

Simajin® FAQs

What language is Simajin written in, what compilers are used, what operating systems and computers is it used with, and how large is it (how many lines of code)?

Simajin is written in ANSI C++. Even more importantly, it is written in machine-portable C++, which is a much stricter requirement, and harder to attain since there is no document that will tell you how to do this. It is something only gained from practical experience. Simajin runs under Windows using Intel or Intel clones, Linux/UNIX on various hardware platforms, and Macintosh OS X on IBM and Motorola processors. It has been used with OS/2 and older versions of the classic Mac OS. Presently we use CodeWarrior for Mac OS X and Windows, and GCC for Linux. We have used the OS/2 IBM compiler; Microsoft Visual C++; and various Sun and Silicon Graphics compilers. We typically turn on all compiler and linker warnings that are practical and treat them as errors.

There are fewer than 60,000 lines of text (read: carriage returns) in Simajin. This includes everything in the header and source code files (blank lines; comments; structure, class and function declarations and definitions; preprocessor statements). This covers the parser generator for the formally defined languages that define the data formats, the pre-processor for the three primary types of data, the simulation execution proper, the graphics, the data analysis and summarization processor, and the interface code to other simulations or graphics displays. This covers the specific code for hosting on Windows, Linux/UNIX, and Macintosh computer platforms, including the various quirks and different functions associated with the implementations of, and interfaces with, OpenGL.

How do you define systems/entities (e.g., tank, APC, dismounted soldier, artillery, helicopter, fixed wing)?

Systems and entities are defined in Simajin in terms of adjectives, nouns, adverbs, and verbs. The underlying meaning of the nouns and verbs are embedded in the source code. The adjectives and adverbs are described by the user in the data bases (or input files). This approach allows Simajin to not only be data-driven, but, as someone once described, data-dominated.

First I will describe the noun part of the simulation architecture, and then I will discuss the verb aspects. Once you see how the nouns and verbs are capable of describing a scenario (remembering that they can be modified by user-determined adjectives and adverbs),

then you will begin to see how Simajin can be applied to many different applications.

I will start off by stating that to define an A-10 in Simajin you could start with a tank and just make it move faster. An aircraft carrier can be an airbase with movement capabilities. The normal assumptions about command and control, the existence of command chains, existence and the number of sides or membership in a side, scenario scope and locales, minimum and maximum detail (where it fits in a simulation hierarchy), types and numbers of entities that often are present do not exist in Simajin. Simajin is not even a military simulation; it is a tool for examining conflicts. Since everything eventually devolves into a conflict in the use of time and other resources, Simajin can be used for a lot of things. It is also important to remember that Simajin is the most evolved member of a family of simulations that have been developed for DNA, various HQ DoD staff elements, NSA, the USAF, USN, USMC, DA, DOT, and other organizations. These simulations have been used for almost 30 years for various studies and projects. Most recently the Joint Strike Fighter Program Office used two of these predecessors as the primary analysis tools for examining the competing aircraft designs. There is a long history of varying degrees of formal verification, validation, and accreditation, as well as practical, time-constrained effective use of this architecture most recently embodied in Simajin.

Nouns

Players

In Simajin real or hypothesized entities are defined within the context of a hierarchical organization belonging to something we call a player. A player has its own perceptions of reality, tactics, orders, attitudes and communicates with other players explicitly according to the level of physics detail the user selects. Within a player communications are immediate – all parts of a player share perceptions, tactics, etc.

Clusters

Players are made up of zero or more things we call clusters. Players defined by a user always have at least one cluster (at least for now). A cluster is defined by a location, possible motion (translational and rotational), and figures (points, lines, areas, volumes). Clusters by default exist at a single point (although offsets in x,y,z can be added to this point for the constituent parts of the cluster). Clusters, by possessing kinematics-related characteristics, are the primary catalyst for causing or allowing physical interactions (sensing, talking, affecting).

