

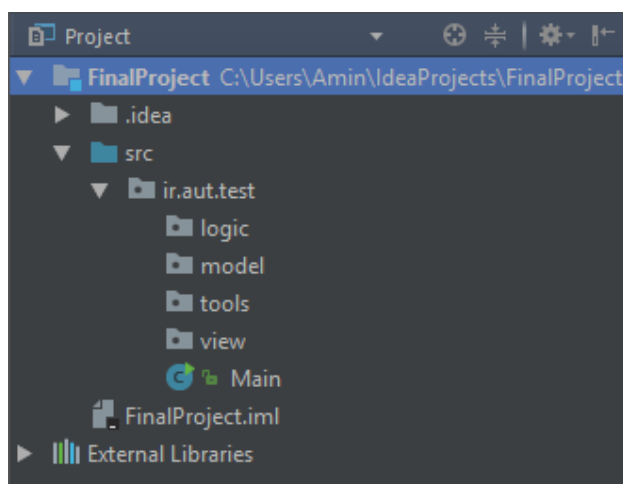


پروژه‌ی پایانی درس برنامه‌سازی پیشرفته بازی حدسی دو نفره به همراه گفتگو بر روی شبکه

- زمان تحویل از طریق بارگذاری در Moodle: پنج‌شنبه ۹۶/۴/۱۵ ساعت ۲۳:۵۵ (عدم بارگذاری در زمان مقرر به منزله‌ی عدم تحویل پروژه بوده و نمره‌ی پروژه صفر در نظر گرفته می‌شود)
- تحویل حضوری در سایت دانشکده: شنبه ۹۶/۴/۱۷ ساعت ۹ الی ۱۲

در این پروژه قرار است برنامه‌ای بسازیم که در آن حین انجام بازی، بتوان بر روی شبکه گفتگو کرد.

در ابتدا لازم است که پکیج‌بندی کلاس‌ها به صورت زیر انجام شود. چنانچه نام پکیج اصلی را (به عنوان مثال) `ir.aut.test` در نظر گرفته باشیم، در داخل آن چهار پکیج با نام‌هایی، همانند شکل زیر باید وجود داشته باشند که در هر کدام کلاس‌های مرتبط با هم، در زیر-پکیج‌هایی قرار می‌گیرند.



`logic`: شامل کلاس‌های مربوط به منطق برنامه شامل شبکه و ارتباطات، منطق بازی و ...

`model`: شامل کلاس‌های مربوط به تعامل با فایل

`tools`: شامل کلاس‌های کمکی و مفید (در صورت وجود)

`view`: شامل کلاس‌های مربوط به واسط کاربری

۱. فاز اول، ایجاد بستر ارتباطی بر روی شبکه

در ابتدای انجام پروژه، به ایجاد بستر ارتباطی بین دو کامپیوتر بر روی شبکه می‌پردازیم و برای این کار می‌خواهیم از کلاس‌های Socket و ServerSocket استفاده نماییم.

۱-۱. جزئیات طراحی

برای انتقال هر گونه پیام بین گیرنده و فرستنده باید **کلاسی مجزا با نامی متناسب** در نظر گرفته شود که از کلاس BaseMessage ارث‌بری نماید و برای آن، یک عدد **ثابت و یکتا** در کلاس MessageTypes در نظر گرفته شود که مشخص کننده نوع پیام است (در زیر کلاس BaseMessage نشان داده شده است که باید از آن **بدون تغییر** استفاده نمایید).

```
public abstract class BaseMessage {  
  
    protected byte[] mSerialized;  
  
    /**  
     * Fields are stored into serial bytes in this method.  
     */  
    protected abstract void serialize();  
  
    /**  
     * Fields are restored from serial bytes in this method.  
     */  
    protected abstract void deserialize();  
  
    /**  
     * Return message type code.  
     */  
    public abstract byte getMessageType();  
  
    public byte[] getSerialized() {  
        return mSerialized;  
    }  
}
```

همچنین در زیر کلاس MessageTypes نشان داده شده است که در آن ثابت‌های عددی مربوط به هر نوع پیام قرار می‌گیرند. همچنین ثابت عددی **PROTOCOL_VERSION** نشان‌دهنده شماره نسخه پروتکل ارتباطی است که گیرنده و فرستنده بتوانند به کمک آن، بررسی کنند که آیا پیام‌ها و محتوایاتی که در آن‌ها قرار دارند دارای قوانین یکسانی هستند یا خیر. بدیهی است که در صورت انجام تغییرات در کلاس‌های مربوط به پیام‌ها به این ثابت یک واحد افزوده می‌گردد.

```
public class MessageTypes {  
  
    /**  
     * Version of communication protocol  
     */  
    public static final byte PROTOCOL_VERSION = 1;  
  
    /**  
     * Code of request Login message  
     */  
    public static final byte REQUEST_LOGIN = 1;  
}
```

