

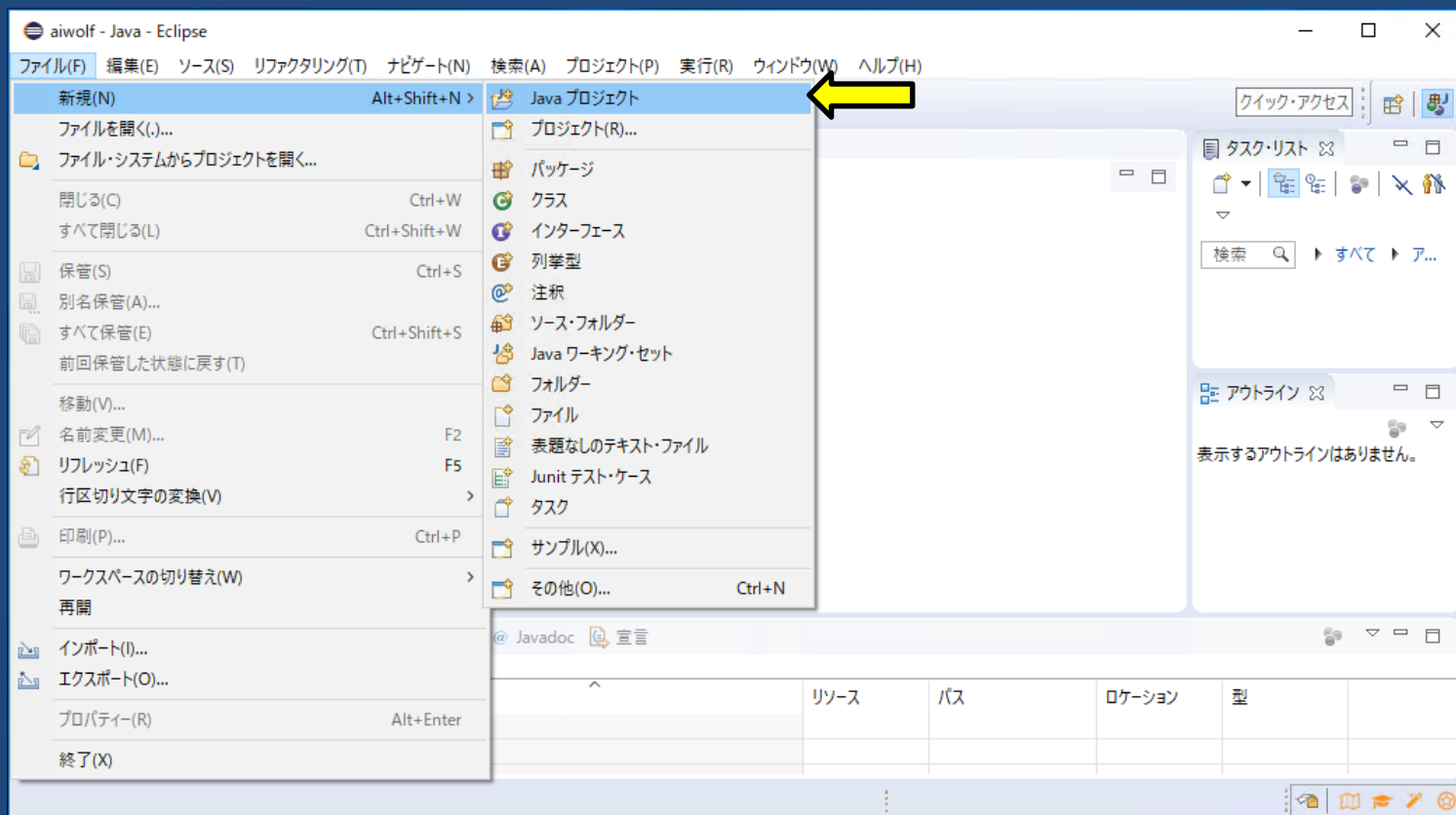
人狼知能エージェント作成方法

目次

- 自作プレイヤーを用いた人狼の実行方法
 - プロジェクトの作成
 - ビルド・パスの構成
 - Javadocを参照
 - 自作プレイヤーでゲーム実行
- 実際にPlayerを実装
 - 占い師を実装

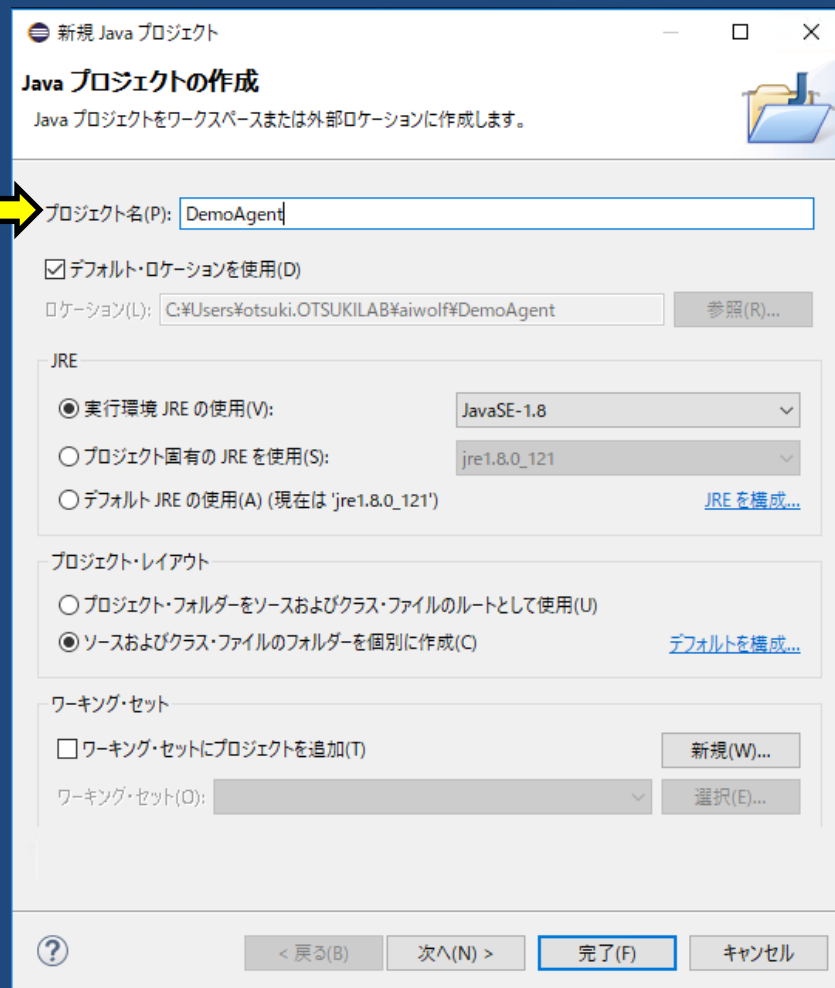
プロジェクトの作成(1)

ファイル > 新規 > Javaプロジェクト



プロジェクトの作成(2)

適当なプロジェクト名
(ここではDemoAgent)
を入力して[完了]



ビルド・パスの構成(1)

人狼知能プラットフォームのダウンロード

- <http://www.aiwolf.org/> から
開発関連 > 人狼知能プラットフォーム
とたどり
aiwolf-ver0.4.5.zip と aiwolf-docs-ver0.4.5.zip
をダウンロード
- aiwolf-ver0.4.5.zipを解凍すると
AIWolf-ver0.4.5フォルダーができる

ビルド・パスの構成(2)

AIWolf-ver0.4.5フォルダーの中身

- 5つのJARファイル
このうち開発に使用するのは以下の2つ
`aiwolf-client.jar` , `aiwolf-common.jar`
- 4つのシェルスクリプトファイル(*.sh)
- 4つのバッチファイル(*.bat)
- 2つのAutoStarter用設定ファイル
(`AutoStarter.ini`, `SampleSetting.cfg`)

ビルド・パスの構成(3)

開発に使うJARファイルの概要

- aiwolf-client.jar
プレイヤーを作成する際に用いる
- aiwolf-common.jar
プレイヤー作成とゲーム実行の両方で用いる

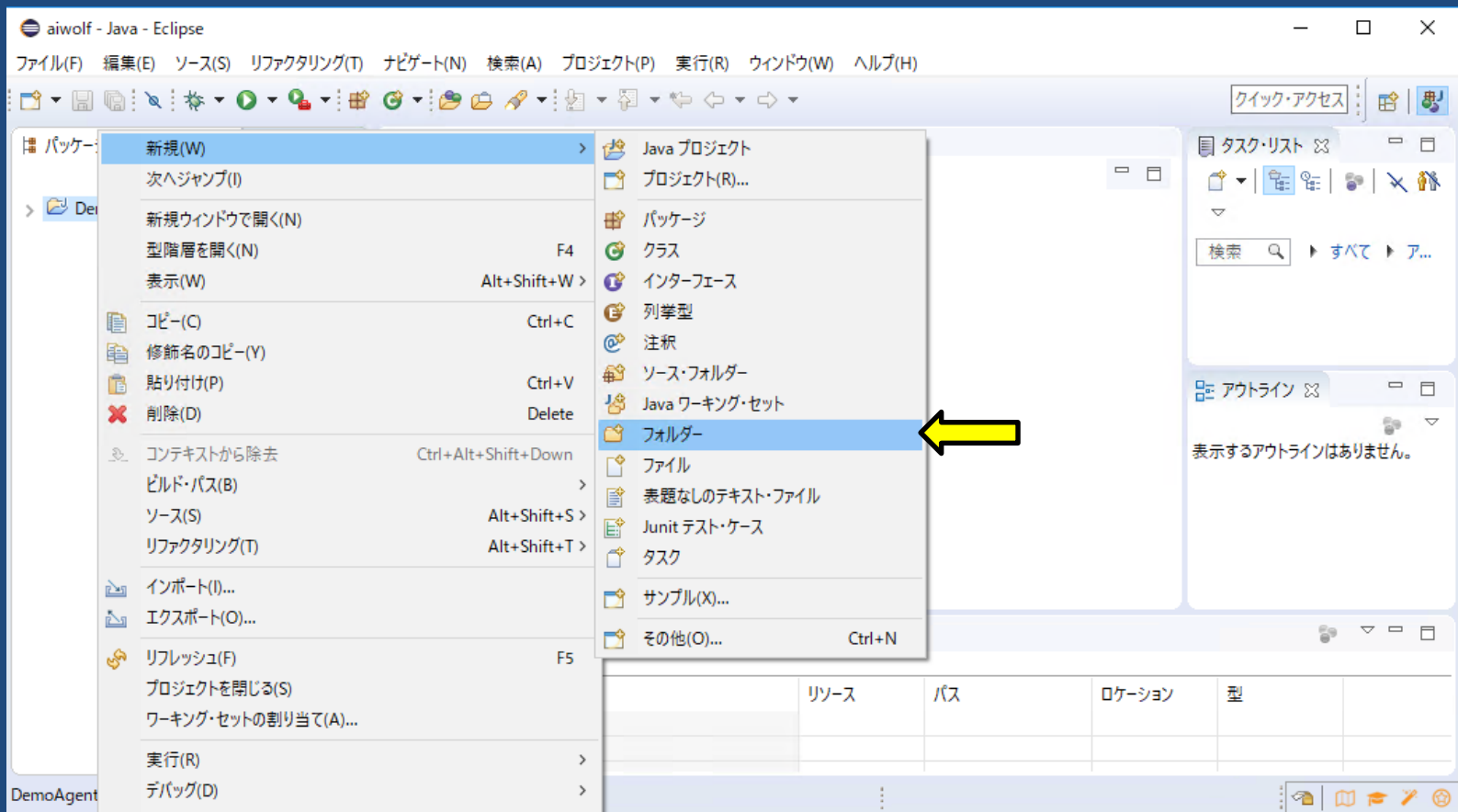
ビルド・パスの構成(4)

他のファイルはゲームを実行するために使用

- StartServer
ゲームサーバーをGUIで起動させる
- StartClient
サーバーに接続するクライアントを起動
- StartGUIClient
サーバーに接続するクライアントをGUIで起動
- AutoStarter
設定ファイル(AutoStarter.iniと
SampleSetting.cfg)の通りにゲームを実行