Elements

Clusters have zero or more elements. The vast majority of the time an active cluster will have at least one element. A cluster without elements primarily exists to cause interference with sensing, talking, and affecting. For example, a dirt berm could exist as a cluster with no elements. In this case the berm could provide cover and concealment to other clusters. The cover could exist in the form of protection against weapons, while the concealment could prevent successful sensing chances.

Elements are the primary catalysts for weapon effects and sensing chances. Signatures, for example, are part of an element's characteristics. Kill probability tables are directed against elements (this will change dramatically in the very near future – refer to the discussion of upcoming weapon representations for more details).

Systems

Elements are mostly the passive aspects of a player. On the other hand elements are made up of systems, and systems give a player the active ability to do something. There are eight types of systems: sensor receiver and transmitter, communication (comm) receiver and transmitter, jammer, mover, affector, and worker/thinker. The first five types are energy-related and have many features in common. Jammers can be thought of as the energy equivalent of effectors. Comm and sensor transmitters can cause problems similar to jammers. The three types of energy emitters (the two transmitters and active jammers) can be sensed, depending on how the user defines the interactions in the data base. In this case the gain patterns for these emitter systems operate similar to an element's active signature. Affector systems can also be given time-varying emissive signatures, such as the infrared, visual, and acoustic signatures resulting from the firing of a tank's main gun. Mover systems can also have signatures associated with the speed of the cluster that owns it. So in one way or another five of the eight system types can affect the detection of their parent element.

Mover System

Mover systems provide a motion capability to the cluster. Rotational motion can be separated from translational motion. Motion can be along the ground or at a more or less constant altitude above the ground (called surface motion in our terms). This type of motion can be thought of as 3 DOF (x, y, and heading) since the altitude is fixed. It is possible to have the altitude above the ground uniformly and smoothly change from one value to another from one path point to the next, but this does not change the underlying fidelity in the representation of the motion.

Motion (translational and rotational) can also be

completely free in all spatial dimensions, resulting in 6 DOF motion (x, y, z, roll, pitch, yaw). This type of motion can be either coordinated (fixed wing aircraft) or not (helicopter). We presently do not have an option to represent the tilting of a naval vessel or ground vehicle as it proceeds through a turn. The lack of this option is completely due to the fact that no one has ever asked for this type of fidelity in how we represent motion. The surface or 3D motion can include turns or not (the latter is called point to point). This results in four basic types of motion: surface with turns, surface without turns, 3D with turns, and 3D without turns. Transitions can occur during different phases of motion. An aircraft, for example, could use surface without turns while en route to the taxiway, surface with turns on the taxiway and runway prior to takeoff, and 3D with or without turns during its flight.

Attributes

Attributes can belong to players, clusters, elements, and systems. Attributes are integer, real, or string variables that can be given default values, initial values before a run starts, or dynamic values during the simulation. Attributes can be sensed, they can be transmitted to other players via messages, calculations can be performed using them, and they can be changed or checked as part of the decisions made by a player. Attributes can be used to represent many different types of application-specific characteristics that in many simulations would either not be present or would have special code for each attribute.

Expendables

Expendables belong to systems. They can represent a fixed set of objects: fuel, ordnance, future players (for disaggregation), and sensor countermeasures such as chaff bundles or flares. Expendables are gradually being replaced by attributes.

Verbs

The verbs in Simajin are closely related to the names of the systems. This is natural, because the systems are the mechanism for the players to perform functions. There is not a one-to-one correlation for practical reasons, primarily because the systems don't exactly map into the verbs. The affect verb, for example, includes affector systems and jammer systems. Verbs also cross boundaries by involving multiple players and different aspects of the player hierarchy. Verbs should be a way of looking at what might happen in a scenario, while the nouns are a guide to how to implement the requisite detail for the desired verbs. We usually refer to verbs as the generic functions of the simulation. The source code in the simulation proper revolves around these generic functions.