کلاس زیر (RequestLoginMessage) نمونه‌ای از کلاس پیامی است که قرار است بر روی شبکه ارسال و دریافت شود. فرض کنید به عنوان مثال می‌خواهیم پیامی شامل دو رشته (نام کاربری و کلمه‌ی عبور) بر روی شبکه تبادل نماییم. برای این کار یک کلاس با نامی مناسب ایجاد کرده که از BaseMessage ارث‌بری می‌نماید. وظیفه‌ی چنین کلاسی ایجاد آرایه‌ای از بایت‌هاست که شامل نام کاربری و کلمه‌ی عبور بوده و همچنین دربردارنده‌ی مقادیر کنترلی طول کلی پیام، نسخه‌ی پروتکل ارتباطی و نوع پیام باشد که به‌صورت منظم و طبق ترتیب مشخص، پشت سر هم قرار می‌گیرند. این وظیفه به کمک متد سازنده‌ی اول انجام می‌گردد. متد سازنده‌ی دوم برعکس سازنده اول عمل می‌نماید و با دریافت یک آرایه‌ی بایتی، مقادیر موجود در آن را به ترتیب جدا کرده و فیلدهای مورد نظر را مقدار دهی می‌نماید. مشخص است که از متد سازنده‌ی اول در هنگامی که می‌خواهیم مقادیری را به صورت آرایه‌ی بایتی درآورده و بر روی شبکه ارسال نماییم استفاده می‌شود و از سازنده‌ی دوم هنگامی که می‌خواهیم آرایه‌ای از بایت‌ها که از شبکه دریافت شده است را به مقادیر اولیه‌ی آن برگردانیم.

```
public class RequestLoginMessage extends BaseMessage {

    private String mUsername;
    private String mPassword;

    public RequestLoginMessage(String username, String password) {
        mUsername = username;
        mPassword = password;
        serialize();
    }

    public RequestLoginMessage(byte[] serialized) {
        mSerialized = serialized;
        deserialize();
    }

    @Override
    protected void serialize() {
        int usernameLength = mUsername.getBytes().length;
        int passwordLength = mPassword.getBytes().length;
        int messageLength = 4 + 1 + 1 + 4 + usernameLength + 4 + passwordLength;
        ByteBuffer byteBuffer = ByteBuffer.allocate(messageLength);
        byteBuffer.putInt(messageLength);
        byteBuffer.put(MessageTypes.PROTOCOL_VERSION);
        byteBuffer.put(MessageTypes.REQUEST_LOGIN);
        byteBuffer.putInt(usernameLength);
        byteBuffer.put(mUsername.getBytes());
        byteBuffer.putInt(passwordLength);
        byteBuffer.put(mPassword.getBytes());
        mSerialized = byteBuffer.array();
    }

    @Override
    protected void deserialize() {
        ByteBuffer byteBuffer = ByteBuffer.wrap(mSerialized);
        int messageLength = byteBuffer.getInt();
        byte protocolVersion = byteBuffer.get();
        byte messageType = byteBuffer.get();
        int usernameLength = byteBuffer.getInt();
        byte[] usernameBytes = new byte[usernameLength];
        byteBuffer.get(usernameBytes);
        mUsername = new String(usernameBytes);
        int passwordLength = byteBuffer.getInt();
        byte[] passwordBytes = new byte[passwordLength];
        byteBuffer.get(passwordBytes);
        mPassword = new String(passwordBytes);
    }
}
```

```

@Override
public byte getMessageType() {
    return MessageTypes.REQUEST_LOGIN;
}

public String getUsername() {
    return mUsername;
}

public String getPassword() {
    return mPassword;
}
}

```

در ادامه قالب کلی برخی از کلاس‌ها آورده شده‌است که رعایت نام و ویژگی‌هایی که برای هر کلاس ذکر شده، الزامی است. (*توجه: در صورت عدم رعایت قرارداد نام‌گذاری، نام و قالب کلاس‌ها، از نمره‌ی پایانی پروژه به شدت کاسته می‌شود)

فیلدها و متدهای مشخص شده در هر کلاس لازم هستند، اما لزوماً کافی نیستند و باید بر حسب نیاز، متد و یا فیلدهایی اضافه گردند.

کلاس **TcpChannel**: این کلاس وظیفه‌ی نگهداری یک socket و کار کردن با آن را بر عهده دارد.

```

public class TcpChannel {
    private Socket mSocket;
    private OutputStream mOutputStream;
    private InputStream mInputStream;

    public TcpChannel(SocketAddress socketAddress, int timeout)
    public TcpChannel(Socket socket, int timeout)

    /**
     * Try to read specific count from input stream.
     */
    public byte[] read(final int count)

    /**
     * Write bytes on output stream.
     */
    public void write(byte[] data)

    /**
     * Check socket's connectivity.
     */
    public boolean isConnected()

    /**
     * Try to close socket and input-output streams.
     */
    public void closeChannel()
}

```

کلاس **NetworkHandler**: این کلاس وظیفه‌ی ارسال و دریافت پیام‌هایی که در شبکه تبادل می‌شوند را بر عهده دارد. واضح است که کلاسی که قرار است با شیئی از این کلاس تعامل کند باید اینترفیس **INetworkHandlerCallback** را پیاده‌سازی نماید.

```

public class NetworkHandler extends Thread {
    private TcpChannel mTcpChannel;
    private Queue<byte[]> mSendQueue;
    private Queue<byte[]> mReceivedQueue;
    private ReceivedMessageConsumer mConsumerThread;

    public NetworkHandler(SocketAddress socketAddress, INetworkHandlerCallback iNetworkHandlerCallback)
    public NetworkHandler(Socket socket, INetworkHandlerCallback iNetworkHandlerCallback)
}

