General Game Playing

Game Management

Programme

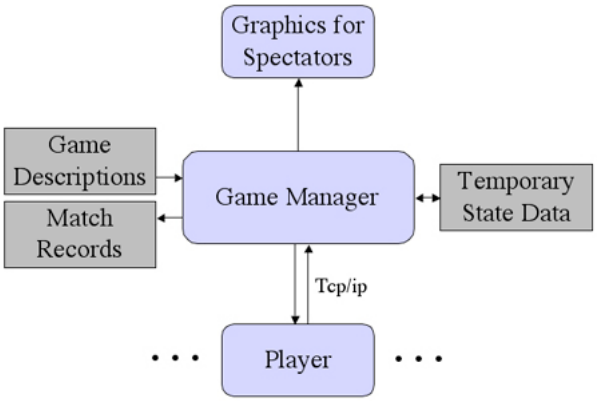
Game / Match Management

Game Communication Protocol

Game Playing Example

This lesson is an overview of game management. More properly, it should be called match management, as the issue is how to manage individual matches of games, not the games themselves. We start with an overview of the General Game Playing ecosystem and the central role of the Game Manager. We then discuss the General Game Playing communication protocol. Finally, we see how it is used in a sample game.

Game Manager



Here is a diagram of a typical general game playing ecosystem. At the center of the ecosystem is the game manager. The game manager maintains a database of game descriptions, maintains some temporary state for matches while they are running, and maintains a database of match results. The game manager communicates with game players using the Internet's TCP/ip protocol. It also provides a user interface for users who want to schedule matches, and it provides graphics for spectators watching matches in progress.

Messages

(info)

(start *id role* (s_1 … s_n) *startclock playclock*)

(play *id* (a_1 ... a_k))

(stop *id* (a_1 ... a_k))

(abort *id*)

In the current GGP communication language, there are five types of messages used for communication between the Game Manager and game players - namely info, start, play, stop, and abort. Let's look at each of them in turn.

The info Message

An *info* message is used to determine if a player is running, communicating, and ready to play a match.

General Form:

```
(info)
```

Replies:

`ready` - ready to play

`nil/busy` - not ready

An info message is used to confirm that a player is running, communication, and ready to play a match. The general form is shown here. Upon receipt of an info message, a player is expected to return ready if it is ready to receive messages. Otherwise, it should return nil. Of course, if a player is not running or not connected, there will be no response. So, in practice info messages sometimes evoke no reply at all.

The start Message

A start message initiates a match.

General Form:

`(start` *id role* `(`s_1 … s_n`)` *startclock playclock*`)`

Reply:

`ready` - ready to begin play

NB: Match begins as soon as all players have replied *or* when *startclock* seconds have elapsed whichever comes first.

A start message is used to initialize a match. The general form of a start message is shown here. The message begins with the keyword start, and this is followed by five arguments, viz. a match identifier (a sequence of alphanumeric characters beginning with a lower case letter), a role for the player to play (chosen from the roles mentioned in the game description), a list of game rules (written as sentences in Prefix GDL), a startclock (in seconds), and a playclock(also in seconds).

Upon receipt of a start message, a player should prepare itself to play the match. Once it is done, it should reply ready to tell the Game Manager that it is ready to begin the match. The GGP protocol requires that the player reply before startclock seconds have elapsed. If the Game Manager has not received a reply by this time, it will proceed on the assumption that the player is ready.

The play Message

A play message is a request for an action.

General Form:

(play *id* nil)
(play *id* (*a*$_1$ ... *a*$_k$))

Replies:

legal action

NB: nil is argument on first step, (*a*$_1$ ... *a*$_k$) thereafter, where *a*$_1$ ... *a*$_k$ are actions of players on preceding step.

NB: If player returns an illegal action *or* playclock has elapsed, Manager substitutes a random legal action.

A play message is used to request an action from a player. The general forms of the play message are shown here. In each case, there is an identifier for the match. On the first request, where there is no preceding move, the second argument is nil. On all subsequent requests, the second argument is a list of the actions of all players on the preceding step. The order of the actions in the record is the same as the order of roles in the game description. Note that if a player fails to reply or does reply in time, the Manager substitutes a random legal action.

Upon receipt of a play message, a player uses the move information to update its state as necessary. It then computes its next move and returns that as answer. The GGP protocol requires that the player reply before playclock seconds have elapsed. If the Game Manager has not received a reply by this time or if it receives an illegal action, it substitutes an arbitrary legal action.

The stop Message

A stop message is used to inform players that the match has terminated successfully.

