IEEE Region 10 Undergraduate Student Paper Contest

A SIMPLE INTELLIGENCE BUILDING SCHEME FOR GAME ENTITIES

February, 2002

Author: Shalin Shodhan

Membership Number: 41290548

Address: 17/328 Satyagrah Society,
Satellite Road
Ahmedabad 380015
Gujarat, India

ABSTRACT

In today's fast paced computer games market there is an insatiable need for new game ideas and games with greater replay value. Artificial Intelligence (AI) plays an integral role in churning out one "killer" game after another. The Simple Intelligence Building Scheme(SIBS) presented here emphasizes upon simplicity of implementation and time-space efficiency. It caters to all the AI needs of modern day computer games. It is modeled upon a combination of the learning process of a human child and the power of discretion of an expert system. Numerous applications of this theory to other fields are presented at the end.

Self-Declaration

I declare that this paper has been completely compiled and prepared by me.

Contents:

List of Figures:

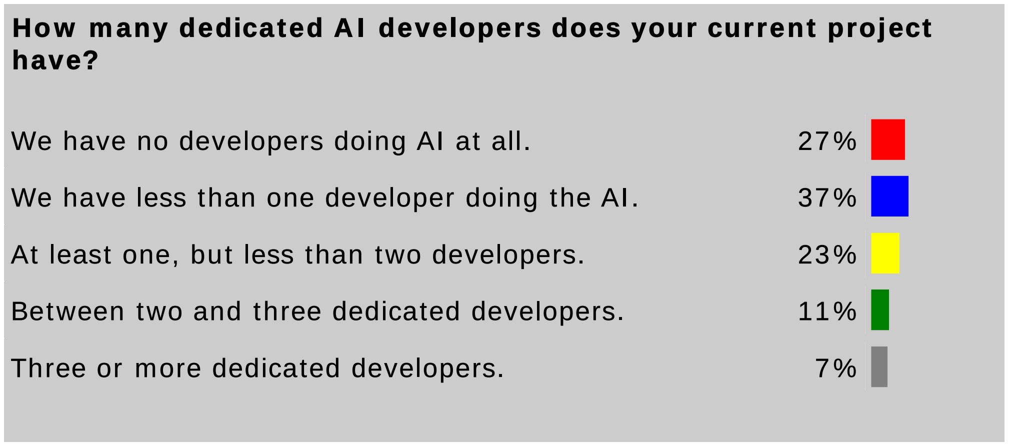
List of Tables:

1. Introduction

“The scary part is not how it thinks, but that it THINKS”
-Quake III, Id Software

AI in games has come a long way since the days of ‘Pacman’ and ‘Space Invaders’. Games like ‘Unreal Tournament’ and ‘Quake III Arena’ boast of intelligence that matches and even betters human players. However gaming is one field where there is a constant need for innovation. No matter how good a game is it can always be made better. One of the main aspects where games look to improve is AI. The “baddies” in the game need to get smarter all the time. This keeps the players coming back to the game. Consequently game developers must invest a lot of resources for AI development. Thus arises the need for an AI technique which is very simple to implement and provides intelligence to all parts of the game. Time saved on AI development can be used to enhance other aspects of the game. Table 1 shows results of an internet poll on how much manpower game developers invest in AI.

Table 1. Dedicated AI Developers



How many dedicated AI developers does your current project have?

We have no developers doing AI at all.	27%	🟥
We have less than one developer doing the AI.	37%	🟦
At least one, but less than two developers.	23%	🟨
Between two and three dedicated developers.	11%	🟩
Three or more dedicated developers.	7%	⬛

(Based on the poll conducted at http://www.gameai.com)

The poll results indicate the poor manpower allotment to AI. Hence AI should be made easy to implement in the game.
A simple example of ‘intelligence building’ in nature is the learning process of an infant. An infant does not have much intelligence or power of discretion. It relies mainly on its parents to show what is right and what is wrong. It also learns from other sources like its own sensory feedback from the world. We base our technique upon the former. Thus our objective is to develop a teaching scheme for game entities. The good thing about machines is that they are great students and do exactly as they are told. A more biological point of view describes the idea thus. The human body is an excellent chemical reactor. The brain stores information using complex chemical reactions. The computer is similar except that it deals with numbers instead of chemicals. We can store almost any sort of information by quantifying it with numbers. This fact lies at the heart of SIBS.
In summary, we are going to develop an economical and fast technique to train game entities, store their knowledge as numbers and make them act as per their training.

