エージェントの作り方

目次

1. エージェント作成前の準備
2. BasePlayerの作成
3. 各役職プレイヤーの作成（村人の場合）
4. 会話の取得方法
5. 発話の生成方法
6. 占い，霊能情報の取得
7. ゲームの流れ
8. エージェント作成前の準備

「Artificial Intelligence based Werewolf （<http://www.aiwolf.org/>）」のページにアクセスし，「資料 ＞ 人狼知能サーバ」から人狼知能サーバの最新版をダウンロードする（6/20時点ではver0.1.5）．

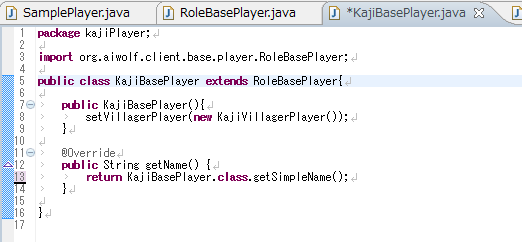


ダウンロードしたファイル内の「jarsファイル」を開くと人狼エージェントを作成するのに必要な4つのjarファイルが入っている．eclipseでエージェントを作成する場合にはエージェントを作成するプロジェクトのビルドパスにこれらのjarファイルを加える．また，「docsファイル」には各クラスのjavadocが入っている．（説明必要？）

1. BasePlayerの作成

　人狼ゲームでは役職によって戦略が異なることと，役職ごとに実装すべきメソッドが異なることから，基本的には各役職ごとのエージェントを作成することになる．org.aiwolf.client.base.player.RoleBasePlayerは，ゲーム開始時にサーバから役職を与えられると，事前にセットしたその役職のエージェントでプレイするようになっているため，このクラスを継承したエージェントを作成すると良い．各役職のエージェントは，コンストラクタ内で「set〇〇〇Player(セットしたいPlayerのインスタンス);」という様にセットする．（下図参照）

例として，KajiBasePlayerクラスを作成する．ここでは役職が村人の時にKajiVillagerPlayerクラスを起動するプレイヤーとしている．他の役職のエージェントも同時にセットすることは可能であり，セットしていない役職のプレイヤーについては，org.aiwolf.client.base.smplパッケージ内の各役職のサンプルプレイヤーが初期状態としてセットされている．また，getName()メソッドをオーバーライドしなければならないが，これはエージェントを示す名前なので好きな名前を返すようにすれば良い．



1. 各役職プレイヤーの作成（村人の場合）

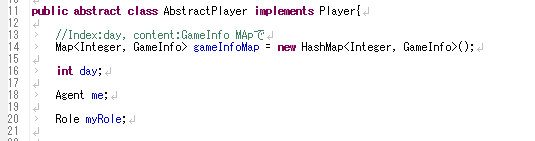
　では，実際に各役職のエージェントを実装してみる．ここではKajiBasePlayerが村人のエージェントとしてセットしていたKajiVillagerクラスを実装する．村人の場合，占いや襲撃のメソッドを実装する必要はなく，実装しなければならないメソッドは，

* update(GameInfo)
* initialize()
* dayStart()
* talk()
* vote()
* finish()

の６つである．org.aiwolf.client.base.player.AbstractVillagerPlayerクラスはこれら以外のメソッドはエラーを返し，またupdate(GameInfo)メソッドとinitialize()メソッドを簡単に実装してある抽象クラスである．KajiVillagerクラスではこのクラスを継承する．

* **update(GameInfo)とinitialize()の実装内容**

　AbstractVillagerPlayer クラスは全役職において必要であろうメソッドを組み込んだAbstractPlayerクラスを継承している．このクラスではinitialize()時に下図のフィールドのme(自分を表すAgentインスタンス)，myRole(自分の役職)フィールドをセットし，update(GameInfo)が呼ばれる際に，gameInfoMap(各日のGameInfo)，day(ゲーム内での現在の日にち)を更新する．

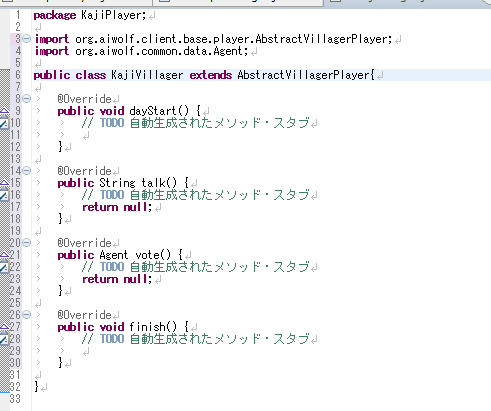


day, me, myRoleの値についてはそれぞれ， getDay(), getMe(), getMyRole()というメソッドで取得でき，gameInfoMapは

