

Carlo Molina

L'asimmetria ontologica gamer-software

(da Carlo Molina, *Age of Empires. La simulazione giocata della vita*, Edizioni Unicopli, Milano, 2003; i documenti pubblicati possono essere utilizzati per fini esclusivamente non commerciali; le citazioni devono recare sempre il riferimento bibliografico, l'URL e il copyright).

Quando è intento a giocare a un RTS come *The Conquerors*, l'utente umano decide e agisce non solo per deduzione, ma segue spesso l'intuito, impara dai propri errori, escogita soluzioni inimmaginabili per il software. Quando può dettare le regole del game, egli può operare secondo motivazioni di ordine irrazionale, e può interpretare i segni prodotti dalle macchinazioni del software in modo del tutto "arbitrario" [...]. Non si può dire, invece, che il software esegua un'autentica produzione di senso: la sua opera è una semiosi "vicaria", un insieme di procedure di elaborazione di dati sulla base di regole prefissate. Il software genera, rileva e media *ogni* segno della comunicazione – ciò che l'utente non è fisicamente in grado di fare – ma non ha "coscienza" di ciò che comunica, non sa quello che fa. O perlomeno non sa come può "sapere" l'uomo. Il software cogita per algoritmi, secondo un procedimento sillogistico inalterabile: il dominio del suo possibile è sintetizzato nella regola if - then - endif. Non può imparare nulla che non conosca già, non può correggere i propri errori quando non è già programmato esplicitamente per farlo.

Esistono, naturalmente, altre differenze macroscopiche. L'uomo che gioca con la macchina è costituzionalmente libero di giocherellare senza un fine, scapricciarsi con lo spirito del gioco, "prendersi gioco" del software – non già ammazzando tutti i villagers che si aggirano per la mappa (che è il divertimento dell'ingenuo software emulator), ma comandando al software di far dire e fare ai villagers cose bizzarre, per spasso o virtuosismo (il divertimento del gamer developer).

Il software non può evolversi che in virtù di modifiche operate dal developer e innesti operati dall'utente: patches, mods, add-ons, updates e upgrades. È ontologicamente inetto (benché tecnicamente predisposto) a maturare: esso esplora ma non ha curiosità, opera alla perfezione ma non conosce ironia, non sa improvvisare, non può meditare quando è spento, è incapace di "pensiero laterale" autenticamente creativo. Paradossalmente, però, può dimostrarsi capace di *comportamenti* più complessi del gamer, "escogitare" soluzioni che all'utente sembrano imprevedibili, stupire, irretire nel gioco, sconfiggere il più versato avversario umano. Per quali motivi?

Se è vero che il software non sa improvvisare, è vero tuttavia che è così rapido nell'esecuzione delle istruzioni che, a giudicare dall'output, è *come se improvvisasse*. Il suo feedback è automatico e istantaneo, quello del gamer non sempre. Le deduzioni algoritmiche della macchina possono battere letteralmente "sul tempo" le più brillanti intuizioni umane. All'interno del suo mondo autisticamente chiuso, il

software può vedere = sapere = potere di più, cioè può disporre di dati continuamente aggiornati sulla mappa e sulle interferenze dell'avversario umano, ne dispone in quantità maggiore del gamer, vi accede e li recupera dalla memoria in modo tempestivo e mirato, elabora non ciò che “vede” ma ciò che ha una certa funzione. Il software non vede per immagini ma per funzionalità: anche se è cieco, spesso, a giudicare dall'output, è *come se vedesse*.

Il software ha anche una diversa “cognizione” del tempo. Il passato, ciò che è *dato*, è per esso sempre presente, in quanto è subito accessibile e non subisce distorsioni e falsificazioni sedimentando nel ricordo. Al contrario, l'uomo può dimenticare o ricordarsi male ciò che non vede da qualche tempo [...]; può ignorare involontariamente ciò che vede molto spesso e che, per così dire, gli diventa trasparente per abitudine; può non “tenere presente” ciò che si trova temporaneamente fuori dal suo campo visivo, specie quando il software gli impone l'equazione “fuori quadro = fuori campo” (Eugeni 1996:198) e, incalzandolo, non gli lascia tempo per “farsi un quadro” di ciò che sta accadendo altrove.

Il software non è soggetto a una curva periodica della capacità di concentrazione, mentre l'uomo può distrarsi e perdere il ritmo, perché è stanco, gli è venuta fame, sta squillando il telefono, o perché la mappa è troppo grande, per *overload* cognitivo. Le mosse del software sulla mappa non sono vincolate alla coordinazione occhio / mano / pensiero, ma si connotano come interventi di/su una matrice di bit. Nel caso peggiore, che è norma e necessità nei videogames [...], i CP possono godere di risorse superiori a quelle che l'utente riceve in dotazione per *default*; spesso all'uomo non resta che bilanciare l'inganno con un altro inganno, il *cheat code* che lo stesso software gli mette a disposizione quando egli, da solo, non ce la fa.

