The 2003 Report of the IGDA's Artificial Intelligence Interface Standards Committee

Alexander Nareyek, Bjoern Knafla, Daniel Fu, Derek Long, Christopher Reed, Abdennour El Rhalibi, and Noel S. Stephens (eds)

Welcome to our committee's report of 2003, which covers the activities of the committee from June 2002 to June 2003. The first section provides a general introduction and overview while the following sections present details on the committee's single working groups. Feel free to post questions, comments or suggestions to our feedback forum.[1]

Sections:

[1] The current text is a PDF rendering (with minimal formatting) of the latest version of the report available on the Wayback Machine - Internet Archive (https://archive.org/web/), the original report was located at http://www.igda.org/ai/report-2003/report-2003.html

1. Preface

On June 17, 2002, our committee on standards for artificial intelligence interfaces (AIISC) was officially launched within the International Game Developers Association (IGDA). The committee's goals are to provide and promote interfaces for basic AI functionality, enabling code recycling and outsourcing thereby, and freeing programmers from low-level AI programming as to assign more resources to sophisticated AI. Standards in this area may also lay grounds for AI hardware components in the long run.

Especially because many institutions from the simulation & military sector are very interested in our initiative, I want to stress right at the beginning that the initiative is directed toward *computer games* (on PCs as well as console platforms), i.e., our goal is *not* to provide interfaces for general artificial intelligence applications. While we hope that our work will also be useful for other domains, computer games pose very specific requirements on issues like available computing resources and goals, and we will primarily direct our design decisions on optimizing the interfaces for the computer-game domain.

Preparations for the committee started more or less with a roundtable at GDC 2002 to check the interest in such an initiative. The roundtable was very successful, and I started organizing the establishment of the committee soon after. It turned out to be an unexpectedly huge undertaking; my related e-mail folders contain far over 2000 individual e-mails until now.

The committee has about 65 members and is composed of multiple working groups, which currently work on the following topics: World Interfacing, Steering, Pathfinding, Finite State Machines, Rule-based Systems and Goal-oriented Action Planning. An additional group on Decision Trees has recently been put on hold until we have a stronger force to tackle this topic. Furthermore, we have a support team, mostly composed of students, which do a great job in supporting the working groups with summaries, documentation and so on. Overviews of the work of all these units during the last year can be found in the following sections.

Every working group has a coordinator (and potentially also a co-coordinator) that manages the group and makes sure that the work goes on. The coordinators and myself represent the committee's steering committee, which focuses on organizational and political work, may set up new or cancel existing working groups, and decides on issues like membership approval.

Our work is coordinated via a project page at SourceForge. We mostly use the forum features for discussions of the single working groups as well as the CVS file system for documentation. This style of operation is far from perfect, and SourceForge's forum features did not turn out to be the most convenient option. Despite these

problems, the discussions become more intense and useful every day. Not even considering group-internal e-mail traffic, 1777 posts have been posted to our forums as of now, with large increases every month.

Especially in the beginning, we faced many problems with respect to member activity. Many people were very interested in participating but obviously did not evaluate their available temporal resources. In consequence, we already had to replace about one third of the initial members. This issue is still not fully overcome, and we continue the process of replacing inactive members.

Pursuing a standardization goal, one is immediately confronted with fears of impeding innovation, needless bureaucratic overhead and possible market-power related goals. We are well aware of the potential drawbacks that can result from bad interface standards, and also implemented measures to counter potential influence from institutions with their own agenda, e.g., not allowing leading positions of the committee to be filled with middleware representatives. It is to say that we seem to be lucky until now, and we mostly have a very harmonious work atmosphere. Besides pushing the standards issue, the committee discussions provide a great learning experience for everyone.

In comparison to other standards in the area of AI, our approach is clearly oriented toward a specific application. It seems to me that for many other AI standards initiatives, the goal was not to enhance specific applications but to standardize specific techniques that might be of use... somehow. I am honestly not sure whether those approaches are useful at all, and whether many AI techniques today are at a level of general applicability - without a relation to a specific application domain. In contrast, application-oriented standardization initiatives, like OpenGL or DirectX in the graphics domain, have been a huge success and led to a tremendous progress in that area. Our "hands-on" application-oriented approach is also reflected in the committee composition: Nearly half of the committee consists of game developers; the other half being equally distributed between middleware representatives and academics.

Creating a report like this is always a trade-off. We certainly want to keep the public informed of our work and progress. On the other hand, our resources are limited, and we would of course like to direct as much work as possible to the creation of the interface standards. We have thus kept the report relatively short, and would like to invite you to also have a look at the online information of our 2003 GDC roundtable on AI Interface Standards: The Road Ahead. More than a hundred slides and many comments on the discussions are available there. Our next larger presentation will be at GDC 2004, where we hope to present and discuss some concrete drafts of the standards.

Finally, if you are interested in joining the committee, we would be happy to receive your application. Please follow our membership application information on the

internet. Unluckily, the page structure of the IGDA's internet presentation is changed now and then. The safest way to locate our pages is thus to use the IGDA's main page at http://www.igda.org as entry point.

I would like to thank everyone who contributed to our committee's progress and this report. The committee is based on voluntary work and we would be nothing without the altruistic support of our members and coordinators. Given the outrageous regular workload in jobs of the game industry, this can hardly be appreciated enough. Special thanks go to Jason Della Rocca, the program director of the IGDA, who probably already has nightmares about e-mails from me coming in faster than he can reply!

I am convinced that - if we really should succeed - our interfaces will represent a giant step for game AI development and the field of artificial intelligence in general. It is certainly not an easy task, but with our great team and excellent support by the IGDA, our chances seem to be pretty good! See you at GDC 2004 for the presentation of our first interface drafts!

Alexander Nareyek
(Committee Chairman)

June 17, 2003
Pittsburgh, PA, USA

2. Interface Formats

Soon after the committee was formed, we realized that we need a forum to discuss our general approach and the way interfaces should be developed and specified. About 200 posts are posted to the forum as of now; however, related discussions sometimes also took over the forum of the working group on world interfacing. Some details on the two main topics that were discussed are given in the following.

2.1. Architectural Infrastructure

One aspect that arose now and then was what type of units we are providing interfaces for - agents vs. functions. A function library would be a standard API with hard-wired function calls. The advantage is that this is a very efficient and can be used independent of the game architecture. An agent-based approach, on the other hand, would send and receive general messages, decoding and encoding them internally. That approach is very flexible but requires an agent-based system architecture with infrastructure like service registry, message passing etc.

There were very different points of view on which approach to adopt, and we discussed the issue at our 2003 GDC roundtable. The preferences became very obvious: All participants voted for a function library, no-one for the agent-based solution, and about 15% were interested in an additional agent wrapper for the function-based approach. Thus, our work will focus mainly on function libraries.

2.2. Specification

Our interface standards must be specified somehow, and the discussions on which specification methods/languages to use quickly emerged. Nearly everyone agreed that multiple levels should be supported, e.g., abstract ontology and XML levels, as well as C/C++ interfaces. C/C++ reference implementations would also be most useful. For the exact formalisms, however, we have given the working groups a bit of freedom to experiment with what they find most useful. We will further pursue this topic once the group's interface suggestions become more concrete.

We also discussed the issue at our 2003 GDC roundtable, and especially whether C or C++ should be targeted. Interestingly, hardly anyone was interested in pure C, while more than 50% of the participants voted for a pure C++ interface. A C approach with a C++ wrapper was supported by 25%. We will consequently focus more on C++ than on C.

3. Working Group on World Interfacing

Ten game AI developers and middleware vendors out of the games and military oriented sector build the AI Interface Standards Committee's (AIISC) world interfacing group. We are the central hub, coordinating ourselves with all other groups to create standardized interfaces to interconnect the games AI, e.g., other AIISC interface implementations, with the non-AI games world representation. Christopher Reed took over the coordinator role from Michael van Lent due to his time constraints last year and is now organizing the groups' work and moderates its forum discussions.

Without any doubt, the world interfacing group is the most active discussing with over 500 forum postings. Many of our arguments circle around the question what it is exactly what is needed by a game world and AI connecting interface and how and on which level to represent it, e.g., on a very concrete level or top-down from a more abstract meta-level.

3.1. Goals for a World Interface Standard

Our primary goal is to provide a functional AI interface to any run-time simulated game world.

Important requirements to realize include:

- language independence,
- architecture independence,
- data-driven design.

3.2. The Game World

What is the game world? The group has struggled with this question from the first day of discussion to the present. The only game world component that is currently in consensus is the "entity".

The game world is composed of entities.

3.2.1. Game World Entities

Though we don't have a concrete definition, examples of entities are easy to come by. Doors, tables, chairs, weapons, ammo, bombs, grenades, missiles, clocks, and characters are all common examples of entities in games. There are many different classes of entities, and many different instances of these classes.

For example, in chess, there are six entity classes (King, Queen, Knight, Bishop, Rook, Pawn). At the beginning of the game, there are 32 entity instances (16 on each side); however, the number of instances changes as pieces are captured or promoted.