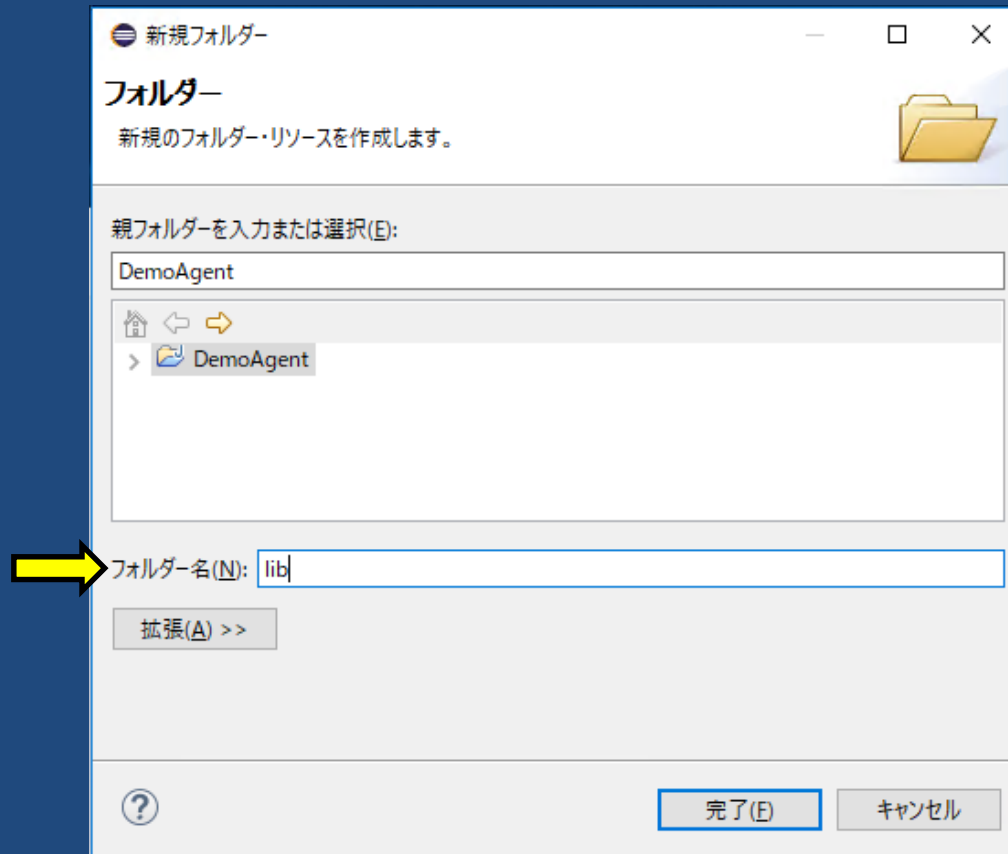
ビルド・パスの構成(5)

プロジェクトを右クリック>新規>フォルダー



ビルド・パスの構成(6)

フォルダ一名(ここではlib)を入力して[完了]

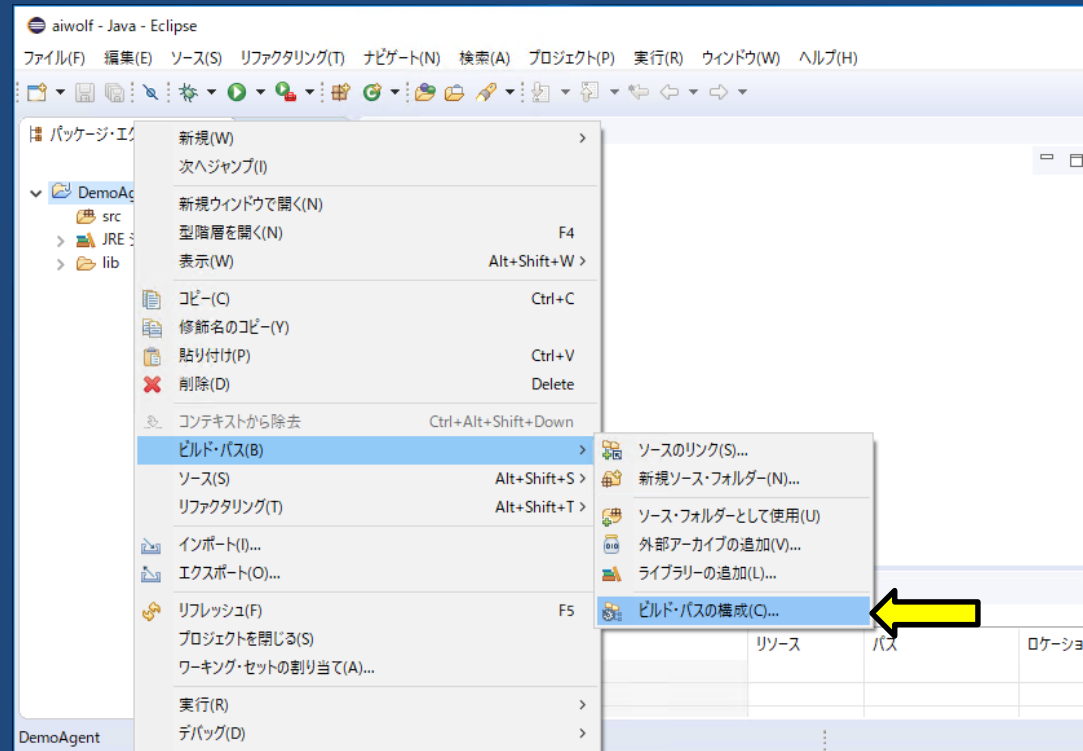


ビルド・パスの構成(7)

1. 作成したフォルダーに
aiwolf-client.jar, aiwolf-common.jar
をドラッグ & ドロップでコピー
2. これらのjarファイルを, 以下に述べる方法で
ビルド・パスに追加

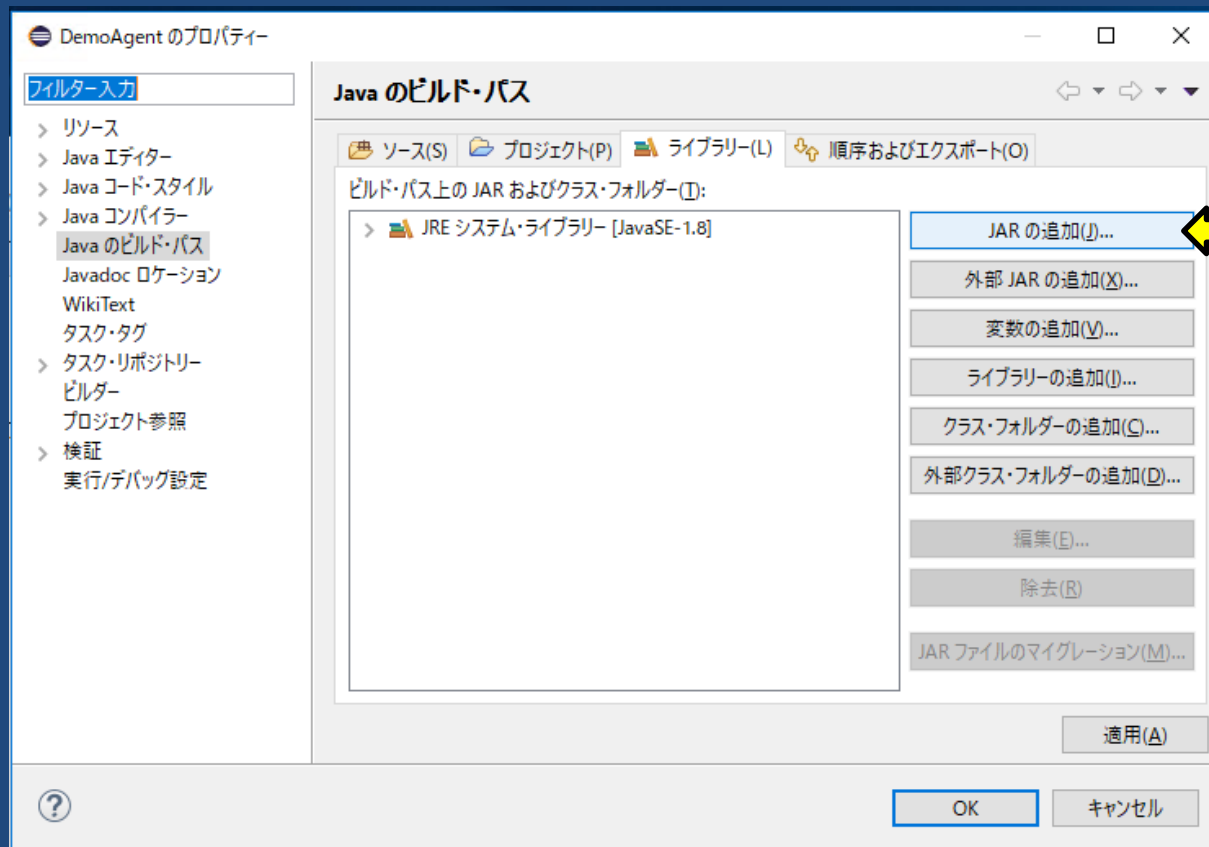
ビルド・パスの構成(8)

プロジェクトを右クリック>ビルド・パス>ビルド・パスの構成



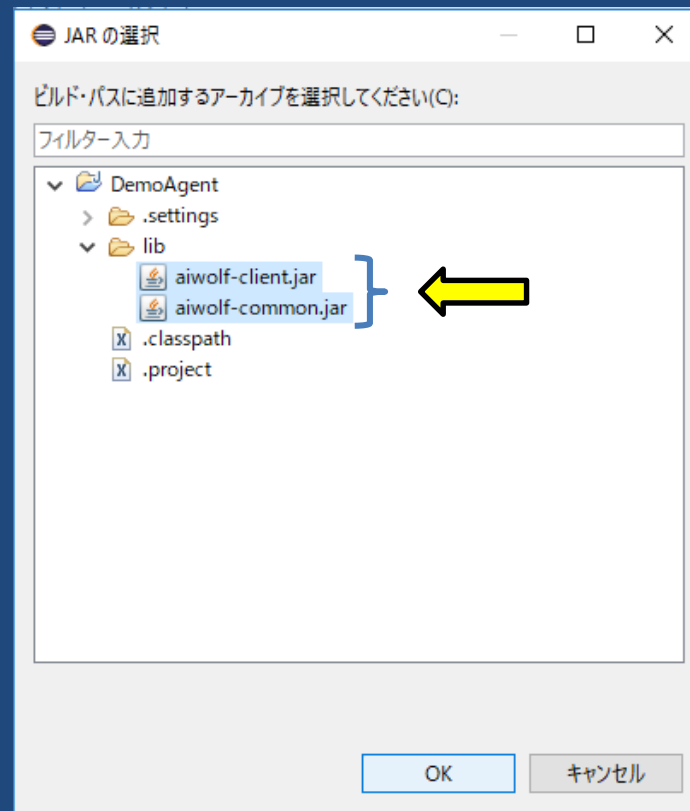
ビルド・パスの構成(9)

ライブラリー>JARの追加



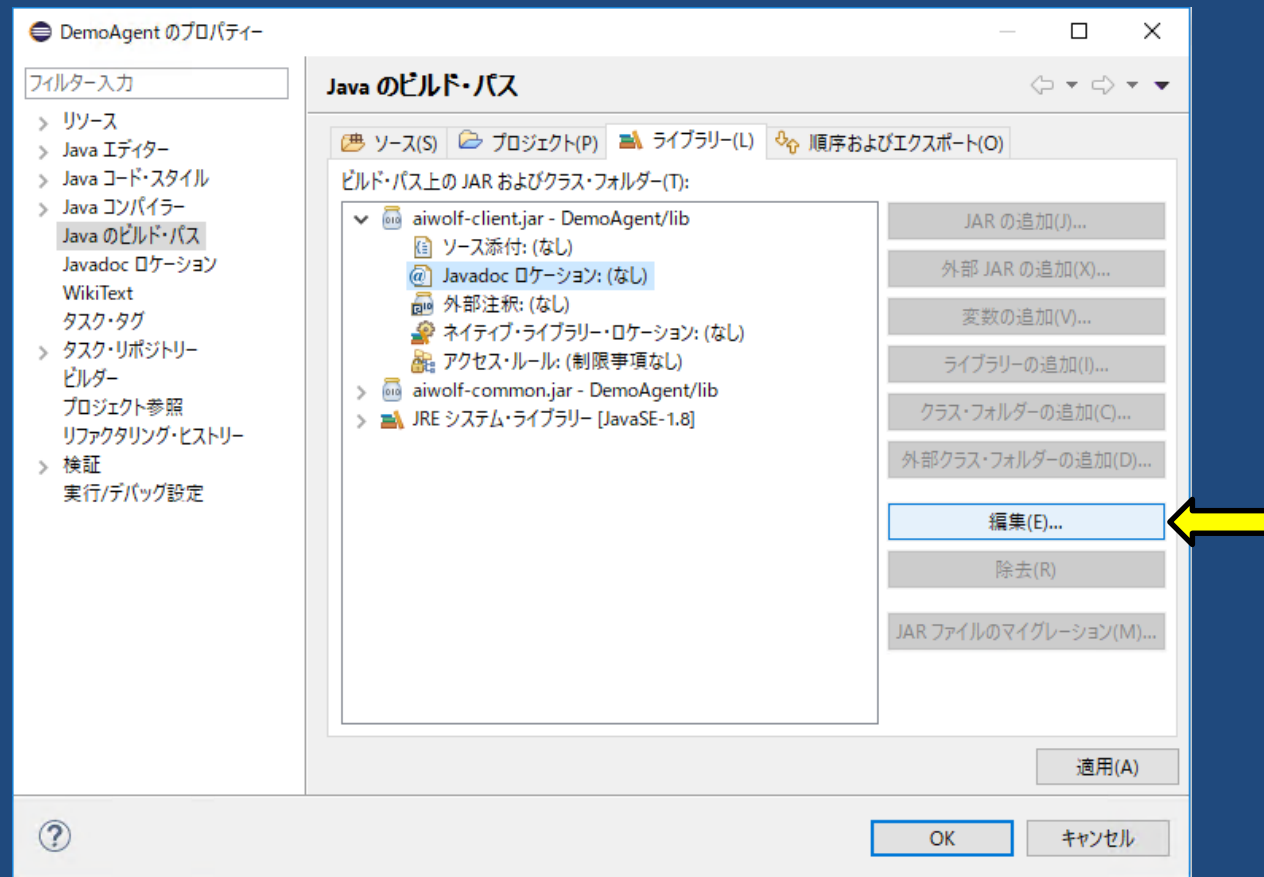
ビルド・パスの構成(10)

