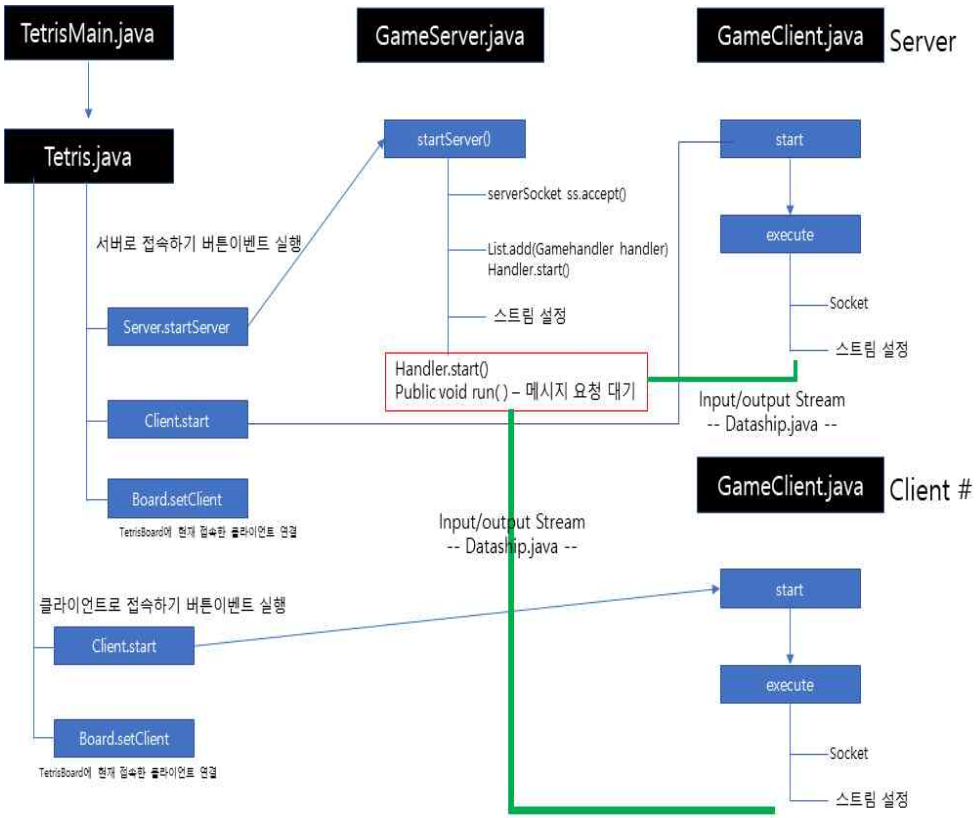
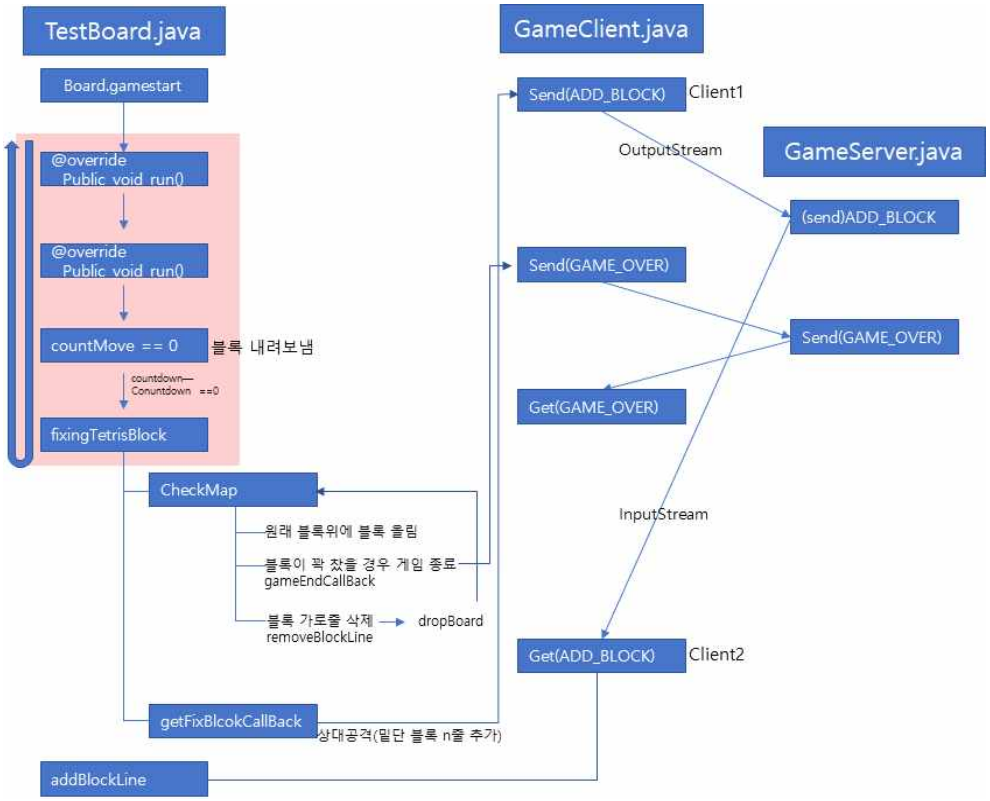
2013112130_컴퓨터공학과_정재엽 (2017-2-OSSP-jalhaebojo-2 / 멀티플레이 테트리스 분석)