Each entity class has any number of member variables representing the state of the entity. These variables are often used for concepts like health, position, velocity, size, animation, and model. In chess, all the entities would have member variables for their positions on the board (X, Y coordinates probably). The values of these variables change for each instance when it is moved.

We are currently working on a language for building these entity classes from a standard template.

3.3. AI from a World Interface Perspective

As with the definition of the world, our group has no official definition for AI and very little consensus of what it involves. In fact, some of what follows may be too recently introduced to really be considered in the consensus, but it's here to help support the concepts that are. While the role of AI may seem intuitively obvious, the only element in consensus is that AI "plays the game".

AI is the controlling force behind all non-human players.

3.3.1. What is a Player?

Non-human AI controlled players - or "players" for short in the following text - vary drastically in shape and function from game to game, but they all have these elements in common:

Objectives

Why the player is participating.

Senses

Describe what the player can see, hear, know, and generally sense.

Actions

What the player can do.

Playing the game then involves responding to senses by doing actions that further the players objectives.

3.3.2. Objectives

There hasn't nearly been enough discussion on this topic to say much more about it than the general principle, e.g., in chess the primary objective of the game is to "Capture the enemy king".

3.3.3. Senses

We've talked in great detail about senses, how they work, and what they do. We have had some trouble trying to come up with concepts that can work in any architecture. We have established two types of senses:

Events (world push to player)

> Events occur at a time in the game and are instantly delivered to players or recorded for later perusal (or both). Events are not necessarily related to entities, though they are often caused by entities.

Queries (player pull from world)

> Queries occur when the player actively uses his available senses to examine the state of the game. Queries are also not necessarily related to entities, though they often are.

Examples for events and queries in First Person Shooters (FPS):

- an event may occur anytime a sound is played,
- a common query might be "Where is my enemy?"

Currently, the exact mechanics of sensory perception is still under debate, as it can be affected by many factors:

1. existence of the sense, e.g., in many games there is no "hearing" (what is there to "hear" in Chess?),
2. players abilities, e.g., even if "hearing" exists, not all players are able to do it, or do it as well as each other.
3. state of the world. For example, even if a player can "hear" anything, sound may not go through walls and closed doors to reach him.

3.3.4. Actions

Actions are the way a player does things in the game. There are two types of actions:

Actuators

These occur at a time in the game and are of a finite duration, e.g., in Chess, an actuator may be "Move diagonal".

Effectors

Effectors occur continuously, applied at all times to the game state. In an FPS, an effector may be "Aim at enemy".

Actions suffer similar problems as senses, not all players can do all actions, and even if a player is technically able to execute an action, the state of the world may prevent it. A further example may illustrate this: in Chess, a Bishop may be able to "Move diagonal", but if it is blocked...

3.4. Interfacing between AI and the World

Though the group mostly agrees on these concepts, the particulars of where they are defined, owned, registered, and created are still very much up in the air. Questions of threads, and synchronous vs. asynchronous architectures have yet to be worked out, although we are confident that eventually, we can make it work. The further interaction with other working groups and their progress will help to advance our work as well, providing more concrete requirements on functionality for the world interface.

3.5. Group Members

Current members of the working group on world interfacing:

- *Group coordinator:* Christopher Reed - Raven Software / Activision
- Tom Barbalet - Noble Ape
- Axel Buendia - SpirOps
- Erwin Coumans - Havok Realtime Physics
- John Morrison - MAK Technologies, Inc.
- Jeff Orkin - Monolith Productions
- Doug Poston - Conitec
- Adam Russell - Pariveda
- Duncan Suttles - Magnetar Games
- Ian Wilson - iNAGO Inc / neon.ai

4. Working Group on Steering

AI game developers, game AI middleware producers, and interested academics form the AI Interface Standards Committee's (AIISC) working group on steering. One of the eight members holds the position of a coordinator, coordinating the groups' discussions and work. Dave C. Pottinger was the first coordinator, but due to time constraints, handed the job over to the current coordinator, Thaddaeus Frogley.

While no final interface has been decided on, the group had very productive periods besides GDC'03, this years' E3, and many job-related deadlines of the members, and produced far over 200 discussion contributions. To focus discussion, examples how to apply steering techniques in games were often used. Papers of Marcin Chady and Craig W. Reynolds provided a starting point how steering systems could be architected and helped to develop a common vocabulary. Missing a precise vocabulary can lead to confusion which has been experienced in some arguments, oftentimes ending in the discovery that the different parties talked about the same thing.

4.1. Goals for a Steering Interface Standard

The goal of the steering group of the AI Interface Standards Committee (AIISC) is to define a standard interface to ease the integration of a games AI with a steering system. A steering system is a system that proposes movements for agents based on their local surrounding, i.e., using the information about the world delivered by their senses. Therefore, concrete but programming language neutral interfaces will be defined.

First, the steering group has defined what is and what isn't steering, and building upon this, identified the main abstractions and concepts needed to analyse and structure the process of steering an agent inhabiting a games world. This process was guided by different examples, e.g., a scenario similar to "capture the flag".

4.2. What is Steering? What isn't Steering?

A steering system controls and implements agent movement, paying attention to multiple goals like not to bump into walls and other agents, while pursuing or evading specific agents, or trying to stay inside a group. It is used for:

- steering agents,
- weapon targeting,
- camera control,
- particle effects,
- etc.

Steering is reactive, non-deliberative, and based on the local environment surrounding every single controlled steered entity. It needs to cope and interact with static and dynamic game world objects in an agent's perceived neighbourhood.

Steering has nothing to do with:

- searching,
- planning,
- backtracking,
- puzzle-solving,
- or anything requiring global knowledge of the game world outside the agent's sensed local neighbourhood.

A steering system cannot be expected to navigate a maze. That is the job of a pathfinder (handled by the working group on pathfinding). Once such a pathfinder/maze-solver outside of the steering system has laid out a path through the maze, the agent's steering system can produce natural-looking motion along this path.

4.3. Abstract Concepts and Components of a Steering System

Problems to solve defining a steering system to use in games are:

- integrate multiple, eventually conflicting steering goals,
- react to (potential) collisions with obstacles, hot/proximity regions, or other agents,
- translate the steering into agent movement in the game world,
- integrating the steering system with other game engine components, e.g., physical simulations,
- allowing for game-specific extensions of the steering system,
- and so on.

Different examples were examined, like the already mentioned "capture the flag", where one agent tries to reach the flag while avoiding obstacles and its hunters. The hunting or enemy agents pursue the flag-seeker while avoiding collisions with other hunters and obstacles. The scenario represented only a small part of "capture the flag" but was and is very helpful in thinking about steering.

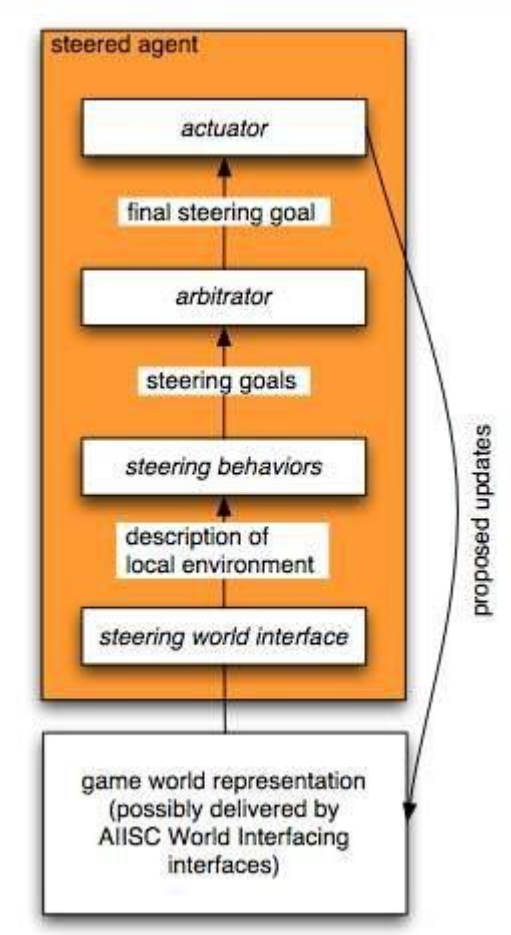
These examples gave an impression of the steering task's complexity and the need to make it simple to combine steering with other AI techniques, which is an important issue to keep in mind for potential standards.

The way chosen to solve these requirements was to create the steering interface as a "grey box", not a black box. The interface user - the game AI developer - should be able to change and adapt a system conforming to the interface standards to his

individual needs while using an established steering foundation that is structured in a way to keep the interface as simple as possible. To meet these needs, the (steering) data to work on is standardized - not the algorithms that use it.

The steering interface standard will define a toolkit of steering-related objects that are glued together through custom code written by the interface user to meet their projects' needs. This simplifies the task of specifying the interface by putting some burden on the end user but won't impede what game developers want to do.

Analysing the steering domain, the following concepts and components describing a steered agent have been identified:



Game world interface

Enables an agent to sense its local environment, e.g., for preventing collisions or for retrieving information about the direction that another agent is facing in to approach it from behind.

