An Investigation into the Simulation of Five-A-Side Football Using Empirical Modelling

- Neil Turner -
1999/00

- Supervised by Dr Steve Russ -
Contents

Abstract

Administrative Information

Keywords

Author's Assessment of the Project

1. Introduction

2. Empirical Modelling Overview
   What is Empirical Modelling?
   How has Empirical Modelling been applied?
   Further Information

3. The Empirical Modelling Approach
   The Empirical Modelling Life Cycle
   Evaluation and Future Development

4. Analysis
   LSD Analysis
   Analysis Implications
   Model Requirements
   Conclusion

5. Design and Implementation
   The Design and Implementation Cycle
   Design
   Implementation
   Experimentation
   Investigation
   Execution of the Design and Implementation Cycle
   Conclusion

6. Program Description
   The Set-up of the Model

   Inactive Agents
   The Pitch
   The Ball
   The Match
   The Teams
Active Agents
The Spectator
User Controlled Agents
An ADM3 Influenced Design
Design Similarities
Additional Design Requirements
The Referee
Outfield Player
Goalkeeper

7. User Interface
The Star Life Cycle
Requirements Specification
Task Analysis
Conceptual and Formal Design
Prototyping
Implementation
Evaluation
Evaluation of the Final User Interface
Conclusion

8. Testing
Testing Strategy
Unit Test
Integration Test
Validation Test
Conclusion

9. Evaluation
How good is the final model?
How Suitable is Empirical Modelling for Creating Real-World Models?
Comparison with Conventional Computer Modelling Techniques

10. Future Development
Improving the Model
Constructing an Autonomous Model
Comparison with conventional computer modelling techniques
Other developments

11. Conclusions

Bibliography

Acknowledgements
Addendices

Visual Basic User Screen Prototypes
  Spectator User Screen Prototype
  Referee User Screen Prototype
  Player User Screen Prototype
  Goalkeeper User Screen Prototype

Model File Descriptions
Abstract

This report describes the complete life cycle from conception to future development for a computer simulation of the well known game of five-a-side football. It looks at the relative strengths and weaknesses of applying an empirical modelling approach to the problem and is one of the first such models to use the ADM (abstract definitive machine) to animate the LSD analysis and deliver a distributed model. The empirical modelling approach is briefly compared to more conventional programming techniques and assessed as a tool for creating accurate real-world computer models of a similar domain.

Administrative Information

Project Supervisor:

Dr Steve Russ,
Dept of Computer Science,
University of Warwick
Coventry,
CV4 7AL.

Keywords

Keywords relating to the project:

- Empirical Modelling
- Computer Simulation
- Five-a-side Football
- Definitive Programming
- LSD
- ADM (Abstract Definitive Machine)
Author's Assessment of the Project

To aid the reader in finding out what might be of interest to them within the report and as a description of the project in a nutshell a brief author's assessment of the project has been included.

What is the technical contribution of this project?
The significant technical contribution of the project is as the first major distributed (i.e. more than one computer is used) computer model developed using the Empirical modelling approach and the ADM3 and Dtkeden tools. This is also significant because the ADM3 tools is used to animate the LSD analysis of the game of five-a-side football, making the LSD specification the most important part of the model.

Why should this contribution be considered either relevant or important to computer science?
This contribution should be considered important for two reasons. Firstly because such an exercise has not been attempted before and secondly because the subsequent investigation into the practical application of Empirical Modelling is important in establishing this technique and Empirical Modelling as a whole as a desirable and recognised computer modelling method.

How can others make use of the work in this project?
The work contained in this project serves many roles. Firstly it can be used to aid someone developing a similar model using the same approach. Secondly it can be used by someone investigating the worth of Empirical Modelling in creating computer models of the real-world and finally it can be used as the basis for a more complex computer simulation of five-a-side football.

Why should this project be considered an achievement?
This project should be considered an achievement because firstly it is the first of its kind, such an ambitious model has never been constructed before using this approach. Secondly because an accurate and very good simulation of the game of five-a-side football has been constructed and finally because a significant contribution has been made to the ongoing research in to Empirical Modelling and its application.
What are the weaknesses of this project?

There are a few weaknesses in this project. Firstly, the model constructed could have been more accurate, certainly it would have been if more time had been available. Secondly, a more thorough evaluation of Empirical Modelling and comparison with more conventional techniques could have been carried out. Finally, the system being modelled could have been more adventurous as a number of models of games have been developed in the past including a previous attempt by Allan Watkins to model association football using Empirical Modelling.
Chapter 1. Introduction

It is claimed that, "The emphasis of Empirical Modelling is to establish a stronger and more permanent link between what we observe in the 'real-world' and our model of it", (Empirical Modelling, 1999). It would seem therefore that Empirical Modelling techniques are ideal for creating accurate computer models of the 'real-world', perhaps better than other more conventional techniques. Is this however really the case? Just how suitable is Empirical Modelling for creating accurate computer models of the 'real-world' and how does it compare with more conventional techniques? To help answer these questions I have decided to investigate Empirical Modelling techniques by putting them to the test and create a computer model of the well known game of five-a-side football. The aim is to create as accurate a model as possible given the time scale available.

The game of five-a-side football was chosen for a number of reasons. Firstly I am very familiar with the game and therefore a minimum amount of background research is required. Secondly it is desirable to limit the scope of the system we wish to model because we do not want a completely open-ended model where anything can happen. This is similar to artificial intelligence where it is desirable to initially create agents that can operate within a limited domain before attempting to create agents that can operate within a much more open ended domain such as the real-world. Finally five-a-side football was chosen because Empirical Modelling techniques have proven successful in constructing a number of computer models of other games such as cricket and tic tac toe although these were not to the level of complexity envisioned for this model.
Chapter 2. Empirical Modelling Overview

Empirical Modelling is still a relatively unheard of computer modelling technique therefore before looking at its application it is recommended that the reader gains a brief overview of the subject area, it is for this reason that this chapter has been included. If the reader is already familiar with Empirical Modelling they may skip this chapter.

What is Empirical Modelling?
Empirical Modelling is an ongoing computer modelling technique developed within the Computer Science Department at the University of Warwick. Research into the area through the Empirical Modelling project has been going on since 1981 and was initiated within the department by Dr. Beynon. The approach attempts to create accurate computer models using observations of the real-world and as we have seen the emphasis is very much on establishing a stronger and more permanent link between the two.

Unlike conventional programming Empirical Modelling is primarily concerned with, "the power of the computer to represent state - in particular, state which is easily interpretable" (Empirical Modelling, 1999). Therefore unlike conventional programming we are interested in a system's state and not the programming and algorithms that have lead to it. To derive our desired states we carry out, "analysis and exploration for the sake of understanding" (Empirical Modelling, 1999) and, "it is characteristic of Empirical Modelling that the focus is upon the construction of environments for interaction and experiment in a domain that is not yet well-understood" (Empirical Modelling, 1999).

The three key elements of Empirical Modelling are:

- Observation
- Dependency
- Agency

Observation is concerned with the idea that, "we must observe and experiment with the thing being modelled to gain an understanding of it" (Empirical Modelling, 1999). Our understanding therefore is
based on that observation and most likely our perceptions of cause and effect within the system being modelled. For example within the game of five-a-side football we can observe such things as the position of a player on the pitch and their distance from the opponents' goal. Dependency is concerned with this perceived relationship between a cause and effect for the system's observables. These dependencies are represented using spreadsheet like definitions such as 'G = BC + RS' which represents the idea that a goal (G) is given if the ball crosses the goal line (BC) and the referee sees this occur (RS). Agents are the objects within the system that can be seen to trigger these sequences of cause and effect therefore for five-a-side football examples might be the outfield players, the referee and the match ball.

How has Empirical Modelling been applied?
Empirical Modelling has been applied to many areas including:

- **Software and system development**
  - Prototyping educational software
  - User-interface design
  - Visualisation for mathematics and science

- **Engineering design**
  - Virtual prototype of lathe shaft design
  - Vehicle cruise control simulation
  - Sailing boat simulation

- **Systems of agents**
  - Railway design and operation
  - Simulation of classroom interaction
  - Simulation of the game of cricket
Further information

For more information about Empirical Modelling please consult the web pages of the Empirical Modelling group within the department of Computer Science at the University of Warwick. The pages can be found at the following address:

http://www.dcs.warwick.ac.uk/modelling
Chapter 3. The Empirical Modelling Approach

The model has been constructed using the Empirical Modelling approach therefore it is important that the reader is familiar with this approach and its life cycle. For the reader's convenience the report has been laid out using the same structure, shown in figure 3.1.

![Figure 3.1](image)

**Figure 3.1** The life cycle for the Empirical Modelling approach.

The Empirical Modelling Life Cycle

As we can see the empirical modelling life cycle is somewhat different from the classic software life cycle model. An analysis of the system being modelled is initially carried out to identify the observables, dependencies and agents. This leads directly to the design and implementation stage where the modeller is concerned with creating, "a computer representation of the domain (system being modelled) in which a script of definitions corresponds to a particular state of the model" (Empirical Modelling, 1999). Using the analysis as a basis a basic script representing the model is designed and implemented and experimentation takes place to investigate dependencies and states within the model:
"Any changes the modeller makes in the definitions will propagate automatically throughout the script (hence also through the model) to maintain the dependencies - very much in the style of a spreadsheet. When the modeller is confident that certain sequences of changes give experimental results correlating closely with similar real world experiments these sequences can be automated, and in this way a full behavioural model can be gradually developed." (Empirical Modelling, 1999)

The model is gradually built up using this process. New model features are designed and added and experimentation and investigation is carried out until the modeller is satisfied that the model behaviours as expected.

Once a satisfactory model has been constructed this then needs to be tested to make sure that it both functions correctly and does indeed closely correspond with the real-world system being modelled. This may lead to further model changes as problems are found.

It is apparent that this process has a great deal in common with both the prototyping software life cycle model in terms of the emphasis on evaluation and experimentation and the incremental software life cycle in terms of the gradual development (Pressman, 1997).

Evaluation and Future Development
Although not a requirement of the life cycle an evaluation of the model and of the Empirical Modelling approach will also be carried out to enable the project objectives to be met. Future development of the model and of the project in general will also be discussed.
Chapter 4. Analysis

As with any software exercise the analysis is perhaps the most important stage of the life cycle because all of the subsequent stages are based on this analysis, "A complete understanding of the software requirements is essential to the success of a software development effort. No matter how well designed or well coded, a poorly analysed and specified program will disappoint the user and bring grief to the developer" (Pressman, 1997: p286). Empirical Modelling is no different. We need to ensure that we know precisely what we require from the model before we start building it and to achieve this a thorough analysis of the thing being model needs to be carried out. In this case an LSD analysis was used to do this. This analysis and its implications for the rest of the model's life cycle are discussed in this chapter.