フォルダー内に入れた2つのjarファイルを選択して[OK]



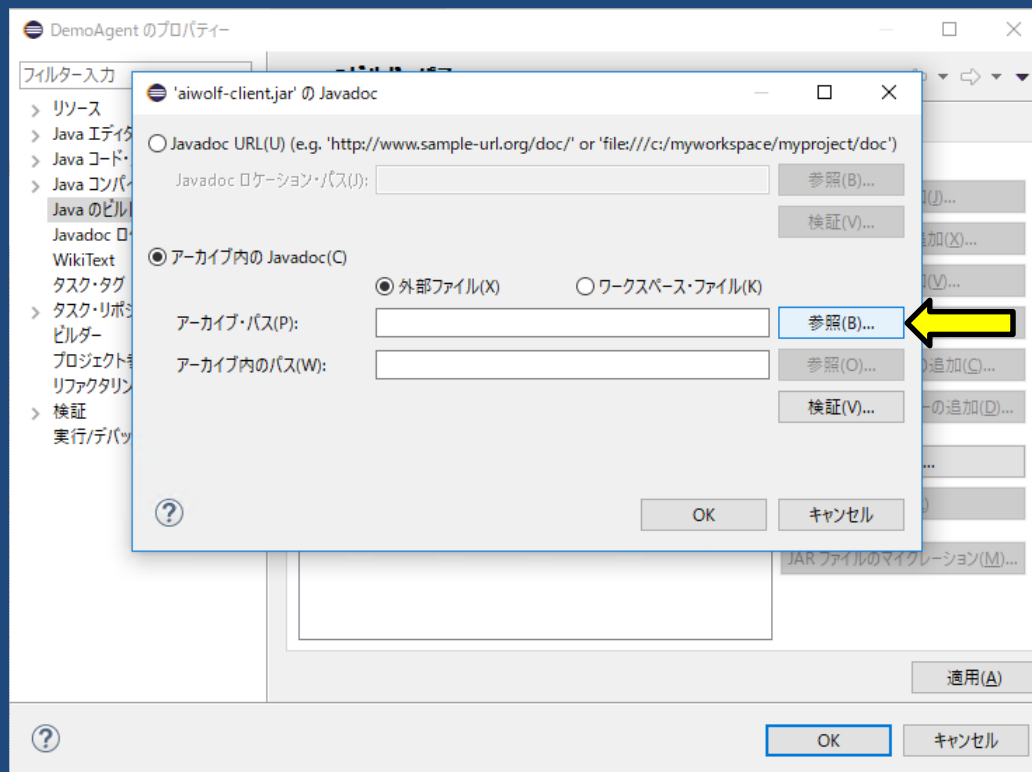
Javadocの参照(1)

aiwolf-client.jar > Javadoc > 編集



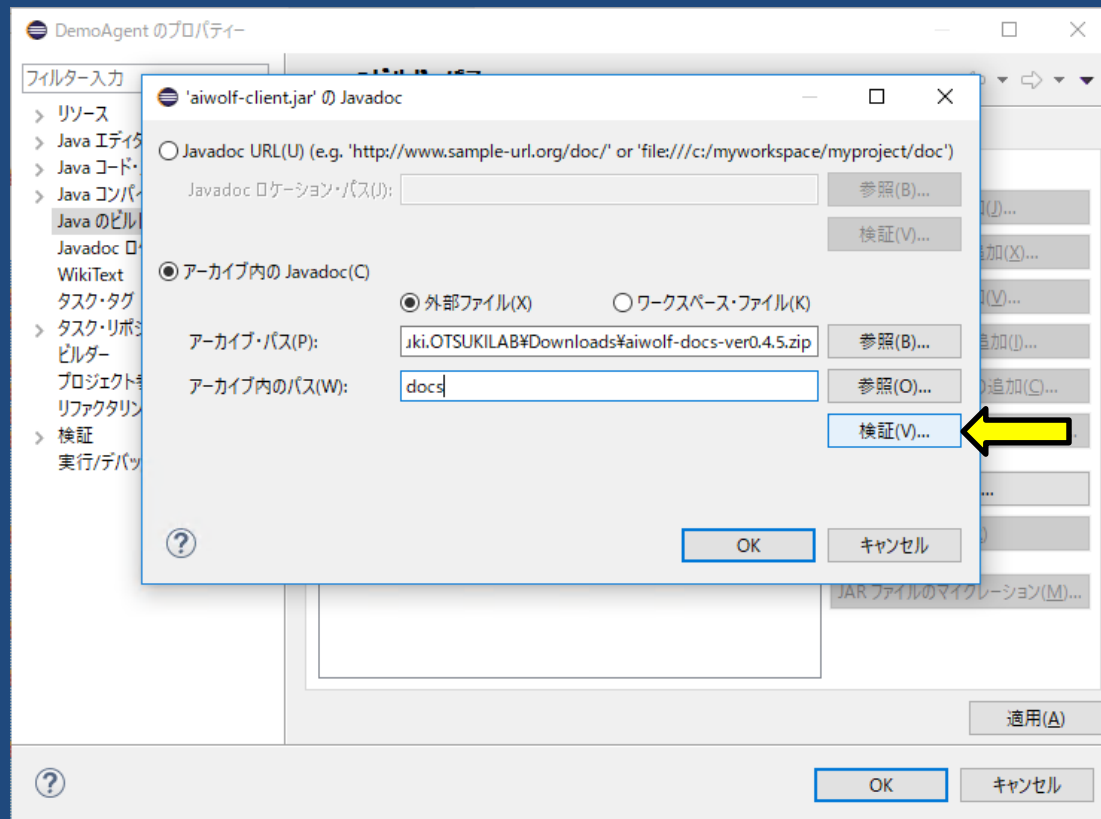
Javadocの参照(2)

アーカイブ内のJavadoc > 外部ファイル > アーカイブ・パス > 参照



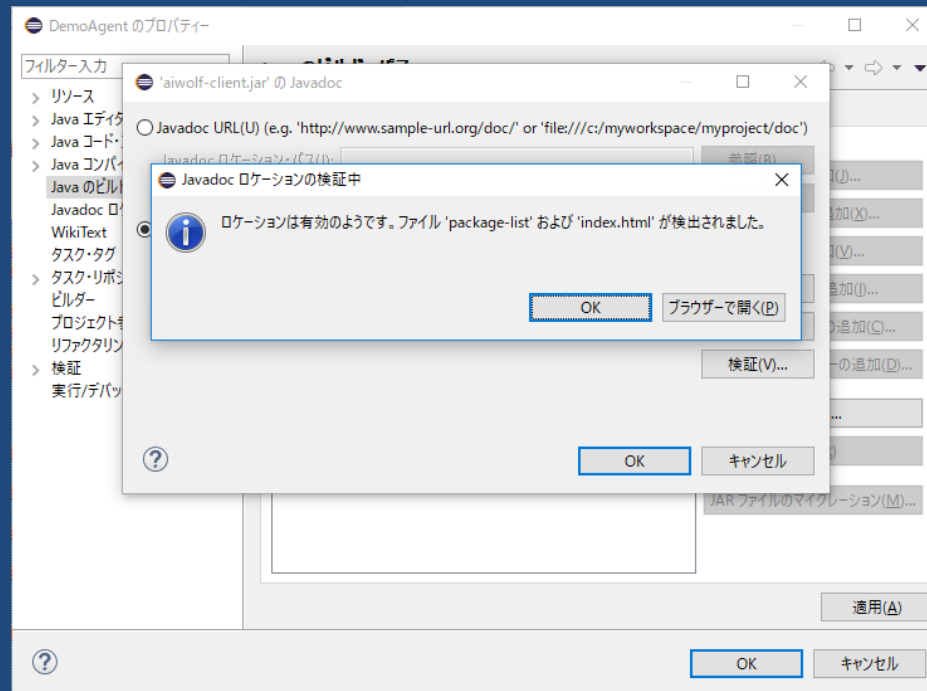
Javadocの参照(3)

ダウンロードしたaiwolf-docs-ver0.4.5.zipを、
アーカイブ内のパスにはdocsを指定し[検証]



Javadocの参照(4)

- ローケーションが有効であることが確認できたら
[OK]
- aiwolf-common.jarも同様にJavadocを参照



自作プレイヤーでゲーム実行

1. AbstractRoleAssignPlayerを継承したクラスを作成
2. ゲーム実行の準備
3. 各役職のPlayerを作成
4. RoleAssignPlayerに各役職のPlayerをセット

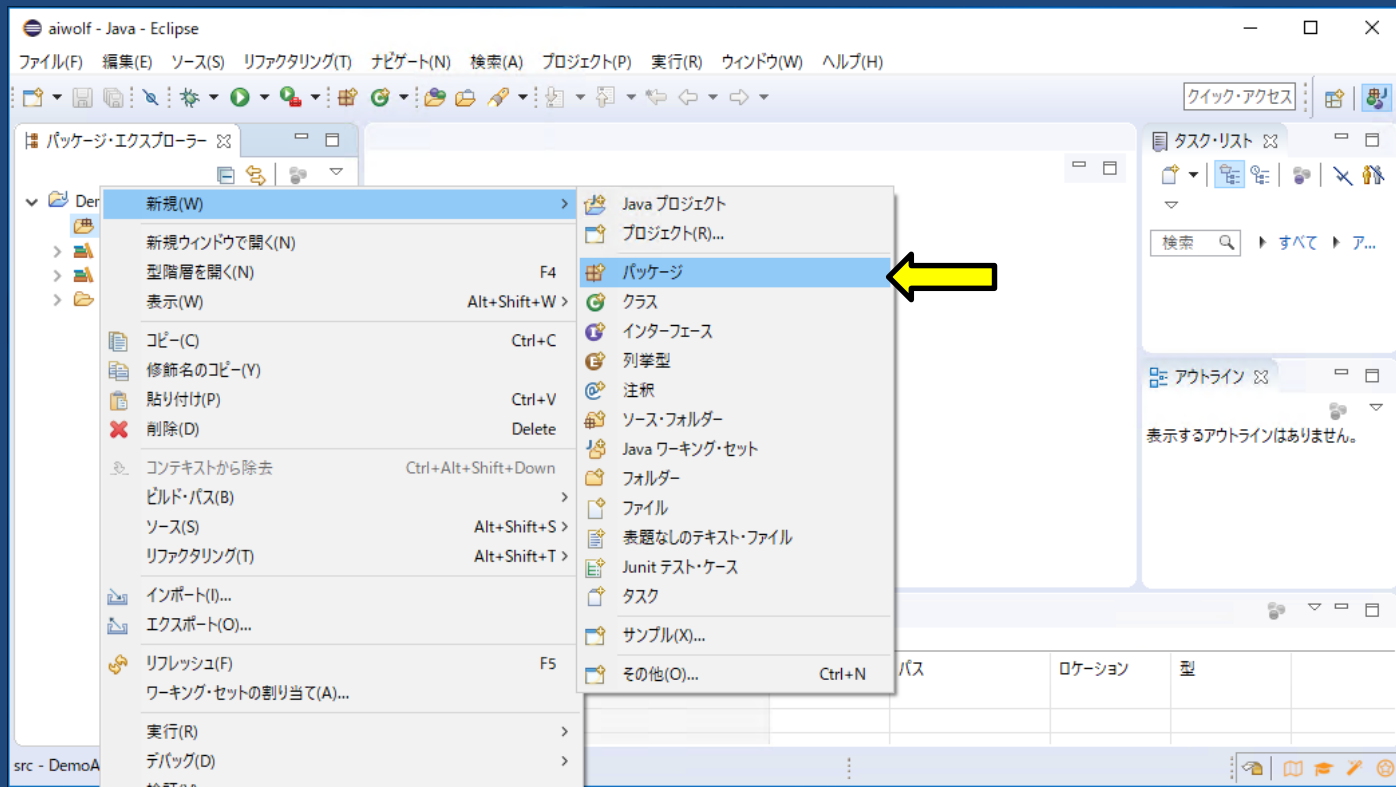
自作プレイヤーでゲーム実行

1. AbstractRoleAssignPlayerを継承したクラスを作成
2. ゲーム実行の準備
3. 各役職のPlayerを作成
4. RoleAssignPlayerで各役職のPlayerをセット

まず実行できる環境を整える

新規パッケージ作成(1)