2. Concept

“You know your game is in trouble ...when you ask the producer about the AI and he says, ‘Those random number generators can do anything’ in an awed voice...”

This idea was initially developed to serve as a path following algorithm and has evolved into a technique called SIBS or Simple Intelligence Building Scheme. It is best explained with an example.

2.1 At the very beginning…

In its earliest form SIBS has been used to make ‘enemy aircraft’ of the 3D game ‘Aerokombat’, race through a canyon. As a first step, let the 3D model, object or the game element which has to be trained be referred to as the “game entity”. Each game entity has a set of controllable “characteristics”. The entity is to be made intelligent with respect to the control of these characteristics.

Example 1
Consider the entity as an enemy aircraft which races through a canyon against the user.

For this initial problem the characteristics of the entity which have to be autonomously controlled during flight are its position and orientation. The next step is to assign controllers to these characteristics. We assign various keys on the keyboard to control translation and rotation about the three co-ordinate axis. Further we must keep a synchronization variable say “timer” which is incremented by 1 in every ‘game loop’. The game is run in “Training Mode”. Training mode defers from the standard “Playing Mode” in the sense that controllers have been assigned to some game entity which is being trained. Once in training mode, the ‘enemy aircraft’ is made to fly through the canyon using the controllers. At the same time, for every value of timer, the values of position co-ordinates and angles of rotation are stored in an array. Thus six variables will characterize the game entity in each loop. When the entire run is over this array is written to a file. Such a file is called an “action file”. Now the controllers are removed from the enemy and assigned back to the user. When the game is run in Playing Mode the action file is read into an array and read back to the characteristics of the entity as parameters. For each value of timer the ‘enemy aircraft’ will behave exactly as it did in Training Mode except that this time it will do so autonomously. This is the purest form of SIBS. It simply involves recording and replaying behavior of an entity.

2.2 SIBS Terminology:

At this point it would be good to understand the terminology associated with SIBS. Since simplicity is the primary objective, there are very few concepts that must be grasped before one can implement SIBS at a basic level. They are:

Entity- The object in the game which we want to make intelligent

Characteristics- The parameters of the entity which are to be made self controlled

Training Mode (Intelligence Acquisition Mode) - Mode where an entity’s characteristics are assigned controllers.

Playing Mode (Intelligence Application Mode) - Normal game mode where entities are autonomous.

Timer - Synchronization variable incremented by one in each ‘game loop’.

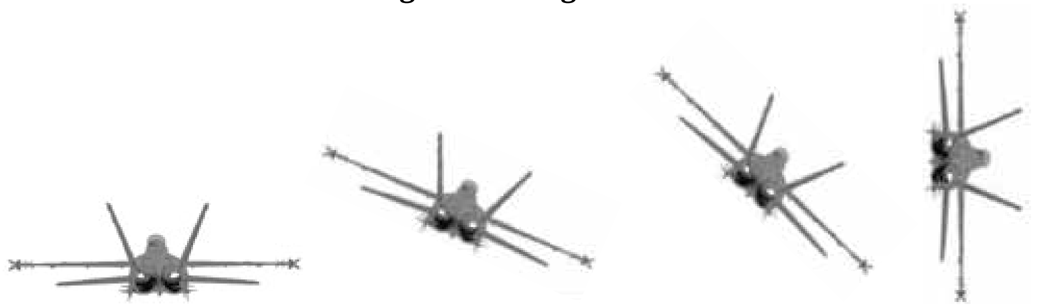
Action File (Intelligence Storage) - A file which is created by training mode and used by playing mode. It is the memory of the entity. It can also be called a “Lesson File”.

Table 2.Sample Action File

TIMER	XROT	YROT	ZROT	YPOS
50	0	0	0	450
51	0	0	15	500
52	0	0	30	550
53	0	0	45	600
54	0	0	60	650
55	0	0	75	700
56	0	0	90	750

Table 2 shows part of an action file of the enemy aircraft for doing a 90 degree clockwise flip while moving right at 50 units per frame as shown in Fig.1.