LSD Analysis

An LSD analysis is concerned with our observations of what is going in within the system we wish to model, "Observation-orientated modelling is used to describe the interaction between agents. It involves the systematic analysis and metaphorical representation of the observables through which stimulus and response are mediated. The LSD notation has been developed for this purpose" (Empirical Modelling, 1999). The LSD notation is shown in figure 4.1.

![LSD Analysis Diagram](image)

*Figure 4.1 The LSD notation (Empirical Modelling, 1999)*
As we can see features of the model are grouped around agents that can be thought of as the objects within the model. In this case an agent might be a striker, the referee or the match ball. The observables based around each agent are variables that we can observe within the system we are modelling such as their distance from the ball, their current position and their current activity. Observables are classified as being a state, an oracle or a handle. States are observables that an agent owns such as a player's shooting skill. Oracles are observables that an agent can react to (i.e. can observe) such as the position of the opponents' goalkeeper when shooting at goal and handles are observables that the agent can change such as the ball's position through kicking it. Within the analysis we also observe dependencies within the model called derivates such as the relationship between the ball crossing the goal line and a goal being given. This is of course assuming that the referee has seen this occur and that there have been no rule infringements. Finally the analysis observes actions that an agent may carry out given certain conditions, these are called protocols. Examples of these include the fact that if the player is in possession of the ball and the match is in open play then he may shoot the ball at either the opponents' goal or his own.

An extensive LSD analysis of five-a-side football was carried out for the model using my own knowledge of the game and this can be found in the appendix. There were often changes after discussions with Dr. Russ as new observations became apparent and the analysis included is the final version used for the initial design. The analysis was carried out using the five-a-side game specified in my guide to five-a-side football, this can also be found in the appendix.

Analysis Implications

It is important to note that the analysis of the game of five-a-side football had very significant implications with regards to the requirements of the planned computer simulation. Although the requirement of an accurate computer model has stayed consistent throughout the type of model required has changed significantly. Initially the aim was create a model with a high degree of autonomy. Players were to know when they should shot the ball, make a tackle or take up a defensive position. However on further deliberation of the problem I came to the conclusion that this would not be feasible to the degree I desired. This is because this is an exercise in Empirical Modelling and not artificial intelligence therefore we should not have to concern ourselves with the hugely complex problem of having to recreate the footballer's brain within the computer model. Although this would
obviously be a worthwhile exercise and was attempted to a lesser degree within Alan Watkin's 1998 project entitled, "Football Simulation using Empirical Modelling Methods" it is the author's opinion that such a feat is not yet achievable. The RoboCup competition (Coradeschi, 1998) has shown that even with cutting edge research artificially intelligent football players are still a very long way away from being as intelligent as their real life counterparts and rather than construct a model with very simplistic behaviour the author would prefer to build one that utilises the user's own intelligence. Therefore a model with user controlled behaviour is required where the user is able to carry out interactions with the model within the five-a-side football environment. This will require the user being able to carry out the actions specified within the LSD analysis.

**Model Requirements**

The requirements of the model have changed significantly due to the switch to user controlled behaviour. A user controlled model suggests that a distributed model will be required so that the multiple agents involved can all be user controlled. Using the LSD analysis we therefore need to identify those agents within the model who are able to influence the five-a-side game through behavioural actions because these actions possibly need to be user controlled. The agents able to carry out such actions and therefore who require separate user interfaces are as follows:

- The referee
- The outfield players
- The goalkeepers

Initially the requirement will be made that each team is composed of a goalkeeper and two outfield players because this minimum that allows all of the actions such as saves, tackles and passes to take place. If time allows the full quota of players can be added to each team.

Using the LSD analysis we can identify the actions that the user must be able to carry out for these agents. For example the analysis states that an outfield player currently in possession of the ball must be able to pass the ball therefore a requirement of the model is that it allows the user to somehow request an outfield player in possession of the ball to pass it. The LSD analysis also provides us with the user's required observables (i.e. oracles), that is the observables that must be somehow displayed to
the user. For example a goalkeeper knows his own name therefore there must be some way for the user controlling the goalkeeper's behaviour to find this information out. These agents will therefore require a user interface to display this information and to allow the user to interact with the computer model (the requirements of each user interface are set out in chapter 7). Finally although you obviously do not have to have a spectator for a game of five-a-side football to take place for completeness I have seen it fit to include the requirement that the spectator agent has a user interface to display the agent's observables. This information could be hidden away from view but I feel that it provides a more realistic model if we have some way of externally viewing the match.

Those requirements not directly set out in the LSD analysis can be derived from the rules for this particular version of five-a-side football, as laid out in the 'Guide to Five-a-Side Football' in the appendix. These rules lead to the following requirements:

- The ball must bounce off the edges of the pitch except when it enters either of the goals.
- The ball can not go above head height therefore only two dimensional co-ordinates for the ball and players are required (i.e. the players will not have to jump to head the ball).
- The model needs to last approximately fifteen minutes (i.e. the length of the real-world game).

Obviously we can not construct an exhaustive set of requirements because we are unable to build up a complete picture of the system being modelled and future model experimentation is likely to lead to further requirements that were not apparent at this stage.

**Conclusion**

The LSD analysis carried out was very effective in highlighting the various requirements of the model and required little knowledge except that of the system being modelled. It is a testimony to the LSD analysis that no other requirement gathering techniques were required before the design and construction of the model could begin.
Chapter 5. Design and Implementation

Software design may be defined as, "the process of applying various techniques and principles for the purpose of defining a device a process or a system in sufficient detail to permit its physical realisation" (Pressman, 1997: p357). Implementation is the process of constructing the physical realisation of the design, usually through some computer-programming tool. The execution of these stages in model's life cycle are discussed in this chapter and their eventual result in the next.

We have seen in chapter 3 that the design and implementation stages of the empirical modelling life cycle are very much intertwined, for the convenience of the reader these stages are outlined again in figure 5.1.

![The design and implementation cycle](image)

**Figure 5.1 The design and implementation cycle**

**The Design and Implementation Cycle**

It is apparent that the design and implementation cycle is an iterative process. An initial design is implemented and experimentation is carried out to see if the model's behaviour correlates with that of the thing we are trying to model. Investigation takes place to assess which parts of the model need to be changed and the process begins again as the model is re-designed to accommodate these changes. This cycle goes on until the modeller feels that the model's behaviour is sufficiently close to that of the thing being modelled to stop. An example of the cycle in practice is the process undertaken to model a player's movement around the pitch. An initial design was made utilising a function using triangulation to generate a movement vector for the player. Given a pair of target co-ordinates the player would
generate a movement vector to move a certain distance towards the target and execute the vector until the target was reached. Upon experimentation however it was found that the player would shoot past the target. This problem was investigated and the solution found to be the introduction of the observable ‘player's distance from target’. This enabled the player to gauge how far he was away from the target and therefore stop in time to reach it.

Design
The design for the model was largely derived from the LSD specification. The agents were directly implemented along with their states, oracles, handles, derivates and protocols. Where in the final design these do not correspond with the specification it is due to experimental changes and design and implementation considerations forced upon the model through the use of the chosen design and implementation tools. The only design work not covered within the specification were the user interfaces for each agent which are discussed in chapter 7 and the functions used. Examples of these include a function to work out the path that a passed ball should take and a function to return a player's field of vision. It is apparent that these functions have not always been faithfully implemented mainly due to experimental changes and design and implementation considerations.

The only design tool utilised during the construction of the model was the model itself which was used for extensive design experimentation and investigation.

A multitude of techniques and practices were used during the design of the model and a significant effort was made to employ good software engineering practice throughout the model's whole life cycle. The techniques and practices employed include:

- The extensive use of constants.
- The use of a shared file where ever different agents required the same data (e.g. the player's constants were placed in a separate file to avoid potential inconsistencies that may occur if each player held their own copy).
- The use of a shared file where ever different agents required the same functions (e.g. the player's functions were placed in a separate file to avoid potential inconsistencies that may occur if each player held their own copy).
• The use of object orientated ideas for the creation of multiple instances of the same agent. For example there are four copies of the player agent and although each player requires their own agent file these are almost identical to the master (object) file and therefore a minimal amount of work is required to keep them consistent. The agents' files only differ in their identification number and the field of vision variable, which needs to be amended to ensure that the players can always see themselves.

• The use of files and functions to create logical program modules.

For a detailed insight in to the final design of the model please see chapter 6, 'Program Description'.

**Implementation**

Implementation is perhaps a more noteworthy stage within the empirical modelling life cycle than more conventional software life cycles because of the emphasis on experimentation and investigation. Often changes can be directly implemented without the need for the design stage although obviously it is important to judge when and where this is appropriate. As my familiarity with both the model and the tools being used increased this often became the case although I made sure that good software engineering practice was maintained throughout.

A number of tools were used for the implementation stage of the life cycle and these included version three of the abstract definitive machine (ADM3), the distributed version of 'EDEN' (Dtkeden) and the Donald and Scout notations. The ADM3 is a notation that allows the animation of LSD specifications. 'Entities' enable the implementation of agents, 'definitions' the implementation of states and derivates, and 'actions' the implementation of oracles, protocols and handles. It has been designed to provide a good abstract model for the proper use of EDEN and therefore through the translation from ADM code to EDEN provides, "a computational model that is based upon definitive representation of state" (Empirical Modelling, 1999). As such, "transitions are represented by parallel redefinition of variables in a definitive script" (Empirical Modelling, 1999), enabling concurrent actions to occur. The ADM3 translates its code into a Dtkeden definitive script. Dtkeden is a distributed implementation of the definitive notation EDEN and as such may be used to write definitive scripts using a syntax modelled on the programming language 'C'. The tool allows distributed modelling using multiple agents over a TCP/IP network and is based on a client/server configuration. Finally the Donald and Scout notations
are also used to construct definitive scripts but are concerned with the user's graphical display. The 'Donald' notation is used for two-dimensional line drawing and the 'Scout' notation for describing screen layout such as the position of windows. For more information on any of these tools please consult the empirical modelling group's web pages, the address of which can be found in the bibliography.

As with the design stage a number of techniques and practices were used during the implementation to ensure that good software engineering practice was employed. The techniques and practices employed include:

- The use of meaningful variable names.
- The extensive use of commenting.
- The use of spacing to ease code readability.

For a detailed insight into the final implementation of the model please see chapter 6, 'Program Description'.

**Experimentation**

Experimentation is a stage of the life cycle that is quite particular to Empirical Modelling and in some ways can be viewed as a very open ended style of testing. The importance of this stage was not as great as it may have been with other Empirical Modelling computer models because the model's behaviour was largely user controlled and because the model's domain was relatively well understood. It was therefore largely a case of making sure that a behaviour could be initiated rather than making sure that it would automatically take place. This was accomplished through a good deal of experimentation with the model itself using both predefined and spontaneous scenarios.

**Investigation**

The investigation stage is in some ways an extension of the experimentation stage and therefore in the same way that a limited amount of experimentation was carried out only a limited amount of investigation was completed. The investigation that was carried out involved the in-depth analysis of results of the experimentation, correlating these results with the model code and looking at ways of
changing the model so that experimental results more closely mirrored those expected. As such the model itself was again used to accomplish this.

