java프로젝트 진행 사항

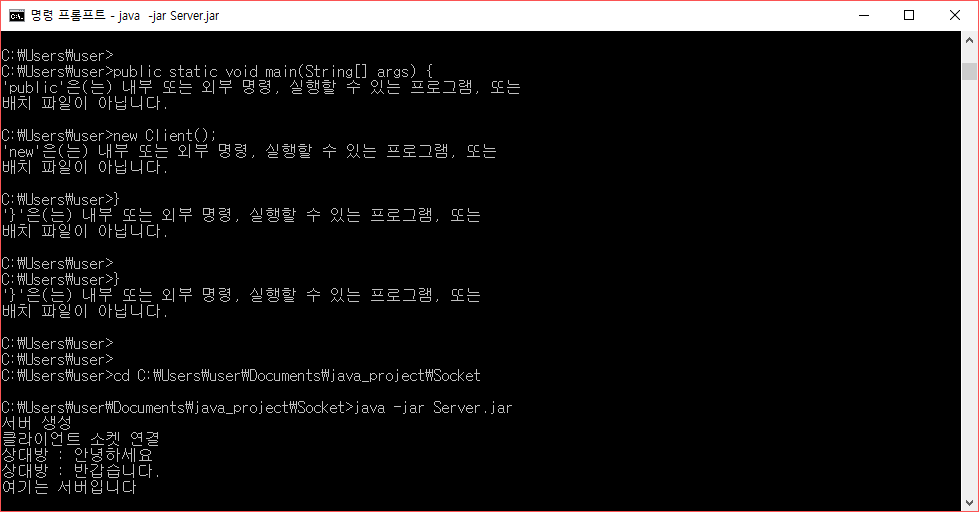
* 메모
  + JFrame.setResizable(boolean) : boolean 값에 따라 사용자가 프레임 사이즈를 임의 조절 할 수 있는지 여부를 나타냄.
  + JFrame.setLocationRelativeTo(null) : 프로그램 최초 실행 시 화면 정중앙에서 실행
  + JFrame.setVisible(boolean) : 창이 보이도록 하는 것. 기본적으로는 false
  + 더블 버퍼링 : 자바에서 이미지를 바로 화면에 출력하는 프로그램은 버퍼링을 심하게 하기 때문에 이용하는 프로그램 기법. 현재 프로그램이 전체화면의 크기에 맞는 이미지를 매 순간마다 생성해서 원하는 컴포넌트만 화면에 출력하는 방식.
  + Graphics.repaint() : 매 순간 새로고침된 페인트를 다시 그려주는 메소드
  + Image.getGraphics() : 이미지에서 그래픽 추출
  + ImageIcon.getImage() : 이미지 추출
  + 클래스파일명.class.getResource(파일루트) : 파일 루트를 클래스파일이 존재하는 디렉토리를 기준으로 찾아줌
  + JPanel.setPreferredSize(Dimension d) : 판넬의 사이즈 설정
  + Dimension(x, y) : 좌표 범위 x, y 크기를 저장
  + JPanel.setBackgraound(Color bg) : 백그라운드 칼라 조정
* 프로젝트 진행상황
  + 9/ 20
    - 게임보드와 상단바 포토샵으로 디자인
    - 이미지 추가를 해보려 했으나 번번히 실패
    - 프로그램 아이콘 설정 완료
  + 10/15/17
    - 이미지를 paint를 통해 추가하는 방법을 익힘
    - 우선 JFrame을 상속받는 ScreenDesign에 게임보드 이미지를 추가해봄
    - 상단바 디자인을 같이 넣어봄
    - JFrame 안에 모든 그래픽설정을 넣었더니 나중에 각각의 위치에 기능들을 넣을때 곤란해질 것을 염려함
    - JPanel을 통해 그래픽 표현하는 방법을 직접 시행착오를 통해 찾아봄
    - 처음에는 JPanel의 사이즈와 위치를 정해주지 않아서 먼저 추가한 JPanel이 나중에 추가한 JPanel의 이미지에 덮여버려서 화면에 보이지 않음
    - JPanel의 사이즈와 위치를 지정하는 방법을 찾아봄
    - 바둑판 라인 디자인 추가
    - 바둑돌 위치를 저장하고 리턴하는 클래스 Stones 제작
    - 바둑돌이 이미 놓인 위치에는 새로 놓을 수 없도록 Stone.isEmpty메소드를 추가
    - 마우스 액션리스너가 반응할 수 있는 리스너 영역을 논리적 알고리즘으로 구현하였다. 다만 액션 리스너를 단일 패널에 적용시키는 법을 찾지 못해서 실패.
  + 10/16
    - 마우스 액션리스너를 통해서 바둑돌을 놓을 수 있게 성공함
    - 히트박스 구현에 성공함
    - 현재 여러 번 클릭시 이미 놓아진 곳 위에 놓아지는 오류 발생
    - 여러 번 클릭 시 돌 색깔이 변하지 않는 오류 발생
  + 11/5
    - 소켓의 기초적인 사용에 대한 연습.
      * 코드
* Server