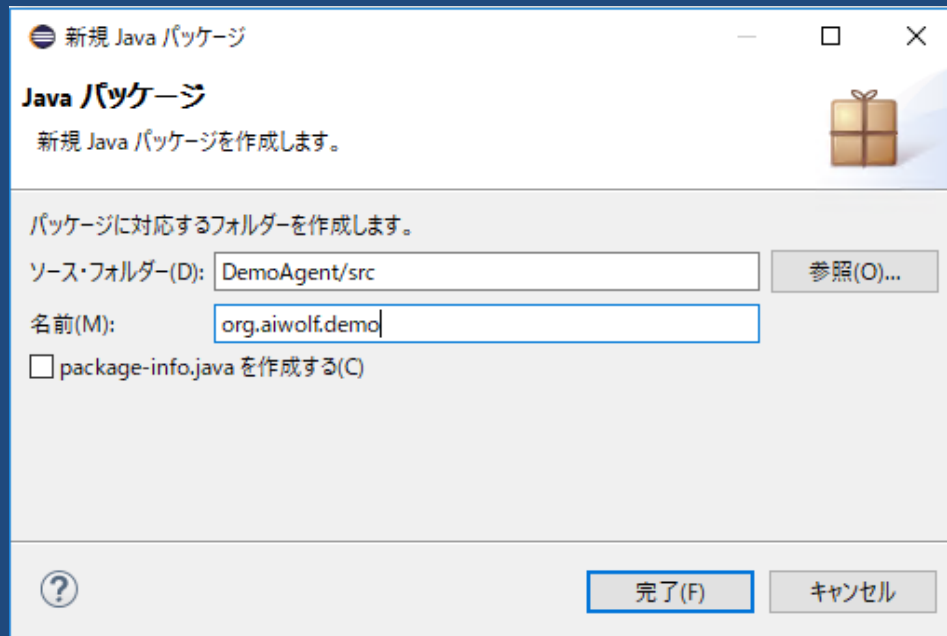
DemoAgent > srcを右クリック > 新規 > パッケージ



新規パッケージ作成(2)

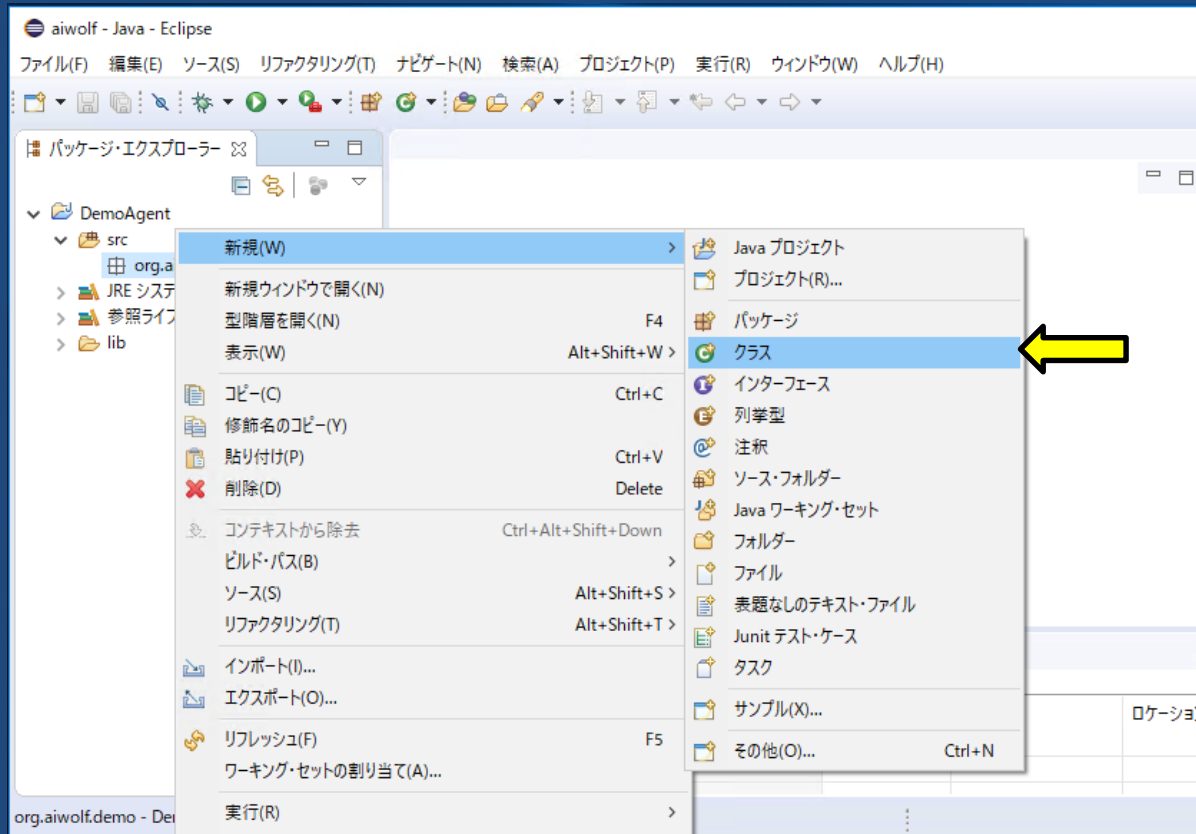
パッケージ名を入力して[完了]

パッケージ名は自分のアドレスを逆から入力
例) demo@aiwolf.org → org.aiwolf.demo



新規クラス作成(1)

作成したパッケージを右クリック>新規>クラス

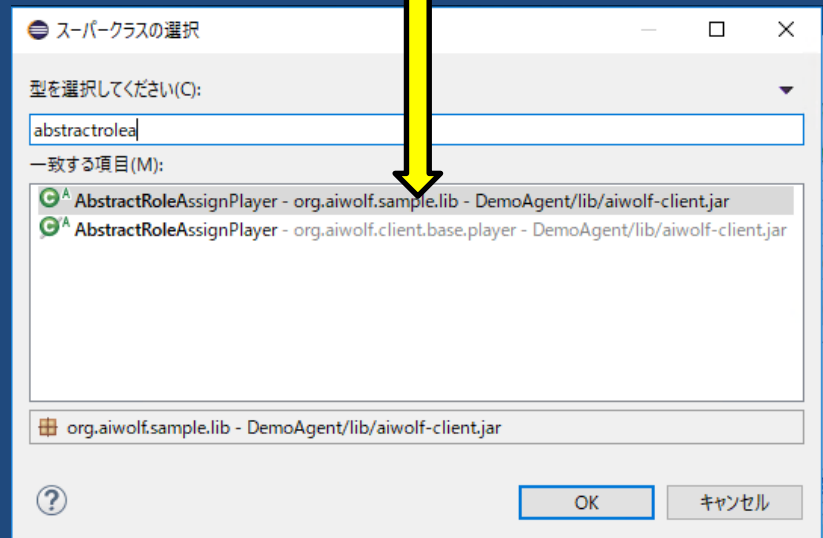
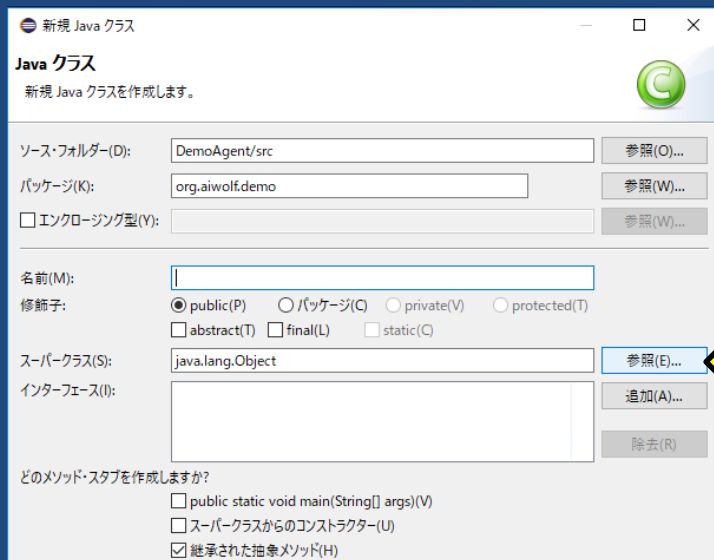


新規クラス作成(2)

スーパークラス>参照から

org.aiwolf.sample.libのAbstractRoleAssignPlayerを選択

※org.aiwolf.client.base.playerの方は
非推奨なのでNG

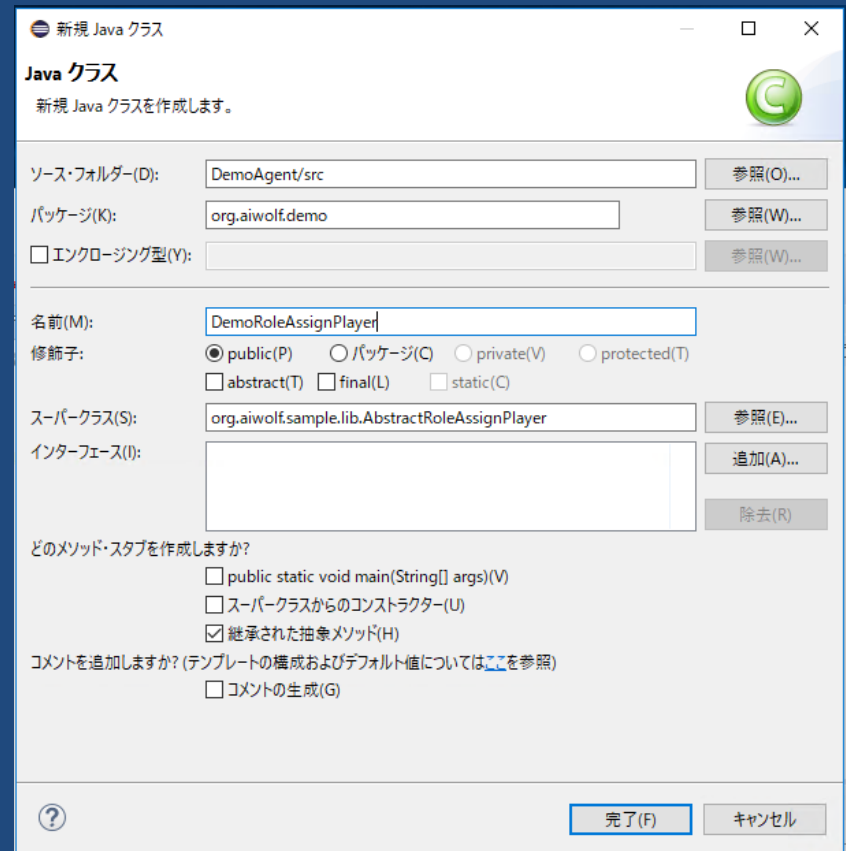


新規クラス作成(3)

クラス名(ここではDemoRoleAssignPlayer)を入力して[完了]



これだけでサンプルプレイヤーと同等のプレイヤーが完成

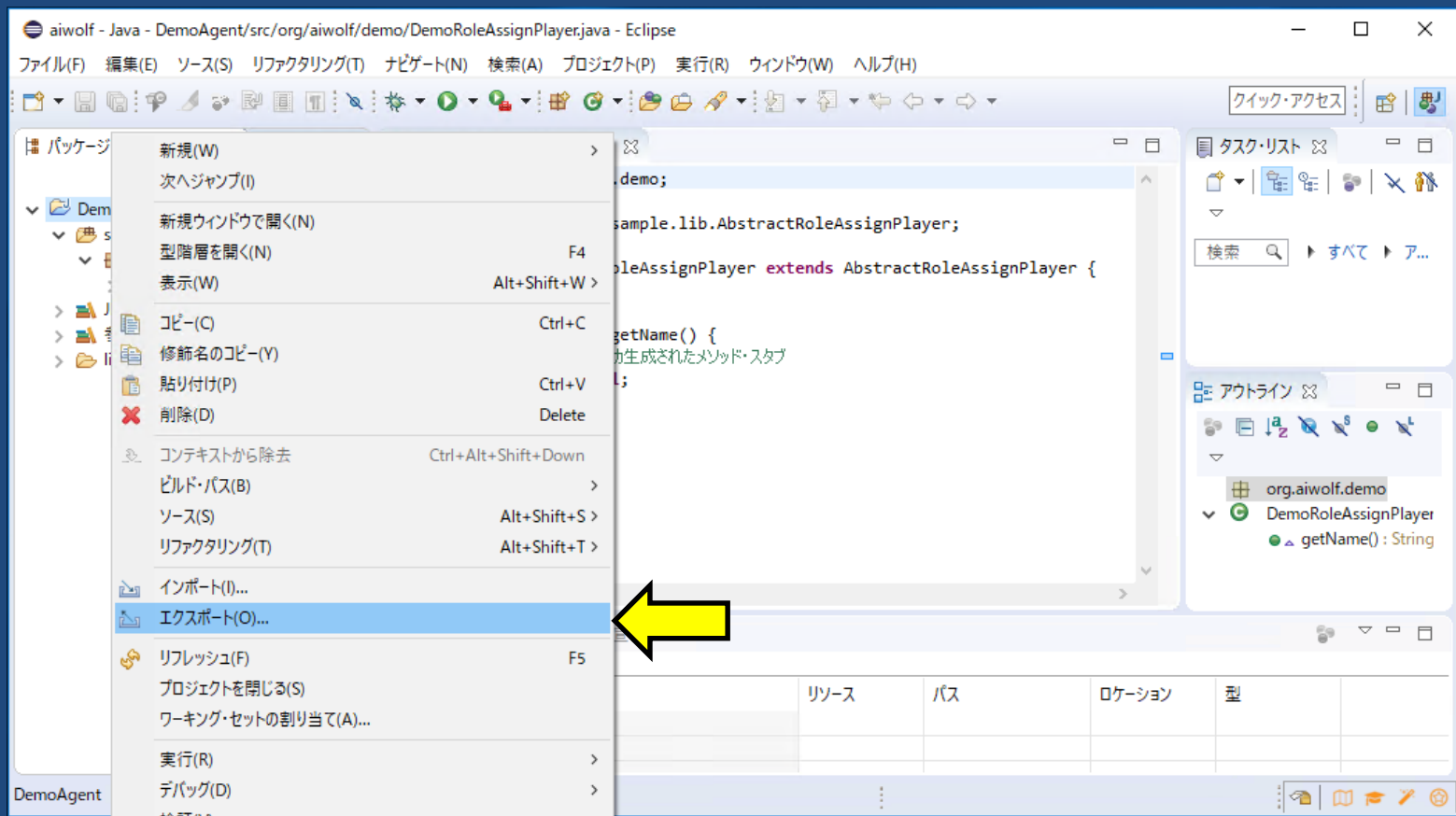
A screenshot of the 'New Java Class' dialog box in an IDE. The dialog is titled '新規 Java クラス' and 'Java クラス'. It contains the following fields and options:

- ソース・フォルダー(D): DemoAgent/src
- パッケージ(K): org.aiwolf.demo
- 名前(M): DemoRoleAssignPlayer
- 修飾子: public(P) パッケージ(C) private(V) protected(T)
- スーパークラス(S): org.aiwolf.sample.lib.AbstractRoleAssignPlayer
- インターフェース(I):
- どのメソッド・スタブを作成しますか?: public static void main(String[] args)(V), スーパークラスからのコンストラクター(U), 継承された抽象メソッド(H)
- コメントを追加しますか? (テンプレートの構成およびデフォルト値についてはここを参照): コメントの生成(G)

Buttons for '完了(F)' and 'キャンセル' are at the bottom right.

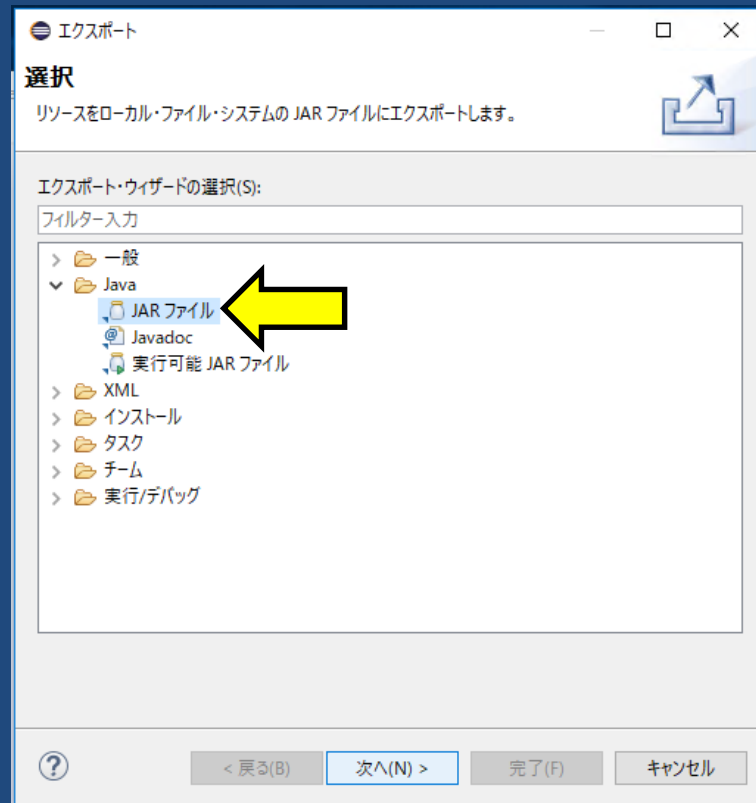
JARファイル作成方法(1)

プロジェクトを右クリック>エクスポート



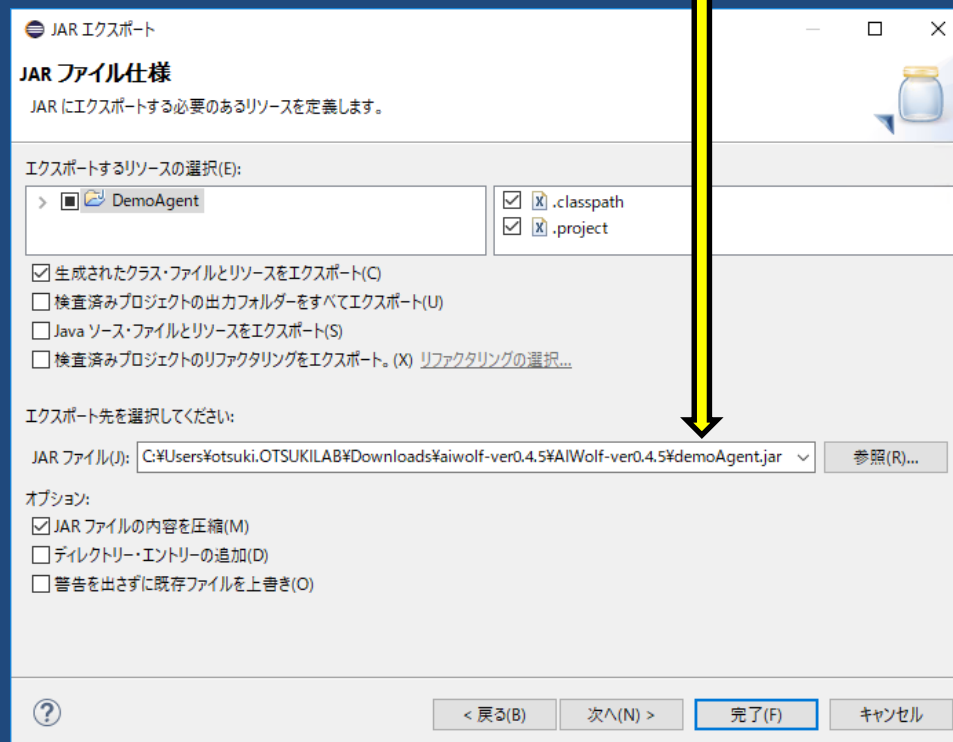
JARファイル作成方法(2)

JAVA > JARファイル を選択して次へ



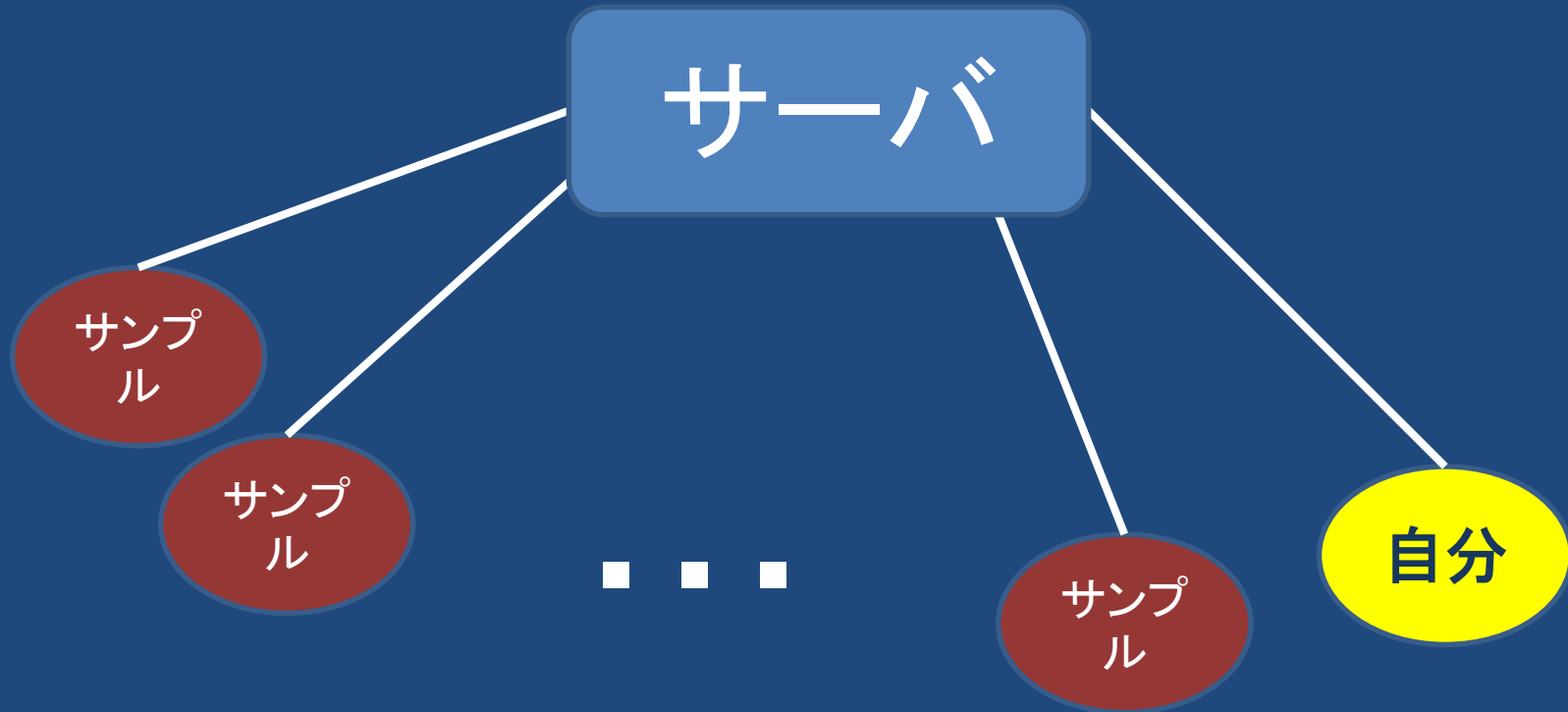
JARファイル作成方法(3)

- エクスポート先はAIWolf-ver0.4.5フォルダー
- JARファイルの名前はdemoAgent.jarとする



実行の構成

1つのサーバに複数のクライアント(人狼知能)プログラムを接続し, ゲームを実行



設定ファイルの記述

- AutoStarter.iniのSample1とSample2の行末にある役職指定(WEREWOLFとSEER)を削除
- AutoStarter.iniのSample5から始まる行をコメントアウト(行頭に#を挿入)
- AutoStarter.iniの末尾に以下の行を追加
[任意の名前(下の例ではDemo)],[パッケージ名].[クラス名]

```
lib=./
log=./log/
port=10000
game=10
view=true
setting=./SampleSetting.cfg
#agent=5
Sample1,org.aiwolf.sample.player.SampleRoleAssignPlayer
Sample2,org.aiwolf.sample.player.SampleRoleAssignPlayer
Sample3,org.aiwolf.sample.player.SampleRoleAssignPlayer
Sample4,org.aiwolf.sample.player.SampleRoleAssignPlayer
#Sample5,org.aiwolf.sample.player.SampleRoleAssignPlayer
Demo,org.aiwolf.demo.DemoRoleAssignPlayer
```