Execution of the Design and Implementation Cycle
As we have seen the design and implementation cycle is very much an iterative process and so it was for the five-a-side model. The model was built up much in the same way that a game of five-a-side football might develop if a group of friends got together to play the game. Each step represents the completion of one design and implementation cycle:

1. The pitch to play the game of five-a-side football on was set up (i.e. the pitch entity was implemented thus initially creating a very simple non-distributed model).
2. A player was introduced and warmed up by practising running around the pitch (i.e. a very basic player entity able to move around the pitch was implemented).
3. The match ball was introduced (i.e. the ball entity was implemented).
4. The player practised losing and then regaining site of the match ball (i.e. the player's field of vision was implemented).
5. The player practised dribbling and shooting the ball (i.e. the player's ability to intercept, dribble and shoot the ball was implemented).
6. A spectator came to watch the player practise (i.e. a very basic spectator agent, able to view the proceedings was implemented therefore a distributed model with the appropriate user displays was set up with the spectator as the server and the player as a client).
7. A second player joined (i.e. a copy of the first player was implemented as an additional client).
8. The two players practised passing the ball between themselves (i.e. a player's ability to pass the ball to another player was implemented).
9. The two players practised tackling each other (i.e. a player's ability to tackle a player with the ball was implemented).
10. A goalkeeper joined in (i.e. a basic goalkeeper agent based on the player agent was implemented as an additional client).
11. The goalkeeper practised saving shots, carrying the ball and throwing the ball out to an outfield player (i.e. the goalkeeper's ability to intercept the ball, carry the ball while in possession and throw it to an outfield player were implemented).

12. Three other players, a goalkeeper and two outfield joined in to form teams (i.e. additional agents were introduced as clients and the team entities were implemented).

13. The players practised only passing to team-mates, only tackling opponents and only shooting at the opponent's goal (i.e. the player's ability to only pass to a team-mate, to only tackle an opponent, to only shoot at the opponent's goal and to only throw the ball to a team-mate were implemented).

14. A referee joined in (i.e. a basic referee agent based on the player agent was implemented as an additional client).

15. The referee made sure that he could hear the spectator shouting when a player was in one of the penalty areas (i.e. the spectator's ability to detect the fact that an outfield player has entered a penalty area and to relay this information to the referee was implemented).

16. The referee made sure that he could hear the spectator shouting when a player was not in their own half (i.e. the spectator's ability to detect when a player is not in their own area and to relay this information to the referee was implemented).

17. The referee made sure that he could hear the spectator shouting when a tackle had been made (i.e. the spectator's ability to detect a tackle and relay this information to the referee was implemented).

18. The referee made sure that he could give a goal when he heard the spectator shouting that the ball had crossed one of the goal lines (i.e. the spectator's ability to detect the ball crossing a goal line and to relay this information to the referee so that he can give a goal was implemented).

19. The referee made sure that he was aware of any illegal tackles that were made (i.e. the referee's ability to deem a tackle as illegal was implemented).

20. The referee made sure that he could start and stop the game (i.e. the referee's ability to initiate 'kick off', to run the match clock and to stop the match on time were implemented).

21. The referee made sure that he could give a penalty for a player entering their own penalty area and a free kick for a player entering their opponents' (i.e. the referee's ability to give a penalty or free kick for penalty encroachment was implemented).

22. The referee made sure that he could give a free kick for an illegal tackle (i.e. the referee's ability to give a free kick for a tackle deemed to be illegal was implemented).
23. The players made sure that they could take a free kick or penalty for their team (i.e. a player's ability to take a free kick or penalty when it has been awarded to their team was implemented).

(Obviously if any of these stages had repercussions for agents other than those mentioned such as the spectator's need to know when a pass has been attempted then these were also implemented within that stage.)

Conclusion

Although the design and implementation cycle proved very effective in the construction of the model the most notable conclusion that can be drawn from this stage is that the LSD analysis and subsequent animation using the ADM3 performed about all expectations. Comparatively little work was required to put the observations contained within the LSD specification into practice and design a model that performed as expected because the two mirrored each other so effectively.

As mentioned the design and implementation cycle was very effective and meant that the model could be constructed gradually. This was important because it meant that I could be sure that the model would continue to behave as expected. It mean that I could explore and experiment before adding new features or changing existing ones because the model behaved unexpectedly, mainly due to inexperience in using the tools.

Perhaps the final conclusion that can be drawn from this stage is the need for further development of the Empirical Modelling tools available. It is evident that these are research tools because as they currently stand they are disjointed, often unreliable, frequently undocumented, sometimes unfinished and not very user friendly. Of all of these faults the lack of good documentation was found to be the most serious because at the moment most documentation available is in the form of technical manuals which often do not even cover all of the features. There is therefore a definite need for both detailed user guides and complete technical manual for all of the Empirical Modelling tools available.
Chapter 6. Program Description

In this chapter the model's final design is described in detail, you may therefore find it useful to relate this information to the corresponding computer code in the appendix.

I will firstly give a brief overview of the model's set-up and then in considerable detail look at each of the agents involved in the model. These include the following inactive agents:

- The pitch
- The ball
- The match
- The teams

And the following active agents:

- The spectator
- The referee
- The outfield players
- The goalkeepers

The Set-up of the Model

As we have already seen the final model is distributed and runs on eight networked workstations using a client/server set-up. This set-up is shown in figure 6.1.
Figure 6.1. The model's set-up
Inactive Agents

The Pitch
As we have seen the pitch was the first agent to be implemented and its dimensions are laid out in file pitch.e. The pitch's two halves are labelled A and B respectively and the distances outlined in figure 6.2 are recorded and used to work out all of the remaining dimensions such as the position of the centre line. This means that the pitch is very flexible because to change the pitch dimensions only these values need to be changed.

![Figure 6.2. The pitch measurements required](image)

The pitch is displayed through the pitch.d file which obtains the required display variables such as the size that the centre circle should appear from file pitch_constants.e. A function is also required by the pitch display and this is obtained from file pitch_functions.e. The function uses trigonometry to return the end co-ordinates for an agent's direction of sight line when given the desired angle of the line, the distance the line is required to be and the line's starting co-ordinates. This enables a line to be drawn when the agent supplies their current position and angle of sight. The pitch dimensions, the circles and labels representing the agents, the line of sight lines and the match ball are
then drawn on the screen using 'Donald'. Finally 'Scout' is used to display the pitch in a window using file \texttt{pitch\_win.e} or \texttt{spec\_win.e}. File \texttt{spec\_win.e} is used if the spectator is displaying the pitch because the window needs to be made insensitive to the user's mouse clicks.

**The Ball**

The ball agent is defined in the file \texttt{ball.e}. It outlines the ball's initial position as being on the centre spot. The variable \texttt{ball\_owner} is used to specify which player is currently in possession of the ball via their id number, a zero means that no one is currently in possession of the ball and this is its initial status.

**The Match**

The match agent is defined in the file \texttt{match.e} which contains details such as the length of the match, the current status of the match (e.g. penalty) and if the match is currently in progress. The match agent also holds details of the current state of play, that is if team one is winning, if team two is winning or if the match is currently a draw. These details are used by the agents to display the match result through the \texttt{display\_result} function. This is carried out after the referee has called time.

**The Teams**

The details of the agents for team one and for team two are held in the files \texttt{team1.e} and \texttt{team2.e} respectively. These files hold a number of important details such as the team's name and jersey colours, whether the team currently has a free kick or penalty and a list of the players in the team. This is important because we need to be able to find out if a given player is in a given team. By listing the players in a team the model is made much more flexible because we only need to change the id of the players in a team to change the combination rather than separately changing each player's individual allegiance.
Active Agents

The Spectator

The structure for the spectator agent is shown in figure 6.3.

Figure 6.3. The structure for the spectator agent
As we have seen the spectator agent serves two distinct roles within the model. Firstly it allows a match to be viewed in its entirety and secondly it serves as the server for the distributed model. Unlike the other active agents the spectator was not implemented using the ADM3 machine because it was felt that in its role as the model's server it was best to attempt to keep the processing overheads as low as possible to allow the server to operate at its maximum. This is because although the ADM3 was very effective in animating the LSD specification it did create definitive scripts with very high processing overheads so instead a definitive script for Dtkeden was implemented directly.

In its role as the server the spectator agent must make sure that each client is kept up to date with any changes in the client's version of the model's state, this is largely handled by the LSD code. For each agent the agent's oracles and handles are listed. If the server finds that a model state variable (observable) is changed then each agent with that variable in their list of oracles must be informed of this change. Model state variables (observables) may be changed either directly by the server or by an agent (client) who has the variable listed as a handle. It is by this method that agents are kept up to date with the model's current state so that for example if the referee awards a penalty or free kick the referee agent will inform the spectator of this and the spectator will relay that information to all of the agents required to know. Clients must therefore explicitly inform the server of any relevant (i.e. non localised) changes in the model's state. Agents are also kept up to date through the server servicing their requests for the current state of a model variable. We find therefore that although agents may store their own local version of the model's current state the true current model state is always stored by the server and it is for this reason that the spectator has been chosen because ideally the spectator will always see what is really going on.

A secondary role for the spectator as server is one of manipulating the ball agent. If for example a player attempts to pass the ball they will send a message to the server requesting it to move the ball accordingly and the server will implement this. There were a number of reasons why the server was chosen to implement the ball agent. Firstly although at first hand it may appear that the referee owns the ball this would obviously make a game of five-a-side football without a referee impossible and this is clearly not the case. It is apparent that although a ball is vital for a game of five-a-side football no active agent actually owns it so it would therefore seem sensible to use the server to implement the ball to allow games to go ahead with any combination of agents. Secondly the ball is the primary trigger agent within the game therefore it is important to use the server to implement it because in doing so we are making sure that the most up to date ball states are always relayed to the other agents. Finally the
The server was used because it has much lower processing overheads than the other agents due to the fact that the ADM3 machine was not used in its construction. This is important because being the main trigger agent the ball's states need to be updated as soon as possible and therefore we do not want any significant delay in doing this because of high processing overheads.

The ball is manipulated either directly through the agent changing the ball's position via the spectator or through the spectator's function, move\textunderscore ball. Given the power applied to the ball and the coordinates to move the ball towards this function will create an initial movement vector for the ball to move towards its target. This vector will be executed and then reduced in accordance with the pitch's level of friction until the ball can be stopped because it is travelling so slowly. The ball hitting any of the pitch walls is detected and the vector changed accordingly and if the ball is found to hit the top or bottom wall between the goal posts then the fact that the ball has crossed a goal line is registered. At the end of each movement cycle each agent is explicitly informed of the ball's new coordinates. This is done because it was found that the server had to be forced to relay the new values as it was too preoccupied with its own local processing to do this automatically and as a result agents appeared to be relayed conflicting values.

