2010.08.20

***1.***

**public** **class** Client **extends** Network

**public** **void** Connect(String ip)

{

**try**

{

connection = **new** Socket(ip, portNum); 10000 에서 portNum 으로 인자 변경.

setOutOb();

setListen();

} **catch**(ConnectException e) {

JOptionPane.*showMessageDialog*(**null**, "IP 와 Port 를 확인하세요.");

//**TODO** 예외발생시 처리해야 함.

}

**catch** (UnknownHostException e)

{

// **TODO** Auto-generated catch block

e.printStackTrace();

} **catch** (IOException e)

{

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

* 포트넘버를 10000 이외의 것으로 변경했을 때 서버에 접속되지 않는 문제 해결.
* Exception error 도 추가로 처리.

***2차 수정***

**public** **void** Connect(String ip) // 주어진 IP 와 portNum 멤버변수를 사용하여 서버에 접속하는 메소드.

{

**try**

{

clientSocket = **new** Socket(ip, portNum); // 서버 소켓에 연결.

setOutOb(); // output 스트림을 설정하여 서버에 데이터를 전송할 수 있게 준비한다.

setListen(); // input 스트림을 설정하여 서버에서 데이터를 수신할 수 있도록 준비한다.

} **catch**(ConnectException e) {

JOptionPane.*showMessageDialog*(**null**, "IP\t: "+ip+"\nPort\t: "+portNum+"\n호스트 또는 port가 올바르지 않습니다.\n\n게임을 종료합니다.");

System.*exit*(0);

//**TODO** 종료 대신 다른 방법으로 ip 와 port 를 새로 받아 접속할 수 있도록 해야 함.

}

* 변수 connection 을 clientSocket 으로 변경
* 예외처리구분에서 ip 와 port 가 부정확할 경우 메시지를 띄우고 프로그램을 종료하도록 수정.
* TODO 리스트 확인

***2.***

LobbyWindow 의 명칭들(한글로 되어있던 것들)을 영어로 변경.

***3.***

ChatWindow 의 Background color 를 Color(192,192,192) 로 변경.

***4.*** ClassCastException in Client 클래스. => 수정해야 함. DataHeader cannot be cast to String class

***5.***

**public** **class** Game **implements** Serializable

**public** **void** getBlock(**int** blockNum) { // center 에 있던 block 을 가져오는 함수. player 의 소유가 되도록 설정된다.

last = blocks.get(blockNum); // 바닥에 깔린 block 들 중 blockNum 에 해당하는 block 을 선택.

hand.add(last); // player 의 block 에 추가한다.

blocks.remove(blockNum); // 바닥에서 그 block 을 제거한다.

// hand.get(hand.size()-1).setOwn(true);

last.setOwn(**true**);

module.m\_NetTaget.SendOb(**new** DataHeader("game2", **new** GameData(module.GC)));

}

* 함수 호출을 길게 하지 않고 위에서 썼던 last 변수를 써서 똑 같은 기능을 구현함. -> 가독성 증가

***6.***

**Client class & server class**

**public** **void** dataEvent(DataHeader data)

{

String head = data.getHeadData();

**if**(head.equals("chat"))

m\_Taget.AddChatString((String)data.getData());

**if**(head.equals("game"))

m\_Game.setGC((Game)data.getData());

**if**(head.equals("game2"))

m\_Game.setGC((GameData)data.getData());

**if**(head.equals("pass")) // 이전 player 가 턴을 pass 했을 때,

**if**(m\_Game.getPlayOrder() == ((Integer)data.getData()).intValue())

{

SendChatMsg("턴 입니다.");

m\_Game.turn();

}

**if**(head.equals("fnsh")) { // client 가 게임을 나간 경우

//**TODO** 게임 상태(block 과 playser) 갱신.

// 그 다음 할 일 생각 안남. 추후 고려.

}

}

* Player 가 게임이 끝나기 전에 게임을 나가는 경우를 위한 데이터 처리 부분을 추가. 기능은 추가하지 않음.

