Creating an AI-Test Platform

Benoit Chaperot
School of Computing
The University of Paisley
Scotland
benoit.chaperot@paisley.ac.uk

Colin Fyfe
School of Computing
The University of Paisley
Scotland
colin.fyfe@paisley.ac.uk

ABSTRACT
In this paper we explain changes made to a motocross game to allow other researchers to develop and experiment their own artificial intelligence techniques in the game. The game is split between an executable and AI DLL's.

Keywords
Motocross game, software architecture, Artificial Intelligence, Dynamic Link Library, AI, DLL

1. INTRODUCTION
Motocross The Force is a motocross game featuring terrain rendering and rigid body simulation applied to bikes and characters. In [2] and [3] we have investigated the use of artificial neural networks (ANN's) to ride simulated motorbikes in this computer game.

The game offers a very good environment to experiment with advanced artificial intelligence and data mining techniques; although the control of the bike is assisted by the game engine, turning the bike, accelerating, braking and jumping on the bumps involve complex behaviours which are difficult to express as a set of procedural rules, and make the use of advanced AI techniques very appropriate.

Different kinds of ANN's have been tested to control the motorbikes: feed forward multi layered perceptrons, radial basis functions networks and self organising maps. While experimenting with these different kinds of networks, some difficulties were noticed:

- The AI source code is in the middle of the game engine code; this makes it difficult to read and maintain; many files have to be modified to change the kind of AI in use in the game; files can not be swapped between researchers due to intellectual property issues.
- Only one kind of AI can be tested at a given time. Direct comparison between different kinds of AI is not possible.

In the research community, it happens often that one researcher presents a new algorithm or AI or data mining technique, and presents the technique in the context of one particular problem or application. The applications used are often very different. It appears that there is a need in the research community for common platforms to evaluate and benchmark individual AI techniques. This has been discussed at the CIG06 conference in Reno, USA.

For data mining it is common practice to use standard datasets to evaluate and compare different classification techniques (see for example [4]). For video games, there seem to be a need for more common platforms to compare AI techniques.

Splitting the motocross game, between the game engine on one side and the AI on the other side, allows for easier AI code maintenance and implementation, and changes the game into a common open platform for many developers and researchers to test and compare different techniques.

In this paper we detail how we have split the game, and how to implement new AI for the game.

2. CHANGES TO THE ARCHITECTURE
The game has been split between the game engine on one side and the AI on the other side, the changes in architecture are detailed below.

2.1 The Original Architecture

Figure 1: Original game architecture.

1. The game is composed of one executable and some data files.

2. The AI source code is in the middle of the game engine code; this makes it difficult to read and maintain.

3. There are some intellectual property issues in that only the authors can implement an AI for the game.

4. Only one kind of AI can be used at a time in the game to control motorbikes. Each motorbike can use its own data file. It is not possible to directly compare AI techniques.

2.2 The New Architecture

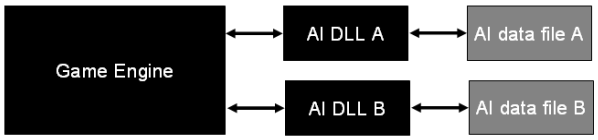

Figure 2: New game architecture.

1. The game is made of one executable, some DLL's and some data files. DLL stands for Dynamic Link Library. Typically DLL's provide one or more particular functions and a program accesses the functions by creating either a static or dynamic link to the DLL. In the context of the new architecture for the motocross game, each DLL implements one kind of AI and is linked dynamically.

2. The AI source code is separated from the game engine code, with one small DLL project per AI; this makes it easy to read, implement and maintain.

3. The AI part is separated from the game engine and is open source. Anyone can implement an AI for the game.

4. Many kinds of different AI's can be used at a time in the game to control motorbikes. Each motorbike can use its own DLL file and its own data file. It is possible and easy to directly compare AI techniques.

3. GAME CLASSES AND STRUCTURES

Before implementing AI for the game, it is important that the user has a good understanding of the various classes and structures in use in the game.

3.1 Track

A track is a course over which races are run. Typically a track is of variable width along its course. The tracks are marked using WayPoints.