Steering behaviours

They are reacting to the surrounding scene provided by the world interface by generating steering goals. Which steering behaviours to use is decided by the game developer. He can also provide his own behaviours. At run-time, the arbitrator dynamically manages the steering behaviours provided by the developer.

Arbitrators

An arbitrator integrates and combines the steering goals produced by the steering behaviours to create the final steering goal intended for the actuator of the agent. The interface user determines which, when and how to use steering behaviours and their outputs by providing his own agent steering arbitrator(s).

Internal information currency

The representation of the steering goals generated by the steering behaviours and the arbitrators. Arbitrators and actuators receive and work on steering goals.

Actuators

> Actuators convert a final steering goal into whatever is needed to propose agent movement to the game.

Steering behaviours react to continuous changes of the environment. Furthermore, the whole steering system of an agent can be controlled by the game logic, e.g., through dynamically changing the used steering behaviours, arbitrator(s) and the actuator, or by modifying the states of the different components of an agent's steering system.

4.3.1. World Interface

Changes in the world state are propagated via the world interface to the steering system. Every steered agent and its different steering behaviours perceive (agent and steering behaviour) relevant entities in their local neighbourhood. A representation of the world is needed that allows defining agents, their properties and the entities building the environment. Especially the detection of collisions or of the possibility of collisions must be provided by the world interface.

Typically, agents hold information about their bounding volume, position, mass, linear and angular velocities, linear and angular accelerations, etc. The world is represented through geometric primitives like circular or spherical bounds, convex hulls, lines, or planes. This set of primitives should be extendable by user provided primitives. Other agents and groups of them, obstacles, walls, vector flow fields, paths, and so on, could influence an agent's steering.

Collision detection and testing interface is needed, for example, to find nearby objects or objects that lie on a ray between two points, or to get the time of a potential collision if an agent is moving further in its current direction. This interface - a hot working group discussion topic - might be provided by the AIISC world interfacing group. What to do with this information is decided by the steering behaviours.

4.3.2. Steering Behaviours

Steering behaviours are the key elements of a steering system. Typical behaviours are:

- seek/flee (for static targets or threats),
- pursue/evade (for dynamic targets like another agent),
- obstacle avoidance,
- wandering around,
- arrival,
- wall following,
- path following,

- flow-field following, and so on.

Combined steering behaviours are for example leader following, queuing and flocking.

They might be fed with the data of the agent they are belonging to and information about their surrounding environment, or they might query the world interface by themselves to get the data needed to do their job. In only working on their local world data, steering behaviours are kind of "myopic".

By choosing the right set of steering behaviours, a game developer might be able to achieve an emergent system behaviour that helps to avoid situations that need backtracking or planning otherwise and would need to be solved by game components other than the steering system.

Based on their sensed world neighbourhood and their agent's state, steering behaviours produce "steering goals" consumed by arbitrators. Steering goals generated by behaviours can potentially also be used as input or hints for other steering behaviours. Steering goals express a behaviour-specific recommendation how to move or change the movement of an agent.

4.3.3. Arbitrators

Arbitrators take the steering goals generated by the steering behaviours they are connected to, handle conflicting steering goals and integrate them into the "final steering goal" dedicated to the agent's actuator.

Because establishing a complex "doing-everything-ever-needed-for-arbitration" solution doesn't seem feasible, the interface user should be able to plug in his own arbitrator using a combination of a variety of provided or self-written arbitration strategies. Plug-ins allow for different arbitration for different steering problems or for different arbitration inside one single steering problem.

Typical approaches of arbitration include:

- (linear) weighting and blending of steering behaviours steering goals,
- discrete selection of just one steering goal,
- prioritizing goals,
- randomly selecting one steering goal,

or more complicated game-specific mechanisms.

The AIISC steering interface standardizes arbitration methods not by standardizing mechanisms but by providing prototypes for integrators/arbitrators, standardizing not the code itself but the data - steering goals - over which arbitration may work.

4.3.4. Internal Information Currency

Steering behaviours, arbitrators and actuators communicate via steering goals - the internal information currency of the steering system. Steering goals encode agent movement or movement-changes as recommended by the generating behaviours or arbitrators. Their value type should be standardized and might be a vector-pair with a mechanism to mark a vector as unnecessary, so arbitration or the actuator won't care about it.

The decision to have a steering goal hold two vectors is mainly backed up by two vectors being a super-set of just one, performance-optimization reasons, and the need to specify the direction of motion separately from the view direction of an agent in a 3D world. The vectors building a steering goal can also be seen as values defining the linear and angular velocity by which the producing steering behaviour wants the associated agent to move with.

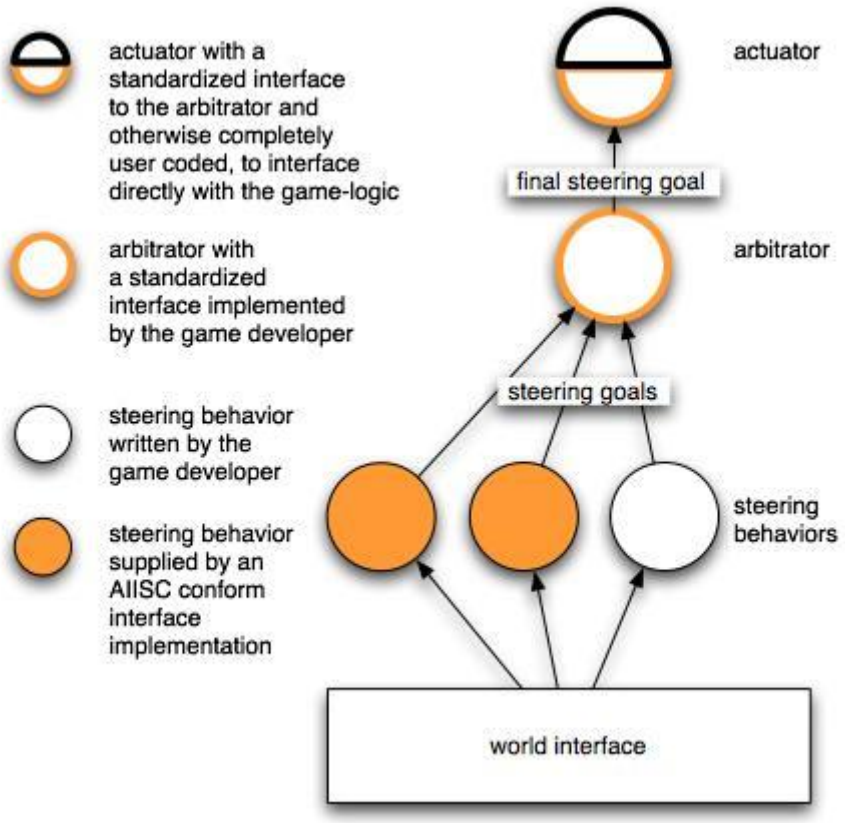
However, until now no real consensus concerning the steering goal vector semantics has been reached.

4.3.5. Actuators

The actuator of an agent takes the final steering goal delivered by an arbitrator. Steering goals are a steering-system specific definition of movement that needn't conform directly with the game-logic way of expressing entity or agent motion. Therefore, the final steering goal is converted into a proposal for the game how the agent should move, e.g., through linear and angular accelerations. It is up to the game to decide what to do with the actuators output.

To allow for direct interfacing with the application and to create whatever is needed by the game-logic coordinating the movement, e.g., a physical simulation component, it is the game AI programmer who provides the actuator and its interfacing with the non-steering systems of the game. The actuator offers a user defined agent- or game-specific meaning to the final steering goal.

Possible outputs of the actuator might be: forces, impulses, velocities, accelerations, displacement vectors or direct movement, that meet the games’ needs.



4.4. State of Work

Currently, the group is in a phase of brainstorming, collecting and discussing high-level concepts. Nothing is implementation-specific or set in stone yet. Terms like "steering behaviour" or "actuator" are just concepts and no classes or objects of a programming language.

The litmus test of every steering interface will be test case implementations and its use in real projects.

4.5. Group Members

Current members of the working group on steering:

- *Group coordinator:* Thaddaeus Frogley - Rockstar Vienna
- Marcin Chady - Mindlathe
- Philippe Codognet - University of Paris 6
- Mike Ducker - Lionhead Studios
- Leon C. Glover - Entropy Unlimited
- Daniel Kudenko - University of York, UK
- Craig W. Reynolds - Sony Computer Entertainment US R&D
- Adam Russell - Pariveda

4.6. Resource(s)

See References section.

5. Working Group on Pathfinding

The working group on pathfinding is combined out of eight members. Noel S. Stephens coordinates the group, which is composed of various game AI developers, middleware producers, academics and educators. He took over this role from Eric Dybsand (who was the successor of Ian Frank). We posted 224 discussion contributions to our forum so far, arguing about the best way to define an interface for pathfinding.

5.1. Goals for Pathfinding Standards

Our preliminary focus/goal is to define the core terminology and techniques associated with commonly used pathfinding algorithms and data structures. Currently, our group is focusing on gathering this past year's collective work, documenting it, and putting application towards the theories discussed.