サンプル4体と
➡ 自分のエージェント1体で
ゲームを行う設定

ゲームの実行

- Windowsの場合 :AutoStarter.batを実行
- Mac/Linuxの場合 : AutoStarter.shを実行

作成したDemoが
ゲームに参加中



Playerの実装

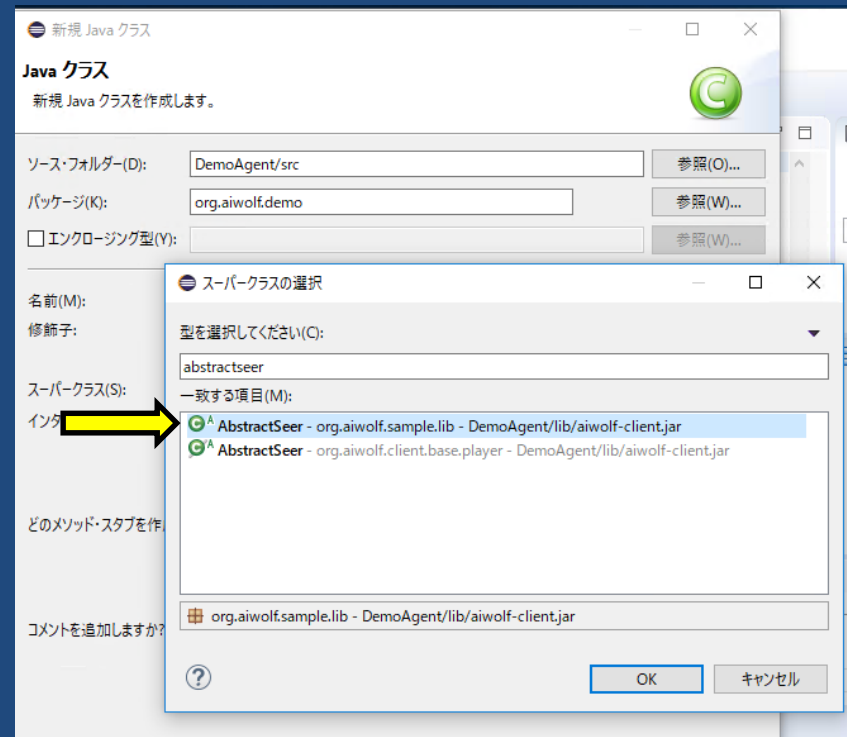
自作プレイヤーでゲーム実行

1. AbstractRoleAssignPlayerを継承したクラスを作成
2. ゲーム実行の準備
3. 各役職のPlayerを作成
4. RoleAssignPlayerで各役職のPlayerをセット

例として占い師を作ってみましょう

占い師用の新規クラスの作成

- RoleAssignPlayerの作成と同様に新規クラスを作成(例えばDemoSeer)
- スーパークラスは org.aiwolf.sample.lib のAbstractSeer



RoleAssignPlayerを変更

- RoleAssignPlayerに次のコンストラクタを追加 (DemoRoleAssignPlayerの場合)

```
public DemoRoleAssignPlayer(){  
    setSeerPlayer(new DemoSeer());  
}
```

- 役職として占い師を振り分けられた時に DemoSeerを呼び出す
- 他の役職の場合はデフォルトでサンプルプレイヤーが呼び出される

ゲーム実行

- もう一度JARをエクスポートし, AutoStarterを実行して動けばOK
- DemoSeerはまだ実装すべきメソッドを実装していないから何もしない

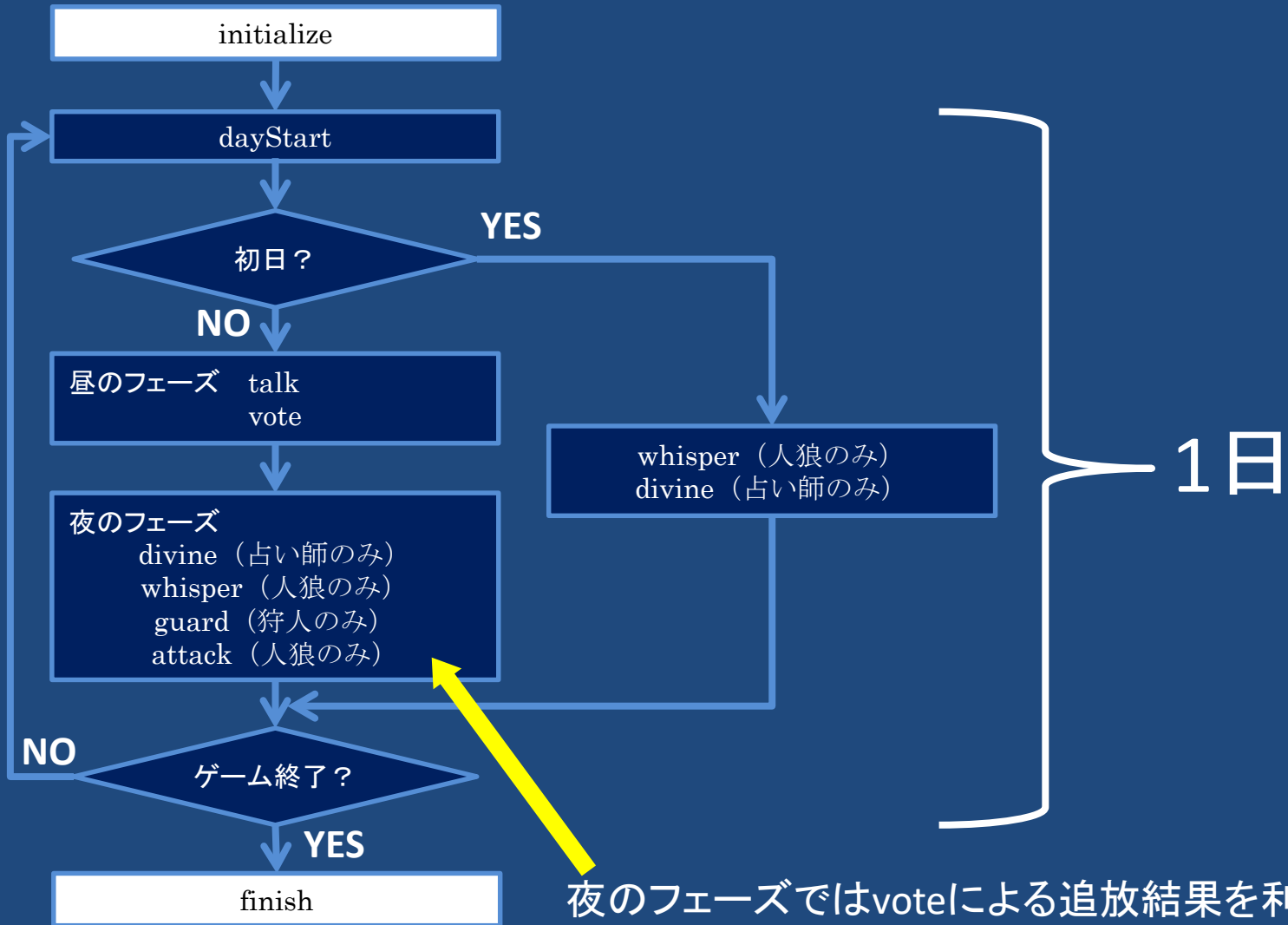
実装すべきメソッドとは？

Playerインターフェース

Playerインターフェース内で定義されるメソッド

- initialize(GameInfo, GameSetting)
 - update(GameInfo)
 - dayStart()
 - whisper()
 - attack()
 - guard()
 - getName()
 - talk()
 - vote()
 - divine()
 - finish()
- ゲームの各タイミングでサーバがプレイヤーのこれらのメソッドを呼び出す
 - updateはinitialize以外のメソッドの直前に呼ばれる

ゲームの流れ



各メソッドの戻り値

- void
 - initialize(GameInfo, GameSetting)
 - update(GameInfo)
 - dayStart()
 - finish()
- String
 - getName() ■ talk() ■ whisper()
- Agent (対象プレイヤーを選択するメソッド)
 - vote() ■ attack() ■ divine() ■ guard()

Seerの実装

- Seer(DemoSeer)が継承したAbstractSeer
 - 占い師に必要なattack(), guard(), whisper()は継承されず、万一呼ばれた場合は例外を投げる
- それ以外のメソッドを実装する

実際に実装してみましよう

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

フィールドの定義

```
/** 自分 */
Agent me;
/** 最新のGameInfo */
GameInfo currentGameInfo;
/** 未報告の占い結果が入る待ち行列 */
Deque<Judge> myDivinationQueue = new LinkedList<>();
/** 白（人間）リスト */
List<Agent> whiteList = new ArrayList<>();
/** 黒（人狼）リスト */
List<Agent> blackList = new ArrayList<>();
/** 灰色（未確定）リスト */
List<Agent> grayList;
/** カミングアウト済みか */
boolean isCO = false;
```

ユーティリティメソッドの定義

```
/** エージェントが活着ているかどうかを返す */
boolean isAlive(Agent agent) {
    return currentGameInfo.getAliveAgentList().contains(agent);
}

/** リストからランダムに選んで返す */
<T> T randomSelect(List<T> list) {
    if (list.isEmpty()) {
        return null;
    } else {
        return list.get((int) (Math.random() * list.size()));
    }
}
}
```

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

getName(), update()の実装

```
public String getName() {  
    return "DemoSeer";  
}  
  
public void update(GameInfo gameInfo) {  
    // currentGameInfoをアップデート  
    currentGameInfo = gameInfo;  
}
```

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

initialize()の実装

前ゲームの情報が残ったままにならないように
フィールドを初期化しておく

```
public void initialize(GameInfo gameInfo,
                       GameSetting gameSetting) {
    // フィールドの初期化
    me = gameInfo.getAgent();
    grayList = new ArrayList<>(gameInfo.getAgentList());
    grayList.remove(me);
    whiteList.clear();
    blackList.clear();
    myDivinationQueue.clear();
}
```

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

dayStart()の実装

```
public void dayStart() {  
    // 占い結果の取り込み  
    Judge divination = currentGameInfo.getDivineResult();  
    if (divination != null) {  
        myDivinationQueue.offer(divination);  
        Agent target = divination.getTarget();  
        Species result = divination.getResult();  
        // 灰色リスト・白リスト・黒リストのアップデート  
        grayList.remove(target);  
        if (result == Species.HUMAN) {  
            whiteList.add(target);  
        } else {  
            blackList.add(target);  
        }  
    }  
}
```

待ち行列の最後に要素を追加

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

vote()の実装

1. 生きている人狼の中からランダムに投票
2. 生きている人狼がいなければ, 生きている灰色のプレイヤーからランダムに投票
3. 上記2項に該当するプレイヤーがない場合は自分以外の生きているプレイヤーからランダムに投票(生存者全員白の場合なのであり得ない状況ではあるが...)

vote()の実装

```
public Agent vote() {  
    // 候補者リスト  
    List<Agent> candidates = new ArrayList<>();  
  
    // 生きている人狼を候補者リストに加える  
    for (Agent agent : blackList) {  
        if (isAlive(agent)) {  
            candidates.add(agent);  
        }  
    }  
    // 候補者がいない場合は生きている灰色のプレイヤーを候補者リストに加える  
    if (candidates.isEmpty()) {  
        for (Agent agent : grayList) {  
            if (isAlive(agent)) {  
                candidates.add(agent);  
            }  
        }  
    }  
}
```

続く

vote()の実装

```
// 候補者がいない場合はnullを返す（自分以外の生存プレイヤーからランダム）
if (candidates.isEmpty()) {
    return null;
}
// 候補者リストからランダムに投票先を選ぶ
return randomSelect(candidates);
}
```

vote()の実装(全体像)

```
public Agent vote() {
    // 候補者リスト
    List<Agent> candidates = new ArrayList<>();

    // 生きている人狼を候補者リストに加える
    for (Agent agent : blackList) {
        if (isAlive(agent)) {
            candidates.add(agent);
        }
    }
    // 候補者がいない場合は生きている灰色のプレイヤーを候補者リストに加える
    if (candidates.isEmpty()) {
        for (Agent agent : grayList) {
            if (isAlive(agent)) {
                candidates.add(agent);
            }
        }
    }
    // 候補者がいない場合はnullを返す(自分以外の生存プレイヤーからランダム)
    if (candidates.isEmpty()) {
        return null;
    }
    // 候補者リストからランダムに投票先を選ぶ
    return randomSelect(candidates);
}
```

次はdivine()を実装してみましよう

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

divine()の実装

まだ占っていない自分以外の生存プレイヤーからランダムに選択

divine()の実装

```
public Agent divine() {  
    // 候補者リスト  
    List<Agent> candidates = new ArrayList<>();  
  
    // 生きている灰色のプレイヤーを候補者リストに加える  
    for (Agent agent : grayList) {  
        if (isAlive(agent)) {  
            candidates.add(agent);  
        }  
    }  
    // 候補者がいない場合は誰も占わない  
    if (candidates.isEmpty()) {  
        return null;  
    }  
    // 候補者リストからランダムに占う  
    return randomSelect(candidates);  
}
```

次は占い結果を発話してみましよう

実装の流れ

1. フィールド・ユーティリティメソッドの定義
2. getName, updateの実装
3. initializeの実装
4. dayStartの実装
5. voteの実装: 占い結果を考慮して投票
6. divineの実装: ランダムに占う
7. talkの実装: カミングアウト, 占い結果の報告

talk, whisperでの発話

org.aiwolf.client.lib.Contentクラスと

org.aiwolf.client.lib.ContentBuilderのサブクラスで生成

以下の手順で得られるtextが発話テキストとなる

```
ContentBuilder builder = 発話の種類に応じたContentBuilder;
```

```
Content content = new Content(builder);
```

```
String text = content.getText();
```

各種ContentBuilderクラス(1)

1. EstimateContentBuilder(target, role): targetの役職はroleだと思う
2. ComingoutContentBuilder (target, role) : targetの役職はroleだ
3. DivinationContentBuilder(target) : targetを占う
4. DivinedResultContentBuilder(target, result) : targetを占った結果resultだった
5. IdentContentBuilder(target, result) : targetは霊媒の結果resultだった
6. GuardCandidateContentBuilder(target) : targetを護衛する
7. GuardedAgentContentBuilder(target) : targetを護衛した
8. VoteContentBuilder(target) : targetに投票する
9. AttackContentBuilder(target) : targetに襲撃投票する
10. AgreeContentBuilder(talkType, talkDay, talkID) : talkDay日目, 種類talkType, talkID番目の発話に同意する