```

```

/**
 * Add serialized bytes of message to the sendQueue.
 */
public void sendMessage(BaseMessage baseMessage)

/**
 * While channel is connected and stop is not called:
 * if sendQueue is not empty, then poll a message and send it
 * else if readChannel() is returning bytes, then add it to receivedQueue.
 */
@Override public void run()

/**
 * Kill the thread and close the channel.
 */
public void stopSelf()

/**
 * Try to read some bytes from the channel.
 */
private byte[] readChannel()

private class ReceivedMessageConsumer extends Thread {

    /**
     * While channel is connected and stop is not called:
     * if there exists message in receivedQueue, then create a message object
     * from appropriate class based on message type byte and pass it through onMessageReceived
     * else if receivedQueue is empty, then sleep 100 ms.
     */
    @Override public void run()
}

public interface INetworkHandlerCallback {
    void onMessageReceived(BaseMessage baseMessage);
    void onSocketClosed();
}
}

```

کلاس **ServerSocketHandler**: زمانی که برنامه در حالت میزبان اجرا شود، یک شیء از این کلاس در انتظار متصل شدن طرف مقابل بوده و پس از اتصال، **NetworkHandler** ای برای تعامل با آن ایجاد می‌نماید.

```

public class ServerSocketHandler extends Thread {

    public ServerSocketHandler(int port, INetworkHandlerCallback iNetworkHandlerCallback,
                               IServerSocketHandlerCallback iServerSocketHandlerCallback)

    /**
     * While server socket is connected and stop is not called:
     * if a connection receives, then create a network handler and pass it through onNewConnectionReceived
     * else sleep for 100 ms.
     */
    @Override public void run()

    /**
     * Kill the thread and close the server socket.
     */
    public void stopSelf()

    public interface IServerSocketHandlerCallback {
        void onNewConnectionReceived(NetworkHandler networkHandler);
    }
}

```

کلاس **MessageManager**: در نهایت به یک کلاس واسط نیاز است که مدیریت ارتباط بستر ارتباطی با مابقی برنامه (بازی و GUI) را برعهده گیرد.

```
public class MessageManager implements IServerSocketHandlerCallback, INetworkHandlerCallback {

    private ServerSocketHandler mServerSocketHandler;
    private List<NetworkHandler> mNetworkHandlerList;

    /**
     * Instantiate server socket handler and start it. (Call this constructor in host mode)
     */
    public MessageManager(int port)

    /**
     * Instantiate a network handler and start it. (Call this constructor in guest mode)
     */
    public MessageManager(String ip, int port)

    /**
     * IMPORTANT: Request Login is an example message and doesn't relate to this project!
     * Create a RequestLoginMessage object and sent it through the appropriate network handler.
     */
    public void sendRequestLogin(String to, String username, String password)

    /**
     * IMPORTANT: Request Login is an example message and doesn't relate to this project!
     * Use the message.
     */
    private void consumeRequestLogin(RequestLoginMessage message)

    /**
     * Add network handler to the list.
     */
    @Override public void onReceivedNewConnection(NetworkHandler networkHandler)

    /**
     * IMPORTANT: Request Login is an example message and doesn't relate to this project!
     * According to the message type of baseMessage, call corresponding method to use it.
     */
    @Override public void onMessageReceived(BaseMessage baseMessage) {
        switch (baseMessage.getMessageType()) {
            case MessageTypes.REQUEST_LOGIN:
                consumeRequestLogin((RequestLoginMessage) baseMessage);
                break;
        }
    }

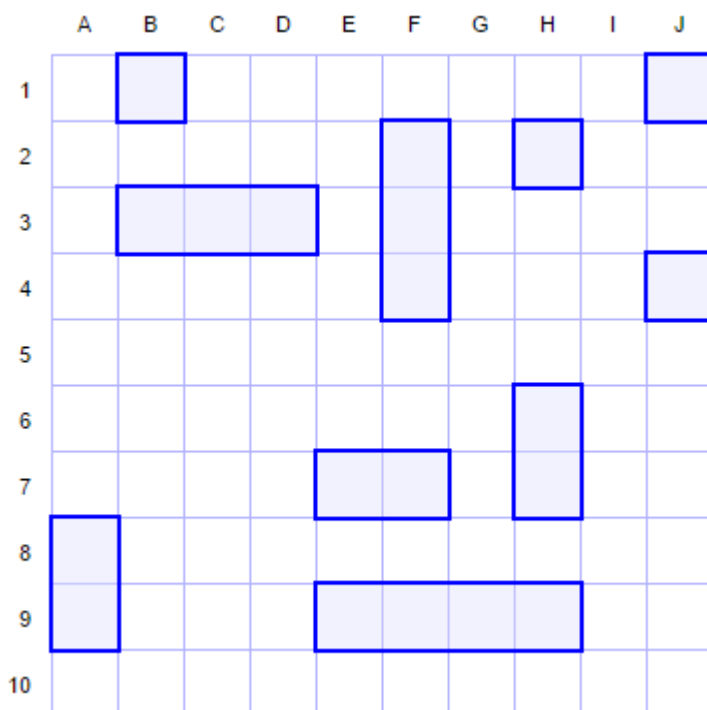
    @Override public void onSocketClosed()
}
```

۲. فاز دوم، ایجاد بازی و واسط کاربری گرافیکی

۱-۲. قوانین بازی

بازی توسط دو نفر انجام می‌شود که هر کدام زمینی جدولی به ابعاد 10x10 خواهند داشت. هر خانه‌ی جدول همانند شکل زیر با عدد (سطر) و حروف (ستون) مربوط به آن آدرس‌گذاری می‌شوند.