Even as a spectator of the match the spectator agent has a couple of roles to play. Firstly it must allow the user to observe all that is going on and secondly it must inform the referee of important events such as a player entering a penalty area, events that he might otherwise not be aware of.

To allow the user to observe all that is going on a pitch display is used along with user prompts and a match commentary. The pitch display plots each agent on the pitch using their coordinates and details such as the print on the back of their shirt, obtained from file names\textunderscore nums and their current angle of sight. The 'field of vision' system as used by the other active agents is utilised except that the items are set as always being visible (for details of the 'field of vision' system please see the design description of the user controlled agents). The spectator's pitch display and various user prompts are derived from the current state of the model and implemented through the spectator.d file. For details of the spectator's user display please see the next chapter. The spectator's match commentary deals with the last twenty or so significant events that the spectator is aware have occurred in the match. When an agent carries out an event that the spectator should be concerned with such as attempting to tackle an
opponent that agent sends the spectator a piece of commentary to add to the list. The commentary is stored in a queue data structure using a list of strings. This is used to maintain chronological order and as such when a new piece of commentary is added the piece at the head of the queue is removed and the queue shuffles along to accommodate the new addition.

The spectator informs the referee of a number of important events that have been observed. Firstly as we have seen the referee is informed of when the ball has crossed one of the goal lines. Secondly the referee is informed of when a player is in one of the penalty areas. Using the function `inside` the spectator can detect when an outfield player has entered one of the areas and relays this information along with the player's id to the referee so that the referee knows whether to give a penalty or free kick. A value of zero means that there is no outfield player currently in the penalty area. The final piece of information that the spectator relays to the referee is when a player is out of their own half so that the kick off can be delayed until every player is in their own half. It is slightly trickier for the spectator to generate this information because he needs to know which team the player plays for and therefore which half the player should start in. This information is found out using the `is_in_team` function.

**User Controlled Agents**

Because all of the user controlled agents are based on the same design it is best to initially look at their design similarities before looking at their differences.

The agents are all very similar in their design because the ADM3 machine has been used to implement them from the LSD specification therefore in order to understand these designs it is recommended that the reader is familiar with the way the ADM3 animates an LSD specification.

**An ADM3 Influenced Design**

We have already seen that states and derivates may be implemented as they would be in EDEN through the ADM3's definitions and the handles. Similarly oracles and protocols may be implemented through the ADM3's actions. The agent's actions are a set of guard statements that when found to be true cause a set of corresponding actions to be executed. For example the following action will cause the referee to increment his match watch counter when the match is in progress and in open play (i.e. no team is currently taking a free kick or penalty):
match_in_progress && (match_status == "open")
    -> ref_watch_counter = ref_watch_counter + 1,

For each execution cycle the agent will test each action's guard statement and carry out those actions found to be true. These actions can be executed concurrently.

Design Similarities
Each agent is based on the same basic design therefore they have a number of design features in common. These features include:

- An agent's field of vision (pitch display).
- An agent's angle of sight.
- An agent's movement.
- An agent's commentary.
- An agent's initial awareness of the positions of other agents.
- An agent's user display.
- Displaying the match result.

An agent's field of vision (pitch display)
An agent's field of vision is displayed in figure 6.4.
An agent's field of vision is implemented by means of a list of Boolean values which represent whether a player can see an agent on their pitch display or not (the player's own Boolean value is always set as true so that a player can always see their own position). The list is in the following order:

1-6. Players 1 to 6.
7. The referee.
8. The match ball.

Obviously as agents are added to the model each agent's list will need to be changed. The function `can_see` is used to decide if an agent is visible or not by using the angle from the observing agent to the agent being tested. If this angle lies within the range of $90^\circ$ either way of the observing agent's angle of sight then it is deemed that the agent being tested can be seen and is displayed on the agent's pitch display along with the print on the back of their shirt. $90^\circ$ either way was chosen because it was felt that this reflected a player's field of vision in a real game. The `field_vision.e` file is used to set those agents not visible as having null co-ordinates, this prevents them from being displayed on the agent's pitch display. The overall method is somewhat different from the LSD specification because
theoretically an agent should not be able to test to see if they can observe another agent because they obviously have no idea of where the agent is. Obviously a compromise has to found and this method can at least hide an agent from a user if not the underlying model.

*An agent's angle of sight*
An agent's direction of sight is stored as an angle and as such when a user attempts to look at a spot on the pitch by right clicking on it trigonometry is used to calculate the new angle required and the agent's direction of sight value is changed accordingly.

*An agent's movement*
An agent's movement is implemented using vectors, much in the same way that it is for the ball. When a user requests to move to a spot on the pitch by clicking on it with the left mouse button the agent's target co-ordinates are changed accordingly and a movement vector is generated using these co-ordinates and the distance the agent can move in one execution cycle. This distance is derived from the agent's speed attribute. On subsequent execution cycles the agent sees that they are not on their target co-ordinates and therefore uses the movement vector to move towards it until the target is reached. The process can be seen in the flow chart shown in figure 6.5.
With the exception of the goalkeeper agents where it is preferable to keep an eye on the game an agent will always look at their target before moving towards it.

An agent's commentary

An agent's commentary is implemented in the same way as it is for the spectator except that it is the agent who generates their own commentary. For example if the player attempts a shot commentary to record this action is added to the agent's list of ten or so recent events. It is not just the agent's recent actions that are recorded as some such as the event of 'kick off' taking place are added because the agent has detected the occurrence of this event. Only significant events that the agent has perceived are added such as the start of the match, a goal being given and the final whistle being blow. Events such as a team mate attempting a tackle are not deemed important enough to be added.
The positions of other agents

Clients must be activated (i.e. logged on to the server) one at a time therefore if an agent is activated before another then they must initialise their values for that agent. Conversely if an agent is initiated after another then they must explicitly ask the server for the current position of that agent. For this reason when the agent is activated they must inform the server of their initial state so that those other agents required to know can be informed. For the outfield players and goalkeepers the states of other agents are initialised and obtained through the get_pos.e file and they obtain initial data from the server through the player_request.e file.

An agent's user display

Each agent implements their own pitch display and set of user prompts and displays. These are discussed in the next chapter and implemented through each agent's respective display file (i.e. ref.d, player.d or keeper.d).

Displaying the match result

Agent's will detect when the referee has called time and proceed to display the match result. The outfield players and goalkeepers accomplish this through the display_result function and the referee through the distribute_result function. An additional requirement of the referee is that he must also inform the spectator of the match result by adding the appropriate commentary to the spectator's list.

Additional Design Requirements

As we have seen the user controlled agents have a lot of design similarities however as their required actions are very different there are also many additional design requirements. Each agent's design will be examined in the context of the additional actions that they are required to undertake.
The structure for the referee agent is shown in figure 6.6.
The referee has the following additional action requirements:

- Initiate 'kick off'.
- Call an end to the match.
- Give a goal.
- Give a free kick for an illegal tackle.
- Give a penalty for encroachment.
- Give a free kick for encroachment.

It should be apparent that the design of the referee agent affords a very high level of flexibility. For example if the respective team sheets change the referee will not require any manual adjustments to accommodate this as all of the referee's relevant adjustments will be made automatically.

*Initiate 'kick off'*

The referee can only initiate 'kick off' if there are no players in either of the areas and each player is in their own half. As previously mentioned this information is obtained from the spectator. Additionally the referee is also aware of when all of the players have been activated (i.e. logged on to the server) as obviously we need to wait for all of the players before starting the match.

*Call an end to the match*

The referee has a watch counter that is used to decide when to call an end to the match. The counter is activated upon 'kick off' and is incremented on each repetition of the referee's execution cycle as long as the match is in open play (i.e. neither team has a free kick or penalty). The watch counter is also used to increment the number of minutes played using the remainder function. The number of execution cycles for each minute was decided by taking the average number for a referee agent being executed on an Sparc Ultra 5 and on an Sparc Ultra 10 workstation.

*Give a goal*

As we have seen the referee will give a goal if the spectator informs him that the ball has crossed a goal line and if the ball is in the referee's field of vision.
**Give a free kick for an illegal tackle**

The referee is informed of new tackles via the spectator because whereas the spectator can see all tackles take place the referee can not. When a player attempts a tackle they inform the spectator of this giving the following list of details:

1. The id of the tackler.
2. The id of the player being tackled.
3. The ferocity of the tackle (i.e. the tackler's level of aggression).
4. The pair of co-ordinates where the tackle took place.

This is relayed to the referee who must check to see if the tackle is within his field of vision and if so will assess the tackle to see if it was illegal or not. Tackles are assessed using the tackler's level of aggression and the referee's leniency. Figure 6.6 shows the percentage of tackles that a referee will give as a foul, the percentages are based upon the tackler's level of aggression. Figure 6.7 shows the percentage of fouls that a referee will ignore, the percentage is based on their level of leniency. A random number is used to ensure that the percentages are closely adhered to.

<table>
<thead>
<tr>
<th>Player's aggression</th>
<th>Percentage of tackles deemed to be fouls</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>25%</td>
</tr>
<tr>
<td>2</td>
<td>20%</td>
</tr>
<tr>
<td>3</td>
<td>15%</td>
</tr>
<tr>
<td>4</td>
<td>10%</td>
</tr>
<tr>
<td>5</td>
<td>5%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Referee's leniency</th>
<th>Percentage of fouls not given</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5%</td>
</tr>
<tr>
<td>2</td>
<td>10%</td>
</tr>
<tr>
<td>3</td>
<td>15%</td>
</tr>
<tr>
<td>4</td>
<td>20%</td>
</tr>
<tr>
<td>5</td>
<td>25%</td>
</tr>
</tbody>
</table>

_Figure 6.6. The percentage of fouls_  _Figure 6.7. The percentage of fouls ignored_

**Give a penalty or free kick for encroachment**

The referee is informed by the spectator of when an outfield player enters a penalty area. Each penalty area has a 'player in area' state which correspond to the id of the player in the area, a value of zero means that there is currently no player in the penalty area. By knowing the id of the player the referee can use the function `is_in_team` to find out which team they play for and thus decide if a penalty should be given because the player has entered their own area or a free kick because the player has
entered their opponents'. Obviously the referee can not give a free kick or penalty if the player in the area is not in their field of vision.
Outfield Player

The structure for a player agent is shown in figure 6.8.
An outfield player has the following additional action requirements:

- Intercept the ball.
- Dribble the ball.
- Attempt to pass the ball.
- Attempt to shoot the ball.
- Attempt to tackle a player with the ball.
- Take a free kick awarded to the player's team.
- Take a penalty awarded to the player's team.

It should be apparent that the design of the player agent affords a very high level of flexibility. The players are distinguishable through their unique identification number and a player's attributes such as their name, position and the team they play for can all be changed without having to change any of the other model code. For example if a player decides to play for the opposition the only manual adjust of the model required is for the team lists to be changed accordingly. The player's states that are dependant on the team they play for such as the one which specifies which goal the player should be shooting at are all automatically adjusted.