Tra i *parigrado* del gioco per computer, gamer e game software, sussiste un'asimmetria ontologica: la loro parità è astratta e paradossale come quella dei due mitici sfidanti, Achille e la tartaruga. Il senso del videogame è che in esso si tenta di ricomporre questa impossibile parità. La macchina non sa di essere una macchina perché non ha coscienza di ciò che è altro da sé, non sa di giocare con un uomo perché ignora che cosa sia un uomo. L'uomo, dal canto suo, ha un atteggiamento curioso: egli sa di essere umano, sa che cos'è una macchina e dunque che *quella* è una macchina – e nondimeno non vuole saperne di avere di fronte *soltanto* una macchina. Il gamer vuole misurarsi con una macchina che sia in tutto pari all'uomo, e chiede al developer di costruirla per lui. La macchina non concepisce contraddizione, l'uomo vuole ricomporre la contraddizione cancellandola. Egli vuole che il software cogiti e agisca a sua immagine e somiglianza, ed è questo che rende la modalità di gioco *single-player* un oggetto di studio così affascinante, e il videogame un oggetto – almeno in teoria – perfettibile ad infinitum.

[...]

*La Mantide di Wiener e la Falena di Hopper*¹ (Bugs e smart agents)

La coordinazione di Computer Graphics e tecniche di A-Life, che sono i poli generativi della simulazione negli RTS, si deve confrontare con due problemi di fondo: il primo è che non sempre l'algoritmo riesce a dissimulare di essere limitato e a simulare una casualità *plausibile*; il secondo è che non sempre il referente “organico” che la A-Life cerca di ridurre a modello è creativo e imprevedibile come, invece, ci si aspetta che sia la sua simulazione. Il software fa finta di giocare come un uomo, ma non è detto che ci riesca. Il motivo per cui perde (quando non è esplicitamente programmato per opporre la minima resistenza al gamer) è che le sue mosse sono troppo prevedibili. Per mezzo delle proprie proiezioni simulacrali (CP agents), esso “si ostina” a colpire il nemico sempre nello stesso punto, non sa veramente “escogitare” strategie creative, si limita a recitare scripts che, per quanto complessi, modulari e scalabili, restano scritti una volta per tutte.

In *The Conquerors*, per es., i CP nemici che scoprono dove si trova il villaggio utente costruiscono – quasi invariabilmente – *Barracks* (Alloggiamenti della fanteria) e Castelli nei suoi pressi, lasciando spesso sguarnita la difesa del proprio insediamento e diventando vulnerabili per troppa aggressività [...]. Altre volte accade che un drappello di unità nemiche attraversi il villaggio utente per recarsi contro un obiettivo prefissato che si trova all'altro capo della mappa: prima di giungere a destinazione, la truppa finisce falciata... dagli arcieri che l'utente ha posizionato sul percorso dopo che il CP nemico aveva perlustrato quell'area, ma prima che esso sferrasse l'attacco. Come scrive Norbert Wiener ne *L'Uso Umano degli Esseri Umani*, “una colonna di formiche seguirà facilmente il capofila, ma se non c'è capofila e la colonna si muove in senso circolare, le formiche continueranno a girare in circolo fino ad essere esaurite dalla stanchezza” (Wiener 1966:83). Il “capofila” è, in questo caso, un AI script non abbastanza sofisticato da consentire un aggiornamento istantaneo delle informazioni sulla mappa, cioè basato sulle LOS attuali delle unità militari in movimento e non sulla LOS dello scout che ha esplorato la mappa in precedenza, quando il CP ha individuato l'obiettivo da distruggere. Si verifica qui una sorta di paradosso temporale: il software esegue un comando automatico (attaccare l'obiettivo) che è logicamente e cronologicamente sequenziale all'input (obiettivo rilevato) e finalizzato a un preciso output (eliminazione dell'obiettivo), ma di fatto, esso opera in base a dati già scaduti, *agisce nel passato*, come se il passato fosse ancora presente. Gli agents “morituri” camminano in un tempo mappato come nebbia di guerra, e il software non sa cambiare tattica in tempo “reale” [...].