General Form:

`(stop` *id* `(`a_1 `...` a_k`))`

Replies:

`done`

A stop message is used to tell a player that a match has reached completion. The general form of the stop message is shown here. There is a match id and a record of the actions of all players on the preceding step.

Upon receipt of a stop message, a player can clean up after playing the match. The move is sent along in case the player wants to know the action that terminated the game. After finishing up, the player should return done.

The abort Message

An abort message is used to tell players that a match has terminated abnormally.

General Form:

`(abort` *id*`)`

Replies:

`done`

An abort message is used to tell players that a match has terminated abnormally. It differs from a stop message in that the match need not be in a terminal state. Upon receipt of an abort message, a player can eliminate any data structures and return to a ready state. Once it is finished, it should return done.

Match Management Procedure

Begins with receipt of request to run a match of a given game with given players and given startclock and playclock.

(1) send info messages to players.
(2) send start message with appropriate parameters.
(3) send play messages to receive plays
(4) send stop message when game terminates
(5) send abort for abnormal termination

In summary, the process of running a match goes as follows. It begins with receipt of a request to run a match. The Game Manager may at its discretion send info messages to the players to ensure that they are ready to play. When it is ready, it sends a start message with appropriate arguments to each player to initiate the match. Once game play begins, the manager sends play messages to each player to get their plays; and it then simulates the results. This part of the process repeats until the game is over. The Manager then sends a stop message to each player. If anything goes wrong, the Manager may send an abort message at any time to terminate the match.

Tic-Tac-Toe Example

```
Game Manager to Player x: (start m23 white description 10 10)
Game Manager to Player y: (start m23 black description 10 10)
Player x to Game Manager: ready
Player y to Game Manager: ready
```

Here is a sample of messages for a quick game of Tic-Tac-Toe. The game manager initiates the match by sending a start message to all of the players, each with a different role. The players then respond with ready. They can respond immediately or they can wait until the start clock is exhausted before responding.

Tic-Tac-Toe Example

```
Game Manager to Player x: (play m23 nil)
Game Manager to Player y: (play m23 nil)
Player x to Game Manager: (mark 1 1)
Player y to Game Manager: noop
```

Play begins after all of the players have responded to the start message or after the startclock has expired, whichever comes first. The manager initiates play by sending a play message to all of the players. Since this is the first step and there are no previous moves, the move argument in the play message is nil. In this case, the first player responds with the action (mark 1 1), one of its nine legal actions; and the second player responds with noop, its only legal action.

Tic-Tac-Toe Example

```
Game Manager to Player x: (play m23 ((mark 1 1) noop))
Game Manager to Player y: (play m23 ((mark 1 1) noop))
Player x to Game Manager: noop
Player y to Game Manager: (mark 1 2)
```

The Game manager checks that these actions are legal, simulates their effects and updates the state of the game, and then sends play messages to the players to solicit their next actions. the second argument in the play message is a list of the actions received in response to the preceding play message. In this case, the first player responds with noop, its only legal action; and the second player responds with (mark 1 2), which is not a good choice.

Tic-Tac-Toe Example

```
Game Manager to Player x: (play m23 (noop (mark 1 2)))
Game Manager to Player y: (play m23 (noop (mark 1 2)))
Player x to Game Manager: (mark 2 2)
Player y to Game Manager: noop
```

Again, the Game Manager checks legality, simulates the move and updates its state, and sends play messages requesting the players' next actions. The first player takes advantage of the situation and plays (mark 2 2) while the second player does noop.

Tic-Tac-Toe Example

```
Game Manager to Player x: (play m23 ((mark 2 2) noop))
Game Manager to Player y: (play m23 ((mark 2 2) noop))
Player x to Game Manager: noop
Player y to Game Manager: (mark 1 3)
```

There is not much the second player can do in this situation to save itself. Instead of staving off the immediate loss, it plays (mark 1 3), while the first player does noop.

Tic-Tac-Toe Example

```
Game Manager to Player x: (play m23 (noop (mark 1 3)))
Game Manager to Player y: (play m23 (noop (mark 1 3)))
Player x to Game Manager: (mark 3 3)
Player y to Game Manager: noop
```

The Game manager again simulates, updates, and requests a move. In this case, the first player goes in for the kill, playing (mark 3 3).

Tic-Tac-Toe Example

```
Game Manager to Player x: (stop m23 ((mark 3 3) noop))
Game Manager to Player y: (stop m23 ((mark 3 3) noop))
Player x to Game Manager: done
Player y to Game Manager: done
```

With this move, the game is over. As usual, in such cases, the Manager lets the players know by sending a suitable stop message. It then stores the results in its database for future references and terminates.


GENERAL
GAME
PLAYING