Development In Business Simulation & Experiential Exercises, Volume 21, 1994

ACTIVITY DRIVEN TIME IN COMPUTERIZED GAMING SIMULATIONS

Precha Thavikulwat, Towson State University

ABSTRACT

Pedagogically and administratively critical to gaming simulations, the treatment of time can differ along three dimensions: scale, synchronization, and drive. Time can be fixed or flexibly scaled, synchronized or asynchronized among participants, and driven either by the administrator, the participants, the clock, or the level of activity. Fixed scaling is more easily programmed; flexibly scaling gives participants more freedom. Synchronization coerces participants unnaturally, but assures that mindless activity is not rewarded. Administrator-driven time is administratively inconvenient, participant-driven time is difficult to coda when decisions among participants are interdependent, and clock-driven time is inherently inadaptable to an irregular schedule. Activity-driven time can involve counting either decisions or accesses. Counting decisions encourages the churning of decisions; counting accesses encourages revolving access and collusive access, and can give rise to inadaptive pacing. Solutions to these problems are discussed. Treating time unconventionally enables gaming simulations to run for many periods without imposing excessive demands on either administrators or participants.

INTRODUCTION

How time is treated in a computerized gaming simulation circumscribes the issues the simulation can address, the procedures that participants must follow, and the work that the administrator must do. The pacing of time is critical. if time moves fast, participants' decision-making depend more on attitude, lass on analysis. If time moves continuously, *when* participants decide can be more consequential than *what* they decide. If time moves by itself, the administrator may lose control over the pace of time. In simulation, as in life, *time is of the essence.* Yet, despite its essential nature, the treatment of time in computerized gaming simulations remains little discussed.

Many computerized gaming simulations treat time as an entity that the administrator advances periodically at the administrator's convenience. Total enterprise business games (Keys & Biggs, 1990), in particular, generally treat time this way. Although relatively easy to program and to understand, this treatment of time imposes severe limits. When time advances at the administrator's convenience, it will not advance much because vary little of most administrator's time is convenient time. Thus, in working with a total enterprise simulation "of reasonably high complexity" (p. 44?), Rollier (1992) found only four to six time advances to be optimal for the entire duration of the simulation. Although four to six time advances may be low, even twice these numbers would not be large numbers.

Other computerized gaming simulations allow participants to advance time periodically at the participant's convenience. Many functional business games (Keys & Biggs, 1990) treat time this way. Although participant control of time is convenient for both participants and administrators, it is more difficult to program in simulations that allow for interdependence among participants' decisions.

Still other gaming simulations advance time by the clock (Chiesl, 1990). Like participant advancement of time, these are convenient for both participants and administrators, and era difficult to program only when interdependent decision-making is allowed. Even so, they era herd to adept to an irregular schedule of breaks, holidays, snow days, and the like.

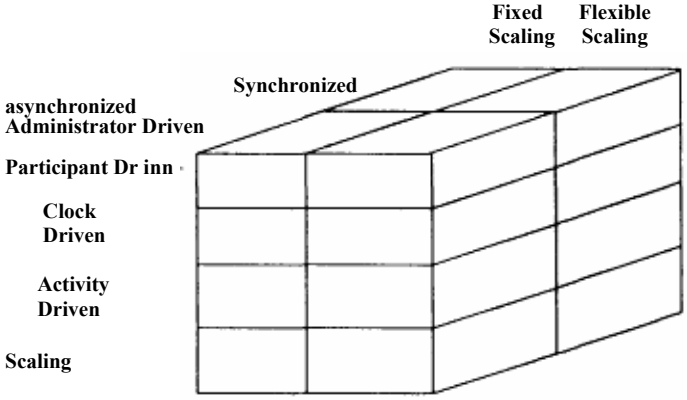
From the standpoint of administrative convenience end schedule adaptability, allowing time to advance with participant activity is attractive. Under this schema, control of time is vested in the collective activity of all participants. Time stands still when participants are inactive; time moves rapidly when participants era vary activity. The problem remains as to how the activities that advance time should be measured, and how participants can be prevented from advancing time by unnatural actions.