***7.***

Class ClientData

**public** **void** close() **throws** IOException

{

outOb.close();

inOb.close();

clientSocket.close();

}

* Client input/output 스트림을 닫는 것은 client 와 연결을 끝내는 것이므로, 소켓도 함께 종료시키도록 코드를 추가.

**8.**

AskDlg class 를 GameWindow class에서 분리하여 독립적으로 사용될 수 있도록 수정.

**9.**

Player class 를 Game class 에서 분리하여 독립적으로 사용할 수 있도록 수정.

***10.***

**public** **class** GameProcess

**public** **void** moveBlock(**int** blockIndex) // center 에 있는 block 들을 player 에게 옮기는 함수.

{

Player me = gameControl.getPlayers().get(myPlayOrder);

**if**(gameControl.getBlocks().get(blockIndex).getNum()==12) // 선택한 block 이 Joker 일 경우 diag 대화상자의 버튼 색을 변경하고, joker 를 놓을 장소를 player 에게 물어본다.

{

String color;

**if**( gameControl.getBlocks().get(blockIndex).getColor() == 0 )

color = "black";

**else**

color = "white";

AskDlg diag = **new** AskDlg("조커가 대신할 숫자를 선택하세요", color);

**int** num = diag.getNum();

gameControl.getBlocks().get(blockIndex).setSortingNum(num);

me.setHasjoker();

}

me.getBlock(gameControl.getBlocks(), blockIndex); // 이 block 을 player 에게 전달.

* Center 에서 player 에게 block 을 전달하는 과정을 수정하여, 이전에 joker 가 player 에게 전달되지 않았던 문제를 해결
* AskDlg 클래스를 독립시킨 후로 joker 를 배치할 위치를 묻기위한 대화상자를 새로이 생성하여 player 에게 직접 joker 의 배치위치를 묻도록 변경.
* gameControl.getBlocks().get(blockIndex).setSortingNum(num);
  + 이 구문은 Block 클래스를 변경하여 block 의 숫자와 정렬될 때의 숫자를 따로 두어서, 모든 플레이어가 joker 를 가진 player 의 blocks 를 정확히 알 수 있도록 하기 위해 추가한 구문.  
    즉 Block 클래스는 block 자체의 숫자를 위한 변수와, sortNum 이라는 정렬을 위한 변수를 따로 가지고 있어서 정렬될 때는 sortNum이 사용되고 block 을 맞출때는 일반 숫자 변수를 사용한다.

***11.***

**public** **class** Block **implements** Serializable {

**private** **boolean** joker;

**private** **int** sortingNum;

Block(**int** color, **int** num){

setColor(color);

setNum(num);

own = **false**; //누군가 가졌는지 여부. 바닥에 깔린 패와 player 가 가진 패 두 가지로 나뉘므로.

open = **false**; //공개여부. player 가 가진 패 중에 공개된 것과 안된 것이 있으므로.

joker = **false**;

sortingNum = num;

**if**( num == 12)

joker = **true**;

}

**public** **void** setSortingNum(**int** sortingNum) {

**this**.sortingNum = sortingNum;

}

**public** **void** setJoker(**boolean** joker) {

**this**.joker = joker;

}

**public** **int** getSortingNum() {

**return** sortingNum;

}

**public** **boolean** isJoker() {

**return** joker;

}

* Block 클래스를 변경하여 정렬할 때 사용할 sortNum 과 block 자체의 숫자를 위한 num 을 따로 두었다. -> joker 의 정렬이 어려운 점을 해결하기 위해 도입.
* Block 이 생성될 때 조커인지 아닌지 자동으로 설정되도록 생성자를 수정함.
* 정렬을 위한 변수인 sortNum 이 추가됨에 따라 이 변수를 위한 get/set 메소드를 추가.