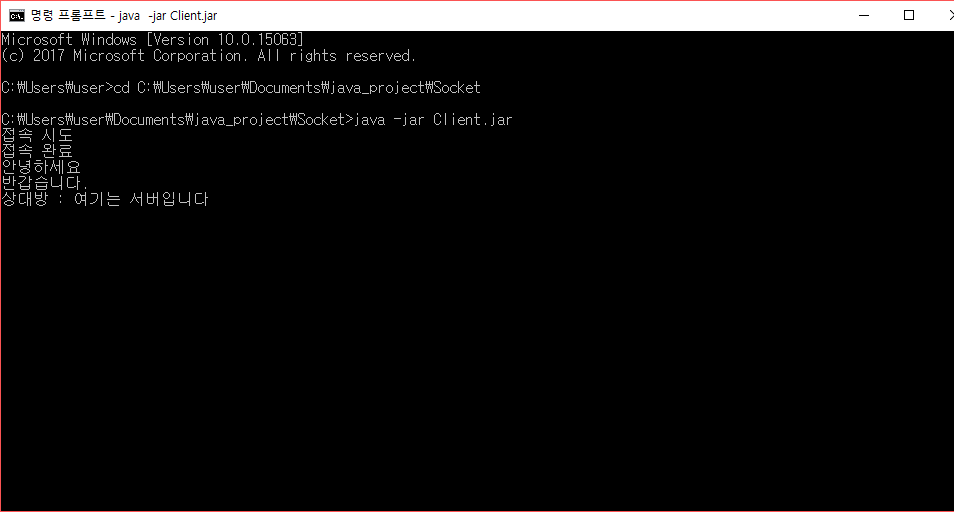
|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | import java.io.DataInputStream;  import java.io.DataOutputStream;  import java.net.ServerSocket;  import java.net.Socket;  import java.util.Scanner;  public class Server {  private ServerSocket serverSocket;  private Socket clientSocket;  private DataInputStream dataInputStream;  private DataOutputStream dataOutputStream;  //1. 데이터를 계속 전송 쓰레드  //2. 데이터를 계속 수신 쓰레드  public void serverSetting() {  try {  //localhost, 10002  serverSocket = new ServerSocket(10005); // 바인드  System.out.println("서버 생성");  clientSocket = serverSocket.accept(); // 어셉트.  //소켓이 접속 완료 된 부분  System.out.println("클라이언트 소켓 연결");  } catch (Exception e) {  }  }  public void closeAll(){  try {  serverSocket.close();  clientSocket.close();  dataInputStream.close();  dataOutputStream.close();  } catch (Exception e) {  }  }  public void streamSetting() {  try {  dataInputStream = new DataInputStream(clientSocket.getInputStream());  dataOutputStream = new DataOutputStream(clientSocket.getOutputStream());  } catch (Exception e) {  }  }  public void dataRecv() {  new Thread(new Runnable() {  boolean isThread = true;  @Override  public void run() {  while(isThread) {  try {  String recvData = dataInputStream.readUTF();  if(recvData.equals("/quit"))  isThread = false;  else  System.out.println("상대방 : "+recvData);  } catch (Exception e) {  }  }  }  }).start();  }  public void dataSend() {  new Thread(new Runnable() {  Scanner in = new Scanner(System.in);  boolean isThread = true;  @Override  public void run() {  while(isThread){  try {  String sendData = in.nextLine();  if(sendData.equals("/quit"))  isThread = false;  else  dataOutputStream.writeUTF(sendData);  } catch (Exception e) {  }  }  }  }).start();  }  public Server() {  serverSetting();  streamSetting();  dataRecv();  dataSend();  }  public static void main(String[] args) {  new Server();  }  } |