One of the postulates underlying Simajin is that the player organization and generic functions (or their

logical equivalents) are necessary and sufficient conditions for representing anything at any level of detail.

Affect

This includes energy, matter, and psychological transfers from one player (or part of a player) to one or more players (or parts of players). Information transfers are part of either sensing (non-cooperative information transfer) or talking (cooperative information transfer). Jamming comes under this generic function. The supplying of electricity or spare parts comes under affect. Weapon effects and praise or condemnation are part of the affect generic function.

How do you define a weapon (e.g., tank cannon, rocket propelled grenade, antitank missile, rifle, machine gun)?

Weapons are, eventually, affector systems. This means that they are part of an element that belongs to a cluster that is part of a player. Affector systems physically affect other players. The effect can be helpful or hurtful. Weapons would normally be thought of as hurtful affector systems. Weapons can affect the target only, or they can also affect others (collateral damage). We are in the midst of dramatically increasing the capabilities of affector systems (and thereby weapons), so I will have to discuss today's capabilities, and then the capabilities that are being added in the next month or so (mid-to-late spring 2004).

Presently weapon systems have stuff that gets used up (and can be reloaded and re-supplied) that represent bullets, bursts of bullets, missiles, artillery shells, warheads, etc. The stuff is either called an expendable or a future player. If the weapon's effect on the target can be approximated by calculating an arrival time and a table lookup for the effect, then this is called an implicit flyout in our terminology. The ordnance (expendable) cannot be sensed or affected by anyone else, except that a maneuver on the part of the target might cause a recalculation of the intercept time for expendables such as missiles that can, themselves, maneuver.

How do you define a sensor (e.g., optical, electro-optical, mmw radar, thermal, infrared)?

Simajin represents transverse and longitudinal wave sensors. The user chooses the propagation speed, frequency, and many other features which I will detail shortly. The sensors can be passive (receiver only) or active (receiver and transmitter). The receiver and transmitter need not be co-located, but, for now at least, they have to be part of the same player.

We use free space (1-way and 2-way range equations) to represent signal losses along with optional

transmission loss tables for cases where the free space losses fail to capture the needed fidelity. Terrain can affect line of sight or not (depending on options selected by the user). Features (buildings, trees, clouds, etc.) can also affect line of sight or not. These two options are selectable by sensor type. The earth can be considered as flat or the user can define an effective planet radius by sensor type. MTI, Doppler minimum and maximum limits, azimuth and elevation limits, internal losses, signal to noise detection thresholds, different frequencies for acquisition, search, track, and fire, etc. are possible data values. Gain patterns can be a single number, or very complex patterns with variable data densities by azimuth and elevation. The gain patterns can be asymmetric in azimuth and elevation and they can be frequency-dependent. The speed of the cluster that owns the sensor receiver can affect the signal detection threshold (sonar, for example). Signatures are defined similarly to gain patterns. The speed of the target can affect its signature (acoustic noise, infrared from heat, etc.). These changes are of course frequency-dependent or not, based on available data and the user's option.

Simajin does not presently represent multi-path, knife-edge diffraction, or similar phenomena except indirectly through changes in the other parameters.

Remember that a sensor could be a player in its own right. It could be dropped from an aircraft, placed somewhere by a ground vehicle, or delivered by artillery. In these cases the "sensor" would also have some means of communication and some processing capability to "think" or "work."

What format is your terrain representation? What terrain features do you represent? What is the data source?

We use DMA DTED formats and USGS formats for the input files. We convert these files into an internal representation using approximately equilateral triangles. We have used, I think, all of the various terrain densities (points per degree) in the DTED format, as well as terrain from all of the different latitude bands, including both north and south of the equator. Additionally, we have used terrain files that cross latitude bands so we know how to handle this issue. We can also take other terrain data that is very dense, using floating point z values, and use this for special areas. We also have several techniques that can be used for generating correlated random terrain altitude values when you need to look at the effects of a wide variety of terrain.

