SYSC-5103 Software Agent Spring/Summer 2016
Group Project

Soccer Agent - Planner Implementation Documentation

Group members:

1) **Babneet Singh (100827441)**
2) **Tosin Agagu (7924240)**
3) **Chen Zhang (7770815)**

Introduction

In a real soccer game, a player reacts according to the current environment state with the objective to win the game. Normally, a experienced player will make quick and smart decisions based upon its understanding of the environment. For our soccer agent, we use a design which consists of four components: **Environment, Planner, Actions, and Executor**. This design implements a STRIPS like planner so that the agent can make sensible decisions while playing soccer.

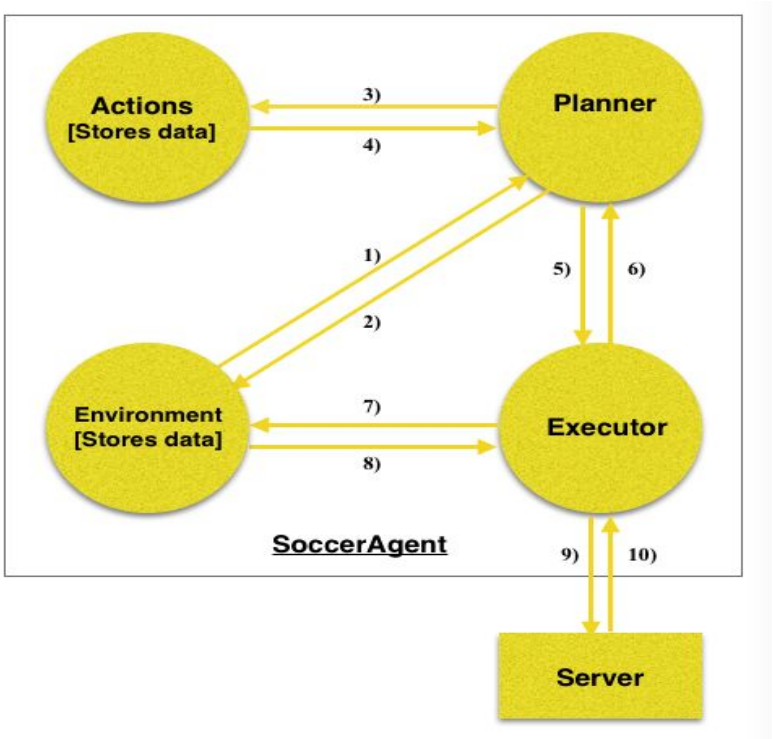

Figure 1: Agent Architecture

Figure 1 above illustrates how the four primary components of the agent interact. “Planner” component is the brain of the agent. It decides what actions to perform after evaluating the environment. “Actions” component stores description of the actions that the agent can perform in the form of STRIPS-like notation. “Environment” component stores information about environment which helps the agent to decide which action to perform. “Executor” component evaluates the various environment properties based upon the information received from the server. It also executes actions for the agent by sending messages to the server.

Environment Properties

Accurate sensing of the environment state is important for the player to make correct judgements. The following are the environment properties or conditions that can be evaluated by our agent:

- **is_ball_visible** - The agent must be able to see the ball for the condition to be true.
- **ball_in_possession** - For this condition to be true, the agent needs to have the possession of the ball.
- **is_being_blocked** - This condition is true if opponent agents are close enough to block or intercept.
- **ball_inside_goal** - This condition is true if the ball and goal location are the same.
- **is_goal_visible** - This condition is true if the agent can see the opponent's goal.

Agent's Actions

Based upon basic actions used by a real soccer player, we define five different actions that can be performed by our agent:

- **pass** - Using this action, the agent can pass the ball to the closest teammate available.
- **locate_ball** - Using this action, the agent will be able to find the position of the ball.
- **intercept_ball** - Using this action, the agent will try to get possession of the ball.
- **locate_goal** - Using this action, the agent will be able to find the position of the goal.
- **score_goal** - Using this action, the agent will kick a ball with the objective to score a goal.

Executor

The executor is implemented in the Executor class. It executes the above defined agent actions, and also evaluates the above defined environment properties. Agent actions are executed by sending a series of messages to the server. Evaluation of environment properties is done by processing messages that are received from the server.

Environment

Environment class implements the environment component. It represents agent's knowledge and perception of the environment. This class evaluates and caches information about the above defined environment properties. The information related to the environment properties is updated whenever new information is received from the server.

STRIPS Notation for Agent's Actions

The agent actions are described in STRIPS like notation in a text file called "AgentActions.txt". The format shown below is used:

action: (intercept_ball) preconditions: (is_ball_visible) additions: (ball_in_possession) deletions: ()

The following three sets of environment properties are defined for each action:

- Preconditions
- Deletions
- Additions

During initialization the description of agent actions is read from “AgentActions.txt”. Then, the text is parsed and the extracted information is stored in instances of AgentAction class. This class stores the following information about an agent’s action: id (name), additions, deletions and preconditions. Additions, deletions and preconditions represent either a single or a collection of environment conditions. For the syntax, the parser handles the following operators: “^”, “,” and “!”. An error is thrown if the above syntax is not followed. The parser is also very strict in terms of spaces outside the brackets. Table 1 below summarizes STRIPS-like description for all agent actions.

Action Name	STRIPS-like Description
pass	preconditions: (ball_in_possession ^ is_being_blocked) deletions: (ball_in_possession) addition: ()
locate_ball	preconditions: () deletions: (!is_ball_visible) additions: (is_ball_visible)
intercept_ball	preconditions: (!ball_in_possession ^ is_ball_visible) deletions: () additions: (ball_in_possession)
locate_goal	preconditions: (!is_goal_visible) deletions: (!is_goal_visible) additions: (is_goal_visible)
score_goal	preconditions: (is_goal_visible ^ ball_in_possession ^ !is_being_blocked) additions: (ball_inside_goal) deletions: (ball_in_possession)

Table 1: Description of agent actions

Planner

The planner is implemented in the Brain class. The planner is given a goal to achieve. For our agent, the ultimate goal is put the ball inside the opponent team's goal. Using the environment information from the instance of Environment class and the description of actions from the instances of AgentAction class, the planner derives a set of actions that will help the agent to accomplish its goal. Once the planner derives the set of actions, it uses the Executor class to execute those actions. This behaviour is performed repeatedly until the end of the game.

No preconditions for the locate_ball action are defined in "AgentActions.txt". This gives the planner a starting point when preconditions for none of the actions end up being true. This prevents the agent from getting stuck.

A new command line option "-mode" is added in the Krislet class: "-mode modified" or "-mode original". This option lets the user decide whether to use the modified version or the original version of Krislet. By default, Krislet now uses the modified implementation.