This paper explores the concept of activity-driven time within a three-dimensional framework. The paper discusses the measurement of activity, the routines that must be coded, and the problems that must be managed.

POSSIBILITIES FOR TREATING TIME

The treatment of time can differ along three dimensions: scale, synchronization, and drive. Scale refers to how time is segmented, the choice being between fixed and flexible scaling. Synchronization refers to the extant to which all participants perceive the same measure of time at the same instance in real time, the choice being between synchronized and asynchronized time. Drive refers to how time is advanced, the choice being among administrator driven, participant driven, clock driven, and activity driven. The three-dimensional framework is illustrated in Figure 1.

FIGURE 1
THREE—DIMENSIONAL FRAMEWORK FOR TREATING TIME



Scaling affects participants' freedom to choose the decision and reporting periods. Most business gaming simulations impose fixed scaling: the "shackling of the decision-making to the [constant] reporting or accounting cycle" (p. 114-115) that Teach (1990a) views as ridiculously unrealistic. Flexible scaling could involve merely allowing participants to select the length of successively shackled periods, as Chiesl (1990) has done with a marketing gaming simulation, or it also could involve complete freedom for participants to specify decision rules that are effective over any interval, as Patz (1990) has proposed, and complete freedom to specify any reporting interval.

The trade off between fixed end flexible scaling is a trade off between ease of coding for the programmer and freedom of choice for participants. Fixed scaling is easier to program than flexible scaling. Furthermore, when e large number of fixed time periods are allowed, when decisions of the past periods carry forward into current periods unless changed. and when participants may choose the periods for which they wish reports, the freedom offered by fixed scaling approaches that of flexible scaling.

Synchronization

Synchronization effect. interrelationships among participants. Gaming simulations designed for independent decision-making by participants ore typically asynchronous; those designed for dependent decision-making among participants are typically synchronous. Although synchronization of activities is common in training, the condition unnaturally coerces all participants to the same degree of experience at the. same instance in real time. Unsynchronized, a participant with six periods of experience might be in competition with another participant with three periods of experience. The participant with more experience generally has a competitive advantage—the reality of business.

On the other hand, synchronization assures that mindless activity is not reworded. In the extreme, an especially energetic participant in a freely asynchronous business simulation could triumph over a loss energetic one whose decision wore better planned, simply because the more energetic participant hod advanced through e greater number of production and marketing cycles over the same reel-time interval. To forestall this perversity, asynchronous simulations must have limits. The limits can simply consist of a minimum number of required periods and e maximum number of allowable periods in the competition.

Drive

Drive effects administrative convenience. For administrators, the ideally convenient gaming simulation requires no administrative action.

An administrator-driven simulation inconveniently requires the administrator to collect and process participants' decisions, leaving to the simulating program the straightforward task of merging interdependent decisions. The programmer's task is eased at the administrator's expense. Nevertheless, administrator-driven designs have had a long tradition and remain pervasive in total enterprise simulations. Like pencil and paper, the technology may be primitive but people understand how it works.

A participant-driven simulation advances time upon the participant's command. Convenient for both administrator and participant, this design is easily programmed provided the consequences of each participants decisions is independent of other participants decisions. To allow for interdependent decisions, however, either the programmer must code complex network-cognizant routines enabling the simulating program to manage multiple simultaneous runs on networked computers, or participants must be restricted to running the simulation on e single designated computer. Because the administrator will not be collecting decisions, they must be collected by an automated arrangement involving either the former or the letter condition.

A clock-driven simulation advances time in concert with the computer's internal clock. Chisel's (1990) marketing simulation incorporates this design. Convenient for both administrator and participants end not especially difficult to program, this design is common in popular video games. The design would be elegant in games that allow for interdependence among participants were it not for its inherent inadaptability to en irregular schedule. Without special arrangements, time running during e break period creates the anomalous condition that decisions made just prior to the break ore especially consequential because they remain fixed for the duration of the break. The condition rewords end punishes arbitrarily if the break results from an unforeseen event that prevent access to the simulation, such as a power outage. e fire alarm, or an adverse weather condition. Even in the absence of such events, the condition gives a substantial advantage to participants whose schedule conveniently enables them to input their decisions just before brooks.