서버 연결



게임 시작 후



이 과정을 거치게 되면 서버로 접속한 클라이언트에서는 시작버튼이 활성화되고
시작하기 버튼을 누르게 되면 오른쪽의 과정이 실행이 된다.
서버 / 클라이언트는 서로 다른 쓰레드를 이용하여 실행.

서버실행시
1) 클라이언트들간의 통신을 위한 쓰레드실행
2) 클라이언트 쓰레드 실행

클라이언트 실행시
1) 클라이언트 마다 쓰레드 실행
2) 따라서 1:1이 아닌 다수의 플레이어가 동시에 연결가능한 테트리스 실행

◆ 프로젝트의 구조는 왼쪽과 같고 주요클래스는 빨간색으로 표시하였습니다.

- com.tetris.classes package
  테트리스 블록의 그래픽 및 이동을 정의
- com.tetris.controller package
  블록의 이동 및 회전을 정의합니다.
- com.tetris.main
  게임의 시작지점
- com.tetris.shape
  블록의 모양 정의
- com.tetris.network
  서버와 클라이언트간의 소켓 통신을 정의
  DataShip을 이용한 소켓 통신.
- com.tetris.window
  ① Tetrs.java : 게임 시작과 종료에 관여
  ② TetrisBoard : 화면 및 키/버튼 이벤트 관여

◆ 서버 접속
TetrisMain.java->Tetris.java
itemServerStart.addActionListner(this)
⇨ public void actionPerformed(ActionEvent e)
⇨ itemServerStart라면
① GameServer.java
server.startServer();를 통해 쓰레드 생성
및 list에 핸들러 추가 및 index(플레이어) 추가
serverSocket.accept()를 통해 요청 대기
handler.start()
② GameClient client 초기화 후 client.start()
③ GameClient.java
public boolean start()에서 execute()실행
public boolean execute를 통하여
client의 소켓 및 스트림 설정을 합니다
④ Tetris.java
서버의 연결준비가 완료되면
시작버튼이 활성화됩니다.
시작버튼을 누름과 게임은 시작합니다.

==========

ServerSocket : 클라이언트 요청을 기다림
Socket : 클라이언트 소켓 요청시 연결
따라서 서버에도 소켓이 2개가 필요

==========

◆ 클라이언트 접속
itemClientStart.addActionListner(this)
⇨ public void actionPerformed(ActionEvent e)
⇨ itemClientStart라면
① clinet.start() 실행 ②GameClient.java에서 public boolean execute실행
③ 소켓연결 및 스트림을 연결합니다.

◈ 게임시작
DataShip.java에서는 소켓통신간 사용할 지정어들을 int로 정해놓았습니다.
Setter&Getter를 통해 소켓통신을 합니다.

```
public static final int CLOSE_NETWORK = 0;
public static final int EXIT = 1;
public static final int SERVER_EXIT = 2;
public static final int PRINT_SYSTEM_OPEN_MESSAGE = 3;
public static final int PRINT_SYSTEM_ADDMEMBER_MESSAGE = 4;
public static final int GAME_START = 5;
public static final int GAME_OVER = 6;
public static final int ADD_BLOCK = 7;
public static final int SET_INDEX = 8;
public static final int PRINT_MESSAGE = 9;
public static final int PRINT_SYSTEM_MESSAGE=10;
public static final int GAME_WIN=11;
```