***12.***

GameDate 클래스가 block 데이터들을 전달하는 방식을 변경

* 기존에 ArrayList<Block> 을 Block[] 배열로 변환하여 전달하던 방식에서 ArrayList<Block> 을 ArrayList<Block> 에 그대로 저장하여 전달하도록 변경함.
* 이렇게 변경함으로써 setGameControl(GameData newBlockState) 메소드에서 ArrayList<Block> 을 전달받는 방법이 매우 간단하게 바뀜.

**기존의 방식**

**public** **class** GameData **implements** Serializable

{

Block[] floor;

Block[][] p;

**public** GameData(Game gc)

{

floor = **new** Block[gc.blocks.size()];

gc.getBlocks().toArray(floor);

p = **new** Block[gc.players.size()][13];

**for**(**int** i=0; i<gc.players.size(); i++)

gc.players.get(i).hand.toArray(p[i]);

}

}

**변경된 방식**

**public** **class** GameData **implements** Serializable

{

// 객체가 생성되면 바닥에 깔린 block 들과 각 player 들이 가지고 있는 block 들의 정보를 저장한다.

ArrayList<Block> floor, tmp;

ArrayList<ArrayList<Block>> blocks;

**public** GameData(Game gc)

{

floor = **new** ArrayList<Block>(); // 바닥에 깔린 block 들을 가져옴.

**for**(**int** i = 0 ; i < gc.blocks.size() ; i++)

floor.add(gc.blocks.get(i));

blocks = **new** ArrayList<ArrayList<Block>>();

**for**(**int** i = 0 ; i < gc.players.size() ; i++) {

tmp = **new** ArrayList<Block> ();

**for**(**int** j = 0 ; j < gc.players.get(i).hand.size(); j++) {

tmp.add(gc.players.get(i).hand.get(j));

}

blocks.add(tmp); // player 0,1,2,3 들이 가진 block( hand ) 들을 p 에 저장한다.

}

}

}

***14.***

GameData 의 구조 변경 이후 클라이언트가 서버의 block 을 맞췄을 때, 서버쪽 게임화면에도 이 내용이 반영되도록 수정되었음.

* 서버가 클라이언트의 block 을 맞췄을 때는 이 내용이 반영되지 않고 있음.
  + 서버쪽의 게임데이터 전달 과정이 문제가 있는 것으로 추정됨.

***15.***

Joker 가 Open되었을 때의 이미지가 기존 이미지( b12, w12 ) 와 같았던 문제가 해결됨.

* 해결한 방법은 명확하지 않으며, 현재는 문제가 발생했을 당시 이미지를 변경한 후 컴퓨터를 재부팅하지 않아, 이전 Joker 의 이미지 캐시가  
  남아있었기 때문 아닌가 하는 생각이 있음.

블록 데이터 객체를 보내야 할 시기

1. 내가 center 에서 블록을 하나 골랐을 때 바로 업데이트 해줘야 한다.
2. 내가 상대방의 블록을 맞췄을 때 상대방에게도 이 사실을 알려주고 둘다 업데이트 해야 한다.
3. 내가 틀렸을 때 내 블록이 업데이트 되어야 하고 그 정보가 상대방에게도 전달되어야 한다.

서버가 데이터를 전달했을 때 open 된 block 의 정보가 전달되지 않는 이유에 대한 고민

1. 데이터 전송상에 문제가 있는 경우
   1. 현재로서는 이럴 가능성은 극히 낮음.
2. 서버가 데이터를 전송하는 시점에 문제가 있어서, block 이 open 된 후가 아니라 open 되기 전의 데이터를 전송하기 때문에 클라이언트가 이를 받지 못하는 경우
   1. Server 와 Client 클래스의 문제가 아니라면, 나머지 클래스들은 서버나 클라이언트 모두에서 동일하게 동작하므로( 또는 적어도 동일하게 동작해야 하므로 ) 이 역시 가능성이 낮으나 1번 경우보다는 가능성이 높음.

***16.***

**public** **class** Game **implements** Serializable { // 게임의 시작을 위한 초기 작업을 수행한다.

ArrayList<Player> players; // 게임 내의 player 들을 저장할 리스트.