各種ContentBuilderクラス(2)

11. DisagreeContentBuilder(talkType, talkDay, talkID) : talkDay日目, 種類talkType, talkID番目の発話に反対する
12. RequestContentBuilder(agent, content) : agentにcontentを要求する
13. OverContentBuilder() : もう話すことはない
以下で定義される定数Content.OVERが用意されている
OVER = new Content(new OverContentBuilder());
14. SkipContentBuilder() : 様子を見る
以下で定義される定数Content.SKIPが用意されている
SKIP = new Content(new SkipContentBuilder());

詳細は, aiwolf-ver0.4.4.zip (1つ前のリリース) 内の「0.4.4での発話生成の方法(修正版2).pdf」を参照

発話生成の例

今日の占い結果を報告する発話を生成

```
//占いの情報の取得
```

```
Judge judge = getLatestDayGameInfo().getDivineResult();
```

```
//発話の作成
```

```
ContentBuilder builder = new
```

```
    DivinedResultContentBuilder(judge.getTarget(), judge.getResult());
```

```
String talk = new Content(builder).getText();
```


talk()の実装

1. 占いで人狼を見つけたらカミングアウト
2. カミングアウトした後は占い結果を報告する

talk()の実装

```
public String talk() {  
    // 占いで人狼を見つけたらカミングアウトする  
    if (!isCO) {  
        if (!myDivinationQueue.isEmpty() &&  
            myDivinationQueue.peek().getResult() == Species.WEREWOLF) {  
            isCO = true;  
            ContentBuilder builder = new ComingoutContentBuilder(me,  
                                                                    Role.SEER)  
            return new Content(builder).getText();  
        }  
    }  
}
```

待ち行列の先頭の要素を取り出さずに取得

続く

talk()の実装

// カミングアウトした後は、まだ報告していない占い結果を順次報告

```
else {  
    if (!myDivinationQueue.isEmpty()) {  
        Judge divination = myDivinationQueue.poll();  
        ContentBuilder builder = new  
            DivinedResultContentBuilder(divination.getTarget(),  
                                         divination.getResult());  
        return new Content(builder).getText();  
    }  
}  
  
return Content.OVER.getText();  
}
```

待ち行列の先頭の要素を取り出す

talk()の実装(全体像)

```
public String talk() {
    // 占いで人狼を見つけたらカミングアウトする
    if (!isCO) {
        if (!myDivinationQueue.isEmpty() && myDivinationQueue.peek().getResult() == Species.WEREWOLF) {
            isCO = true;
            ContentBuilder builder = new ComingoutContentBuilder(me, Role.SEER)
            return new Content(builder).getText();
        }
    }
    // カミングアウトした後は、まだ報告していない占い結果を順次報告
    else {
        if (!myDivinationQueue.isEmpty()) {
            Judge divination = myDivinationQueue.poll();
            ContentBuilder builder = new DivinedResultContentBuilder(divination.getTarget(),
                                                                    divination.getResult());
            return new Content(builder).getText();
        }
    }
    return Content.OVER.getText();
}
```

次は他の人の発話を読み込んでみましょう

会話内容の取り込み

- 会話のリストは `GameInfo.getTalkList()` で `List<Talk>` 型として取得
- Talkクラスのメソッド
 - `getAgent()`: 発話したAgentを取得
 - `getText()`: 発話内容 (String) を取得
 - `getDay()`: 発話日 (int) を取得
 - `getIdx()`: その日の何番目の発話か (int) を取得
 - `getTurn()`: その日の何番目のターンの発話か (int) を取得

会話内容の取り込み

Talk.getText()で得られる文字列は人狼知能プロトコルに準拠

例：“DIVINED Agent[04] HUMAN”

この文字列をparseするには、Contentクラスのコンストラクタの引数として与える

//Contentクラスのコンストラクタの引数に発話内容のStringを入れると自動的にパースされる

```
Content content = new Content(talk.getText());
```

Contentクラスの使い方

戻り値	メソッド名	説明
String	getText()	発話内容をそのまま返す
Operator	getOperator()	発話内容の演算子を返す. 発話が単文の場合はnull
Agent	getSubject()	発話内容の主語を返す
Topic	getTopic()	発話内容のトピックを返す (COMINGOUTやDIVINED等)
Agent	getTarget()	発話内容の目的語となるプレイヤーを返す (例えば"DIVINED Agent[01] HUMAN" → Agent[01])
Role	getRole()	発話の目的語となる役職を返す (例えば"COMINGOUT Agent[02] SEER" → SEER)
Species	getResult()	占い(霊媒)の結果を返す (例えば"IDENTIFIED Agent[03] WEREWOLF" → WEREWOLF)
TalkType	getTalkType()	TopicがAGREE/DISAGREEの時, 対象発話のタイプ (TALK/WHISPER)を返す
int	getTalkDay()	TopicがAGREE/DISAGREEの時, 対象発話の発話日を返す
int	getTalkID()	TopicがAGREE/DISAGREEの時, 対象発話の発話IDを返す
List<Content>	getContentList()	発話内容が複文・重文の場合, 節のリストを返す

会話内容を取り込むように update()を修正(1)

```
/** GameInfo.talkList読み込みのヘッド */  
int talkListHead; // dayStart()中で0に初期化しておくこと  
  
public void update(GameInfo gameInfo) {  
    currentGameInfo = gameInfo;  
    // GameInfo.talkListからカミングアウト・占い報告・霊媒報告を抽出  
    for (int i = talkListHead; i < currentGameInfo.getTalkList().size();  
        i++) {  
        Talk talk = currentGameInfo.getTalkList().get(i);  
        Agent talker = talk.getAgent();  
        if (talker == me) {  
            continue;  
        }  
        Content content = new Content(talk.getText()); // 発話をparse
```

続く

会話内容を取り込むように update()を修正(2)

```
switch (content.getTopic()) {
case COMINGOUT:
    // カミングアウト発話の処理
    break;
case DIVINED:
    // 占い結果報告発話の処理
    break;
case IDENTIFIED:
    // 霊媒結果報告発話の処理
    break;
default:
    break;
}
}
talkListHead = currentGameInfo.getTalkList().size();
}
```

修正したupdate()の全体像

```
/** GameInfo.talkList読み込みのヘッド */
int talkListHead; // dayStart()中で0に初期化しておくこと

public void update(GameInfo gameInfo) {
    currentGameInfo = gameInfo;
    // GameInfo.talkListからカミングアウト・占い報告・霊媒報告を抽出
    for (int i = talkListHead; i < currentGameInfo.getTalkList().size(); i++) {
        Talk talk = currentGameInfo.getTalkList().get(i);
        Agent talker = talk.getAgent();
        if (talker == me) {
            continue;
        }
        Content content = new Content(talk.getText()); // 発話をparse
        switch (content.getTopic()) {
            case COMINGOUT: // カミングアウト発話の処理 ← 実際にカミングアウトの情報を
                break; // 取り込んでみましょう
            case DIVINED: // 占い結果報告発話の処理
                break;
            case IDENTIFIED: // 霊媒結果報告発話の処理
                break;
            default:
                break;
        }
    }
    talkListHead = currentGameInfo.getTalkList().size();
}
```

カミングアウト情報の取り込み例

```
/** カミングアウト状況 */
Map<Agent, Role> comingoutMap = new HashMap<>(); // initializeで初期化しておくこと

public void update(GameInfo gameInfo) {
    currentGameInfo = gameInfo;
    // GameInfo.talkListからカミングアウト・占い報告・霊媒報告を抽出
    for (int i = talkListHead; i < currentGameInfo.getTalkList().size(); i++) {
        Talk talk = currentGameInfo.getTalkList().get(i);
        Agent talker = talk.getAgent();
        if (talker == me) {
            continue;
        }
        Content content = new Content(talk.getText()); // 発話をparse
        switch (content.getTopic()) {
            case COMINGOUT:
                // カミングアウト情報の取り込み
                comingoutMap.put(talker, content.getRole());
                break;
            case DIVINED: // 占い結果報告発話の処理
                break;
            case IDENTIFIED: // 霊媒結果報告発話の処理
                break;
            default:
                break;
        }
    }
    talkListHead = currentGameInfo.getTalkList().size();
}
```

カミングアウト情報が取り込めたら

- 例えば、偽の占い師が現れたことがわかる

– 自分のカミングアウト前だったらどうする？

– 偽占い師は人狼か、それとも裏切り者か・・・

などについて、対応を考えてみましょう

補足

Playerの各メソッドの説明

- initialize(GameInfo)
 - ゲーム開始時に一度だけ呼ばれる
 - サーバから送られてくるGameInfoを取得
- update(GameInfo)
 - 各行動の前に呼ばれる
 - サーバから送られてくるGameInfoを取得
- dayStart()
 - 日の初めに呼ばれる
- finish()
 - ゲームが終了した時に呼ばれる

Playerの各メソッドの説明

1日の終わりに呼ばれるメソッド

- `vote()`
 - 投票する相手を選択する
- `attack()`
 - 襲撃する相手を選択する
 - 人狼のみ呼ばれる
- `divine()`
 - 占いする相手を選択する
 - 占い師のみ呼ばれる
- `guard()`
 - 護衛する相手を選択する
 - 狩人のみ呼ばれる

Agentを返す必要あり

Playerの各メソッドの説明

発話のメソッド

- talk()
 - 全体に対して発話する
 - 全員呼ばれるメソッド
- whisper()
 - 人狼だけに対して発話する
 - 人狼のプレイヤーだけが使用するメソッド

Stringを返す必要あり

GameInfoの説明

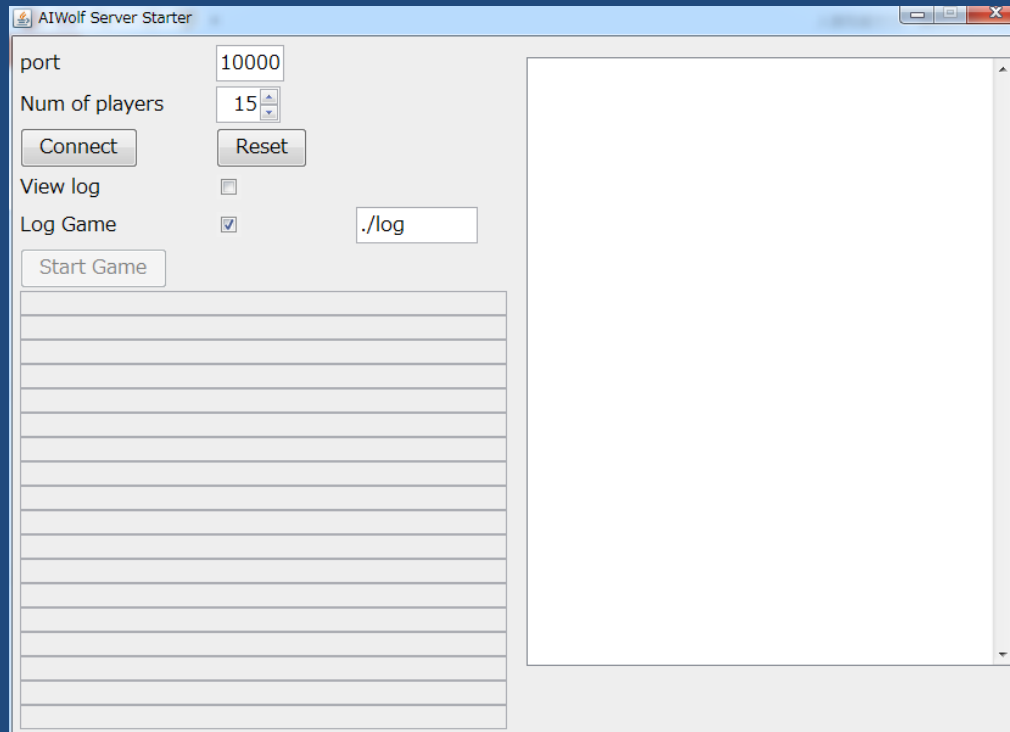
戻り値	メソッド名	説明
int	getDay()	日にちを返す
Role	getRole()	自分の役職を返す
List<Agent>	getExistingRoles()	このゲームに存在する役職のリストを返す
Agent	getAgent()	自分(Agent型)を返す
List<Agent>	getAgentList()	全プレイヤーのリストを返す
Species	getMediumResult()	霊媒結果を返す (霊媒師のみ)
Species	getDivineResult()	占い結果を返す (占い師のみ)
Agent	getGuardedAgent()	前日護衛したプレイヤーを返す (狩人のみ)
List<Agent>	getLastDeadAgentList()	前日死亡したプレイヤーのリストを返す (呪殺された妖狐を含む)
Agent	getExecutedAgent()	前日 追放されたプレイヤーを返す
Agent	getLatestExecutedAgent()	当日 追放が決まったプレイヤーを返す. 決定前はnull
Agent	getAttackedAgent()	襲撃の成否にかかわらず, 襲撃したプレイヤーを返す (人狼のみ)
List<Vote>	getVoteList()	前日 の追放の際の投票リストを返す
List<Vote>	getLatestVoteList()	追放再投票の場合, 前投票のリストを返す
List<Vote>	getAttackVoteList()	襲撃投票のリストを返す (人狼のみ)
List<Vote>	getLatestAttackVoteList()	襲撃再投票の場合, 前投票のリストを返す (人狼のみ)
List<Talk>	getTalkList()	会話のリストを返す
List<Talk>	getWhisperList()	囁きのリストを返す (人狼のみ)
List<Agent>	getAliveAgentList()	生きているプレイヤーのリストを返す
Map<Agent, Status>	getStatusMap()	各プレイヤーの生死の状態を返す
Map<Agent, Role>	getRoleMap()	各プレイヤーの役職を返す. ゲーム中は自分の分かる役職のみ (村人なら自分だけ, 人狼なら仲間の人狼も). ゲーム終了時は全員の役職.