3.2 WayPoint

WayPoints are markers positioned on the centre of the track, every metre along the track, and are used:

- To give course information to computer controlled bikes, i.e. position, direction and width of the track.
- To monitor the performance: a bonus can be given to a computer controlled bike for passing a WayPoint.

Figure 3: Screen shot taken from the game, the white crosses on the left hand side represent WayPoints.

3.3 Game Engine

There is one game engine, it is implemented by the executable. It is everything but the AI, and is responsible for updating the simulation.

3.4 AI

There is one AI per computer controlled bike. Each AI can be written to or read from an AI data file. Each AI makes use of an AI DLL. More than one bike can share the same DLL. An AI is trained to make a decision given a situation.

3.5 Situation

The situation is the general state of the bike, position, orientation and velocity relative to the ground.

3.6 Decision

These are commands and are the same as the controls for the human player:

- Accelerate, brake.
- Turn left, right.
- Lean forward, backward.

3.7 SampleData

SampleData is the main structure used for communication between the game engine (executable) and the AI DLL. Typically the game engine fills the situation fields of the structure and pass the structure to the AI DLL; the AI DLL fills the decision fields of the structure, given the situation, and passes it back to the game engine; the game engine updates the state of the corresponding computer controlled bike and the simulation accordingly.

3.8 Training Set

A training set is a structure used for the training of AI. A training set is a collection of **SampleData**'s made from the recording of a human player playing the game. Each sample contains a situation and the corresponding human player's

decision. AI's are trained to make the same decision as the human player, given a situation.

3.9 Terrain
Structure used to give ground height information.

3.10 Weight
A Weight is an AI parameter that is to be optimised using for example Genetic Algorithms. Typically a weight is a connection strength between two neurons in an ANN.

3.11 Genetic Algorithms
Training is considered as an optimisation. GA's are used to improve the AI's performance by modifying AI weights. GA are implemented by the executable.

4. IMPLEMENTING NEW AI

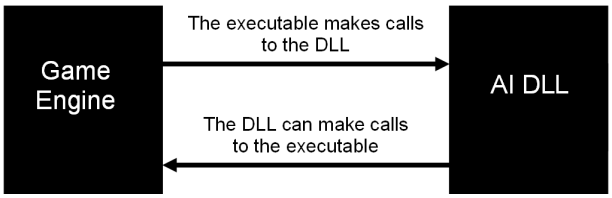

Figure 4: Communication between the game engine executable and the AI DLL.

An important feature of the architecture is that the executable calls DLL functions, for example for decision making, but the AI DLL's can also call executable functions, for example to obtain more information about a situation, before making a decision. It is a two way communication process.

4.1 DLL Functions
In order to be recognised by the game executable each AI DLL must be placed in a particular folder and implement a set of functions; these functions can be grouped into categories.

4.1.1 General Operation Functions
The general operation functions are:

- Creation: this function creates an AI and returns a void pointer on the newly created AI to the executable.
- Destruction
- Decision Making: this function returns a decision to the executable, given a situation.
- Render: this function is called every time the game is rendered; this gives the opportunity to the AI DLL to display AI information; this is mainly used for debugging purposes.

After AI creation, the executable keeps a void pointer to the AI and passes it as a parameter to all subsequent calls to the AI DLL.

4.1.2 Training Functions
These functions are typically used for training the AI using back propagation techniques.

- Generate AI From Training Set: this function is called every time the user wants an AI to learn from a training set. The DLL loads and processes the training set.
- Generate AI From Training Set Update: this function is called once per game update: the DLL updates the training or generation of an AI from a training set; typically there are 25 backpropagation iterations per game update and 100 game updates per second.
- Is Generating AI From Training Set: returns true if the AI is currently training from training set.

4.1.3 Weights Functions
These functions are typically used for training the AI using genetic algorithms techniques.

- Put Weights
- Get Weights
- Set Generation
- Get Generation
- Save Weights
- Get Number Of Weights

4.1.4 Version Functions
- Get AI Name: returns AI DLL name, one name per AI DLL.
- Get AI Version: returns version of AI implemented.
- Get Debug: returns true if this is a Debug version of AI DLL.