서버접속이 완료가 되었으면 TetrisBoard에서 client를 받고있습니다.
TeetrisBoard에서는 client들의 키이벤트를 받습니다.

```
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_ENTER){
        messageArea.requestFocus();
    }
    if(!isPlay) return;
    if(e.getKeyCode() == KeyEvent.VK_LEFT){
        controller.moveLeft();
        controllerGhost.moveLeft();
    }else if(e.getKeyCode() == KeyEvent.VK_RIGHT){
        controller.moveRight();
        controllerGhost.moveRight();
    }else if(e.getKeyCode() == KeyEvent.VK_DOWN){
        controller.moveDown();
    }else if(e.getKeyCode() == KeyEvent.VK_UP){
        controller.nextRotationLeft();
        controllerGhost.nextRotationLeft();
    }else if(e.getKeyCode() == KeyEvent.VK_SPACE){
        controller.moveQuickDown(shap.getPosY(), t
        this.fixingTetrisBlock();
    }else if(e.getKeyCode() == KeyEvent.VK_SHIFT){
        playBlockHold();
    }
    this.showGhost();
    this.repaint();
}
```

서버로 접속한 클라이언트가 게임시작버튼을 누름으로 시작됩니다.
게임시작버튼을 누르면 아래와 같이 게임이 시작이 됩니다.
① TetrisBoard.java
public void actionperformed(ActionEvent e)
⇨ client.gameStart
② GameClient.java
public void gameStart(int speed)를 통해 서버로 GAME_START메세지를 보냅니다.

③ GameServer.java
public void gameStart
⇨ GameClient.java로 GAME_START 메시지를 보내 게임을 시작상태로 만듭니다.
④ GameClient.java
public void reGameStart
⑤ Tetris.java
tetris.gameStart
board.gameStart
⑦ TetrisBoard.java
public void gameStart // 게임이 실행됩니다.

==========

◈ 게임 컨트롤 부분
이렇게 게임이 시작되면 TetrisBaord에서 키이벤트 입력을 받습니다.
TetrisBlock.java를 통해서 블록들을 이동과 회전을 담당하게 됩니다 .

com.tetris.shape packge에서는 블록들의 모양을 설정합니다.
com.tetris.controller package는 블록의 이동 및 회전 명령을 받습니다.
com.tetris.classes packge 블록 이동/회전 로직 부분
- Block : 블록의 좌표 로직을 설정합니다.
- TetrisBlock.java블록의 이동 logic을 실행합니다.

① TetrisBoard.java
public void gameStart(int speed)
⇨ 게임화면(가운데부분) 맵, 도형, 고스트 뷰들을 세팅하고
세팅을 토대로 화면의 HOLD부분과 NEXT부분을 출력합니다.
shap = getRandomTetrisBlock()
ghost = clone_getBlockClone(shape, true)
shap은 도형을 받아오며, getBlockClone은 도형이 나타나는 최초위치 및 고스트뷰 설정을 합니다.
⇨ @Override
public void run()의 while문을 통해 게임 진행.
< 변수설명 >
- countMove : 속도가 빨라질수록 countMove의 크기는 작아지며 TetrisBlock의 moveDown호출이 증가하며 테트리스 블록이 내려오는 속도가 빨라집니다
- countDown : 블록이 움직일수 없는 경우(controller.moveDown() == false), 블록이 위에 놓이도록 하는 메소드호출 fixingTetrisBlock실행 전 딜레이를 줍니다. 따라서 countDown에 들어가는 수가 클수록 블록이 놓이는 데까지 걸리는 시간이 오래걸립니다. 이 시간컨트롤을 통해 초보자에겐 느리게 놓이도록, 숙련자에게는 빨리 놓이도록 해볼 수도 있습니다.

private void fixingTetrisBlock
- public void CheckMap()
⇨ 내려온 블록을 원래 있던 블록위에 올립니다.
⇨ 만약에 줄이 꽉 찼을 경우 게임을 종료합니다. (gameEndCallBack)
⇨ 가로줄이 가득찼다면 그 줄을 지웁니다. ⇨ 다시 CheckMap을 실행하여 줄을 다지웁니다.
- getFixBlockCallBack
⇨ 지워진 줄만큼 addBlock을 현재클라이언트에 전달
⇨ 현재 클라이언트 ADD_BLOCK 메시지 보냄 ⇨ 서버 ADD_BLOCK 응답
⇨ 상대 클라이언트 ADD_BLOCK 메시지 받음
⇨ 상대 클라이언트 board.reAddBlock ⇨ 밑단 블록 증가

==========

< 현재 소스에서 사소한 문제 >

현재 주어진 소스에서 TetrisBoard.java의 checkMap의 게임끝내는 조건이 잘못돼있습니다.
다른 블록들의 시작지점은 가운데 블록들 내려오는 화면을 기준으로 좌상단을 (0,0)으로 했을 때 (4,1)에서 시작하도록 되있기에 게임 종료 조건으로 (2 < x < 7 && y ==1)로 되어있습니다.
하지만 위의 스크린샷과 같이 사각형의 블록의 경우 0,4에서 시작하기 때문에 위와 같은 상황이 발생할 수 있었습니다.