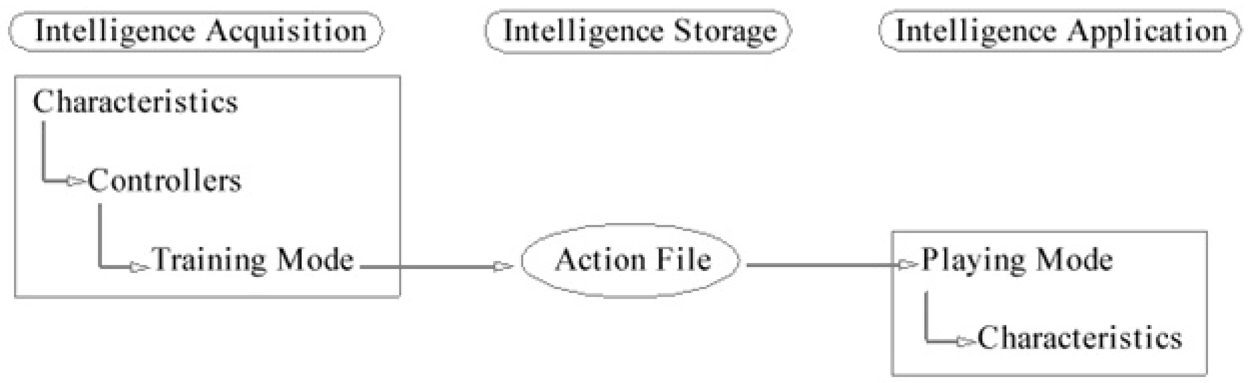
Fig. 1 Training Mode



Using keys to control and train the enemy aircraft.

2.3 In Summary…

Fig. 2 Block diagram of a SIBS implementation.



SIBS has three main logical components- Intelligence Acquisition, Storage and Application as shown in Fig. 2. Intelligence Acquisition deals with teaching the entity how to use its characteristics. Intelligence Storage consists of creating the data structure of the action file and the deciding the method of quantifying the characteristics. Intelligence Application is the process of choosing and playing an action file in response to an event or based on the flow of the game.

The barebones form of SIBS can be implemented by the following steps.
Take the game entity to be trained and assign controllers to the appropriate characteristics. Run the game in Training Mode. An action file is created containing characteristics represented as numbers for each value of timer. When training is complete, run the game in Playing Mode. The action file guides the entity's behavior.

There are many ways of interfacing the action file with the characteristics of the entity. In the subsequent sections we shall explore the various methods for the same and hence study various forms of SIBS implementation.

3. Data Structures and Complexity

“You know your game is in trouble...when your AI says, in a calm, soothing voice, ‘I'm afraid I can't let you do that, Dave’**....”**

Designing the data structures is the most important part of SIBS. The data structures are always implementation specific. The main points to be taken care of are as follows.

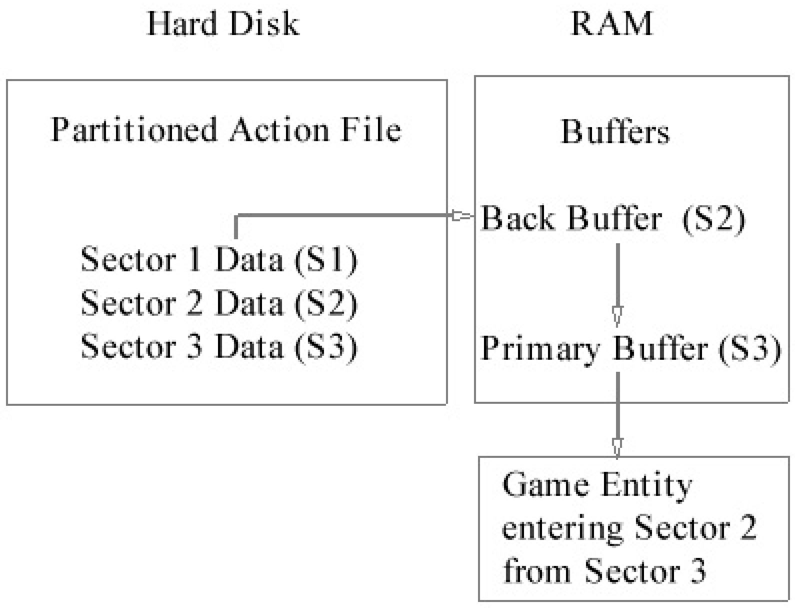
3.1 Characteristics: The characteristics that must be stored in action files. They vary for each entity. Care must be taken to quantify ‘entity behavior’ in a minimum number of characteristics so that the game runs optimally. In example 1 the aircraft behavior in context of the game was fully quantified by its positional and rotational parameters. Enumerating the characteristics to be assigned controllers is not very difficult since all characteristics that can change have to be programmed to do so and hence the coder has direct access and control over them. The data type of characteristics is also an issue since it may not be possible to use numbers at all times.