5.2. Common Terminology

AI

> For the purposes of this group's documents, when the term "AI" is used by itself, it refers specifically to the pathfinding algorithm or agent.

Agent

> An entity in a simulation. For the purposes of this group, an agent is the entity (be it a character, vehicle, or whatever) which is attempting to pathfind.

Connection

> A path between two nodes. Every node must have one or more connections to another node in order to exist in the graph.

Explicit Graph

> An explicit graph is one in which the graph is held in a data structure that encapsulates the graph structure.

Graph

> A representation of a terrain model made up of an arbitrary mesh of nodes and connections.

Grid

> A representation of the terrain in which all of the nodes have a fixed number of connections in fixed directions.

Implicit Graph

An implicit graph is one in which the graph's structure is exposed through an interface. It is more representationally powerful than an explicit graph.

Node

An atomic entry in the navigational graph. When the AI navigates to a node, its destination is considered to be the centre of the node.

Pathfinding

This is the process of searching for waypoints that form a path, which can be used by agents as a guide for movement through terrain and obstacles within a virtual world.

Region

A collection of nodes. The region is a collection of "large scale" navigational data such that the AI can use this to determine a larger path it needs to cover or not cover in order to reach its destination.

5.3. High-Level Specification

The pathfinding group has been discussing several concepts in regard to storing and retrieving data pertinent to that of the typical pathfinding algorithms. Thus far, we have come up with a Graph, Node, Connection concept.

The Graph is the underlying container/descriptor for any given region.

Nodes are the sub-sections that break the graph into regions that are more manageable. The nodes will hold navigational related information that the agents can use to determine the most optimal path. The weighted information held within the node is dynamically assigned to the node and directly affects the weight/cost of the node.

Each node will be connected to another node to form a grid of nodes that the agents can parse through in order to formulate the most optimal path. The connections will also have a weighted value assigned to them and thus will have the ability to have dynamically assigned navigational data members associated with them.

There will be the ability within both nodes and connections to register callback methods that will be able to handle/over-ride weighted calculations. This allows agents the ability to control how they view the Graph, Node, connection sets uniquely to the agent itself. The idea is to have a series of container classes that have the ability to have data members stored within them which defines the container

class itself in regards to pathfinding algorithms as well as have the ability to over-ride fundamental pathfinding algorithms that will enable for a more flexible system.

As of recently, the pathfinding group has defined the first stage to achieve the goal for the before mentioned pathfinding concepts. These three sub-groups within the pathfinding group are:

- Terminology and Theory Documentation Group
- Graph, Node, Connection Set Class Construction Group
- Navigational Mesh File I/O Group

5.3.1. Terminology and Theory Documentation Group

The members of this group are responsible for documenting both terms and concepts discussed throughout the pathfinding forums. The end result for this group is to create a nicely formatted document that contains terms commonly used within the discussions (with definitions) and documentation that the laymen mind might not be familiar with as well as concepts discussed through the forums that the pathfinding group agrees are pertinent to the standard said group is constructing.

This group should query the two other groups for any new terminology or concepts. This group should not feel like they are liable in coming up with the terminology, but rather will act as a conduit to organize and prepare the information discussed in the pathfinding forums.

Members of the group:

- *Lead:* Miranda Paugh
- *Additional support:* Ian Frank

5.3.2. Graph, Node, Connection Set Class Construction Group

The members of this group are assigned the task to construct a series of classes that correspond to the Graph, Node, Connection topology and theory discussed thus far in the pathfinding group. The first stage of this task is to construct a simple Windows application that will allow a user to construct a graph, create nodes within the graph, and then create connections between the nodes.

The classes should be pure to their purpose and thus should not contain any elements that require any Windows API support. Any external library support should be contained in classes outside of the Graph, Node, Connection classes. The result should be a series of generic classes that can be ported to any platform with little effort.

Members of the Group:

- *Lead:* Mike Ducker
- *Additional support:* Ian Millington and Syrus Mesdaghi

5.3.3. Navigational Mesh File I/O Group

The members of this group are assigned the task to construct a set of classes/libraries that can be used to read in a navigational mesh file from a specific format. The data that gets read in should have a fairly simple set of access methods that will enable access to the data read in from the file.

The members of this group need to keep in mind that there needs to be a layer between the file I/O methods and the actual code/classes that handle storing and retrieving the data. This will allow for portability later in the year. A good starting example of this type of code could almost be borrowed directly from Game Programming Gems 1 section 3.6 "Simplified 3D Movement and Pathfinding Using Navigational Meshes".

Members of the Group:

- *Lead:* Stephane Maruejouls
- *Additional support:* Eric Dybsand

5.4. Design Decisions

Thus far, we have come up with the agreement that pathfinding is very unique based on the game genre and demands. In order to provide the most flexible pathfinding standard we will need to create class containers that can be defined externally yet have methods that allow external systems to configure them for the specific pathfinding requirements needed at the time. We have felt that there is no universal means to pathfinding, yet there are similar traits and data sets that can be configured/assigned to allow for typical internal algorithm (i.e. A*) use.

We have agreed that such a task will require refinement of the concepts, and before we come up with an entire system, we need to come up with sub-systems first. Now that we have a well-defined theory foundation, we have broken our group into sub-groups with the assignments of either tracking/documenting new concepts/terminology, creating a very fundamental Graph, Node, Connection class set, and finally coming up with a means to testing our Graph, Node, Connection class set (Navigational Mesh I/O).

It has been agreed upon to test our current theories through common application before seeking further analysis or further depth to our suggested theories. Upon completion of our first application stage, we will analyse our implementation and

discuss how we can better improve the fundamental concepts used as well as the interface to said concepts.

5.5. Group Members

Current members of the working group on pathfinding:

- *Group coordinator:* Noel Stephens - Atari Games (Paradigm Division)
- Mike Ducker - Lionhead Studios
- Eric Dybsand - Glacier Edge Technology
- Ian Frank - Future University-Hakodate
- Stephane Maruejouls - MASA Group
- Syrus Mesdaghi - Full Sail Real World Education
- Ian Millington - Mindlathe
- Miranda Paugh - Magnetar Games

6. Working Group on Finite State Machines

To create standards interfaces for game related finite state machines - that is the mission of the ten members of this AI Interface Standards Committee's (AIISC) working group. Professions include game AI and non-game AI developers, academics, consultants and AI middleware vendors. Nick Porcino manages the work of the group in the role of its coordinator. Until now, we contributed around 180 postings to our forum.

6.1. Goals for a Finite State Machines Interface Standard

The primary goal of the finite state machines (FSM) group is to define a standard interface that facilitates integration of finite state machines with game engines. To that end, our major goals have been the following:

1. identify how FSMs are commonly embedded in games,
2. define a common language to describe finite state machines,
3. define an XML description of FSMs for interoperability between tools,
4. define a C++ API whereby FSMs can be efficiently and easily interfaced with other systems.

6.2. FSM Committee Motivation

Finite state machines are arguably the most popular technology in game AI programming today. They are conceptually simple, efficient, easily extensible, and yet powerful enough to handle a wide variety of situations.

The finite-state machine (or finite-state automaton) exists in its simplest form from computational theory, defined as a set of states *S*, an input vocabulary *I*, and a transition function *T(s,i)* mapping a state and an input to another state. The machine has a single state designated as the start state, where execution begins, and zero or more accepting states, where execution terminates.

Less abstractly, an FSM is a concise, non-linear description of how the state of an object can change over time, possibly in response to events in its environment. The implementation of FSMs in games always differs from the theoretical definition. Some code is associated with each state so that as the object's state changes, its behaviour changes accordingly. Moreover, the transition function is broken up and its internal logic distributed among the states so that each state "knows" the conditions under which it should transition to a different state.

FSMs are often depicted graphically using flowchart-like diagrams in which states and transitions are respectively drawn as rectangles and arrows. Graphical

representations of FSMs are popular, so popular that the Unified Modelling Language (UML) reserves one of its nine diagram types just for state machines.

Because game character behaviour can be modelled (in most cases) as a sequence of different character "mental states" - where change in state is driven by the actions of the player or other characters, or possibly some feature of the game world - game programmers often find that finite state machines are a natural choice for defining character AI. In an FSM-based behaviour, the states describe how the character will act, and the transitions between states represent the "decisions" that the character makes about what it should do next. This "decision-action" model has the advantage of being straightforward enough to appeal to the non-programmers on the game development team (such as level designers), yet still impressively powerful. FSMs lend themselves to being quickly sketched out during design and prototyping, and even better, they can be easily and efficiently implemented.

The FSM committee seeks to define a standard that captures the simple yet powerful way in which FSMs can be put to use.

6.3. FSM Committee Progress

So far, the committee has worked on the first two phases, exploring how game developers use FSMs, and reaching a consensus on a common language to describe an FSM.