*Intercept the ball*

As specified in the requirements a player will automatically intercept the ball when it is near enough to do so, is not currently in possession and is in the player's current field of vision. How close the ball needs to be for the player to intercept it is defined in file `player_constants.e`.

*Dribble the ball*

If a player carries out any movement while in possession of the ball then the ball is also moved using the player's position and movement vector. This makes the ball appear slightly ahead of the player to give the impression that the ball is being dribbled.

*Attempt to pass the ball*

As specified in the user interface requirements a player attempts to pass by middle clicking near a team-mate while in possession of the ball. The function `click_near_team_mate` is used to see if
the click was near a team-mate. This function runs through the other players in the player's team's list (excluding the goalkeeper) and sees if their current position is close to the co-ordinates of the mouse click. The proximity required to be defined as 'close' is defined in file `player_constants.e`. The first team-mate found close to the mouse click is passed towards by requesting the server to execute the `move_ball` function. The power applied is derived from a minimum passing power and the player's level of passing skill, obviously the higher the skill the faster the pass.

**Attempt to shoot the ball**

As specified in the user interface requirements a player attempts to shoot by middle clicking in their opponent's penalty area while in possession of the ball. Each player is therefore aware of the dimensions of their opponents' penalty area and can use the `inside` function to see if a middle mouse click was registered inside it. The server is again requested to execute the `move_ball` function to carry out the shot. The power applied is derived from a minimum shooting power and the player's level of shooting skill, obviously the higher the skill the faster the shot.

**Attempt to tackle a player with the ball**

As specified in the user interface requirements a player attempts to tackle an opponent with the ball by middle clicking near them when close enough to carry out the challenge. The `click_near_team_mate` function is therefore used again but for testing if the player with the ball is a team-mate if the player were to play for the opposition. The proximity required to be defined as 'close' is defined in file `player_constants.e`. Figure 6.9 shows the percentage of tackles won depending on the player's level of tackling skill. A random number is used to make sure that the percentages are closely adhered to.

<table>
<thead>
<tr>
<th>Player's tackling skill</th>
<th>Percentage of tackles won</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>50%</td>
</tr>
<tr>
<td>2</td>
<td>55%</td>
</tr>
<tr>
<td>3</td>
<td>60%</td>
</tr>
<tr>
<td>4</td>
<td>65%</td>
</tr>
<tr>
<td>5</td>
<td>70%</td>
</tr>
</tbody>
</table>

*Figure 6.9. The percentage of tackles won*
To simplify matters and because it is very hard to quantify, a player's level of dribbling skill was not taken into consideration. If a player has been tackled or a tackle has deemed to be illegal then a player will acknowledge this.

*Take a free kick awarded to the player's team*

A player will automatically acknowledge when they have conceded a free kick for either an illegal tackle or for entering their opponent's penalty area. Penalty encroachment will give possession to the opposition's goalkeeper but an illegal tackle will allow a player from the opposing team to request to take the free kick. Players can detect when a free kick has been given to their team and whether the player can take it or not is determined by the player's distance from the ball (i.e. if they can intercept it) and if no other player has previously requested to take it. This is reflected by the colour of the player's 'freekick' button which is green if they can request to take the free kick but red if they can not. To simplify matters a requirement was not made for the opposing players to be a set distance away from the free kick as is usually the case. Once a player has signified that they will take the free kick the player's usual actions are restored with the exception of intercepting the ball (i.e. they can pass or shoot). The player signifies that open play has resumed after taking the free kick.

*Take a penalty awarded to the player's team*

A player will automatically acknowledge when they have conceded a penalty for entering their own penalty area and place the ball on the penalty spot. Players can detect when a penalty has been given to their team and can then request to take it. Whether the player can take the penalty or not is determined by the player's distance from the ball (i.e. if they can intercept it) and if no other player has previously requested to take it. This is reflected by the colour of the player's 'penalty' button which is green if they can request to take the penalty but red if they can not. Once a player has signified that they will take the free kick the 'shoot' action is used to take it (the goalkeeper is free to move while the penalty is being taken). The player signifies that open play has resumed after taking the penalty.
Goalkeeper

The structure for a goalkeeper agent is shown in figure 6.10.

Figure 6.10. The structure for a goalkeeper agent
A goalkeeper has the following additional action requirements:

- Intercept the ball.
- Carry the ball.
- Attempt to throw the ball.

It should be apparent that like the player agent the design of the goalkeeper agent affords a very high level of flexibility. Like the players the goalkeepers are distinguishable through their unique identification number and a goalkeeper's attributes such as their name, position and the team they play for can all be changed without having to change any of the other model code.

*Intercept the ball*

The goalkeeper will automatically intercept the ball when it is close enough to do so, this proximity is defined in file `keeper_constants.e`. The goalkeeper's level of saving skill is not taken in to consideration when determining the level of success because there was not sufficient time to implement such a feature.

*Carry the ball*

If a goalkeeper carries out any movement while in possession of the ball then the ball is also moved accordingly using the same co-ordinates as the goalkeeper.

*Attempt to throw the ball*

This action mirrors an outfield player's passing action except that a minimum throwing skill, defined in file `keeper_constants.e` and the goalkeeper's own throwing skill is used to determine the power applied.
Chapter 7. User Interface

The user interface is of great importance to any computer model, "(human-computer interface design) is now recognised as a vital component of successful computer applications" (Preece, 1994: p'v'). This importance is made all the more great when there is a high level of user interaction, as is the case with this model. The design of the user interface was therefore taken very seriously and substantial time was allocated for it to be undertaken and completed.

Star Life Cycle

A human-computer interaction design approach was utilised in the form of the 'star life cycle' (Preece, 1994). The star model representing the star life cycle is shown in figure 7.1.

*Figure 7.1 The star model (Preece, 1994)*

There are a number of reasons as to why the star life cycle was chosen for the user interface design. Firstly it is geared towards developing systems with high levels of user interaction, "The star life cycle is primarily orientated to the particular demands of developing interactive systems that will be usable by people" (Preece, 1994: p381). Secondly it is evident that this is an approach very much centred on evaluation. This is in tune with the Empirical Modelling approach that also stresses the importance of evaluation and therefore the two appear to compliment each other very well. Furthermore the star life cycle also, "stresses rapid prototyping and an incremental development of the final product" (Preece
1994: p380), again very much in tune with the Empirical Modelling approach. Finally the star life cycle is very flexible because it allows the development to begin at any stage (indicated by the entry arrows) and to be followed by any stage (indicated by the double-headed arrows). This is very important because Empirical Modelling requires a very high level of flexibility in the interface design. We may not be able to completely assess the model's requirements or design until we have carried out some experimentation and therefore may need to amend these stages at a later date. Obviously this will often have implications for the user interface design so therefore we need an approach that affords us this level of flexibility.

I will now take a look at how each stage of the life cycle was carried before undertaking a brief evaluation of the final user interface constructed. Although there is no formal stage order within the star life cycle for ease of reading I will lay the stages out in a conventional order.

Requirements Specification
The requirements specification has been split up in to three different specifications. The functional requirements, the data requirements and the usability requirements. These requirements were initially set out and then subsequently amended if evaluation of the user interface proved that this was necessary.

Functional Requirements
The functional requirements specify what the system must do (Preece, 1994). They were derived from the LSD analysis which laid out both the observables that needed to be presented to the user and the actions that a user is required to be able to undertake for a given agent. For example the LSD analysis insists that a player be able to shoot the ball at an opponents' goal therefore a requirement of the user interface is that there exists a method to allow the user to initiate this action. Similarly a player is required to know their current position on the pitch so this information will need to be presented to the user in some way. No formal documentation was completed for the functional requirements because they are all contained within the LSD analysis.
**Data Requirements**

The data requirements specify the structure of the system and the data that must be available for processing to be successful (Preece, 1994). Again the data requirements were derived from the LSD analysis. Although the analysis does not explicitly define data types and their meaning it does largely define them implicitly. For example it is clear that a player's name should be displayed as a string of characters, that a team's score should be displayed as an integer number and that the match clock value should also be displayed as an integer number (after all if a football player asks the referee how many minutes have been played in a real game of five-a-side football he is unlikely to reply, "five minutes and twenty seven seconds"). Where there was ambiguity I used my knowledge of the domain to decide upon a suitable data type. For example it is unclear as to how player skill levels should be represented but I used integer numbers because this is the method used by many football computer games such as 'Championship Manager 2' and 'Premier Manager 2'. Similarly player positions on the pitch are represented graphically from an over view in the same way that they are in 'Sensible World of Soccer '96/97'. Again I felt it unnecessary to complete any formal documentation for the data requirements because they are largely contained within the LSD analysis.

**Usability Requirements**

Usability requirements specify the acceptable level of user performance and satisfaction with the system (Preece, 1994). Being a primarily interactive system the usability requirements for the model were very high. These requirements can be set out in terms of the model's requirements for learnability, throughput, flexibility and attitude. The learnability is concerned with how easy it is to learn how to use the system. In this case an inexperienced user must be able to very quickly learn how to operate the model with very little prior training. The throughput is concerned with how easy the model is to use and the speed of operation for experienced users. Five-a-side football is a game played in real time therefore a model of five-a-side football must strive to operate in real-time. For this reason experienced users of the model must be able to easily carry out tasks with considerable speed. Flexibility is concerned with the extent to which the system can accommodate changes to the tasks and environments. Significant flexibility will be required within the user interface because both the tasks and the environment are likely to change during the development of the model. Finally attitude is concerned with the positive attitude instilled in users by the system. A user interface with high attitude will be required for the model to keep the user engaged and to hold their attention for the full duration.
of the match. The author felt it unnecessary to complete any formal documentation for the usability requirements because these requirements were informally laid out in the author's head prior to the design of the user interface. If a group had designed the user interface then formal documentation would have been completed.

Task Analysis

Task analysis is concerned with the learning and knowledge that users utilise and steps that they go through in achieving a specific goal (Preece, 1994). Formal task analysis was not carried out for three reasons. Firstly because of time constraints. Secondly because the author felt that he had suitable experience and knowledge of user interfaces for it to not be essential and finally because this is a project primarily concerned with the investigation of Empirical Modelling and not the construction of user interfaces. A formal task analysis may prove useful in the future if the user interface were to be significantly re-designed.

Conceptual and Formal Design

The conceptual design is concerned with questions of what the system should do (Preece, 1994). The conceptual design was drawn from the requirements specification and took the form of listing the actions and observables required for each agent. The initial conceptual design can be found in the appendix. It differs somewhat from the final design because this initial design was partially prototyped, implemented and then evaluated which led to a few changes.