* getGameInfoMap()：Map全体を取得
* getGameInfo(int)：指定した日にちのgameInfoを取得
* getLatestDayGameInfo：現在の日にちのgameInfoを取得

というメソッドで取得できる．

AbstractVillagerPlayerを継承し(下図)，実装したい戦略に沿って必要なメソッドを実装する．



1. 会話の取得方法

　update(GameInfo)で取得するGameInfoにはその日の会話のログも含まれており，GameInfo.getTalkList()でList<Talk>を取得できる．Talkクラスが持つフィールドは

* day：発話された日にちを表す
* idx：その日の何番目の発話であるかを表す
* agent：発話者を表す
* content：発話内容を表す(String型)

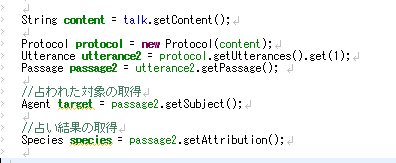
の４つで，それぞれgetDay(), getIdx(), getAgent(), getContent()で取得できる．

contentには人狼プロトコルで書かれた文章が入っているため，中身を理解するためにはパースする必要がある．org.aiwolf.client.libパッケージにあるProtocolクラス，Utteranceクラス，Passageクラスは発話のパースのためのクラスであり，例えば，

**「私は占い師であり，Agent01は人狼であった．みんなAgent01に投票してほしい．」**

という発話は以下のような構造で表される．

すなわち，Protocolは発話全体，Utteranceは発話文，Passageは基礎文を表す(<http://www.aiwolf.org/>内の人狼プロトコル参照)．実際にこれらのクラスを使って，Passage2の「誰が占われ，その結果が何であったか」という情報を取得すると次のようになる．



ProtocolクラスのコンストラクタにString型の発話を引数として入れると，自動的にUtteraceやPassageが作られる．Protocol.getUtterances()でList<Utterance>を取得することができ，Utterance.getPassage()で基礎文を取得できる．この例ではPassage.getSubject()で占われた対象を取得し，Passage.getAttribution()で占いの結果を取得しているが，その他の情報を取得するメソッドについてはjavadoc参照．

1. 発話の生成方法

　次に自分が発話を行う方法の説明をする．先に述べたとおり，このゲームは人狼プロトコルを用いて進行するが，org.aiwolf.client.lib.TemplateTalkFactoryクラスを用いれば人狼プロトコルを知らずとも発話の生成が可能である．現段階で生成可能な発話は7種類．

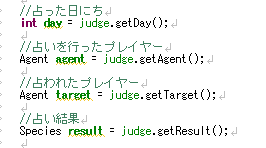
1. estimate(Agent agent, State state)：**「agentはstateだと思う．」**
2. comingout(Agent agent, State role)：**「agentがroleだとカミングアウトする．」**
3. inspected(Agent agent, Species species)：**「agentを占った結果，speciesだった．」**
4. medium\_telled(Agent agent, Species species)：**「霊能の結果，agentはspeciesだった．」**
5. guarded(Agent agent)：**「agentを護衛した．」**
6. skip()：**様子見(発話することは無いが，まだ会話は終わらせたくない)．**
7. over()：**もう発話しなくて良い．**

これらのメソッドでUtteranceクラスを生成できる．サーバに発話を送る時にはString型で送らなければならないため，以下のような手順でString型に変換する．

1. TemplateTalkFactoryクラスでUtteranceを作成(複数の発話なら複数個)
2. ProtocolクラスのコンストラクタにUtterance，またはArrayList<Utterance>を引数として入れてProtocolクラスのインスタンスを作成
3. Protocol.getText()でString型に変換してサーバに送る．
4. 占い，霊能情報の取得

占い師(または霊能者)でプレイする場合，update(GameInfo)で得られる情報の中に，占い結果(または霊能結果)が入っており，GameInfo.getDivineResult()（霊能結果は，

GameInfo.getMediumResult）で取得できる．占い結果と霊能結果はJudgeクラスで表されており，下図の様に各情報を取得できる．



1. ゲームの流れ

Ver0.1.5における人狼サーバでは下記の流れでゲームを行う

update(GameInfo)

initialize()

vote()

divine()（占い師のみ）

attack()（人狼のみ）

guard()（狩人のみ）

update(GameInfo)

talk()

whisper()（人狼のみ）

update(GameInfo)

dayStart()

update(GameInfo)

finish()

**会話終了**

**ゲーム終了**