6.3.1. FSMs in Practice

Early on, we examined the ways in which FSMs were implemented in practice. We determined that two common methods were "flat" and "hierarchical". By "flat" we mean the standard way in which FSMs are thought of - as atomic states and transitions between them. A "hierarchical" (or "nested") state refers to states that encapsulate other FSMs. Thus, being in a "hierarchical state" really amounts to being in a set of atomic states.

"Inherited" states are object-oriented implementations of states. This type of state inherits all the functionality (and possibly transitions) of a pre-existing state but can change its behaviour.

Another implementation distinction we uncovered was "polling" versus "event-driven." Polling implementations have FSMs actively query states of the world when executing transition logic. Event-driven approaches, by contrast, wait for the game engine to signal some event. This event will drive state execution and transition logic. Event-driven approaches are most efficient and have been the current focus.

6.3.2. Representing FSMs

As it turns out, there are already a number of existing pieces of work that implement or depict finite state machines. For example, SourceForge hosts a number of FSM projects, such as the generically-named Finite State Machine, a State Machine Compiler, a Qt-based FSM called Qfsm, a Finite State Machine Language, and our dear leader's project Fizzim.

In terms of authoring and depicting FSMs, we examined a number of graphical methods, such as GraphViz and an emerging graph standard called GXL.

6.4. State of Work

Currently the FSM group is in a stage of discussing requirements on a common language to describe an FSM. From there, we will prescribe an XML-based ontology and game engine interface.

6.5. Group Members

Current members of the working group on finite state machines:

- *Group coordinator:* Nick Porcino - LucasArts Entertainment
- Daniele Benegiamo - AI42
- Sam Calis - Universal Interactive
- Scott Davis - Black Cactus Games
- Daniel Fu - Stottler Henke Associates
- Mark Gagner - WMS Gaming
- Ben Geisler - Raven Software / Activision
- Dave Kerr - Naturally Intelligent
- Linwood H. Taylor - University of Pittsburgh
- Darren Ward - Bits Studios

This section was written with the help of Ryan Houlette of Stottler Henke Associates.

7. Working Group on Rule-based Systems

In this report, we introduce the current work carried-out by the Artificial Intelligence Interface Standards Committee (AIISC) working group on Rule-based Systems (RBS).

Due to work-commitment of most of the members including the coordinator, the group has started very slowly. Many points of discussion have been started without having been concluded or reached any agreement. To allow a better progression of the group toward its identified goal, a more clear strategy and distribution of tasks will have to be defined, with each member of the group responsible of a specific task. This scheme will be in place and monitored in the forthcoming weeks.

For now, we present some of the background concepts and issues discussed on the aspects of specification, design and development of a Rule-based System for game AI to implement challenging game agents (i.e. NPC). We are investigating the components and possible architectures of RBS, and the different applications of game AI in general and of RBS in particular, depending on game genres.

This is a preliminary report on the current work of the RBS group.

7.1. Goal

Programming game AI is one of the most challenging enhancements that a game developer can implement. The real-time performance requirements of computer game AI and the demand for human-like interactions, appropriate animation sequences, and internal state simulations for populations of scripted agents have impressively demonstrated the potential of academic AI research and game AI technologies.

Many academic AI techniques have been used and are currently used to implement game AI. These include, without being exhaustive: *finite state machines* (FSM), *decision trees* (DT), *pathfinding*, *steering*, *goal-oriented action planning* and *rule-based systems* (RBS).

Regardless of their differences, many concepts from academic AI can be implemented in games.

To be believable, the AI in a game must simulate cognition, sense the environment realistically, and act convincingly within that context. In defining game AI, the programmer will have to code agent activity and behaviour so that characters appear intelligent and respond realistically to perceived conditions and situations.

The aims of these pages are to introduce the work carried-out by the Artificial Intelligence Interface Standard Committee (AIISC) working group on Rule-Based Systems (RBS).

Rules Based Systems are comprised of a database of associated rules. Rules are conditional program statements with consequent actions that are performed if the specified conditions are satisfied.

The aims of the group are to discuss and develop a set of standards on RBS applications and architectures suitable to games. This is a preliminary report on the current work of the RBS group, and which will be followed-up by recommendation and a proposal for a game-AI RBS standard in future reports.

7.2. Games Genres and AI

Many game genres have been created, and while they might be different in the aims, the gameplay and the setting, they share some commonalities in term of game AI.

7.2.1. Action Games

Action games involve the player in the exploration of fantasy worlds in the form of running, jumping, climbing or leaping with the goal of discovering the doorway or exit into the next stage or level. Generally, these titles feature cute characters battling a cast of "baddies" and often involve a simple plot such as rescuing a princess. Platform games like Donkey Kong as well as maze games such as Pac-Man fall into this category. Another game genre that can be considered as action games are shooters.

In action game genre, it is in creating intelligent opponents that the most obvious possibilities for integration of sophisticated AI arise. NPCs cooperating with the player character (PC), is another area in which there is a real opportunity for further application of sophisticated AI such as RBSs.

7.2.2. Adventure Games

Adventure games take players on a journey in which they visit strange lands, find keys to unlock mysterious doors, and often they must gather inventory such as keys and sometimes weapons to solve puzzles. The combination of certain inventory items and clues received along the journey are the keys to moving on to hidden areas to discover more mysteries. An overall story, sometimes of epic proportion, or comedic intent is the drive of each small task.

Two interesting applications of AI and RBS to the genre are the creation of more realistic and engaging NPCs and maintaining consistency in dynamic storylines.

7.2.3. Role Playing Games

Role-Playing games require players to take on the role of a person or group of people. While role-playing is generally associated with sword and sorcery and fantasy, it can take place in any setting or time. These games typically send players on a journey where they can interact with other characters and attain new skills and abilities, often by fighting battles.

The differences between RPGs and adventure games arise from the scope involved. RPGs take place in far larger worlds and the player has more freedom to explore the environment at their own pace. Also, underlying RPGs is some rule set stemming from the original, and quite complex, Dungeons & Dragons rules.

The RPG format offers the same kind of challenges to the AI developer as the adventure game.

7.2.4. Strategy Games

Strategy games require the player to take on a leadership role (general, king, god-like figure, etc.) and oversee every detail of the provided scenario(s). Generally, strategy games require the user to move and to deploy troops or units, to manage resources and to attain set goals.

Two distinct classes of game have emerged from the strategy genre. Turn based strategy (TBS) games involve each player taking their turn to move units, order production, mount attacks and so on, one after another. The Civilization (www.civ3.com) series is the definitive example of this kind of game. Real time strategy (RTS) games, as the title suggests, take place in real-time with players moving units, ordering production etc. in parallel. The Age of Empires (www.ensemblestudios.com/) and Command & Conquer (westwood.ea.com) series, along with Total Annihilation (www.cavedog.com), stand out as good examples of this genre.

AI in strategy games needs to be applied both at the level of strategic opponents and at the level of individual units. AI at the strategic level involves the creation of computer opponents capable of mounting ordered, cohesive, well-planned and innovative campaigns against the human player. At the unit level, AI is required in order to allow a player's units to carry out the player's orders as accurately as possible. Challenges at unit level include accurate path finding and allowing units a degree of autonomy in order to be able to behave sensibly without the player's direct control.

In RPGs or RTSGs, rules can be used to program the behaviour of your own PCs or units. For example, in Baldur's Gate you can specify the behaviour of the characters in your group using a scripting language. This would have been easier using rules.

Rules can be used also to specify scenarios. For example, in most RTS games map editors, you can describe a scenario by specifying rules applied to a map section (if a number of units enter this section, then trigger this action).

7.2.5. Game-AI Behaviour

The sections above show some of the possible application for AI in game in general and RBS in particular. These are the creation of interesting opponents, realistic and engaging NPCs and maintaining consistency in dynamic storylines.

To help in making the game immersive and allow suspension of disbelief, the creation of the NPCs must provide different types of behaviour. Two categories of generic behaviour in NPCs are commonly used: reactionary and spontaneous.

NPCs behave in a reactionary manner whenever they are responding to a change in their environment. If an enemy spots you and starts to run towards you and shoot, then they have acted as a reaction to seeing you.

NPCs behave in a spontaneous manner when they perform an action that is not based on any change in their environment. A NPC that decides to move from his standing guard post to a walking sentry around the base has made a spontaneous action.

For more details Please see: Howland G, A Practical Guide To Building A Complete Game AI, 1999.

7.2.6. SOAR-BOT Example

Example: SOAR-Quake, courtesy of J. Laird.

```
IF
        enemy visible and my health is < very-low-health-value (20%)
OR
        his weapon is much better than mine
THEN
        propose retreat
```

7.3. RBS Definitions

7.3.1. Introduction

One form of AI that can be used is a *rule-based system*.

Rule-based systems differ from standard procedural or object-oriented programs in that there is no clear order in which code executes. Instead, the knowledge of the expert is captured in a set of *rules*, each of which encodes a small piece of the expert's knowledge.

Each rule has a left-hand side and a ride hand side. The left-hand side contains information about certain facts and *objects* which must be true in order for the rule to potentially fire (that is, execute).