The formal design is concerned with questions of how things set out in the conceptual design can be achieved (Preece, 1994). The author therefore used HCI (human-computer interaction) design guidelines, derived from a previous academic course on HCI and his own experience of developing user interfaces to sketch the initial formal design for each agent. Scenarios were used to help envision the design problems. These sketches have been included in the appendix. Again they differ somewhat from the final design because firstly the layout was largely formulated during the prototype state and secondly because subsequent evaluation of the implemented design led to several design changes.
Prototyping
Conventional prototyping was not used for the user interface because a version of the model with limited functionality was not constructed. Instead Microsoft Visual Basic 5.0, a 'drag and drop' visual-programming environment was used to create mock screen layouts. This allowed design considerations such as the screen layout and use of colour to be investigated and improved upon with considerable speed and ease. Microsoft Visual Basic 5.0 was used because it allows screen designs to be constructed and changed in a very short period of time. The prototyping process used is shown in figure 7.2.

![Prototype Screen Layout](image)

**Figure 7.2 The prototyping process**

The resulting screen layouts are included in the appendix and formed the bases for the implemented agent screen designs.

Implementation
The interfaces were implemented using the Donald and Scout notations. Donald is a definitive notation for two dimensional drawing and was used to draw among other things the five-a-side pitch, the circles representing the players and the circle representing the football. Scout is a definitive notation for
describing screen layouts and as such was used to define the screens layouts for each interface. The forced use of these notations had a few important consequences, namely:

- Buttons could not be directly implemented, instead a window had to be used. This meant that a conventional command button's affordances and feedback could not be utilised.
- Complex graphics proved difficult to implement.
- The implementation proved time consuming because layouts had to be specified numerically (i.e. using co-ordinates).

**Evaluation**

Evaluation was perhaps the most important stage of the life cycle because it proved the real acid test for each of the agent user interfaces. Scenarios were used to test each interface against the requirements set and after each session a list of problems and areas where improvements were required were drawn up. Design solutions to these problems were generated and these were implemented directly within the interface (the prototyping stage eliminated the vast majority of screen layout problems so little further consideration was given to that stage). Examples of design problems and solutions found during the evaluation stage include:

- It was found that allowing goalkeepers to move to any area of the pitch caused problems because users would invariably click outside the area during experimentation with the model. This resulted in a penalty being given when the referee saw the goalkeeper leave his own area. The solution therefore was to only allow goalkeepers to move inside their own area although this did mean sacrificing realism for usability.

- A tackle can only be executed if a player is near enough to an opponent with the ball to do so. It was found helpful therefore to have a system of indicating when a tackle was possible. This was accomplished by introducing a box that turns green to indicate that a tackle is possible but is otherwise red.

- It was realised that the spectator and the referee user screens did not display the two team names therefore a couple of boxes were introduced displaying team names and in the team colours.
The process of evaluation and redesign was continued until the author believed that a satisfactory interface had been constructed. Unfortunately due to time constraints the author was unable to obtain the beneficial input from a novice user during this stage.

Evaluation of the Final User Interface
For the evaluation of the final user interface I will firstly look at some of the HCI ideas that have been utilised in its design. I will then evaluate the user interface with respect to the requirements initially set out and look at possible ways of developing the interface in the future.

Many HCI theories and ideas have been utilised in the design of the user interface. For an explanation of these please see Preece (1994) or any other good book on HCI. The ideas laid out below have been referenced to the user interface for a player in figure 7.3 so that the reader can observe their application in practice:

1. **Affordances** e.g. The button used to signify that the player will take the free kick.
2. **Visibility** e.g. The user will visibly see that clicking on a point on the pitch will cause the player to move to that point. The model's current operation is always made very visible to the user.
3. **Colour Coding** e.g. The use of green to signify when the player is in possession of the ball and red when he is not.
4. **Segmentation** e.g. The windows for the player's attributes are grouped and are all the same colour to emphasise the segmentation.
5. **Memory Model** e.g. The match score is displayed so that the user does not have to remember what it is.
6. **Guiding Attention** e.g. The 'Player Can Tackle' window will turn from green to red to draw the user's attention to the fact that the player can now tackle an opponent with the ball.
7. **Knowledge in the Head** e.g. The instructions to operate the model are displayed so that the user does not have to store all of this information in their head (although an expert user probably will).
8. **Mental Model** e.g. The user can refer to their functional mental model of the game of five-a-side football when operating the model.
9. **Metaphors** e.g. The user can use their previous knowledge of computer applications, computer games or indeed the game of five-a-side football as a suitable metaphor for the interface.

10. **Learning** e.g. The user can use their previous knowledge of computer applications (e.g. buttons) and games as an analogy for the interface. Learning by experimentation has also been taken in to consideration by making the interface as simple, visible and robust as possible.

11. **Constraints** e.g. The penalty button will only turn green (i.e. be active) when the player is eligible and able to take the penalty.

12. **Direct Manipulation** e.g. When the user requests to move the player there is the minimum of delay before the player will move.

13. **Mapping** e.g. When a user clicks on a point on the pitch to look at that point there is a direct mapping between clicking on the point and looking at the point.

14. **User Support** e.g. An on-line user guide has been developed to aid with more advanced user support.

15. **Consistency** e.g. The exclusive use of the mouse as the only means of user interaction.

16. **Patterns** e.g. When a player attempts a tackle and is successful the 'Player Can Tackle' window will change from green to red, thus changing from a green, red pattern to a red, red pattern.

17. **Feedback** e.g. When a player attempts to undertake an action the window displaying the player's current action will change to provide the user with vital feedback.
**Application of HCI ideas for the player user interface**

**HCI Ideas**

1. Affordances
2. Visibility
3. Colour Coding
4. Segmentation
5. Memory Model
6. Guiding Attention
7. Knowledge in the Head
8. Mental Model
9. Metaphors
10. Learning

*Figure 7.3 The user display screen for an outfield player*
11. Constraints
12. Direct Manipulation
13. Mapping
14. User Support
15. Consistency
16. Patterns
18. Feedback

As the user interface stands it fulfils both the functional and data requirements by both displaying all of the relevant information and by enabling the user to carry out all of the required actions. The real question therefore is whether the interface fulfils the usability requirements? Firstly the interface appears to be relatively easy to learn. On-screen Instructions aid inexperienced users and the interface itself is relatively simple, is very visible and has good mappings. Of course it would require substantial evaluation by novice users before such a claim could be substantiated. This could be carried out through the use of observation, interviews and questionnaires and these actions are recommended to be undertaken in the future. Secondly the throughput of the interface appears to be adequate for the expert user. Design features such as the exclusive use of one input device (i.e. the mouse) and the fact that interaction is concentrated within the five-a-side pitch area allow the expert user to swiftly carry out tasks and accomplish goals. Again further evaluation might prove beneficial and this could be accomplished through benchmark and performance testing of the interface. The interface appears be flexible enough to meet the requirements because it has been able to accommodate numerous design changes as well as facilitating the construction of multiple interfaces. Finally from the limited exposure to multiple users that the interface has received it is apparent that it consistently instils a positive attitude within that user. This is the case because firstly the system is very visible, therefore the user is always aware of the model's current operation and secondly because the interface borrows many ideas from HCI research and so has quite a 'natural' feel to it. Further evaluation would however be recommended to substantiate this claim and this could take the form of user observation, interviews and questionnaires.

Conclusion
The user interface appears to serve its purpose very well. The star life cycle has done a very good job of complimenting the Empirical Modelling approach and has been very successful in developing an
interface that fulfils all of its requirements. Further evaluation is however recommended because the interface could undoubtedly be improved upon.
Chapter 8. Testing

Testing is becoming an increasingly important stage in any piece of software's development life cycle, "Software testing is a critical element of software quality assurance" (Pressman, 1997: p464). Thoroughly testing the model therefore was taken very seriously and in addition to the continual testing carried out a whole week was given away to testing the final model.

Testing Strategy
The software testing strategy was concerned with both verification and validation of the model. Verification can be seen as making sure that the product is being built correctly and validation as making sure that the correct product is being built (Pressman, 1997). The testing strategy used can be seen in figure 8.1

![Figure 8.1 The software testing strategy (Pressman, 1997)]

**Figure 8.1 The software testing strategy (Pressman, 1997)**

Unit Test
Unit testing is concerned with making sure that the individual software units such as functions and procedures function correctly (Pressman, 1997). White-box testing, a method that uses the control structure of the unit of code to derive test cases (Pressman, 1997) was used to test this and was carried out in unison with the coding. For example the function `dir_line_points` within the file `pitch_functions.e` has eight different control paths so test data was created to make sure that all eight of these control paths functioned correctly. Each procedure and function was tested in this way.
Integration Test
Integration testing is concerned with making sure that the software functions correctly after the units have been integrated (Pressman, 1997). In this case incremental integration (Pressman, 1997) was used as the model was constructed and tested in small segments and black-box testing, a method that tests the functional requirements of a program (Pressman, 1997) was used to make sure that the model operated correctly after each unit addition. For example after the addition of the unit of code that allowed a player to intercept the ball this function was tested by ensuring that the player could intercept the ball anywhere on the pitch, intercept both a stationary and moving ball and only intercept the ball if it was not under the possession of another player.

Validation Test
"Validation testing provides final assurance that software meets all functional, behavioural and performance requirements" (Pressman, 1997: p507). Black-box testing was used to make sure that the functional and behavioural requirements were met and alpha testing (Pressman, 1997) was used to assess performance requirements and also some functional and behavioural requirements. The test cases used for each agent are included in the appendix although more spontaneous test cases (e.g. If I click on my own player's circle will it do anything?) are not included.

The validation testing carried out suggests that the functional and behavioural requirements have largely been met and given a workstation with the recommend requirements (i.e. a SUN Ultra Sparc 10 or 5 workstation) the performance requirements have also been met. A number of problems have however been found and these include:

- Players and goalkeepers seem unable to intercept the ball if it is travelling at speed.

This problem will be discussed further in chapter 10, 'Future Development'.

Conclusion
It is always very difficult to assess the success of a piece of software's testing stage because, "testing cannot show the absence of defects, it can only show that software errors are present" (Pressman, 1997: p466). Therefore although the testing highlighted many errors it is still unclear as to how many were
left undiscovered. This problem was made worse by the very nature of the model. Firstly this is a highly complex model therefore it is simply not possible to test all of the model states that may exist and secondly this is a distributed model so it often took a great deal of time to set the model up for testing and therefore less testing could be achieved in the time available.

Although the testing strategy proved very successful in highlighting many errors within the model more testing is recommended if we are to be sure that the model is truly robust. It is recommended that someone other than the author, perhaps in the form of an independent test group carries out this future testing to maximise the destructive potential of the testing (Pressman, 1997: p506). Beta testing (Pressman, 1997) is ultimately recommended because it means that the model will be subjected to demanding use under much more realistic conditions.
Chapter 9. Evaluation

Now that we have looked at the model's life cycle and the final design we can evaluate these and answer some of the questions posed at the start of the report. Firstly how good is the final five-a-side football model constructed? Secondly what can it tell us of the suitability of Empirical Modelling for creating accurate computer models of the real world? And thirdly how does it compare with more conventional computer modelling techniques?

How good is the final model?

In answering this question I shall pose a number of other questions:

- Does the model fulfil the requirements laid out?
- How accurate is the model?
- How well engineered and designed is the model?