We have used DFAD formats when available for areas and volumes such as forests, plowed fields, etc. We can represent "horizontal" areas such as a grassy area, and

“vertical” areas for tree lines along a road that block line of sight. We can also represent single trees.

How do you set up the scenario sequencing?

The sequence of events in the scenario is not something that is an explicit part of the data bases. The user defines the initial conditions, and from that, in general, the scenario proceeds. This means that if you stop the simulation and write out the state of the simulation, then this can be used as the initial conditions for another run.

There are a few options for the user to cause some specific events to occur during the simulation. We call these exogenous events, since they originate outside of the simulation’s processing architecture. These events can be used to turn systems on and off or set them non-operational; they can be used to cause new players to enter the scenario at specific points and times or randomly along lines or in areas or at random points in time.

Other than these exceptions the scenario sequences through time based on the initial conditions. To give you an idea of how this works I will explain what happens at game time zero, which has been referred to as the “big bang” beginning of the simulated universe since it is distinctly different from the rest of the simulation.

When the user places players in the scenario, this implies several types of events that must occur immediately to get things started. The three general categories of events are:

- initiation of interactions and movement;
- system status changes; and,
- initiation of cognitive activities;

Where, if any, is there any randomness in the simulation?

A complete answer to this question has several levels, based on how directly the randomness is associated with a simulated phenomenon. We will start with the most direct answer, the distributions represented and where functions having pseudo-random algorithms are used.

Probability Density Functions

There are two distributions explicitly represented: uniformly distributed random variates, and a Gaussian (bell curve) distribution. Regardless of the distribution or number of places where these functions are called in the source code, there is only random number stream whose initial value is under the control of the user.

Event Sequencing

The “leftist tree” algorithm is used for sorting the events by time. For our purposes here it is important to

know that this algorithm only gives a “semi-sorted” tree. This means that you are only guaranteed that the root node of the tree is the next event. You cannot traverse the tree in any fixed manner to determine the order of events.

Randomness from Randomness

The “butterfly effect” occurs in Simajin. A small random difference in one event can cascade through subsequent events, causing fairly dramatic differences in the final simulation state.

How is "sensing" modeled?

In Simajin sensing refers to only the physical phenomena; the cognitive aspects are part of what we call noticing and digesting. In addition there is the issue of how sensing gets started, and then how it stops, if ever. This is associated with the runtime efficiency of the simulation, since the determination of sensing interactions is the primary measure of how much computer time a simulation will need. This discussion will cover all five parts, starting with the determination of sensing interactions, proceeding to the physical representation of sensing and the termination of sensing chances, which are covered by the observe event, and the cognitive aspects of sensing.

Sensing: the Neighbor Event

Although players have mover systems, and although movement is one of the generic functions or verbs, there is no movement event in Simajin. Clusters that are in motion have paths that extend some amount into the future. Where a cluster is, though, is dependent upon who needs to know about it and when. In a very real sense movement does not occur in Simajin until and unless there is a need to determine if an interaction might occur or to resolve an interaction. The interaction can be endogenous to the simulation, meaning that the interaction is between simulated entities residing within and represented by Simajin. The interaction can be exogenous, meaning with something outside the control of the simulation. This exogenous interaction could be with the user via graphics or data saved for post-processing, or with another simulation or simulator.

Sensing: the Digest Events

A single data assimilation is represented with two events. Some preliminary features are performed during the start event; however, the bulk of the assimilation occurs during the end event. The data from a sensing chance can potentially update all of the data known about the object.

There is presently an assumption that a perception has as a basis something that exists in the simulation. In other words there are no true false targets, or phantasms as we call them. Explicitly represented false targets can be simulated using specially designed players, but this

is not completely satisfactory for all situations.

When simulating indirect fire at simply a point, whether there is anything there can be somewhat successfully accomplished by creating players that represent important geographic points that might be of interest, such as road intersections, or registration points. Then collateral damage weapon effects can be used along with sensor and weapon Gaussian errors to represent sensing errors and weapon aiming errors and flyout dispersion to potentially affect clusters that might be near the detonation point.

