정상적으로 동작하는 경우, 아래의 메소드들이 구현된 것.

- PlayStateInterfaces

getPlayStateInterface()

- PlayStateInterface

startInputThread()

startBlockThread()

generateBlock()

printScore()

addBlockToBoard()

isInputEntered()

getInputValue()

moveBlockLeft()

moveBlockRight()

moveBlockUp(

moveBlockDown()

printScreen()

isValid()

rotateBlockRight()

rotateBlockLeft()

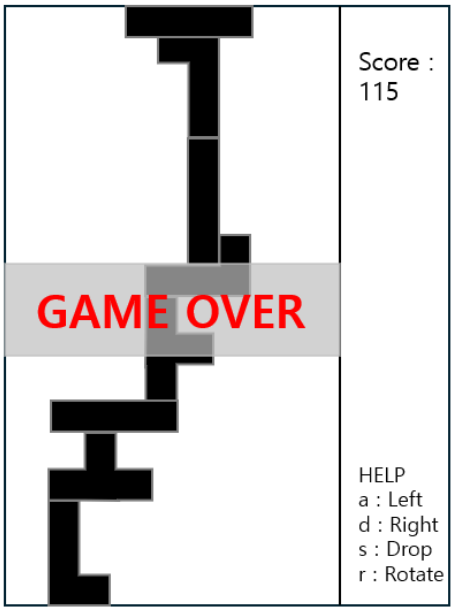
isEnd()

clearLine()

setCurrentBlockNull()

3) 게임 종료

3-1) 블록이 화면 상단까지 쌓이면 게임이 종료된다.

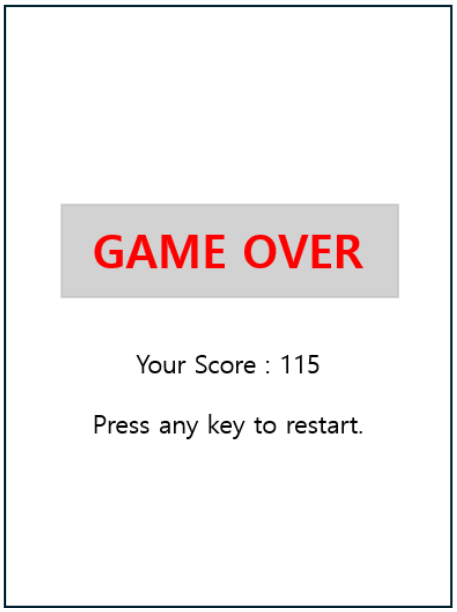


정상적으로 동작하는 경우, 아래의 메소드들이 구현된 것.

- PlayStateInterface

isGameOver()

3-2) 최종 점수와 함께 게임 결과가 표시된다.



정상적으로 동작하는 경우, 아래의 메소드들이 구현된 것.

- PlayStateInterface

endInputThread()

endBlockThread()