Does the model fulfil the requirements laid out?

The model does indeed appear to fulfil all of its requirements, both in terms of its verification and its validation. In terms of its verification all of the required functionality has been implemented because we have built the model as outlined within the LSD specification. In terms of the model itself the aim was to create as accurate a model as possible given the time available therefore to evaluate the model in terms of its validation we really need to answer the next question.

How accurate is the model?

Although it is obviously extremely difficult to quantify just how accurate a computer model is it is apparent that the model created delivers a reasonably accurate simulation of the game of five-a-side football, all be it with only six players. Without any prior knowledge someone familiar with five-a-side football can tell that the model is a simulation of the game and should be able to participate with only the additional knowledge required to use the user interface. Although many inaccuracies do exist within the model, some of which are discussed in chapter 10 certainly the core components of the game have been included and the foundations have been laid for perhaps a much more accurate simulation.
How well engineered and designed is the model?

Although it is extremely difficult to quantify just how well a piece of software is engineered and designed I believe that it is apparent that the model is both well engineered and well designed. We have seen in chapter 5 that although there were many forced design constraints (e.g. the unavailability of an object orientated design) a number of design techniques have been used to ensure this is the case. This has resulted in a model that is very flexible, logical in its design and can be easily changed and upgraded.

How Suitable is Empirical Modelling for Creating Real-World Models?

In answering the question of how suitable Empirical Modelling is for creating accurate models of the real-world I shall use my experience gained from constructing the model to identify what I feel are the strengths and weaknesses of Empirical Modelling in this context.

Strengths of Empirical Modelling

- Empirical Modelling is more natural than conventional computer programming because it mirrors the real world. The modeller can identify parallels between the real-world system and the model through principles such as dependencies, observables and protocols. *
- Empirical Modelling tends to result in models that closely mirror the system being modelled because they are not purely the modeller's interpretation of that system. *
- Empirical Modelling requires less programming knowledge than conventional computer programming because the programmer is not required to interpret the system being modelled and encompass it in their design so much. The real knowledge required is that of the system being modelled. *
- Empirical modelling tends to result in models that are very flexible and allow considerable experimentation. The modeller is able to try out new ideas and change the model with considerable ease.
- Empirical modelling is very easy to test and debug because alterations can be made ‘of the fly’. There is no need to change and recompile the model and any changes made immediately propagate through the model.
- The Empirical Modelling life cycle allows gradual construction of model, this means that the required level of accuracy can be built because this level should increase with the construction time.
• Empirical Modelling guarantees the modeller some end model because the model is built up gradually. We can not have the situation where a finalised design is found not to work and therefore a model is not delivered.

• The gradual development of models means that the model requirements are likely to be fulfilled because continual evaluation of the model can take place and therefore any changes that are necessary to steer the model towards its requirements can be made.

• The Empirical Modelling life cycle is very flexible, we are not confined to the rigid structure of traditional software life cycle models.

Weaknesses of Empirical Modelling

• There is very little good documentation available for Empirical Modelling and few past examples to look at. *

• The Empirical Modelling tools are still very much research tools therefore many still contain bugs, they are not very user friendly and are not as integrated as they might be. *

• The 'hands on' nature of Empirical Modelling can lead to badly engineered models because the design stage is neglected in favour of the implementation stage. Also changes can be made rather haphazardly rather than carrying out proper redesigns. *

• Each Empirical Modelling tool uses a slightly different syntax which can cause obvious problems if the modeller has to use up to four tools in parallel.

• The computational performance of the models constructed using Empirical Modelling is often poor.

• It is extremely difficult to conclusively test a model because of the huge number of possible states that may exist and therefore need testing.

• The models that are developed are not very portable because all of the tools are purely UNIX based.

• Complex graphics are not very easy to implement for the models.

• It can be extremely difficult to trace bugs because if a variable is mistyped the system believes that the user wishes to create a new variable and therefore the error is often not picked up.

• Empirical Modelling often forces the modeller to reuse code because there is only very limited support for objects through 'virtual agency'.

• The development interface is quite basic. For example there is no support for 'drag and drop' development of the user interface.
- It can take considerable time for an experienced computer programmer to come around to the different way of thinking required by Empirical Modelling.

(* It is the author's opinion that these are the most significant strengths and weaknesses)

I believe after studying both the strengths and weaknesses of Empirical Modelling it should be apparent that it is indeed a very suitable technique for creating accurate computer models of the real-world. There are obviously still difficulties to overcome, largely because this is still very much a research topic but the models constructed do tend to closely correspond with the system being modelled, both in terms of functionality and design and in terms of accuracy which has to be ultimate goal. It would appear therefore that Empirical Modelling does indeed deliver its promise of establishing a stronger and more permanent link between what is observed in the 'real-world' and our model of it.

**Comparison with Conventional Computer Modelling Techniques**

I will briefly compare the Empirical Modelling approach with the construction of a model of five-a-side football using an object orientated programming language such as C++ or Java and the life cycle shown in figure 9.1 because this is the method by which the vast majority of software programs are constructed, "The linear sequential model is the most widely used paradigm for software engineering." (Pressman, 1997: p35). For details of this software life cycle model please see Pressman (1997) or any other good book on software engineering.
Figure 9.1. The linear sequential or 'waterfall' software life cycle model.

Figure 9.2 shows the comparison between a model of the same system constructed using the 'waterfall' software life cycle model and an object-orientated programming language and one constructed using the Empirical Modelling approach.
<table>
<thead>
<tr>
<th>Waterfall - Object Orientated Design</th>
<th>Empirical Modelling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programmer's interpretation of the real-world.</td>
<td>Closely resembles observations of the real-world. *</td>
</tr>
<tr>
<td>Requires extensive programming knowledge and experience.</td>
<td>Requires knowledge of system being modeled but significantly less programming knowledge or experience. *</td>
</tr>
<tr>
<td>Object Orientated (Encapsulation, Inheritance, Information hiding etc..) *</td>
<td>Procedural (Code often re-used).</td>
</tr>
<tr>
<td>Encourages good design (e.g. modularity).</td>
<td>Allows poor design and is more open to abuse.</td>
</tr>
<tr>
<td>Object paradigm.</td>
<td>Agent paradigm.</td>
</tr>
<tr>
<td>Supports many data types.</td>
<td>Supports only primitive data types.</td>
</tr>
<tr>
<td>Many in-built functions and third party modules available. *</td>
<td>Few inbuilt functions and no third party modules.</td>
</tr>
<tr>
<td>High performance possible.</td>
<td>Typically delivers relatively poor performance.</td>
</tr>
<tr>
<td>Very portable, e.g. cross platform.</td>
<td>Less portable, predominately UNIX based.</td>
</tr>
<tr>
<td>Complicated client/server distributed models.</td>
<td>Simplified client/server distributed models. *</td>
</tr>
<tr>
<td>Uncertainty must be resolved.</td>
<td>Uncertainty can be left and resolved later with the aid of experimentation.</td>
</tr>
<tr>
<td>Relatively hard to debug, test and experiment.</td>
<td>Very easy to debug, test and experiment.</td>
</tr>
<tr>
<td>More manageable testing due to improved modulation.</td>
<td>Complex testing due to potential number of model states.</td>
</tr>
<tr>
<td>Inflexible due to the need to get each stage of the life cycle correct before undertaking the next.</td>
<td>Great flexibility afforded by the life cycle. *</td>
</tr>
<tr>
<td>Relatively little experimentation encouraged therefore arguably more chance of product not fulfilling requirements.</td>
<td>Great deal of experimentation encouraged therefore arguably more change of delivering product that fulfills requirements.</td>
</tr>
<tr>
<td>Mistakes in life cycle are much more serious therefore end product is not guaranteed.</td>
<td>Mistakes in life cycle are less important and you are always ensured some end product.</td>
</tr>
<tr>
<td>Very well understood and proven.</td>
<td>Still relatively unknown (research area) and unproved.</td>
</tr>
<tr>
<td>Lots of documentation and support available. *</td>
<td>Very little documentation available.</td>
</tr>
<tr>
<td>Predominately compiled.</td>
<td>Interpreted therefore very easy to debug.</td>
</tr>
<tr>
<td>Extensive graphical capabilities.</td>
<td>Limited graphical capabilities.</td>
</tr>
<tr>
<td>Open ended models not encouraged. The programmer should always know what the model will do.</td>
<td>Creates very open ended models where the model's behaviour is not always known in advance.</td>
</tr>
<tr>
<td>Creation of models usually takes considerable time.</td>
<td>Allows very swift creation of simple models.</td>
</tr>
<tr>
<td>Advance development environment.</td>
<td>Basic development environment.</td>
</tr>
</tbody>
</table>

(* It is the author's opinion that these are the most significant advantages)

It would seem therefore that both methods have significant advantages and disadvantages over each other. The main advantage of the 'waterfall' model using an object orientated language would appear to be its use of objects and the advantages they bring, the fact that it is very well known and therefore has a deal of support and the amount of aid that it provides the programmer. The main advantage of the Empirical Modelling model would appear to be its close correspondence with the system being
modelled, both in terms of its design and functionality, the increased flexibility the approach delivers, the ease with which distributed models can be constructed and the lesser amount of programming knowledge required.

In terms of constructing models it would seem therefore that although the 'waterfall' method using an object orientated language allows the creation of a superior piece of software (i.e. in terms of performance, graphics and elegance etc.) the Empirical Modelling method allows the creation of a more accurate model because it is more the simulation of the real-world system and less the imitation and interpretation. Empirical Modelling also appears to allow models to be created with greater speed and with much less effort and experience.
Chapter 10. Future Development

It is important that any substantial project work has significant scope for future development because any well chosen project area will always have capacity for additional research and development.

Future development for this project is likely to take one of three routes. Firstly it is likely to be concerned with making the model more accurate. Secondly it is likely to be concerned with using the model constructed as the initial framework for an automated model, that is a model with agents that possess some degree of autonomy. Thirdly it is likely to be concerned with a much more in-depth comparison of empirical modelling techniques and conventional computer modelling techniques.

Improving the Model

There is arguably infinitesimal scope for making the model more akin to the real-world game of five-a-side football therefore I have seen it fit to only discuss those improvements that I see as both desirable and readily obtainable. The reader may decide for themselves which improvements are more desirable than others:

- Improving the interception of the ball
- Introducing a degree of uncertainty
- Introducing three dimensions
- Improving the physics engine
- Introducing more players
- Introducing communication
- Improving the model's graphics
- Increasing the number of attributes
- Dynamic attributes

Improving the interception of the ball

As the model currently stands an outfield player or goalkeeper will only be successful in intercepting the ball if it is either stationary or travelling very slowly. Obviously this introduces problems because a player will very often not intercept a pass and a goalkeeper not make a save because the ball is
travelling too fast to do so. It would therefore be of great benefit to correct this problem, the cause of which is believed to be the fact that a player's or goalkeeper's execution cycle is too slow in interpreting the situation and realising that the ball should be intercepted. In the time that the player or goalkeeper should realise that they should attempt to intercept the ball the ball travels past the agent and therefore makes an interception impossible.