هر بازیکن تعدادی شیء با طول‌های 1، 2، 3، 4 و به عرض 1 در اختیار دارد که آن‌ها را به نحوی دلخواه قبل از شروع بازی در زمین خود می‌چیند (تعداد و نوع این اشیاء در اختیار دو بازیکن یکسان است). این اشیاء می‌توانند به صورت افقی و یا عمودی در خانه‌های متوالی قرار بگیرند، به گونه‌ای که هیچ دو شیئی در یک خانه مشترک نباشند (روی هم قرار نگیرند). همچنین هیچ دو شیئی نباید مجاور هم باشند، یعنی بین آن‌ها باید حداقل یک خانه فاصله وجود داشته باشد.



تعداد برای هر بازیکن	اندازه‌ی شیء
۱	۴
۲	۳
۳	۲
۴	۱

پس از اینکه هر کدام از بازیکنان اشیاء را در زمین خود قرار دادند، بازی به صورت نوبتی شروع می‌شود. در هر نوبت، بازیکن خانه‌ای از زمین حریف که حدس می‌زند شیئی در آن‌جا وجود دارد را مورد هدف قرار می‌دهد. اگر اصابت به خانه‌ای خالی صورت گرفته باشد، علامتی مشخص‌کننده‌ی این وضعیت (مثلا دایره) در آن نشان داده شده و نوبت به حریف می‌رسد. در غیر این‌صورت، اصابت به یک شیء صورت گرفته و ضمن نشان دادن علامتی مشخص (مثلا ضربدر) در آن خانه، تا زمانیکه بازیکن بتواند خانه‌های غیرخالی را هدف قرار دهد، نوبت را در اختیار خواهد داشت. بازی تا زمانی ادامه پیدا خواهد کرد که یکی از بازیکنان تمامی خانه‌های غیرخالی زمین حریف را مورد اصابت قرار دهد.

(می‌توانید بازی را در [اینجا](#) به صورت آنلاین بازی کنید).

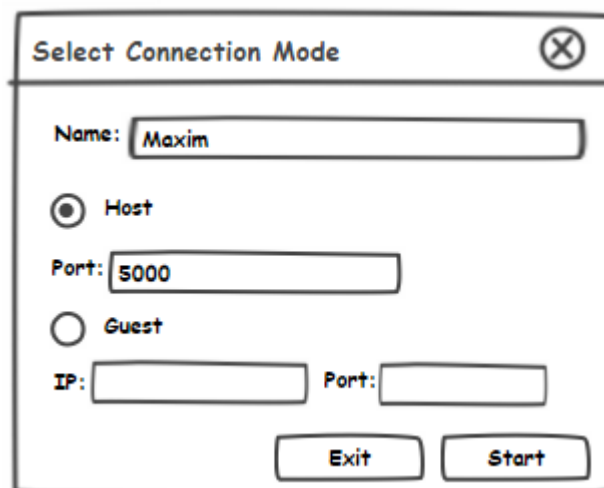
۱-۲. پیاده‌سازی

نکته‌ی مهم در پیاده‌سازی بازی آن است که هر کدام از بازیکن‌ها هیچ اطلاعی از خانه‌های اصابت نشده‌ی زمین حریف نباید داشته باشند. یعنی هنگامی که بازیکنی یکی از خانه‌های زمین حریف را مورد اصابت قرار می‌دهد، پیامی از طریق بستر شبکه‌ی ایجاد شده، حاوی آدرس آن خانه به حریف ارسال می‌نماید. برنامه‌ی حریف که وضعیت آن خانه را می‌داند، (زمین خودی) پیامی حاوی این که اصابت به خانه‌ای خالی یا غیرخالی صورت گرفته است را به طرف اول برمی‌گرداند و طرف اول با توجه به آن، وضعیت خانه را در زمین به‌روز رسانی می‌نماید.

۲-۲. روال اجرای برنامه

در ابتدای اجرای برنامه، پنجره‌ای به کاربر نشان داده می‌شود که در آن کاربر انتخاب می‌کند که قرار است میزبان بازی باشد یا میهمان. همچنین نام و مشخصات اتصال خود را وارد می‌کند.