Perceptions of clusters have track IDs. These are assigned by the observing player and passed along with the perception when messages are sent. Within the simulation players then use these track IDs to correlate perception updates from multiple sources.

Presently perceptions are updated with whatever the latest information might be. If there were sensor errors from one detection then the new data will overwrite the existing data without “smoothing” the data. This has been satisfactory for previous applications. Future applications that need additional detail in this aspect of data fusion do not present any architectural issues. It is simply a feature of the simulation that no one has requested fidelity improvements.

How are matter and energy represented? Is the simulation adiabatic?

Simajin does not conserve matter and energy. Players can be added to the simulation completely independent of the initial conditions by the user. These types of events are technically called miracles in our terminology. It is important for the user to consider the consequences of miracles because they can cause discontinuous and unexpected changes in a player’s behavior when the pre-existing player happens to observe the miracle.

Matter is not explicitly represented in Simajin, and therefore momentum is not explicitly represented either. Maneuvers and motion are typically defined in the data bases to represent the effects of mass and momentum. For example, acceleration and deceleration values will limit how fast the speed of a cluster can change, as if this were due to inertia, thrust, drag, friction, etc.

Energy requirements in many cases are not explicitly treated. For example, users often ignore the electrical power requirements for radars, heating and air conditioning in buildings, etc. This is starting to change now that attributes have been added. Food, fuel for generators, energy storage and energy usage, and other related factors can be simulated with attributes, including the decisions that must be made when energy

reserves have been depleted.

How is motion represented?

Translational motion is represented as a sequence of points. The intervals between the points are either straight lines (infinite turn radius) or arcs of circles. From one interval to the next the plane of the arc can change. This results in a continuous and differentiable everywhere position function. For velocity it means that there is a possible instantaneous change in the unit velocity vector at the end of a segment. The length of the arcs is dependent upon user-defined data related to information processing rates and speed of the object.

For any given segment acceleration is constant, meaning that speed (magnitude of velocity) changes continuously and is differentiable everywhere. Acceleration thus changes in a step-wise manner. Jerk (the time derivative of acceleration) is always zero except at each point in the path where it can be \pm infinity when acceleration instantaneously changes.

Pure rotational motion can be represented with roll, pitch and yaw rates and durations without any associated translational motion. Rotational motion can also occur as a consequence of translational motion. For example, when an aircraft is turning there can be a bank angle and a climb angle that change as the turn progresses.

How are objects represented in terms of their shape, size, physical extent?

By default objects (clusters) are point objects.

All energy-related calculations are performed using far field assumptions.

How are interactions decoupled in order to progress through time?

The progression of time is represented by a series of events that are considered to be instantaneous. In those cases where a phenomenon takes an amount of time that cannot be assumed to be at an instant, then multiple events are used to start, continue, and stop the process or activity.

Indirect fire has two issues or differences from direct fire. First, indirect fire typically includes one or more observers that are not part of the unit that owns the weapon, or they are not co-located with the weapon.

What type of output data is generated? What format is it in?

The basic output, which is used for listing file summaries, graphics, interaction with other simulations, etc., is a sequence of game time-tagged sentences. The

sentences always have a subject and verb. Except for completely internalized cases, there is also a direct object, and sometimes an indirect object. Additionally there will be extra amplifying data that is peculiar to the specific sentence. There are about 100 types of sentences covering internalized mental activities, status changes, sensing, moving, talking and affecting.

The format of the raw data is contained in binary files created while the simulation is running. The user can choose to have none, some or all (normally some or none during a study) of these sentences written to a listing file while the simulation is running. The user can also choose to drive the graphics display with these sentences using something we call circumstances. A circumstance is like a compound sentence, where the user can select which subjects, verbs, direct and indirect objects are part of the circumstance. Additionally the amplifying information and the game time can be used to further describe the circumstance.