Example 2
Consider an entity which has conversations with the player. When we train it to converse, it will build an action file of sentences and words, not numbers. Also conversations are based on user input and hence their sequence may not be pre-recorded.

3.2 Choice of Timer: The synchronization variable is the common value joining intelligence acquisition with application. Normally a timer variable which is incremented with every game loop is used to accurately log actions at all times. Such a timer variable can quantify any event by its time of occurrence in terms of the number of game loops after which it occurred. However sometimes timer may not be a linearly increasing numeric quantity. Such a situation commonly arises when the action file must respond to some input and not to an event. The entity in example 2 is such a case. The action file contains sentences which must be used based on the user’s actions or messages. Thus the synchronization variable is an action or a message from the user. Such actions or messages can be numerous. Hence the action file must incorporate the associative logic for each action and corresponding textual response. Timer in this case is the left hand side variable of the associative logic, that is, the action or message from the user.

3.3 Structure of the Action File: Many aspects influence the structure of the action file. Type of content is one such factor. An action file for example 2 must be dynamic and respond to input. It has an inbuilt associative logic called a ‘decision engine’. Normally decision making part is separated into a neural net or an expert system and action file is kept static. More about decision engines is discussed in section 5.1
The main aspect of structuring the action file is speed of operation. To ensure best performance the action file must be easily readable by the game and small enough to be loaded into memory. If it is not small enough it may be partitioned and loaded in buffers.
Only those portions which are needed at the current stage may be loaded. As the need arises, more data may be swapped in from the buffer. A large action file which is double buffered and partitioned will perform smoothly. Flow of information when using a sector based approach with a partitioned action file and double buffering can be seen in Fig. 3

Fig. 3 Flow of data in partitioned action file with double buffering



3.4 Complexity

When we consider time-space requirements of SIBS the most important point to ponder upon is the array based storage at run time or in Playing Mode. Even though data is only in numbers the amount of memory required may become too large in certain cases. In such cases buffering schemes must be applied. Since the algorithm requires no processing at runtime and simply involves streaming data from a pre-created action file its time efficiency is very good. Space efficiency depends on the type of entity and detail of characterization. Method of quantification also has great effect on space efficiency.

3.5 Alternative Quantifications

All data in SIBS is quantified and stored as numbers. But there are exceptions to this scheme, see example 2. The numbers themselves may be modified to improve space efficiency. For example, instead of storing each characteristic value in each game loop only those values which change may be stored. Also, instead of storing the value itself, only the change in the value may be stored. Numerous compression techniques may be used to keep the action data as small as possible.

SIBS is open to modifications. Each developer may use his own data structures and methods for quantifying entity information.

4. Coding

“You know your game is in trouble...when even your AI won't play the game - it just deletes itself...”

The main part of coding SIBS is directing the flow of information and switching modes.

4.1 Flow of intelligence data

Code must be written to for every characteristic that we want to make self-controlled.
During training mode the code must interface the characteristics with the controllers and during playing mode it must interface the action file with the characteristics.
The code must also control the direction of flow for each mode. Such a flow must be as streamlined as possible since it has a direct relationship with the efficiency of operation. To stream data to and fro between action files and entities arrays or any other types of buffers may be used. Various partitioning and buffering schemes discussed in section 3.3 must be implemented with code.

4.2 Switching modes

There are two modes for SIBS, the Training Mode and the Playing Mode. In the former the entity being trained has to be understood and appropriate characteristics linked with controllers. The choice of controllers is only limited by the developer’s hardware constraints. Another variation to mode switching is providing the end user with his own training facility as discussed in section 6.1. In such a case coding must allow of invocation of Training Mode at the end user level.