(a) در حالت میزبان:



Select Connection Mode

Name: Maxim

☒ Host

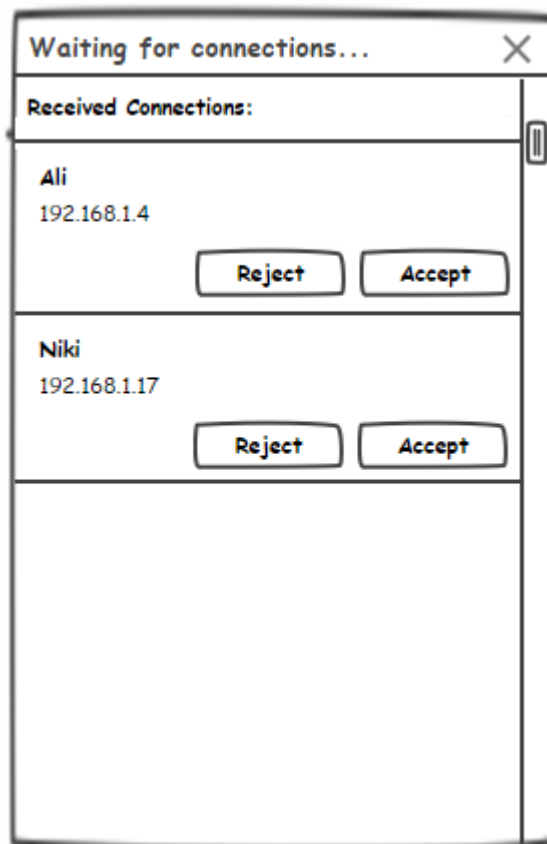
Port: 5000

☐ Guest

IP: Port:

Exit Start

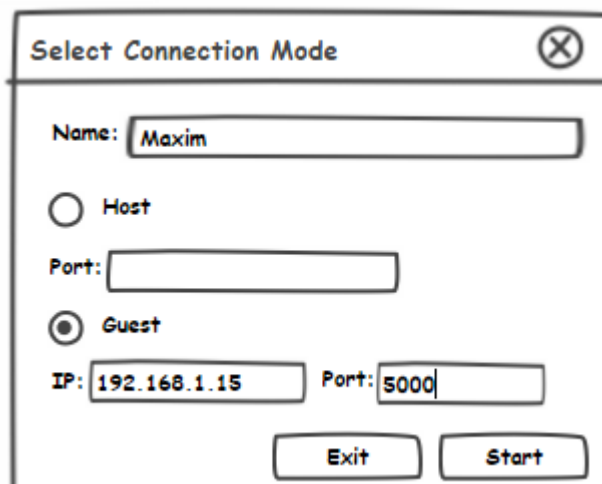
پس از فشردن Start:



A window titled "Waiting for connections..." with a close button (X) in the top right corner. Below the title bar is a section labeled "Received Connections:". It contains two entries, each with a name and an IP address, and two buttons labeled "Reject" and "Accept".

Received Connections:
Ali 192.168.1.4
Niki 192.168.1.17

(b) در حالت میهمان:



A window titled "Select Connection Mode" with a close button (X) in the top right corner. It contains a "Name:" label followed by a text box with "Maxim". Below this are two radio buttons: "Host" (unselected) and "Guest" (selected). Under "Host" is a "Port:" label followed by an empty text box. Under "Guest" are "IP:" and "Port:" labels followed by text boxes containing "192.168.1.15" and "5000" respectively. At the bottom are "Exit" and "Start" buttons.

Name: Maxim

☐ Host

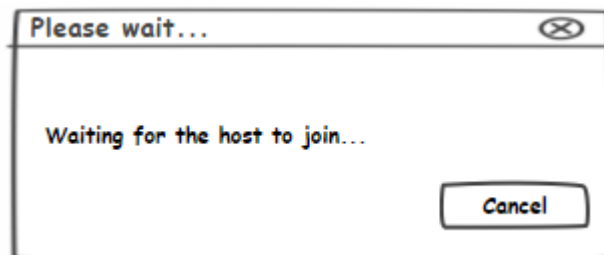
Port:

☒ Guest

IP: 192.168.1.15 Port: 5000

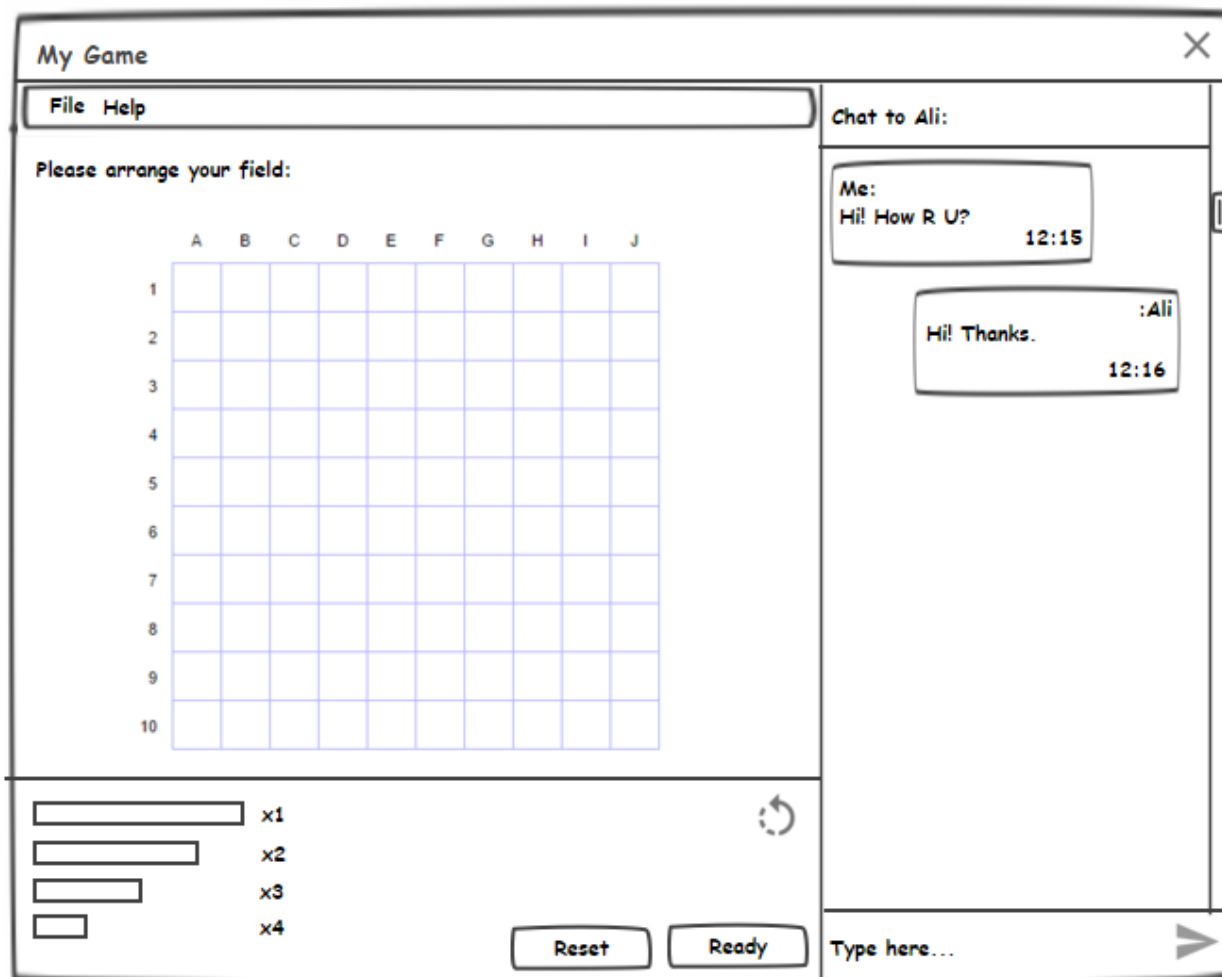
Exit Start

پس از فشردن Start:

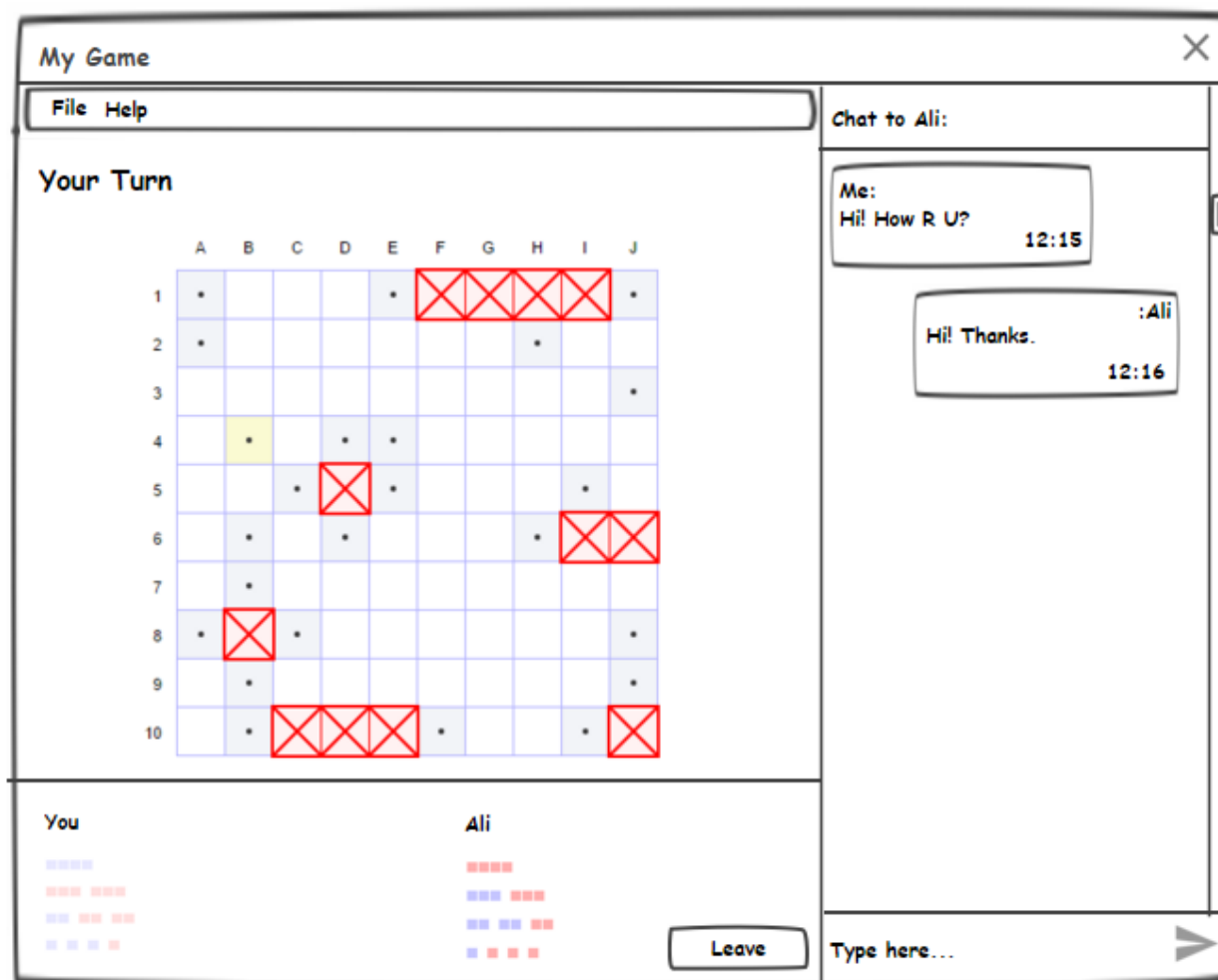


پس از اتصال هر دو کاربر می‌توانند با یکدیگر به گفتگو بپردازند.

در ادامه هر کدام از کاربران می‌بایست به چیش زمین خود بپردازند. هنگامی‌که کاربر یک شیء را در زمین خود قرار می‌دهد با زدن کلید ترکیبی Ctrl+R می‌تواند شیء جاری را Rotate نماید. (افقی به عمودی و بالعکس)