All these functions were found useful to carry out our experiments. To make the architecture simpler, an AI DLL is required to implement all these functions in order to be recognised by the executable. If some functions are not needed (for example the user does not want to use GA), then the user can simply create empty functions.

The executable and AI DLL's make use of the DirectX library for vector and matrix structures and operations.

4.2 Executable Functions
The executable makes the following functions available to AI DLL's.

4.2.1 WayPoint Functions
AI DLL's can make calls to the executable to obtain the following interpolated information about WayPoints:

- Transformation matrix
- Position

- Direction
- Width

The information is interpolated between WayPoints. The functions take two parameters, a WayPoint index, with one WayPoint every metre along the track, and a distance in metres along the track from this WayPoint.

4.2.2 *Drawing Functions*

AI DLL's can make calls to the executable to draw the following kind of primitives on the screen:

- 3D Vertices
- 3D Lines
- Text
- Rectangles

These functions are used mainly for debug purposes and take a colour as one of their parameters.

4.2.3 *Track Functions*

AI DLL's can make calls to the executable to obtain information about tracks and terrain:

- Height: a function returns the terrain height at a given position.
- Track creation: the DLL can load tracks; this is useful when processing training sets; a training set can contain training data from more than one track.
- Track unique identifiers: the DLL can check that a track has not changed since the time the training set was generated.

4.2.4 *Other Functions*

- Get Version: returns version of the executable.
- Forward transform: a function returns a space centred at the origin of the motorbike; the Z axis points up and the Y axis follows the horizontal velocity direction. This space is more convenient than bike space to represent and transform world objects in relation to the bike.

4.3 Operation

The new AI system works as follow:

1. The executable loads all DLL's contained in a given directory; if the DLL implements all functions described above, and versions match, then it is kept loaded, otherwise it is unloaded.
2. Each computer controlled bike loads its own AI data file; each AI data file is to be associated with an AI DLL. The association between data files and DLL's is done by matching the name contained in data file header with the names given by the DLL's.
3. The AI is created using the matching AI DLL **CreateAI** function, and all future AI function calls will call the matching AI DLL functions.

5. ILLUSTRATIVE RESULTS

It was feared that this new AI system would run slower than the original AI system because of the high number of calls between the executable and the DLL's. It actually runs faster, despite the high number of calls between the executable and the DLL's, because the AI code is now tidier and smaller than before.

Two different kinds of AI have been implemented using the new architecture, feed forward multi layered perceptrons (MLP), and self organising map (SOM). The source code is inspired by [1]. Source code and a free version of the game will be made available at the following address:

`http://cis.paisley.ac.uk/chap-ci1`

It is possible to easily compare in real time the two kinds of AI. The two kinds of ANN's used the same number of weights and were trained using the same training data. It appears that the MLP performs better but the SOM is less computational intensive. The ANN's behave differently and make different decisions and mistakes. Increasing the number of weights for the SOM networks enables for better performance but the performance is always less than that of MLP networks.

The main problem with the SOM's is that their internal operation prevents them from differentiating between important and not so important inputs. The networks fail to make decisive decisions.

6. CONCLUSION

This new architecture for the game allows us to easily develop and experiment with AI techniques. Source code and a free version of the game will be made available soon. We hope that many researchers will join us at developing new and innovative AI.

Future work may include the creation of a **Motocross The Force Cup**, where many different AI's will compete at racing bikes in the game.

Other work may involve the use of large scale distributed processing to optimise existing AI using genetic algorithms.

7. REFERENCES

[1] M. Buckland. http://www.ai-junkie.com/. Technical report, 2005.

[2] B. Chaperot and C. Fyfe. Motocross and artificial neural networks. In *Game Design And Technology Workshop 2005*, 2005.

[3] B. Chaperot and C. Fyfe. Improving artificial intelligence in a motocross game. In *CIG06*, 2006.

[4] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, editors. *Machine learning, neural and statistical classification*. Ellis Horwood, 1994.