Since there is no specific way of coding SIBS numerous methods may be used to implement it. It is language and platform independent.

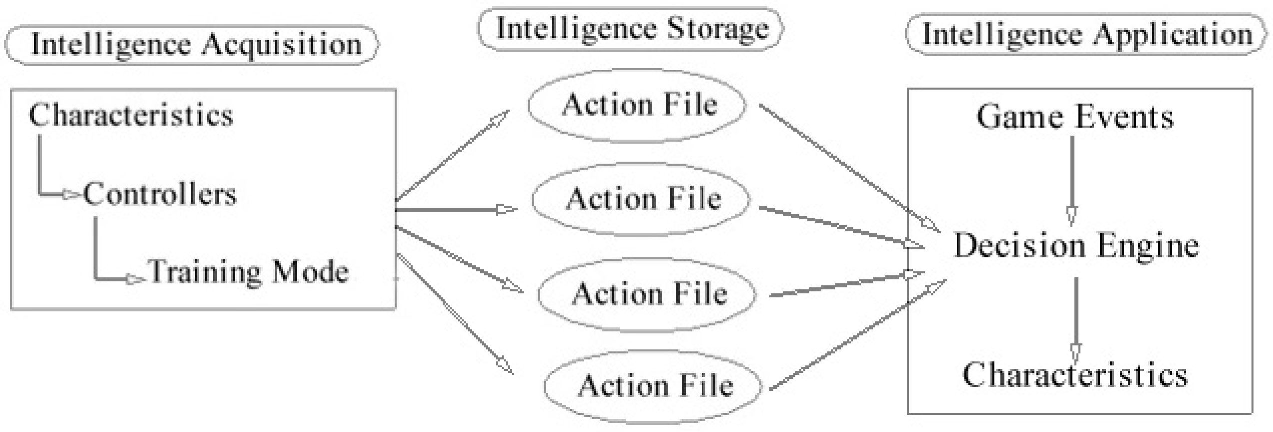
5 SIBS Enhancements

Various enhancements to the basic idea enable SIBS to serve all the AI requirements of a game. When we see SIBS as a part of the big picture it is evident that the real power of SIBS lies in the great number of applications that it has .Since we can train any entity and store any number of characteristics of that entity into any number of action files we can build a complete database of actions that the entity can have. This is just like teaching a baby a multitude of actions or building on various traits of a child's nature. However with each enhancement there is a reduction in the simplicity of implementation of SIBS. Then again,

"Everything should be made as simple as possible, but not simpler."
-Albert Einstein

5.1 Multiple Action Files and Decision Engines

Fig. 4 Block Diagram of SIBS with a Decision Engine



Existence of multiple action files raises the problem of choosing a single action file at a time. Which action file is chosen at a particular time depends on various factors. While action file creation is still quite simple, choosing actions may require the use of a decision engine. A typical decision engine must take the events of the game as input and choose appropriate action file which makes the entity perform. Thus the intelligence building scheme is generic but assignment of controllers and selection of actions files, if needed, are specific for each entity. This enhancement would make SIBS a 'very good learner' with the 'power to decide' of an expert system.

Conventional AI concepts may be used to implement the decision engine. It may be a neural network or a rule based inference engine. An interesting concept is the use of SIBS itself in training the decision engine. Such nesting is discussed next.

5.2 Nesting

When one SIBS implementation uses another SIBS module to train some of its parts we can say that it is a nested used of SIBS. For example, the decision engine discussed in section 5.1 may be 'trained' to select action files. It may acquire such training through SIBS. Thus the developer must simply train the decision engine rather than code a complex AI mechanism into it. The neural network or expert system built into the decision engine can be replaced by an action file which serves the exact same purpose. It is important to note that an action file is static and hence will not respond differently to different events, which is the primary objective of a decision engine. Thus it cannot completely replace the neural network or expert system but it can simplify their creation. An example of such simplification is a combination of an action file and a simple ‘if-then’ construct. Together they will behave as a complex decision maker. Thus SIBS can be nested as many times as required.

5.3 Non-Linearity

"For a smart material to be able to send out a more complex signal it needs to be nonlinear. If you hit a tuning fork twice as hard it will ring twice as loud but still at the same frequency. That's a linear response. If you hit a person twice as hard they're unlikely just to shout twice as loud. That property lets you learn more about the person than the tuning fork."