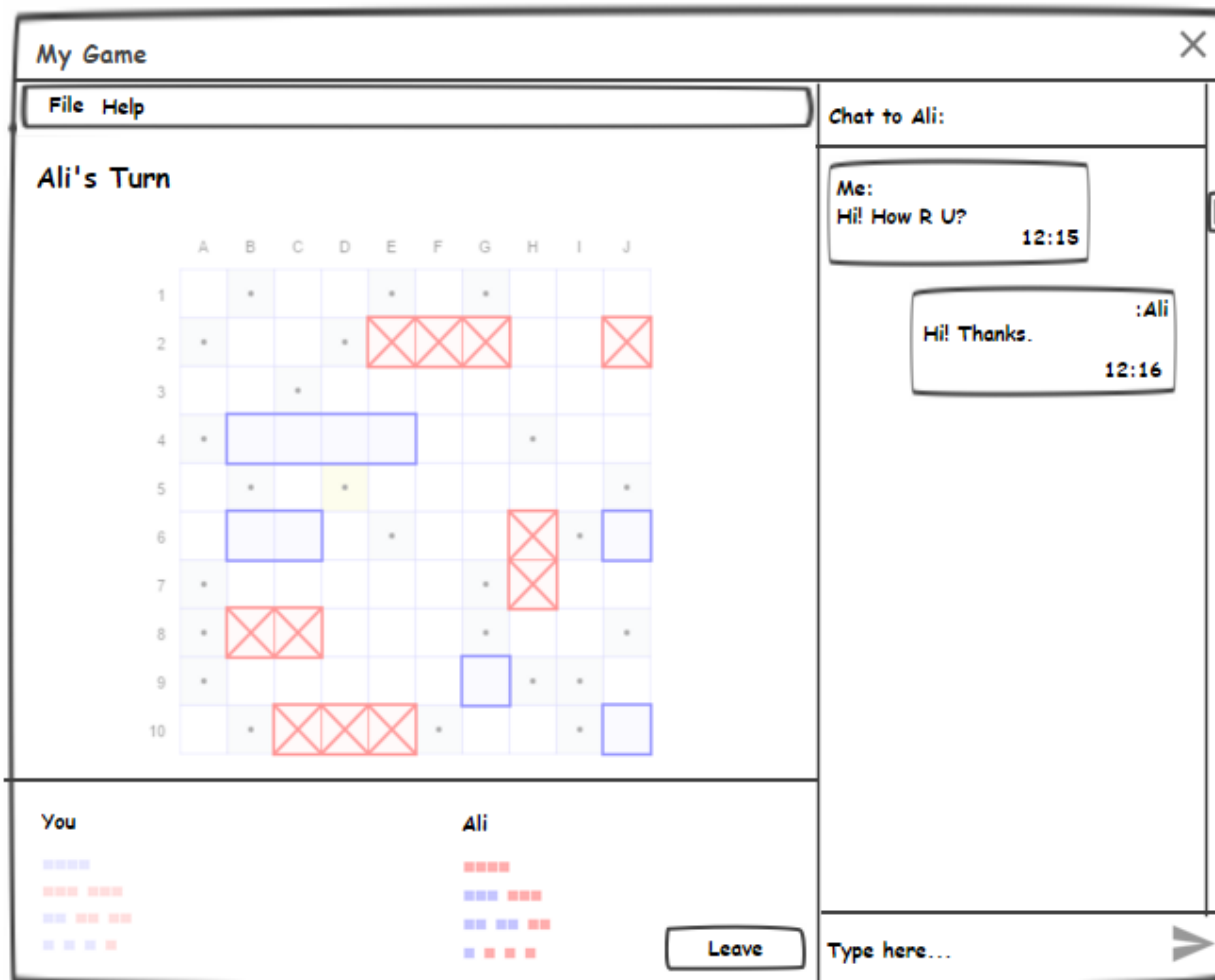


پس از آن که کاربر زمین خود را آماده نمود، با زدن کلید Ready می‌بایست منتظر اعلام آمادگی طرف مقابل بماند. (ممکن است طرف مقابل آمادگی‌اش را زودتر اعلام کرده باشد) لازم به ذکر است که پس از فشردن کلید Ready، دیگر نمی‌توان در چپ‌نشان انجام شده تغییری به‌وجود آورد، همچنین کلید Reset به Cancel تبدیل می‌شود که اعلام آمادگی را لغو نموده (به طرف قابل نیز اعلام شده) و می‌توان در زمین تغییرات ایجاد نمود. پس از آن که هر دو کاربر آماده‌ی بازی شدند **میزبان به‌صورت تصادفی** انتخاب می‌کند که نوبت اول در اختیار خودش باشد یا حریف.



برای هر دو کاربر زمینی قابل رویت است که قرار است روی آن اصابتی صورت گیرد و با تغییر نوبت، زمین قابل مشاهده عوض می‌شود. واضح است که هر بازیکن در زمان نوبت دیگری فقط می‌تواند زمین خود مشاهده کند و با کلیک روی خانه‌های آن اتفاقی صورت نمی‌گیرد.

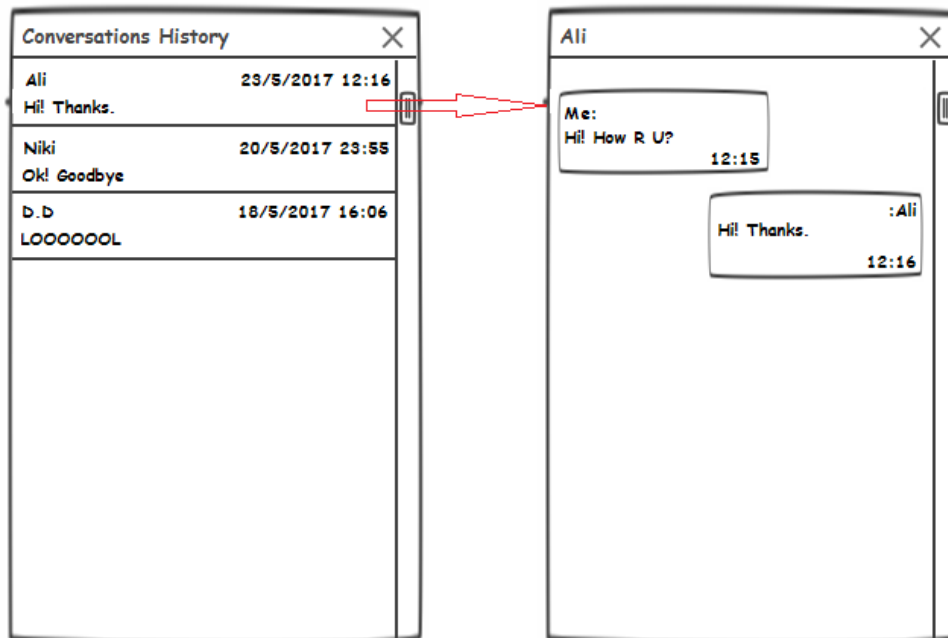
پس از آن که بازیکنی که نوبتش است حرکت خود را انجام داد، نتیجه‌ی اصابت به مدت ۳ ثانیه نشان داده شده، سپس زمین قابل رویت به زمین دیگر تغییر می‌کند.



***توجه:** در صورتی که در اتصال با طرف مقابل خطایی ایجاد شد، یکی از طرفین بازی را ترک کرد، میزبان در خواست اتصال میهمان را رد نمود، میهمان انتظار برای اتصال را قطع کرد و یا ... می‌بایست به طرف مقابل پیغام خطای مناسب به کمک دیالوگ نشان داده شود.

۳-۲. ذخیره‌سازی تاریخچه‌ی گفتگوها

در منوی File گزینه‌ای به نام Conversations History وجود داشته باشد که با فشردن آن لیستی از گفتگوها بر حسب جدیدترین به صورت نزولی نمایش داده می‌شود. با کلیک کردن بر روی هر کدام از آن‌ها صفحه‌ای دیگر شامل پیام‌های تبادل شده نمایش داده خواهد شد.



گفتگوهایی که بین کاربران تبادل می‌شود می‌بایست در فایل ذخیره شده تا بعداً قابل مشاهده باشند. برای این کار پیام‌ها به کمک کتابخانه‌ی org.json در قالب JSON در آمده و به صورت متنی در فایل‌هایی با پسوند ".json" ذخیره می‌شوند. هر آیتم لیست گفتگوها متناظر با یک فایل است که گفتگوی مورد نظر در آن ذخیره شده است.

به عنوان مثال در JSONObject ریشه می‌بایست نام طرف مقابل، آدرس IP، یک شناسه (id) برای این مکالمه در نظر گرفته شود و هم چنین یک آرایه از JSONObject‌ها که پیام‌های تبادل شده را ذخیره می‌کند که هر کدام شامل متن پیام، زمان پیام، ورودی و یا خروجی بودن پیام و ... است.