

Automated Commentaries for Simulated Soccer

CommentaryProducer Class

Audience	All
Author	Justin Hogg
Scope	
Date Created	05/03/2007
Version Number	0.1
Version History	0.1 – Original Document (JH)
Reviewed	Yes
Last saved by Justin Hogg, 18/03/2007	

Sign-off sheet

Date: 18/03/2007

Document Author Signature: JH*

Document Author Name: Justin Hogg

Quality Assurance Signature: AM*

Quality Assurance Name: Ahsan Mussa

Project Manager Signature: AM*

Project Manager Name: Ahsan Mussa

* By signing this document you approve that the entire contents of the deliverable has been reviewed and is in line with the objectives of the project.

Automated Commentaries for Simulated Soccer

1. Introduction

This class is responsible for producing text-based comments that relate to game events on the simulated football field (the soccer monitor). This is achieved by making calls to the `CommentTemplateStore` class for comment templates, and marking them up with any required variables.

1.1 Overview

The `CommentaryProducer` (CP) class defines methods to return a `String` comment for each type of event that can occur during a football game played on the system. These cover simple events such as pass, losing possession of the ball, missed shots, goals, and more intelligent commentary such as counter attacks and opportunities for a team to cross the ball for a goal attempt.

1.2 `CommentaryProducer` (CP)

The `GameAnalyser` (GA) class analyses the football game, identifies game events and formations as they occur, and makes calls to the CP. These calls are named after the game event they represent – for example – the method to represent a pass is called `speakPass()`. The GA includes any necessary parameters in such calls; a pass takes two integers representing the passing player and the receiver.

The CP method then makes a call to the `CommentTemplateStore` (see `CommentTemplateStore0.1`) to obtain a suitable comment object (see `Comment0.1`) for that type of event, and again, the methods are named accordingly – for a pass the method is `getPassComment()`.

Once the CTS call returns the comment object, a full comment sentence must be formed. A few comment objects contain a full comment held in a string, and no variables are necessary. In this instance, the single string can be returned via a call to the comment object, and passed to the `CommentScheduler` (CS) (along with that comments importance rating – returned via another call to the comment object) for entry into the proposition pool (see `CommentScheduler0.1`).

Most comments however, require mark-up with variables and this type of comment object contains a number of string parts that are split at the point a variable is required. In this case, the CP will extract, in turn, each string part via calls on the comment object and add them to a new string to represent the complete comment. It also adds the variables (provided as parameters in the call) in the appropriate place.

Once the complete comment is formed, it is passed to the CS, along with its importance rating, for adding to the proposition pool.

Automated Commentaries for Simulated Soccer

Some code illustrates this process:

```
public void gameStart(String teamL, String teamR) {  
  
    Comment matchStart = cts.getMatchStartComment();  
  
    String completeComment = matchStart.getPartOne() + teamL +  
                             matchStart.getPartTwo() + teamR +  
                             matchStart.getPartThree();  
  
    cs.addToPool(matchStart.getImportance(), completeComment);  
  
}
```

The CP class contains methods of this type for all game events identified by the system.

In addition it contains a method to mark-up player numbers, and a method to help filter calls from the GA that make more than one call for the same occurrence of a game event.

Player numbers are passed from the GA as an integer in the range of 1-22. It is then necessary to determine the name of the team and the player number within that team. The teams do not change name in our system and are called "Reds" and "Blues". A parameter in the range of 1-11 is marked up with the team name (blue), and a parameter of 12-22 is marked up with the team name (red) and the parameter minus 11, to give a player number of 1-11 within the red team.

Certain event calls from the GA are repeated for the same instance of a game event:

```
speakFreeKick()  
speakTeamkeepingPossesion()  
speakWinsPossesion()  
speakOneTwo()  
speakPossessionLost()
```

Without a control mechanism, several comments relating to a single instance of an event in the game can occur, and this repetition is undesirable and unrealistic. To prevent this, the five methods above query the time of the last occurrence of that type of event. If the previous occurrence was over 2000ms ago, then the comment is entered into the proposition pool. If the last occurrence was less than 2000ms ago, it is disregarded.