* Client

|  |
| --- |
| import java.io.DataInputStream;  import java.io.DataOutputStream;  import java.net.Socket;  import java.util.Scanner;  public class Client {  private Socket clientSocket;  private DataInputStream dataInputStream;  private DataOutputStream dataOutputStream;  //1. 데이터를 계속 전송 쓰레드  //2. 데이터를 계속 수신 쓰레드  public void connect() {  try {  System.out.println("접속 시도");  clientSocket = new Socket("127.0.0.1", 10005);  System.out.println("접속 완료");  } catch (Exception e) {  e.printStackTrace();  }  }  public void dataRecv() {  new Thread(new Runnable() {  boolean isThread = true;  @Override  public void run() {  while(isThread) {  try {  String recvData = dataInputStream.readUTF();  if(recvData.equals("/quit"))  isThread = false;  else  System.out.println("상대방 : "+recvData);  } catch (Exception e) {  }  }  }  }).start();  }  public void dataSend() {  new Thread(new Runnable() {  Scanner in = new Scanner(System.in);  boolean isThread = true;  @Override  public void run() {  while(isThread){  try {  String sendData = in.nextLine();  if(sendData.equals("/quit"))  isThread = false;  else  dataOutputStream.writeUTF(sendData);  } catch (Exception e) {  }  }  }  }).start();  }  public void streamSetting() {  try {  dataInputStream = new DataInputStream(clientSocket.getInputStream());  dataOutputStream = new DataOutputStream(clientSocket.getOutputStream());  } catch (Exception e) {  }  }  public Client() {  connect();  streamSetting();  dataSend();  dataRecv();  }  public static void main(String[] args) {  new Client();  }  } |

* + - * 실행 화면





* + - 다중 Client 연결로 변경
      * 위 Server 코드에서 일부분을 변경
        + private Socket clientSocket;

**private** Map<String, Socket> clientSocket = **new** HashMap<>();

* + - * + serverSetting() 함수에 추가된 부분

int i = 0;

while(i<3) {

i++;

clientSocket.put(String.*format*("%s", i),serverSocket.accept());

// 소켓이 접속 완료 된 부분z

System.*out*.println("클라이언트 소켓 연결");

}

* + - * + closeAll() 함수에 추가된 부분

**for**(**String** **x** : clientSocket.keySet())

clientSocket.get(x).close();

}

* + - 그러나 inputStream과 outputstream을 어떻게 변경 해주어야 할지 모르겠어서 고민중임.
  + 11/13/17
    - 기존 마우스 리스너의 비효율적인 계산 방식 개선
      * 기존에는 바둑판에 존재하는 모든 크로스 포인트의 좌표를 1차원 배열에 저장한 뒤, 마우스 입력이 들어오면 배열의 모든 원소들에 대해 좌표사이의 거리를 구하는 피타고라스 정리를 이용한 식으로 해당 크로스포인트에 해당하는지 확인하여 이를 행렬로 변환하였다가 다시 좌표형식으로 바꾸어 화면에 뿌리는 식으로 구현을 했었다. 하지만 이 경우 시간 복잡도가 O(n)으로, 너무 많은 자원을 잡아먹기때문에 알고리즘 변화를 시도하였다. ( 비효율적인것은 알고 있었으나, 클라이언트 초기 구성 당시에 화면구성과 마우스 리스너의 이용방법을 익히는데에 생각할 것이 많아서 생각나는대로 구성을 했기 때문에…)
      * 기존의 방식에서 사이즈 변화에 대응 할 수 있으면서 시간 복잡도를 낮출 수 있는 방법으로 변형시켰다.
        + 마우스 리스너를 통해 좌표 값을 받음