Any rules whose left-hand sides match in this manner at a given time are placed on an *agenda*. One of the rules on the agenda is picked (there is no way of predicting which one), and its right-hand side is executed, and then it is removed from the agenda. The agenda is then updated (generally using a special algorithm called the *RETE algorithm*), and a new rule is picked to execute. This continues until there are no more rules on the agenda.

7.3.2. Rule Based Systems Components

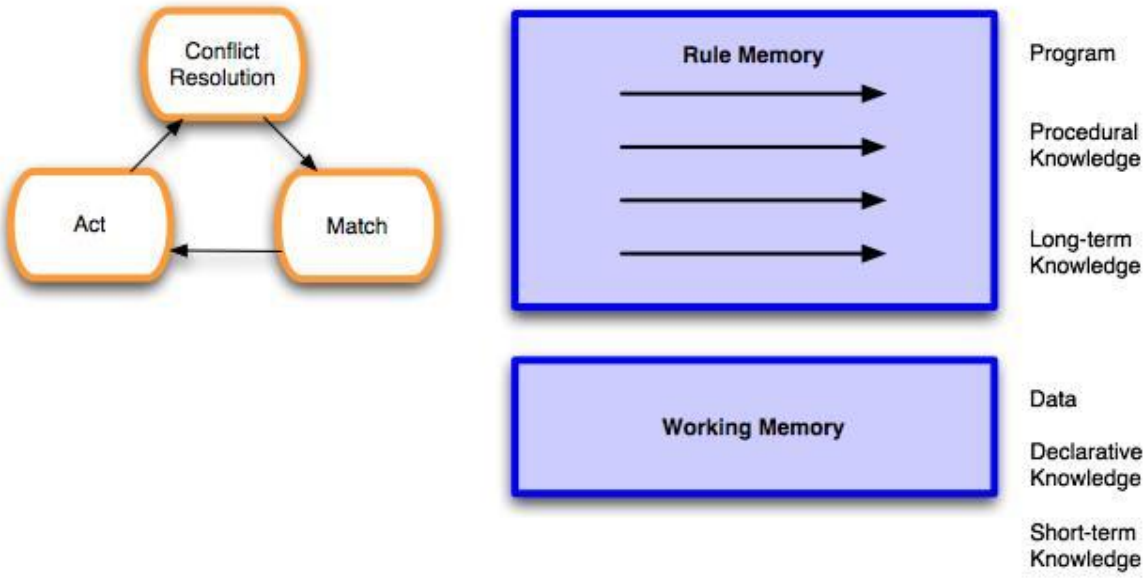
Rule-based systems consist of a set of rules, a working memory and an inference engine. The rules encode domain knowledge as simple condition-action pairs. The working memory initially represents the input to the system, but the actions that occur when rules are fired can cause the state of working memory to change. The inference engine must have a conflict resolution strategy to handle cases where more than one rule is eligible to fire.

A rule-based system consists of:

- a set of rules,
- working memory that stores temporary data,
- inference engine.

The inference mechanisms that can be used by inference engines are:

- Backward Chaining:
  - To determine if a decision should be made, work backwards looking for justifications for the decision.
  - Eventually, a decision must be justified by facts.
- Forward Chaining
  - Given some facts, work forward through inference net.
  - Discovers what conclusions can be derived from data.



7.3.3. Extensions to Rule-based Systems

Rule-based systems support formalisms with different level of expressiveness. Examples of these include:

- propositional logic,
- first-order logic,
- events and temporal constraints,
- probability associated with rules,
- fuzzy logic,
- etc.

All of these can be used to provide better AIs, e.g., you can imagine a bot that hides when it is being shot at. Then it waits for five seconds before trying to shoot back if there is no other shooting and no incoming noise.

7.3.4. RETE Algorithm

A possible inference engine is the RETE Algorithm. The RETE Algorithm is widely recognized as by far the most efficient algorithm for the implementation of rule-based systems. It is the only algorithm whose efficiency is asymptotically independent of the number of rules. Although a number algorithms implementing

production rules have been considered, based on actual, empirical evidence, the RETE Algorithm is orders of magnitude faster than all published algorithms with the exception of TREAT algorithm. RETE is usually several times faster than TREAT for small numbers of rules with RETE's performance becoming increasingly dominant as the number of rules increases.

The typical RBS has a fixed set of rules while the knowledge base changes continuously. However, it is an empirical fact that, in most RBSs, much of the knowledge base is also fairly fixed from one rule operation to the next. Although new facts arrive and old ones are removed at all times, the percentage of facts that change per unit time is generally fairly small. For this reason, the obvious implementation for an RBS architecture is very inefficient. The obvious implementation would be to keep a list of the rules and continuously cycle through the list, checking each one's left-hand-side (LHS) against the knowledge base and executing the right-hand-side (RHS) of any rules that apply. This is inefficient because most of the tests made on each cycle will have the same results as on the previous iteration. However, since the knowledge base is stable, most of the tests will be repeated. You might call this the *rules finding facts* approach and its computational complexity is exponential.

A very efficient method known is the RETE algorithm. It became the basis for a whole generation of fast expert system shells: OPS5, its descendant ART, RETE++, CLIPS, JESS, and ILOG-Rules.

For more information on RETE see:

- Forgy, C. L., "RETE: A fast algorithm for the many pattern/many object pattern match problem". *Artificial Intelligence*, 19(1) 1982, pp. 17-37.
- Giarratano and Riley, *Expert Systems: Principles and Programming, Second Edition*, PWS Publishing, Boston, 1993.

7.4. Specification

The aim is to propose a general game AI engine organized around the components mentioned in the RBS definitions section. This should make the implementation of NPCs easier by providing a suitable interface with the game-world, a common inference engine and different knowledge base suitable for a large variety of games.

A summary of RBS components is below with possible choices:

7.4.1. Knowledge Representation

- Production Rules
- Frames
- Object Oriented Representation

7.4.2. Inference Algorithm

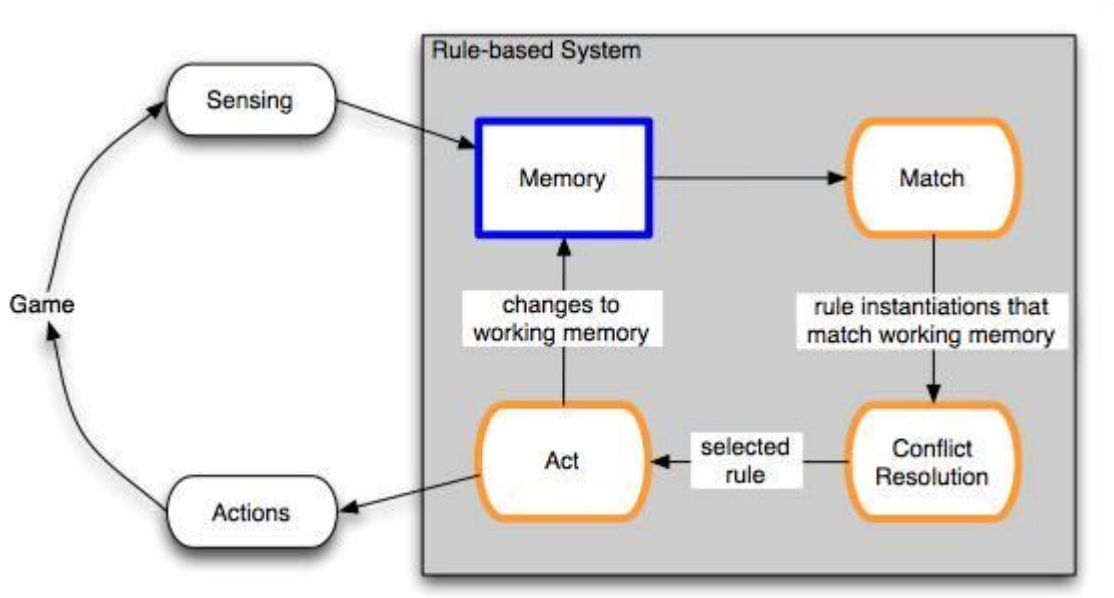
- RETE
- TREAT

7.4.3. System Architecture

- Centralized
- Multi-Agent
- Blackboard

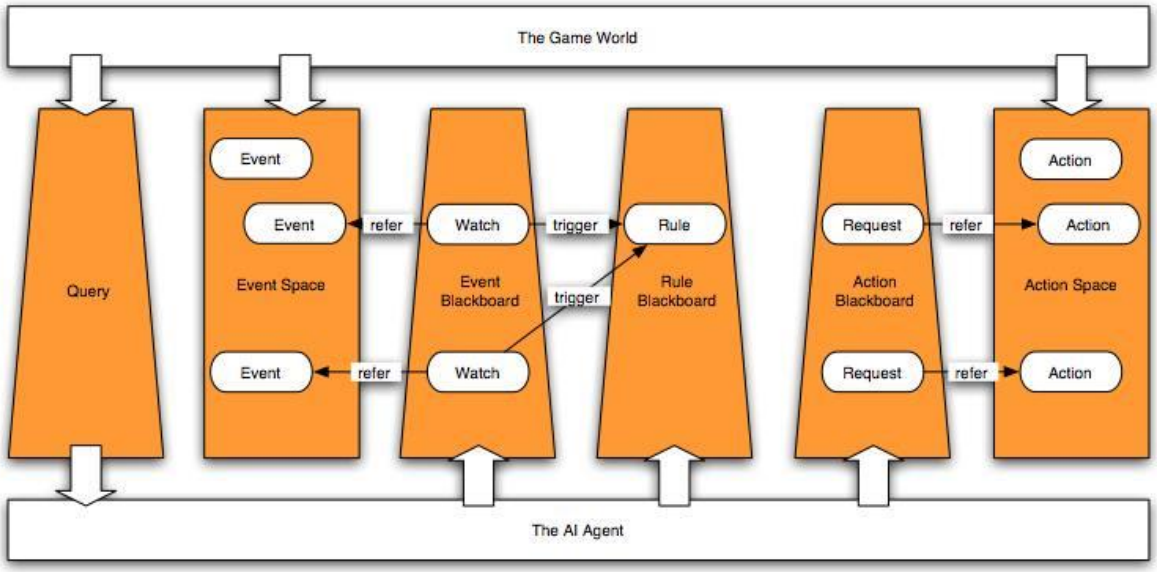
7.4.4. RBS Game AI Cycle

The diagram below shows the "thinking" process.



7.4.5. Interface with Game World

The following architecture shows an example how an RBS could be integrated. Note that the architecture is of course very dependent on what the final world interface will look like.



7.5. Group Members

Current members of the working group on rule-based systems:

- *Group coordinator:* Abdennour El Rhalibi - Liverpool John Moores University
- Jean-Louis Ardoint - ILOG
- Daniele Benegiamo - AI42
- Nathan Combs - BBN Technologies
- Hannibal Ding - Interserv Information Technique
- Clay Dreslough - Sports Mogul
- Frank Hunter - Adanac Command Studies
- Gerard Lawlor - Kapooki Games
- John Mancine - Human Head Studios
- Miranda Paugh - Magnetar Games

7.6. WG Appendix A: Concepts and Terminology

Blackboard architecture

A Blackboard Architecture is an AI solution where Knowledge for a domain is shared between numerous KS (Knowledge Sources). Each KS represents an expert bringing its own set of knowledge to the blackboard and uses the knowledge published through the blackboard to build assumptions, make deductions etc.

Condition-action rule

A condition-action rule, also called a production or production rule, is a rule of the form:

if condition then action.

The condition may be a compound one using connectives like *and*, *or*, and *not*. The action, too, may be compound. The action can affect the value of working memory variables, or take some real-world action, or potentially do other things, including stopping the production system. See also inference engine.

Conflict resolution

Conflict resolution in a forward-chaining inference engine decides which of several rules that could be fired (because their condition part matches the contents of working memory should actually be fired.

Conflict resolution proceeds by sorting the rules into some order, and then using the rule that is first in that particular ordering. There are quite a number of possible orderings that could be used.

Frames

Frames are a knowledge representation technique. They resemble an extended form of record (as in Pascal and Modula-2) or struct (using C terminology) or class (in Java) in that they have a number of slots which are like fields in a record or struct, or variable in a class. Unlike a record/struct/class, it is possible to add slots to a frame dynamically (i.e. while the program is executing) and the contents of the slot need not be a simple value. There may be a demon present to help compute a value for the slot.

Demons in frames differ from methods in a Java class in that a demon is associated with a particular slot, whereas a Java method is not so linked to a particular variable.

Heuristic

A heuristic is a fancy name for a "rule of thumb" - a rule or approach that doesn't always work or doesn't always produce completely optimal results, but which goes some way towards solving a particularly difficult problem for which no optimal or perfect solution is available.

Inference engine

A rule-based system requires some kind of program to manipulate the rules - for example to decide which ones are ready to fire (i.e., which ones have conditions that match the contents of working memory). The program that does this is called an inference engine, because in many rule-based systems, the task of the system is to infer something, e.g. a diagnosis, from the data using the rules. See also match-resolve-act cycle.

Knowledge base

Collection of the data and rules that suitably represent the problem domain.

Match-Resolve-Act cycle

The match-resolve-act cycle is the algorithm performed by a forward-chaining inference engine. It can be expressed as follows:

loop

1. match all condition parts of condition-action rules against working memory and collect all the rules that match;
2. if more than one match, resolve which to use;
3. perform the action for the chosen rule until action is STOP or no conditions match.

Step 2 is called "conflict resolution". There are a number of conflict resolution strategies.

RETE

Algorithm used to optimize forward chaining inference engines by optimizing time involved in recomputing a conflict set once a rule is fired.

Rule-based system

A rule-based system is one based on condition-action rules.

Search

Search is a prevalent metaphor in artificial intelligence. Many types of problems that do not immediately present themselves as requiring search can be transformed into search problems. An example is problem solving, which can be viewed in many cases as search a state space, using operators to move from one state to the next.

Particular kinds of search are breadth-first search, depth-first search, and best-first search.

Working memory

The working memory of a rule-based system is a store of information used by the system to decide which of the condition-action rules is able to be fired. The contents of the working memory when the system was started up would normally include the input data - e.g. the patient's symptoms and signs in the case of a medical diagnosis system. Subsequently, the working memory might be used to store intermediate conclusions and any other information inferred by the system from the data (using the condition-action rules).

7.7. WG Appendix B: Links and References

See References section.

8. Working Group on Goal-oriented Action Planning

This group has been slow to coordinate activities, due to heavy work-load commitments of its former coordinator, Ruth Aylett, and of its recently appointed new coordinator, Derek Long. This report summarises the most important elements of the discussion that has been pursued so far and outlines the directions that will be explored in the medium term.

8.1. What is Planning?

Most of the discussion of the group has centred on the role of action planning in games, and the problem of defining terms. *Planning*, *actions* and *goals* are all terms that are used by different people in different ways within the group, resulting from the diversity of backgrounds from game developers to academics. Furthermore, the difference between action planning and other techniques for action-selection like FSMs, or decision trees, was not always clear. General agreement has been reached that goals are conditions that an agent desires to bring about and that actions are the means that agents have to achieve these goals. Planning is the problem of assembling a coherent programme of actions to achieve specified goals. In contrast to that, rule-based systems, finite state machines, or decision trees represent hard-wired sets of actions and correspond more to reactive, fixed plans to deal with specific situations. Common drawbacks of these techniques are missing 'individualism', too shallow goal hierarchies, repetitive failures, or other obvious FSM-like behaviours. The advantage of goal-oriented action planning over these techniques is the ability to provide more flexible and more diverse behaviours for NPCs, since plans are constructed 'online' from atomic actions and adapted specifically to the current goal.

There are still important open questions in this overall framework: a planner might need to work with abstractions of actions if it is to produce a plan that is robust to possible changes in the world and fast enough to construct. For example, there is little point planning a detailed programme of actions to, say, capture the enemy flag, using actions at the level of *move to position X*, *face direction D*, *extend right arm* and so on, since the opponent will be responding in a way that undermines much of such a plan before it is executed. On the other hand, a plan at an abstract level: *deploy decoy move on left flank*, *deploy strike team in deep penetration around right flank*, *on signal, send in sprint team with heavy cover in scattered formation* and so on, can be robust to responses by the opponent (that is, this plan anticipates certain forms of response and can still be executed regardless of the details of the response, provided that the anticipation is broadly valid).

Execution of a plan is then an interesting challenge in mapping the high-level abstract actions into lower level executable steps. This might be achieved through some sort of mapping to finite-state machine models of execution behaviours.

8.2. Talking to Planners

If we assume that the objective of a planner is to put together a coherent programme of actions, there still remain the challenges of communicating to and from the planner. A planner must be told the current state of the world (as it is known by the agent planning) and it must know, or have access to, descriptions of the actions available to the agent at the appropriate level of abstraction for the plan being sought. It must also be told the goals. Generation of goals is non-trivial and there are many questions about why certain goals might be adopted. Once a plan is constructed, communicating the plan to the external executive is also a challenge. There must be means to monitor the execution of a plan and the abstractions will make this harder because it will make it difficult to know when an action is completed - the translation of an abstract action into concrete steps must be flexible enough to account for different possible states of the world. For example, when is the action of deploying a decoy move completed? When some set of team members is in place? Where must they be precisely to count as deployed? Must they have been detected? How can we be sure that they are decoying the opponent?

One language that has been proposed for communicating with a planner is PDDL, used in the academic planning community. There are many reasons why this is probably inappropriate for specialised target domains such as in games, but speed is obviously one crucial issue: there is no way that game time can be spent constructing and parsing PDDL documents. The actions available to an agent in a particular game domain will be hard-wired into the planning system for that game, in order to achieve efficient performance. Of course, there might be a role for PDDL or something like it in game development, to be compiled into dedicated planner machinery for a given domain.