-Neil Gershenfeld, When Things Start to Think, 1999

Another interesting enhancement to SIBS would be the addition of non-linearity. Since an action file will cause the entity to perform the same thing every time the game might become predictable. To make SIBS non-linear, Pseudo-Random number generation may be applied to choose from a multitude of action files or also to make subtle modifications to the action files. Again, SIBS nesting may be used to implement non-linearity. A separate action file may be created with the specific purpose of modifying other action files at runtime to add non-linearity.

6 Other Applications

"You know your game is in trouble...when your game's AI is offered a better paying job than you."

This section deals with application of SIBS in new game ideas or other fields.

6.1 Game idea: AI wars

This is very much the age of network gaming. People from all over the world battle it out over various networks. Internet gaming has its own legends and Gods and makes for intriguing wars on the web. A game idea that spawned from SIBS is to develop modules for training battle droids. These battle droids will then be released on a controlled battle arena online. They will do battle with other droids and the better trained droid will be victorious. It is somewhat like a software version of autonomous sumo wrestling robots. It requires no user participation during battle. The user's task is to train and update training for his droid. An interesting variation to the standard network gaming idea is that in this game both users need not be online to play. All they need to do is train droids and make them available for battle. Thus Training Mode becomes the main "hands-on" part of the game and Playing Mode is more like watching your child in the 'school play'.

6.2 Movie Scripting

Computer animated movies were among the highest grossing films in the past year. SIBS has great use in this field. Movie entities may be controlled in the same way as game entities. An entire movie may be stored as action files. Movie making becomes a computer aided puppet show except that the puppets will act on their own once they are trained.
Like movies, some games require events to occur according to a particular script. Instead of coding for such scripts it would be much easier just to run the game and play it out. While playing out each event, action files can be generated and incorporated into the game as a live storyline. Non-Linearity may also be added at this point to make the game adapt to the user.

6.3 Recording and Replay

The essence of the SIBS theory lies in capturing characteristics of game entities in numbers and storing them. This concept can be broadened to not only entities but events as well. Thus a game can be recorded in just a bunch of numbers and this makes storing 'replays' very economical. All that is to be done is to list every characteristic of each event and entity which may change during the course of the game. Since nothing changes on its own and all changes have to be programmed, enumerating these should not be a problem. Also, each change can easily be quantified in form of numbers and hence an action file may be created to store the proceedings of the game.

6.4 Applications to Robotics

Since SIBS is an AI technique it may be used to train robots and make action files. The possibilities are immense and since teaching a robot is faster than teaching a child, rapid development of intelligence can be brought about without worrying about techniques for making the robot learn on its own. SIBS has applications in all things AI.

6.5 Game Physics

SIBS makes its way into all areas of gaming. While game physics is not very affected by intelligence of entities, the other way round it has the greatest effect on building intelligence. Physics specifies the rules of ‘gameplay’. These rules strictly govern the actions of entities. Application of SIBS to physics has nothing to do with intelligence, but a lot to do with the essence of the SIBS concept, which is quantification and storage of information as numbers. Collision data, freedom of movement, response to forces and all other aspects can make use of SIBS data. Such data can be collected during training more by the game developer and incorporated into the game rather than having to formulate complex algorithms for the same. Thus SIBS is a handy tool for building simulators.

7. CONCLUSION

The true power of SIBS as a concept lies in its simplicity of implementation and multitude of applications. It is a highly extensible idea that can be seamlessly integrated with any conventional AI technique. It reduces the complexity of developing intelligent programs by balancing the weight of the “thinking” part of AI and the “learning and remembering” part of AI. As an ‘intelligence manager’, SIBS looks into all three aspects of intelligence that is, acquisition, storage and application. SIBS can be thought of as a “parenting scheme” to create intelligence in game entities by teaching them like children.
To end on a philosophical note, SIBS is dedicated to all the Parents of the world who mould children into better people and to Mother Nature from whom we get all our ideas.

“A human being is part of a whole, called by us the Universe... he experiences himself, his thoughts and feelings, as something separated from the rest - a kind of optical delusion of his consciousness. This delusion is a kind of prison for us... Our task must be to free ourselves from this prison by widening our circles of compassion to embrace all living creatures and the whole of nature in its beauty.”

-Albert Einstein