This works well as events are sent from the GA in this fashion about every 100ms. 2000ms is sufficient for the state of play on the field to have changed and a call relating to a new state of play to be available.

Automated Commentaries for Simulated Soccer

Whilst most repetitions are eliminated, during extended periods of low-activity the game event relating to one of the five listed events may still be taking place on the field, and a second call may be successfully made i.e. the string describing the event is added to the proposition pool. As these strings are different for subsequent calls to that method, it will produce a different phrase describing the same event, and it is preferable (as a conscious design choice) to have a slightly different phrase output as audio by the system, than a sustained period of silence.

2. Previous versions

Section 1.2 describes the final version of the CP class, and the prioritisation & scheduling of the audio commentary is dealt with in the CommentScheduler (CS) class.

Early iterations of the project did not include the CS class. The very first iterations could handle all comments in a timely manner. It wasn't until the number of events that could be identified and passed to the CP from the GA increased, that prioritisation became necessary – this was due to the fact that the total duration of all comments exceeded the time available to play their audio counterparts.

This was dealt with by means of identifying two levels of comment:

Compulsory – this type of comment must always be spoken, examples being goals, full time comments and goal shots.

Optional – this type of comment does not always need to be spoken. Examples include passes, loss of possession and a player dribbling the ball.

If there is a compulsory comment available it will take precedence by the system and the optional comment will be discarded. Another case where an optional comment may be discarded is if there are too many comments for all of them to be output as audio without producing a lag between the audio commentary and the events taking place on the soccer monitor (the simulated football field). In this instance the latest comment (the comment corresponding to the most recent event taking place in the game) will be output, and the previous ones discarded.

This functionality was built into the methods that return an optional comment type, within the CP class. All comments that are output as audio timestamp a variable to identify the start time of the speech.

When calls are made from the GA for optional comments (the most frequently occurring), a check is made to determine if the previous comment output as audio was over a certain time ago – this time is the length of the longest comment – and ensures the previous audio comment has finished. If this is the case the next optional comment can be output, if not it is discarded. The same check is not made for compulsory comments and as a result, this

Automated Commentaries for Simulated Soccer

method was too crude to handle a high number of compulsory comments effectively and some goals were not spoken. This was due to the fact that the audio classes have no temporary buffering mechanism (i.e. queue) for multiple comments, and if several comments are passed to it in a short space of time, that is, if more than one comment is passed to it before the current one finishes, then the latest comment to be passed overwrites any earlier ones, and only the latest is converted to audio.

As a more complex method was necessary, I decided to move it into a separate class for clarity, that would handle scheduling and prioritisation of comments (see CommentScheduler0.1).

3. Evaluation

Similar to the CTS, this class has also been continually updated over the course of the project, as the GA class was able to determine an increasing number of simulated game events, the CP required methods to be coded to mark-up and produce comments for those events.

The final version of the CP receives calls from the GA relating to game events, and produces a complete comment string that includes any variables in the correct place (as detailed in section 1.2). It also marks up the player numbers and greatly reduces repetitions of comments for the same instance of a game event.

The class works well without any negative impact on the system performance as a whole. String manipulation is fast and does not produce a lag on the audio commentary.

Although player mark-up does not affect performance, it would be more efficient to identify each player correctly in the GA when the system starts, and pass the correct player number and team as a parameter. This would reduce the number of method calls by eliminating the need for the markUpPlayer() method.

Similarly, it would be better practice for the GA to determine if an occurrence of one of the five repeating method calls to the CP, discussed in section 1.2, was a repetition of the same game event instance. As the GA does not do this, it creates unnecessary work for other classes. It is not the purpose of the CP to eliminate redundancy, but to mark-up and produce comments. The purpose of the GA is to analyse the game and make calls to the CP based on the events taking place in the football game. The GA should only make one call per game event to ensure prioritisation mechanisms only handles what is necessary.

As the final version of the CP class does not deal with prioritisation and scheduling, this will be detailed and evaluated in CommentScheduler0.1.