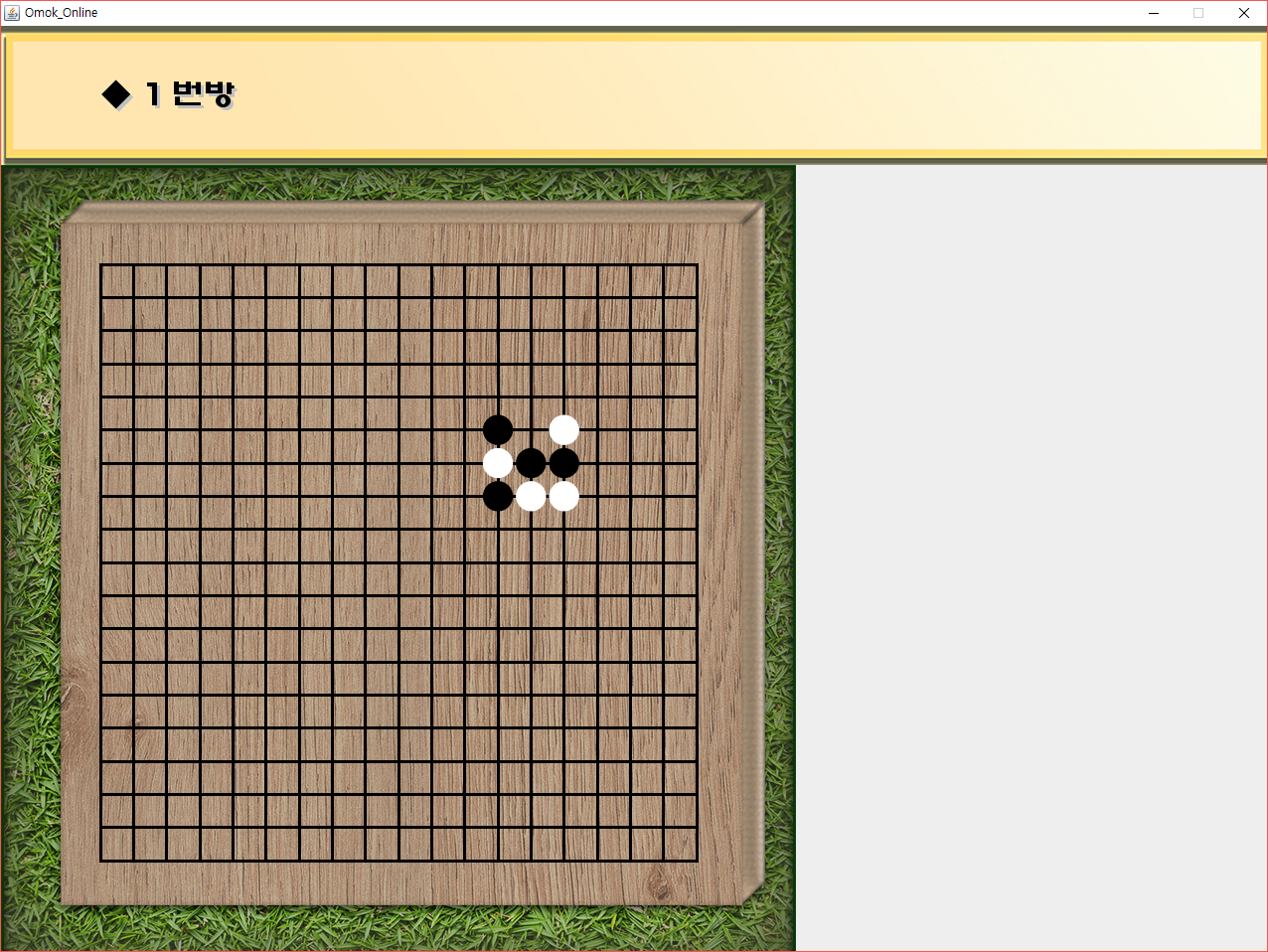
좌표 값에서 BOARDLINE\_START\_WIDTH을 우선 뺀다.

위 값을 LINE\_SPACE로 나누어서 몫과 나머지를 각각 저장해 둔다.

나머지가 LINE\_SPACE의 절반보다 크다면 몫+1 위치의 크로스포인트, 절반보다 작다면 몫 위치의 크로스포인트를 리턴하여 준다.

이렇게 나온 값은 마우스 클릭 위치에서 가장 가까운 점을 O(1)의 시간 복잡도로 리턴해준다.