ArrayList<Block> blocks; // 게임중 바닥에 깔려있을 block 들을 저장할 리스트.

**public** **void** setPlayers(**int** numOfPlayer) // 주어진 인자 만큼의 player 를 생성한다.

{

**기존의 방식**

// Player[] p = new Player[numOfPlayer];

// for(int i = 0; i < numOfPlayer; i++)

// {

// p[i] = new Player();

// players.add(p[i]);

// }

**변경된 방식**

**for**(**int** i = 0; i < numOfPlayer; i++)

players.add( **new** Player() );

}

* 게임을 시작하는 플레이서 수만큼의 객체를 생성할 때 배열 객체를 생성한 후 그 각 객체들에 대해서 다시한번 객체를 생성한 뒤 ArrayList에 추가하는 과정이 불필요하므로 수정.

***17.***

public class Block implements Serializable {

private int color; // 블럭의 색깔. 검정 == 0, 하양 == 1

private int num; //블럭의 숫자. 조커는 12.

private boolean joker;

private boolean open;

private int sortingNum; // 조커와 일반 블럭을 구분하지 않고, block 을 정렬하기 위해 선언된 변수. 조커는 기본적으로 12이지만 사용자의 선택에 따라 그 외의 숫자가 되기 때문에 이 변수가 필요함.

private boolean own; // 소유된 블럭인지 아닌지를 저장할 변수. transient는 직렬화할 때 제외할 변수임을 명시하는 기능을 함.

public void setJoker(boolean joker) {

this.joker = joker;

}

* setJoker() 메소드는 블록이 생성될 때를 제외하고는 다른 곳에서 쓰여서는 안되는 메소드이므로 삭제함.

***18.***

public class Client extends Network

{

Socket clientSocket;

ObjectOutputStream OOutStream; // 서버로의 객체 output 스트림을 연결할 변수.

ObListener OInStream; // client 가 server 로부터 들어오는 데이터를 계속 받을 수 있는 input 스트림을 연결할 변수.

class ObListener extends Thread // server 로부터 들어오는 input스트림을 받아 알맞은 동작을 하도록 하는 클래스.

{

ObjectInputStream OInStream;

boolean listenning = true;

// void dataEvent(DataHeader data) - done

public void dataEvent(DataHeader data) {

//입력된 데이터의 처리는 여기에 추가할것.

int flag = data.getFlag();

System.out.println("client dataEvent called");

switch(flag) {

case DataHeader.CHAT: // 데이터 헤더가 대화 이벤트일 경우.

myRoomWnd.AddChatString((String)data.getData());

break;

case DataHeader.GAME:

//if(gameProcess.gameControl == null || !gameProcess.gameControl.equals((Game)data.getData()))

gameProcess.setGameControl((Game)data.getData());

break;

case DataHeader.GAMEDATA:

gameProcess.setGameControl((GameData)data.getData());

break;

case DataHeader.PASS: // 턴 넘김 메시지를 받고서, client 자신이 플레이할 차례가 되면 턴을 시행한다.

if(gameProcess.getPlayOrder() == ((Integer)data.getData()).intValue())

gameProcess.turn();

break;

case DataHeader.MYORDER: // 해당 client 의 순서를 전달받으면 그 순서로 세팅.

gameProcess.setPlayOrder(((Integer)data.getData()).intValue());

break;

case DataHeader.TOTALCOUNT: // 총 인원수를 전달받으면 그 숫자대로 게임 GUI 를 세팅.

gameProcess.gameWndGUI.Setting(((Integer)data.getData()).intValue());

break;

}

}

}

}

* data.getHeadData() 메소드를 통해 data 객체가 어떤 객체인지를 판단했던 과거의 방식 대신, DataHeader 클래스에 public static final int 타입의 flag 들을 설정한 후 이 값을 받아와서 switch 문을 통해 작업을 할 수 있도록 수정함.
* 이렇게 함으로써 코드가 좀더 깔끔해지는 효과와 flag의 이름 변경이 용이해지는 효과가 있음.