Introduction a degree of uncertainty
As the model currently stands many actions are carried out with an absolute level of certainty. For example passes, shots and throws are always directed exactly at their target and interceptions (if the system allows them) are always successful. This obviously conflicts with the real-world game of five-a-side football and therefore it would be desirable to introduce a degree of uncertainty to these actions in the same way that tackles are not always guaranteed to be successful. The similar use of a random number to adhere to predetermined percentages could be used so that for example an outfield player with a passing skill of four would reach their passing target about 80% of the time. More urgently perhaps a system could be introduced that uses a goalkeeper's saving skill to determine the percentage of shots that they are likely to save because as the model currently stands a goalkeeper will theoretically save 100% of the shots that they can reach.

Introducing three dimensions
The current model only recognises two dimensions therefore such actions as lofting and heading the ball are not possible. Although it has been noted that a 'no over head height' rule can be stipulated meaning that the model is not too far out of touch with a real-world game of five-a-side football it would still be desirable to introduce a third dimension to add extra depth to the model. This would allow a much greater range of actions to be implemented and make for a more complete model.

Improving the physics engine
The model currently possesses a very rudimentary physics engine such that factors such as acceleration, momentum and friction are either not modelled or modelled with a low degree of realism. It would be desirable therefore to introduce a more realistic physics engine so that for example a player's acceleration, deceleration and momentum are all taken in to consideration when modelling a player's movement around the pitch. This would deliver a more realistic model of the game.
Introducing more players

The model is of a game of five-a-side football and therefore although trivial the introduction of two more players on each side would make the model more realistic. It is apparent that the final model only incorporates six players to ease development and especially testing and the introduction of a further four would require:

1. The construction of the additional agent files.
2. The necessary changes to the spectator's LSD code.
3. The necessary changes to each agent's field of vision.
4. The necessary changes to the pitch display.
5. The necessary changes to the initialisation of yet to be activated agents and the request for information from previously activated agents.
6. The necessary changes to the team lists.

Introducing communication

At the moment no provisions are given for players to communicate with one another despite the fact that this is an important part of five-a-side football because as a team game communication is vital to ensure that shared team tactics are carried out. Therefore although the lack of communication within the model might not matter if the operators are within talking distance problems are obviously introduced if this is not the case (remember clients need only to be on the same network as the server). It would therefore be desirable to incorporate some provisions for cross player communication. This might take the form of a communications box where messages can be sent and received or a number of predetermined communications messages such as, "the team is now playing a defensive style".

Improving the model's graphics

The model currently utilises very basic line drawing graphics. Although these are sufficient it would be a very interesting exercise to attempt to develop a three dimensional graphics (3D) engine for the model which would allow for a much more accurate graphical representation of an agent's current perception. For example at the moment no matter how far away it is an agent can always see the ball if
it is in their field of vision. A 3D graphics engine would allow the ball and other objects to appear larger closer up and smaller further away thus adding to the model's level of realism.

*Increasing the number of attributes*
As the model currently stands players possess a relatively small number of attributes. For example within the model an outfield player possesses just five skill attributes whereas within the football management game, 'Championship Manager 2' outfield players possess twenty one! A greater number of attributes arguably allow for a more realistic representation of that player's true level of ability and means that more factors can be taken into consideration when determining the outcome of events. For example when a player attempts to tackle an opponent with the ball we might take additional factors such as the opponent's level of dribbling skill, the opponent's prominent foot (i.e. a tackle coming in from the left on a player dribbling with their left foot is likely to be more successful than from the right because the right foot can then be used to shield the ball) and the if the opponent is currently injured or not into consideration.

*Dynamic attributes*
Player's currently posses static attributes. A player's attributes such as their level of fitness, shooting skill and tackling skill are constant through out the match and it maybe argued that this does not mirror a real-world game of five-a-side football. Within a real-world game attributes will change. For example a player will become tired so their level of fitness and speed will decrease, a player's confidence might increase because they have scored a couple of goals making their shooting skill increase and a player might become injured causing their passing skill to diminish. It would therefore be desirable to incorporate such features into the model to increase the level of realism.

*Constructing an Autonomous Model*
As previously mentioned a very interesting exercise would be to use the model as the framework for an autonomous model, that is a model where agents possess a high degree of autonomy or, "self-government" (Collins, 1994). This would mean that agent will carry out their actions automatically and show some degree of intelligence in choosing when are where to do so. Ideally we would like a model that could be seen to be acting out what appears to resemble a real-world game of five-a-side football. Obviously this would very much be an exercise in artificial intelligence but the existing model could
provide a very good framework for someone attempting to investigate the intelligence required to play such a game as five-a-side football. Being a team game areas such as shared team tactics, communication and shared goals and objectives could be investigated.

**Comparison with conventional computer modelling techniques**

Although a brief comparison with conventional computer modelling techniques has been made a more in-depth analysis would prove an interesting exercise. This could involve the direct comparison with a five-a-side football computer model constructed using conventional techniques and therefore would necessitate either the construction of such a model or the acquisition of the source code for one similar. This would allow many more comparisons to be made and perhaps highlight new areas where empirical modelling possess significant strengths or weaknesses.

**Other developments**

As a final thought I have seen it fit to briefly discuss future development that might emerge as empirical modelling evolves. As a distributed model the model constructed would seem to be a prime candidate for an exercise in running an empirical model over a wide area network (WAN), perhaps using web based tools to implement it over the Internet. Such tools are currently in development and would certainly open up a whole new area for the prospective modeller to explore.
Chapter 11. Conclusions

Within this chapter the conclusions drawn from the project are discussed.

The first conclusion that can be drawn is that the project objectives were met and were set just about right. An accurate simulation of five-a-side football with considerable scope for improvement was constructed using an Empirical Modelling approach and this was utilised in evaluating Empirical Modelling and in comparing it with more conventional computer modelling techniques.

The second conclusion that can be drawn from the project is that the Empirical Modelling approach used appears to be very suitable for constructing accurate computer models of the real world, models that themselves correspond very closely with their real-world counterparts. Although more research is required this project suggests that the Empirical Modelling approach used does indeed appear to deliver its promise of establishing a stronger and more permanent link between what is observed in the 'real-world' and our model of it. It is unclear however if the approach would deliver similar successes in modelling other domains that are not as limited and well understood as five-a-side football.

The third conclusion that can be drawn is that Empirical Modelling compares very favourable with more conventional techniques for constructing accurate computer models of the real world. Although more research is required this project suggests that in terms of ease of construction and overall accuracy Empirical Modelling offers a better solution although perhaps at the price of an inferior piece of software in terms of graphics, performance and elegance of design etc..

The final conclusion that can be drawn from this project is with regards to project management. Although the project was largely on schedule a significant number of problems were experienced which should be taken in to consideration for future exercises of this nature. Problems included the unexpectedly long time required to learn new tools and concepts. The extremely limited amount of documentation available on Empirical Modelling. The problems experienced due to the fact that the tools used were very much still research tools and perhaps most significantly of all the fact that due to unexpected problems stages always took longer than expected and therefore this must always be taken into consideration when estimating time scales. Other lessons learnt include the importance of
scheduling and timetabling to make sure that work gets completed. Although considerable time was
given away for each stage it often took longer than expected and therefore some rescheduling was
required to ensure that subsequent stages were not delayed. Another important lesson was the
importance of co-operation and mutual aid, in this project obtained through frequent meetings with
other students undertaking Empirical Modelling projects and the importance of choosing objectives that
are obtainable and not wholly over ambitious. Overall this was an extremely useful exercise in good
project management and displays just how difficult a skill it really is.
Bibliography


'Premier Manager 2', Gremlin Interactive, 1995.


Acknowledgements

The author would like to thank Dr Steve Russ, Dr Meurig Beynon and the Empirical Modelling group at the University of Warwick for their considerable help in completing this project.
Appendices

The Visual Basic screen prototype for the mock spectator screen layout

The Visual Basic screen prototype for the mock referee screen layout
The Visual Basic screen prototype for the mock player screen layout

The Visual Basic screen prototype for the mock keeper screen layout
Model File Descriptions

1. agent_functions.e
   • This file details the shared functions of the referee, the outfield players and the goalkeepers.

2. ball.e
   • The file implements the ball agent.

3. field_vision.e
   • This file sets up the field of vision dependencies for the agents.

4. get_pos.e
   • This file finds out for an agent the positions of agents previously activated and initialises those yet to be activated.

5. keeper.d
   • This file outlines the user display for a goalkeeper agent.

6. keeper_constants.e
   • This file sets out the constants as used by a goalkeeper agent.

7. keeper3.adm3
   • This is the file for the first goalkeeper agent. The second goalkeeper agent's file (keeper6.adm3) is the same save for the id number and the field of vision list.

8. match.e
   • This file implements the match agent.

9. mouse.e
   • This file sets up mouse constants used by the user controlled agents.

10. names_nums.e
    • The file sets out the names and numbers of each of the agents.

11. pitch.d
    • This file outlines the display of the five-a-side pitch and graphical representation of all of the agents involved in a match.

12. pitch.e
    • This file sets out all of the dimensions for the five-a-side pitch used in each match.

13. pitch_constants.e
    • This file sets out all of the constants required by the pitch display such as the size of the penalty spot.
14. **pitch_functions.e**
   - This file outlines all of the functions required for an agent's pitch display.

15. **pitch_win.e**
   - This file outlines the window used to show the pitch display by all of the agents required to detect any user input (i.e. the referee, players and goalkeepers).

16. **player.d**
   - This file outlines the user display for an outfield player agent.

17. **player_constants.e**
   - This file sets out all of the constants used by the outfield player agents such as the proximity required to intercept the ball.

18. **player_functions.e**
   - This file outlines all of the functions exclusively used by the player agents.

19. **player_request.e**
   - This file outlines all of the information requested by each player from the server upon their initial activation.

20. **player1.adm3**
   - This is the file outlining the first outfield player agent. The other player agent files (player2.adm3, player4.adm3 and player5.adm3) are the same save for the id number and the field of vision list.

21. **ref.adm3**
   - This is the file for the referee agent.

22. **ref.d**
   - This file outlines the user display for the referee agent.

23. **ref_constants.e**
   - This file sets out all of the constants used by the referee such as the number of execution cycles that represent one match minute.

24. **ref_functions.e**
   - This file outlines all of the functions used exclusively by the referee agent.

25. **spec_pitch_win.e**
   - This file outlines the window used to show the pitch display by the spectator agent (i.e. no user interaction is required to be detected).

26. **spectator.d**
   - This file outlines the user display for the spectator agent.
27. spectator.e
   • This file outlines the spectator (server) agent.

28. spectator_functions.e
   • This file sets out the functions used exclusively by the spectator agent.

29. team1.e
   • This file outlines the details required for team one such as the team list.

30. team2.e
   • This file outlines the details required for team two such as the team's name and colours.