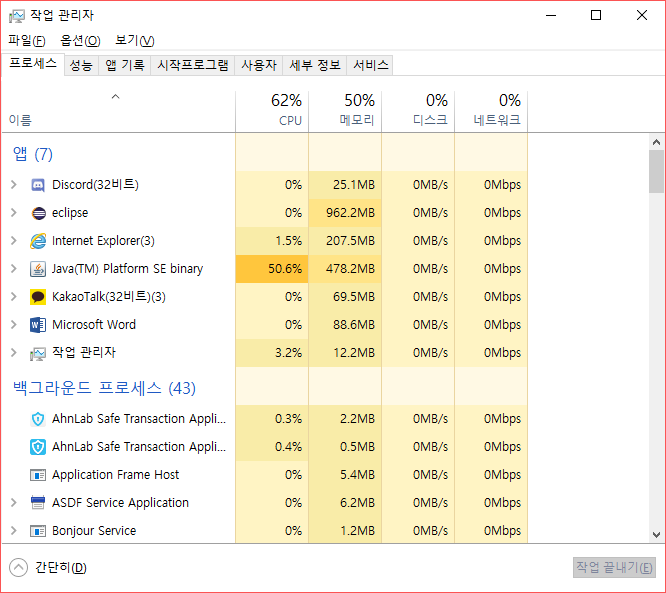
* + - * 위 방법을 통해 해당 문제를 완벽하게 해결하였다.
    - 서버를 통해 stone 정보를 주고받을 수 있게 만들기 위하여 소켓 관련 내용을 검색함. 아직은 방법을 찾지 못함.
  + 11/20/17
* 역할 분담을 바꿈. GUI디자인 담당으로 바뀜.
* 기본적인 GUI는 이미 완료 했으므로 코드 내용 전달과 보수 위주로 진행하기로 함.
* 상단 바 디자인과 함께 게임시작 버튼을 추가했음. (버튼의 실행 내용은 아직 구현하지 않음.)
  + 변경 전 GUI



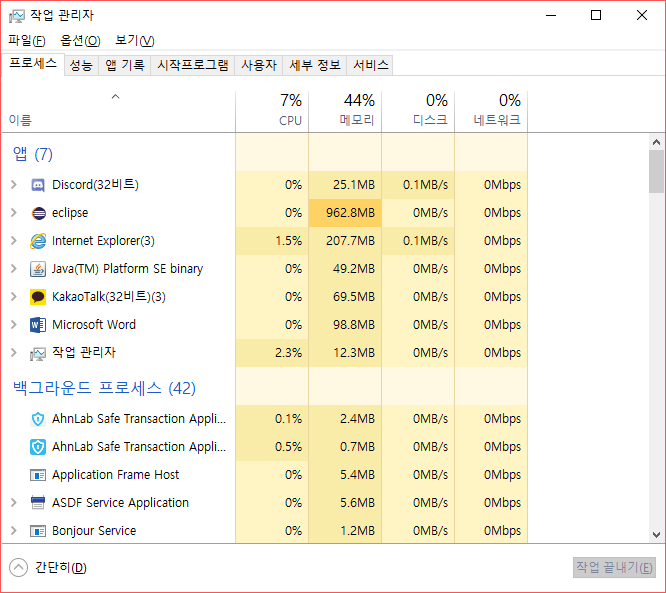
* + 변경 후 GUI

  
(오른쪽 아래 프레임은 채팅창을 위한 프레임. 다른 사람이 담당.)

* + 11/22/17
* 상단바 JPanel에 게임 스타트 버튼 클릭 감지 마우스 리스너를 만듦.
* 클릭이 감지되면 게임시작 버튼이 클릭되었는지 확인한 후 원하는 역할을 할 수 있게 만듦. ( 현재는 서버와 통신을 위한 클래스 구현이 되지 않았기 때문에 단순히 “게임시작”을 콘솔 라인에 출력하는 것으로 되어있음.)
* 상단 바와 바둑판에 마우스가 올려져 있을 때 사용자 가시성을 위한 움직임을 넣기 위해 방법을 모색 중.
* 스크린이 너무 끊임 없이 리셋 되는 것을 막기 위해 바둑판에 돌이 놓아진다는 신호가 올 때 마다 화면이 새로 출력 되게 함.
  + 기존에는 GameBoardDesign 안에 직접 리페인트를 넣는 바람에 화면이 모니터 주사율 이상으로 끊임없이 재 출력 되느라 CPU 사용량을 어마어마하게 많이 차지했다.



기존에 CPU 사용량을 보면 6세대 i3 데스크탑 CPU 기준으로도 이만큼이나 먹는다…



이후 repaint를 stone 클래스에서 돌을 놓는 입력이 있을 때 마다 화면을 재출력했더니 어마어마한 성능 향상이 되었다. GUI는 기존과 다름 없이 제대로 작동함.