An activity-driven simulation captures the advantages of the clock driven simulation without that design's inherent inadaptability. In concept, an activity-driven simulation advances time when participants ore active. It pauses during breaks, because brooks are intervals without activity. Activity-driven design, however, is difficult to program and less easily understood. Unless carefully coded end administered, they can be too slow, too fast, too erratic in pace, or too easily manipulated by the mischievous participant. resulting in a computer-hindered simulation rather then a computer-assisted one (Jones. 1991).

ACTIVITY-DRIVEN DESIGN

The first consideration in designing an activity-driven simulation is the measure of activity. At the most elemental level, each decision is an activity; at the most aggregated, each access by e participant of the simulation is an activity.

It the simulation measures activity by counting decisions, then safeguards must be in place to discourage participants from churning decisions: changing them repeatedly merely to advance time. Churning can be discouraged by a program that flags each category of decisions as they are made so that time is advanced only once f or any number of changes in each category over a set real-time interval. Even so, because time advances when decisions change, participants will be inclined to change decisions merely to advance the time, a misdirection of their efforts.

If the simulation measures activity by counting accesses, then safeguards must be in place to forestall revolving access and collusive access. Revolving access occurs when one participant accesses the simulation repeatedly merely to advance time. Collusive access occurs when a group of participants access the simulation reportedly in concert merely to advance time.

Revolving Access

Revolving access can be made ineffectual by the program recording the period of last access and advancing time only it the period of last access precedes the current period. For example, consider a business simulation that measures time in periods and subperiods. Assume that e period is composed of five subperiods, that time advances in increments of one subperiod. and that each production and accounting cycle is one period. Assume further that the current time is subperiod 1 of period 20. When the simulation is accessed by a participant who last accessed it in period 15, the program advances the current time to subperiod 2 of period 20, and updates the participants last-access period to 20. Should the participant quit the simulation end reassess it immediately after, the program, noting that the participants last-access period of 20 now matches the current period, will leave the current time unchanged.

Collusive Access

Collusive access is possible only when the number of conspirators exceed the number of subperiods in a period. In collusive access,

The conspirators access the simulation in turn, quit immediately after gaining access, reassess it again, quit again, repeating the cycle merely to keep advancing the time. Continuing the example above, six conspirators whose last-access period was less that the current period could by accessing the simulation in turn quickly advance the time until it reaches subperiod 0 of period 21. If each conspirator quits the simulation immediately after gaining access, all excepting the lest will have a lest-access period of 20 or earlier. Thus, the five conspirators with last-access periods lower that 21 could reassess the simulation in turn again end advance the time five subperiods until it reaches subperiod 0 of period 22. The cycle can be repeated indefinitely.

The obvious solution to collusive access is to raise the number of subperiods in a period to the point where collusion becomes impractical. This slows the pace of the simulation, possibly to the point where it could be intolerable in small classes. In a class of 20, for example, if the number of subperiods in a period were set to 9, half the class would have to access the simulation before it could advance e period. If the average participant accesses the simulation twice a week, the simulation's time would advance by no more than 40 subperiods a week, thus limiting the pace of the simulation to no more than 4.4 (40 divided by 9) periods a week.

To enable a faster pace with a large number of subperiods, the program can be coded to advance time one subperiod when accessed by a participant whose last-access period is one less than the current period, and two subperiods when accessed by one whose last-access period is two or more periods less than the current period. Besides enabling a faster pace, this scheme gives a make-up advantage to participants who are less active. Although the less active access the simulation fewer times, each access generally advances time twice as much, and thus is about twice as consequential as an access of the more active. The make-up advantage limits the value of sheer activity, a favorable byproduct.

Inadaptive Pacing

A final problem emerges when a simulation that measures activity by counting accesses is installed to allow simultaneous access on a local area network. In this setting, even if the number of participants simultaneously accessing the simulation exceeds the number of subperiods in a period, the simulation's time will not advance unless at least a number exceeding the number of subperiods in a period quit end reassess the simulation. The problem arises because the administrator may occasionally wish to have the entire class participate in the simulation simultaneously, and to have the simulation's time advance rapidly in that interval.