8.3. Planning Strategy

Planning in the academic community has traditionally been concerned with the domain-independent task of constructing programmes of activity from primitive action descriptions, using weak-heuristic search techniques. Much work on planning has also considered more scripted activity such as hierarchical task network style planning (HTN planning). In considering the ways in which planning can play a role in games it is important to question what degree of flexibility is really sought and what form of planning is really necessary. It is important that agents exhibit intelligent goal-directed behaviour, but not necessarily that they are capable of constructing near-optimal plans in all situations. Sensible default behaviours to enter

safe states from which deliberation can proceed to identify new plans might be just as effective as an ability to re-plan from arbitrary initial states.

It is extremely unlikely that heavy-duty computation can be expended in planning, so search intensive approaches are unlikely to find favour. Anytime algorithms might be more promising, but there still needs to be a good performance guarantee for time-to-first-plan. On the other hand, once a plan is in place and an agent is pursuing it, there is likely to be some opportunity to continue to develop and extend the plan. Thus, a good forward-planning strategy is likely to be more promising than a backward-planning strategy, since it will generate actions for execution quickly before completing the entire plan. The combinatorial explosion involved, however, renders this less directed forward-planning approach often infeasible.

8.4. Plans for the Future

The group needs to progress these ideas, and this will commence with a short-term activity to identify concrete scenarios in which planning can play a role, together with a clear description of what would be a useful plan to have constructed in examples of these scenarios. Once the role of plans and planning has been more clearly identified, further activities will be focused on constructing the framework for communication with a planner, with a view to clear separation of the roles of planning, plan-dispatch, plan-execution monitoring and, possibly, re-planning.

8.5. Group Members

Current members of the working group on goal-oriented action planning:

- *Group coordinator:* Derek Long - University of Durham
- *Co-coordinator:* Ruth Aylett - University of Salford
- John Funge - iKuni Inc.
- Massimiliano Garagnani - The Open University
- Phil Goetz - Intelligent Automation
- John J. Kelly III - Model Software
- Craig Lindley - Zero-Game Studio, The Interactive Institute
- Ian Millington - Mindlathe
- Jeff Orkin - Monolith Productions
- Brian Schwab - Sony Computer Entertainment America
- R. Michael Young - North Carolina State University

9. Support Team

The AI Interface Standards Committee (AIISC) support team consists of ten students enthused about game AI from all around the world. There are different characters in the team, some of whom try to take the initiative whenever possible, and others who prefer to be assigned to tasks explicitly. Last year, Paul-Etienne Belloncik held the position of the group coordinator but has left the AIISC due to time constraints founding his own game developing company (named "Unlikely Games"). Bjoern Knafla was elected the next coordinator and builds the main communication link between the committee chairperson Alexander Nareyek and the support team now. Additionally, it is the coordinators task to try to distribute the workload onto all team members.

Our forum discussions count 150 postings but most of the work is organized by emailing team members or the AIISC working group coordinators. Being far outnumbered by the other committee members - the experts - and all of us studying actively, it is sometimes hard to deliver immediate support when asked for. Nonetheless, we are striving to offer the best help possible.

9.1. Tasks

Support tasks involve:

- summarizing the discussions of the different AIISC working groups,
- collecting questions of experts and answers given in the AIISC forums into so called "Expertly Asked Questions" (EAQ) documents,
- working into software, APIs or other standards (e.g., XML schemas or OpenGL) seemingly useful for the experts and providing short reviews of them,
- creating activity reports of all AIISC members,
- assisting in the creation of presentation slides and reports,
- helping with technical problems mostly concerning the usage of SourceForge.net and accessing its CVS repositories,
- and having an open ear to all needs and problems that might occur in the daily committee work, e.g., developing slide templates for conferences like GDC.

Every one of us monitors at least one committee discussion forum to react quickly on support demands and to protocol the posted arguments in summaries. In general, our role is not to participate actively in the experts' discussions (not to disturb them with greenhorn talk) but sometimes it is very hard to hold our horses, i.e., when we think that we could provide some expertise as well.

Until now, most discussion summaries were written by single members, but the steadily growing number of postings won't be overcome this way. In the future, more

and more group work will have to take place, mainly between the supporters assigned to the same AIISC working group. All of us will need to dedicate more time for the AIISC or more support team members are needed.

9.2. Motivation

We are proud to support the world's best game AI experts and to learn from them at the same time. The AIISC is a team, and we are helping to make a difference with our support - and have fun besides, too. However, we aren't sure if all of the experts (apart from some exceptions - mainly working group coordinators) really noticed and utilized our hard work. Directly collaborating together with the AIISC working group coordinators and experiencing their passion and patience helping us to improve our work is a blast though.

9.3. Group Members

Current members of the support team:

- Stephen D. Byrne - Hiram College
- Alex J. Champandard - University of Edinburgh
- Ting Feng - Northeastern University
- Cengiz Gunay - University of Louisiana at Lafayette
- Daniel Hartrell - University of Toronto
- Alexander Hornung - RWTH Aachen
- Börje Karlsson - Universidade Federal de Pernambuco
- Bjoern Knafla - University of Bielefeld (*currently elected group coordinator*)
- Jayaraman Ranjith - International Institute of Information Technology
- Hugo da Silva Sardinha Pinto - Universidade Federal do Rio Grande do Sul

References

[1] Craig W. Reynolds. Steering Behaviors for Autonomous Characters. Game Developers Conference (GDC), 1999. URL: http://www.red3d.com/cwr/steer/gdc99/

[2] G. Howland. A Practical Guide to Building a Complete Game AI: Volume I. Gamedev.net, 1999. URL: https://www.gamedev.net/articles/programming/artificial-intelligence/a-practical-guide-to-building-a-complete-game-a-r784/

[3] G. Howland. A Practical Guide to Building a Complete Game AI: Volume II. Gamedev.net, 1999. URL: https://www.gamedev.net/articles/programming/artificial-intelligence/a-practical-guide-to-building-a-complete-game-a-r785/

[4] John E. Laird and John C. Duchi. Creating human-like Synthetic Characters with Multiple Skill Levels: A Case Study Using the SOAR Quakebot. AAAI 2000 Fall Symposium Series: Simulating Human Agents, 2000.

[5] C. L. Forgy. RETE: A fast algorithm for the many pattern/many object pattern match problem. Artificial Intelligence, 19 (1), 1982.

[6] J. C. Giarratano and G. D. Riley. Expert Systems: Principles and Programming, Second Edition, PWS Publishing, 1993.

[7] AI Depot. Artificial Intelligence Depot, 2002. URL: https://ai-depot .com/Main.html

[8] Intrinsic Algorithm. Game AI, 2003. URL: http://www.gameai.com/

[9] Analía Amandi, Marcelo Campo, and Alejandro Zunino. JavaLog: a framework-based integration of Java and Prolog for agent-oriented programming. Computer Languages, Systems & Structures, 2004.

[10] Alejandro Zunino. JavaLog Website, 2003. URL: https://www.exa.unicen.edu.ar/~azunino/javalog.html

[11] FS Media. Warcraft 3 Preview. November 2000. URL: https://www.firingsquad.gamers.com/games/war3preview/

[12] Dave C. Pottinger. Implementing Coordinated Unit Movement. Game Developer Magazine, February 1999. URL: https://www.gamasutra.com/features/19990129/implementing_01.htm

[13] JCP. JSR 94: Java Rule Engine API, 2004. URL: https://www.jcp.org/en/jsr/detail?id=94

[14] eXpertise2Go. Introduction to Expert Systems, 2001. URL: https://www.expertise2go.com/webesie/tutorials/ESIntro/

[15] Alison Cawsey. Forward Chaining Systems. In Databases and Artificial Intelligence 3 - Artificial Intelligence Segment, 1994. URL: https://www.cee.hw.ac.uk/~alison/ai3notes/subsection2_4_4_1.html

[16] Steve Rabin. AI Wisdom - Game Artificial Intelligence Articles & Research, 2003. URL: https://www.aiwisdom.com/

[17] I. Wright and J. Marschall. More AI in Less Processor Time: “Egocentric” AI. Gamasutra, 2000. URL: https://www.gamasutra.com/features/20000619/wright_01.htm

[18] Marco Pinter. Realistic Turning between Waypoints. AI Game Programming Wisdom. Charles River Media, 2002.
[19] J. E. Laird and M. van Lent. Developing an Artificial Intelligence Engine. In Proceedings of the Game Developers' Conference, 2000.
[20] I. Frank. Explanations Count. AAAI 1999 Spring Symposium on Artificial Intelligence and Computer Games, Technical Report SS-99-02, AAAI Press, 1999.
[21] D. Miranker. TREAT: A new and efficient match algorithm for AI production systems. Pittman/Morgan Kaufman, 1989.
[22] A. Rollings. Game Architecture and Design, Coriolis Technology Press, 2000.
[23] Drew McDermott et al. PDDL–the planning domain definition language–version 1.2. Yale Center for Computational Vision and Control, Tech. Rep. CVC TR-98-003/DCS TR-1165, 1998. URL: https://homepages.inf.ed.ac.uk/mfourman/tools/propplan/pddl.pdf