* + 11/23/17
* 다른 팀원들의 프로젝트 코드 독해 속도를 늘리기 위해 effect 메소드를 추가하여 코드를 전부 읽을 필요 없이 effect메소드에 내용을 추가하면 되도록 수정함.
  + 12/06/17
* 기존에 Stones 클래스에서 putStone 호출시마다 repaint 하던 것을 RePaintThread를 생성하고 JPanel을 생성자 파라미터로 받게 하여 GameBoardDesign과 TopDesign 클래스의 생성자에서 RePaintThread 객체를 생성한 뒤 일정 간격(1000/60)ms 마다 화면을 다시 그리도록 하였다.
* TopDesign 클래스에 마우스 모션리스너를 추가하였고, 마우스가 움직이는 동작이 있는 경우 해당 위치가 게임시작 버튼이 위치한 곳인지 판단하여 마우스를 게임시작 버튼 위에 올릴 경우 사용자가 피드백 받을 수 있게 움직이도록 만들었다.
  + 마우스 리스너 호출시 TopDesign에서 this를 마우스 리스너의 생성자 파라미터로 전달.
* 몇가지의 비효율적인 상수 코드 가시성 확보를 위해 변경.
* 게임 보드 위에 마우스 포인터가 올라가 있을 경우 유효 범위 안쪽인지 확인하여 돌이 놓일 위치를 미리보기 시켜줌.
* Stones 클래스를 삭제하고 재구성함
  + Stones 클래스의 경우 필요 이상으로 많은 메모리 공간을 차지하고 돌이 겹쳐 놓여지는 경우가 생겼다.
  + Stone 클래스는 byte[][] 자료형을 사용하여, 각 행렬 좌표의 위치에 어떤 돌이 놓여 있는지 혹은 비어 있는지를 바로 바로 확인할 수 있어서 오버헤드가 줄어들었다.
  + 12/06/17 +)추가
* 이미지 파일 path를 수정. Jar 파일 실행시 Main.ClassPath의 위치를 기준으로 잡던 이미지 위치가 잡히지 않아 수정 함.
* 마우스 클릭 이벤트로 putStone과 게임시작을 구현하다보니 마우스가 조금이라도 움직이면서 눌리게 되면 드래그로 인식하여 실행이 안되는 자잘한 버그가 설계 초기부터 존재했는데, 사고를 통해 마우스 클릭이 아닌 릴리즈(해제)시에 실행되게 변경함으로써 좀 더 사용성을 향상시켰다.
* 마우스가 눌린 상태로 이동되는 경우에 바둑돌 미리보기가 처음 클릭 된 위치에 남아있고 따라오지 않는 오류가 발생하여 이것이 드래그로 인식되어 발생하는 오류임을 인지하고 드래그 리스너에도 무브드 리스너에 있던 내용을 똑같이 추가하여 문제를 해결하였다.

* + 12/11/17
* 소스코드 통합에 들어감
* 이종석 팀원에게 Client 파일을 받아서 기존에 직접 Stone 에 바둑돌을 추가하던 것을 수정함.
  + GameBoardMouseListener 클래스의 MouseRealesed 메소드를 통해 실행되는 effect메소드에서 putStone을 빼고, 싱글턴 스레드인 Client에 Client.getInstanse().dataSand() 로 서버에 돌을 놓았다는 프로토콜을 생성하여 보낸다.
  + Stone클래스의 putStone은 Client 클래스의 dataRecv 메소드의 스위치문을 통해 바둑돌이 놓였다는 프로토콜을 받으면 호출되도록 수정하였다.
* 게임시작 전에는 게임보드에 바둑돌이 놓이지 않도록 수정함.
  + 게임시작 버튼으로 인한 실행인 effect 메소드에 버튼 눌림 확인용 출력문 대신 dataSand를 통해 서버로 게임시작 요청을 보낸다.
  + Client 클래스의 dataRecv 메소드의 스위치문을 통해. 게임시작 승인 프로토콜을 받으면 GameBoardDesign의 static boolean 변수인 isStarted 를 true로 바꿔주고, 이 때부터 게임보드에 바둑돌을 놓을 수 있도록 구현하였다.
* 게임이 종료된 상태에서 다시 한번 게임이 시작된 경우, 게임시작 승인이 되면 게임을 이어서 할 수 있도록 Stone클래스의 stone을 전부 초기화하여 바둑돌을 지우는 메소드 resetMatrix를 구현하였다.
  + 12/17/17
* 서버와 연결하는 작업과정에서 일부 GUI를 개선함. 기존에는 한 클라이언트에서 두가지 색의 바둑돌을 모두 놓았기 때문에, 바둑돌 미리보기 역시 두가지 색 모두 표시했지만, 서버와 연결 후에는 자기 자신의 미리보기만 나타내면 되기 때문에 서버에서 게임시작시 받아오는 myId를 이용하여 자신의 색을 받아 myColor에 저장하여 자신의 차례일 때만 미리보기가 나타나게 하였다.