付録: GUIによるゲームの実行

GUIによるゲームの実行

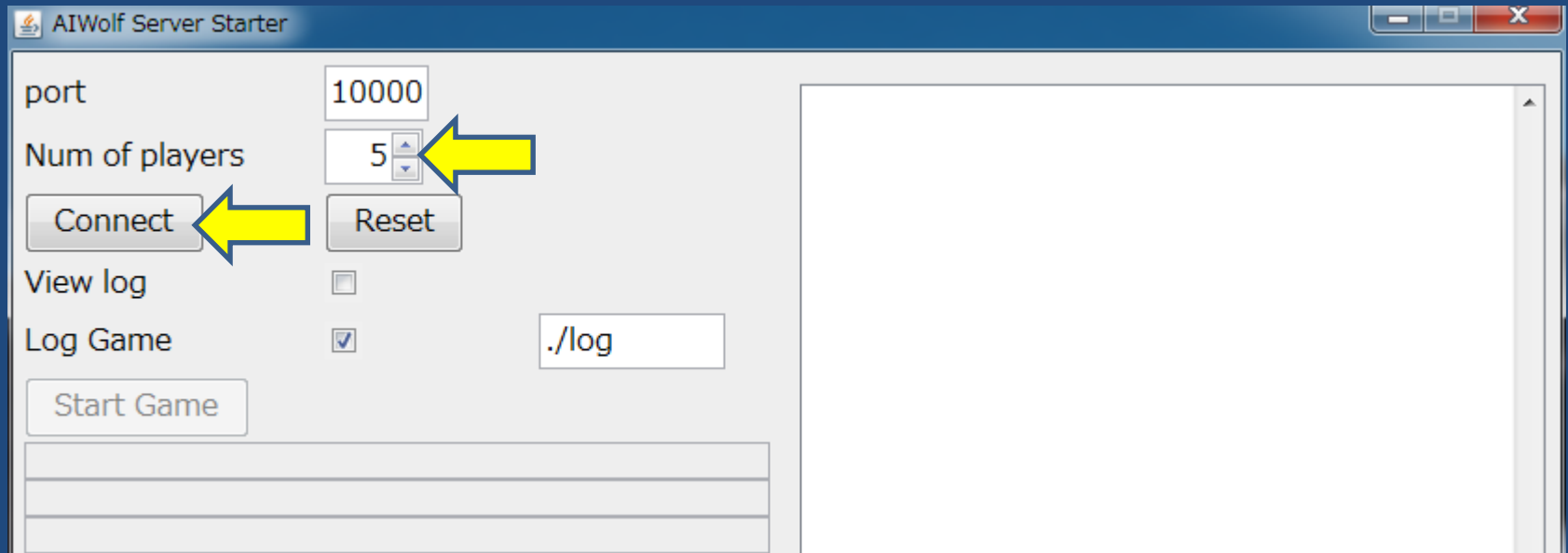
- サーバの起動

- Windowsの場合 : StartServer.batをダブルクリック
- Mac/Linuxの場合 : StartSerber.shをダブルクリック



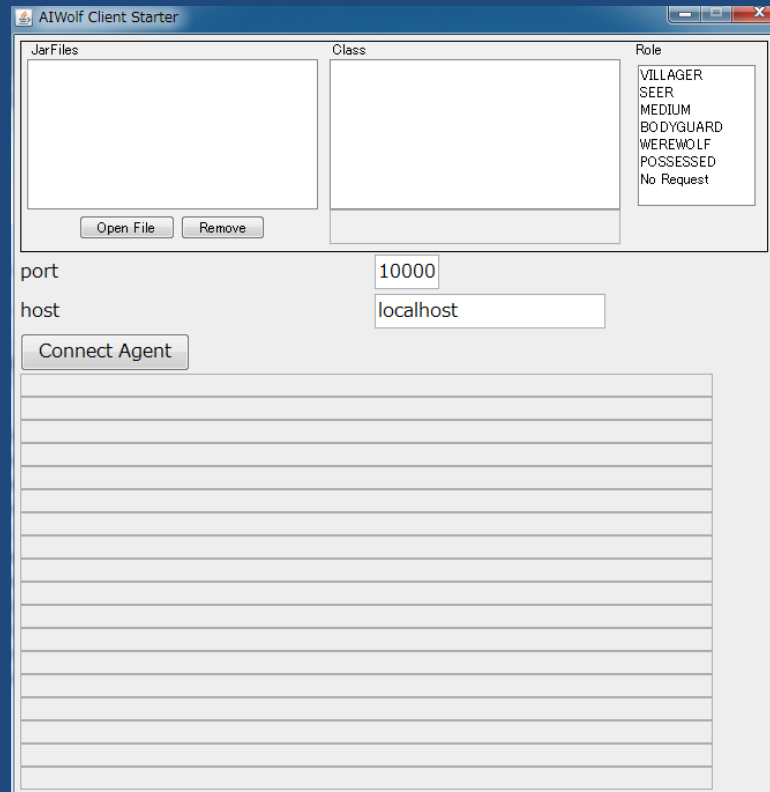
サーバの設定

- Num of playersを5に設定
- Connectをクリックしてサーバを起動



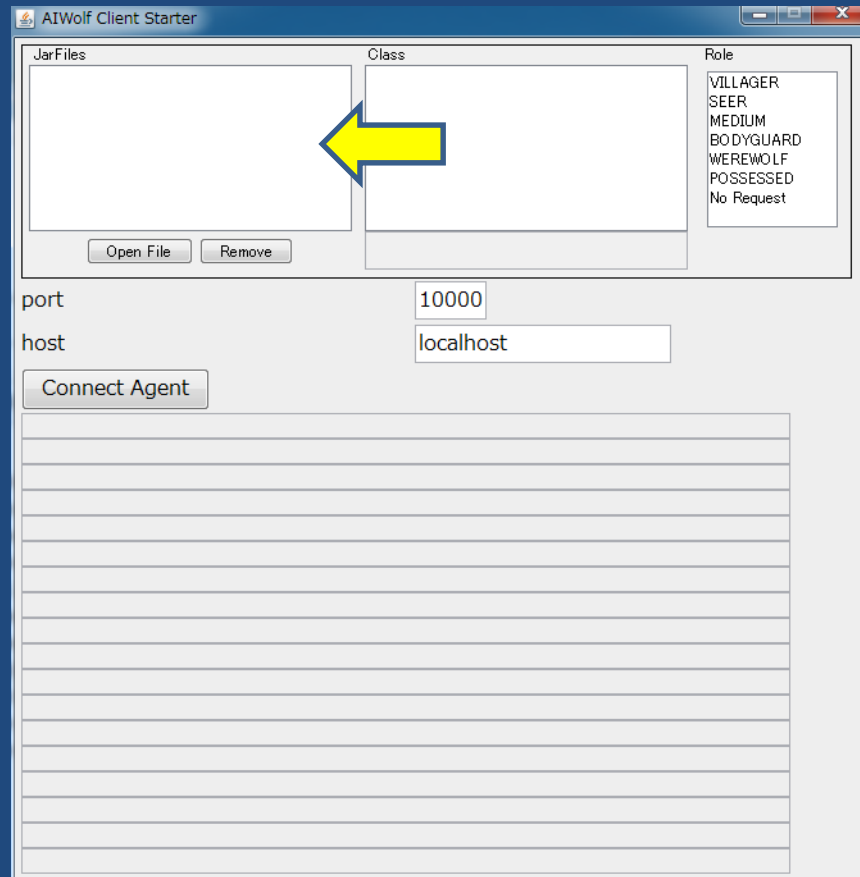
クライアントの接続 (1/3)

- クライアント接続用プログラム
 - Windowsの場合 :StartGuiClient.batを起動
 - Mac/Linuxの場合 : StartGuiClient.shを起動



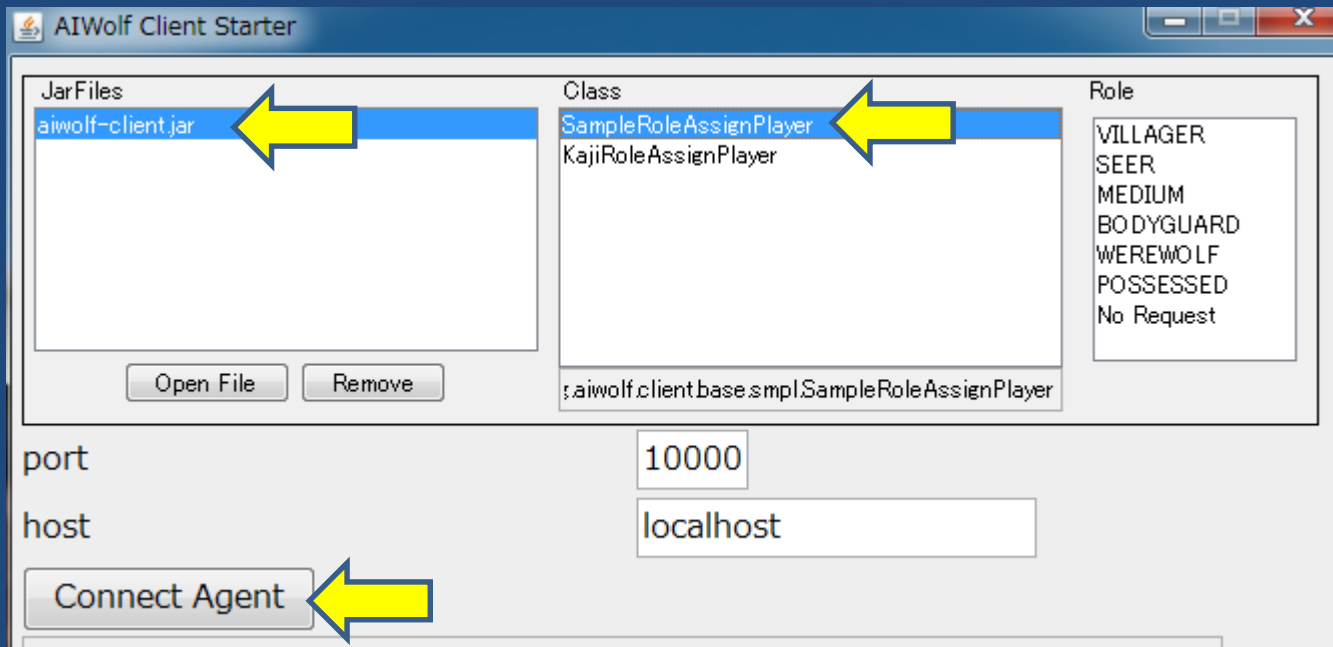
クライアントの接続 (2/3)

- 先ほど作成したjarとaiwolf-client.jarをJarFiles欄にドラッグアンドドロップ



クライアントの接続 (3/3)

- Jarファイル, 接続するプレイヤークラスを選択
- Connect Agentを選択
- Role欄で役職をリクエストすることもできる



ゲームの実行

- 自分のエージェント1体と SampleRoleAssignPlayerを4体Connect
- Server StarterでStart Gameをクリック

AIWolf Server Starter

port: 10000

Num of players: 5

Connect Reset

View log

Log Game ./log

Start Game

jp.ac.cu.hiroshima.info.cm.nakamura.Agent.NakamuraRoleAssignPlayer

SampleRoleAssignPlayer

SampleRoleAssignPlayer

SampleRoleAssignPlayer

SampleRoleAssignPlayer

SampleRoleAssignPlayer

Here comes Agent[01] jp.ac.cu.hiroshima.info.c
m.nakamura.Agent.NakamuraRoleAssignPlayer

Here comes Agent[02] SampleRoleAssignPlayer

Here comes Agent[03] SampleRoleAssignPlayer

Here comes Agent[04] SampleRoleAssignPlayer

Here comes Agent[05] SampleRoleAssignPlayer

5体のエージェントが接続されているのを確認 (ServerStarter)

実行結果

- 実行結果が右側の欄に表示される