This problem of inadaptive pacing can be solved by having the simulation program advance time by the clock when the number of participants simultaneously active exceeds the number of subperiods, that is, when e *quorum* is obtained. Each of the simultaneously running program on the network detects a quorum by frequently reading the time values from a common file, and noting when the period value advances independent of its own actions. When the quorum is detected, the program starts its clock on a countdown. At the end of the countdown, the program advances the time and resets its countdown, provided another program has not already done so in the interval. Otherwise, the program merely resets its countdown without advancing the time.

For example, assume the countdown interval is ten minutes, the number of subperiods in a period is nine, end the current time is subperiod 0 of period 23 in the absence of participant access. Assume further than ten participants proceed to access the simulation at about the same time, and that all ten have simultaneous access. If each participant access advances time one subperiod,

The first participant access advances the time to subperiod 1 of period 23; the second, to subperiod 2 of period 23; until the tenth, which advances it from subperiod 9 of period 23 to subperiod 0 of period 24. Shortly afterwards, each of the programs activated by the nine earlier-accessing participants, recognizing that the period number has changed from 23 to 24 independent of its own action, independently starts a ten-minute countdown. The first of the nine reaching the end of its countdown advances the clock to subperiod 1 of period 23 and resets its countdown. The remaining eight, upon noting that the time has again advanced independent of their actions, resets their countdowns only.

Provided the countdown interval is not noticeably long, participants will see no gain in cycling access to advance the time. Quitting a program and relaunching it takes effort, the loss of which may not be sufficiently compensated by any resulting acceleration of time.

CONCLUSION

Activity-driven simulations are convenient for simulation participants end administrators, and are workable. Problems of pace and mischievous manipulation can be solved by proper programming. Activity-driven design, however, is only one of four drive-design possibilities, and drive design is only one of three dimensions of how time can be treated in gaming simulations.

A gaming simulation must first and foremost address the right issues. Although time itself generally is not a pedagogical issue, its treatment limits the issues that can be addressed. Critical to business strategy, for example, is the issue of forgoing short-term profitability for long term returns. To address this issue, the gaming simulation must allow a clear distinction between the short and long term. Such a distinction can hardly be made in business simulations that are run for only a few periods. As Teach (1990b) has observed, "Evaluating students who participate in business games on the basis of cumulative profits over a designated number of plays, typically 8 to 1 6 simulated periods, emphasizes the short-term perspective and does not provide incentives f or long-term planning" (p. 17). Activity-driven time solves this problem by allowing the number of periods to increase by an order of magnitude, without imposing excessive demands on either administrators or participants.

REFERENCES

Chiesl, N. E. (1990) Interactive real time simulation In J. W. Gentry (Ed.), *Guide to business gaming and experiential learning (ABSEL).* (p 14 1-158). East Brunswick, NJ: Nichols/GP Publishing.

Jones, K. (1991) Using computer-assisted simulations and avoiding computer-hindered simulations. *Simulation & Gaming, 22,* 234-238.

Keys J. B., & Biggs, W. D. (1990) A review of business games. In J. W. Gentry (Ed.), *Guide to business gaming and experiential /earning (ABSEL),* (pp. 48-73). East Brunswick, NJ: Nichols/GP Publishing.

Patz, A. L. (1990) Open system simulation In J. W. Gentry (Ed.), *Guide to business gaming and experiential learning (ABSEL),* (pp. 159-176). East Brunswick, NJ: Nichols/GP Publishing.

Rollier, B (1992) Observations of a corporate facilitator *Simulation & Gaming, 23,* 442-456

Teach R. D. (1990a). Designing business simulations. In J. W. Gentry (Ed.), *Guide to business gaming and experiential learning (ABSEL)* (pp. 93-116). East Brunswick, NJ: Nichols/GP Publishing.

Teach R. D. (1990b). Profits: The false prophet in business gaming. *Simulation&Gaming,27,*12-26.