“In generale fra gli insetti”, seguita Wiener, “anche quando un insetto da preda, come la mantide, ha già divorato l'addome della sua vittima, cosicché questa è ormai *in articulo mortis*, ciò che resta dell'animale continuerà nel suo compito di nutrirsi come se nulla fosse accaduto” (ib.) [...]. In generale tra gli agents

¹ “Relay n°. 70, Pannello F, (falena) nel relay. Primo caso effettivo di bug rilevato”. Con questa nota su un foglio di registro, Grace Murray Hopper (l'inventrice del linguaggio COBOL) segnalò il ritrovamento di una falena nei circuiti del calcolatore Mark II. L'insetto era rimasto “fulminato” nella macchina, causandone il crash. Grace Hopper lo incollò con dell'adesivo accanto alla nota: è la prima testimonianza di un “bug” nella storia della Computer Science.

costruiti con tecniche di A-Life, come i villagers di *Age of Kings*, sussiste una certa coazione a ripetere i propri “automatismi a previsione” anche in situazioni che l’uomo interpreta come assurde. Il motivo è che tanto gli insetti, quanto gli smart agents, difettano dei percetti che rendono possibili certe capacità di interpretazione, oppure elaborano “troppo tardi” la retroazione adeguata all’ultimo percetto rilevato dall’ambiente (naturale o simulato che sia). La mosca si ostina a sbattere contro il vetro perché non sa che cosa sia il vetro, e non lo sa perché, probabilmente, non è in grado di vederlo controluce²; il *Man at Arms* di un CP si ostina a colpire il *Market* utente perché non sa che, contemporaneamente, un villager utente sta riparando l’edificio da un altro lato della costruzione (sicché esso resta integro), e non lo può sapere, perché la sua priorità è quella di distruggere il Market e gli manca l’intuizione che, se eliminasse prima il villager, in definitiva sarebbe più rapida anche la distruzione del Market. La zanzara si lancia nella luce azzurra che la arrosterà viva perché quello spettro luminoso è per essa un’irresistibile attrattiva, e il cavaliere CP si lancia in un attacco suicida contro il Castello utente (che lo tempesterà di frecce) perché questo è ciò che il software gli comanda di fare a prescindere dalle circostanze.

“Nella sua forma più semplice, il principio della retroazione significa che il comportamento viene periodicamente confrontato con il risultato da conseguire, e che il successo o il fallimento di questo risultato modifica il comportamento futuro. La sua funzione è di rendere il comportamento di un individuo o di una macchina relativamente indipendente dalle cosiddette condizioni di carico” (Wiener 1966:84). Si può affermare che la retroazione della mosca e della zanzara è inadeguata perché esse sono costituzionalmente impossibilitate a “concepire” gli oggetti del vetro trasparente e della resistenza elettrica, oppure a “interpretarne” la trasparenza, e la fluorescenza, come gli uomini interpretano questi segni in *questo* contesto. La retroazione difettosa degli agents, invece, dipende in sostanza da un ritardo nell’aggiornamento dei percetti (il software processa correttamente dei dati obsoleti) o dall’assegnazione di priorità esecutive troppo rigide e non più adeguate alle mutate condizioni del mondo simulato (dunque, anche se in modo indiretto, il feedback difettoso dipende ancora da un ritardo nell’aggiornamento dei percetti).

Sempre Wiener, in *Cybernetics*, scrive che “prevedere il futuro di una curva significa eseguire una certa operazione sul suo passato” (Wiener 1968:28). Ciò che, talvolta, sembra mancare perfino al software di uno strategy game in “tempo reale”, è la capacità di operare sul presente. In altre parole, la A-Life simula sì dei comportamenti, ma si tratta di *comportamenti senza apprendimento*.

[...] Il paradosso del software che simula il comportamento di un sistema vivente è che il software funziona *di per sé* come un sistema autistico, chiuso in se stesso, nel quale tutte le combinazioni *probabili* delle esperienze funzionali al mantenimento della propria circolarità sono già note, tutte le modifiche comportamentali previste, tutti i comportamenti assemblati in base a certe premesse – ma questo

² Ma solo un istante più tardi, quella mosca che si accaniva contro il vetro invisibile per volare via, cammina noncurante sulla stessa superficie, disposta però “orizzontalmente” sotto di essa e dunque percepita in modo diverso, come se fosse un *oggetto* diverso.

sistema, che in sé sarebbe perfettamente funzionale, è a sua volta inglobato in un sistema più ampio, che include un gamer cui bisogna per forza lasciare aperta una “backdoor”. Il software, una volta avviato, è capace di badare a sé: esso innesca ed “esplode” tutte le combinazioni del proprio caos deterministico, emulando il sistema olistico di un pianeta vivente ma isolato in se stesso ove innumerevoli forme viventi sono generate, si riproducono, combattono tra loro e periscono. Il gamer “accade” in questo sistema chiuso come un bombardamento di meteoriti dal cosmo: la sua interferenza, quando è veramente creativa, è un *deep impact*, l'imprevisto contro il quale per il software non esiste possibilità di comportarsi.

Il game nel suo insieme è un sistema in equilibrio precario, *una monade con finestra*, e in questo senso è la simulazione ideale di un organismo o ecosistema condannato a morte. [...] Il gamer, sottosistema-uomo, collide con la macchina precisamente come la falena scoperta da Grace Hopper fece col calcolatore Mark II: nel sistema-game egli è il dato disfunzionale, l'errore logico – idealmente, il *Bug* che causa l'arresto. Come fossero bugs, egli orienta i propri agents contro quelli del software, ed estende la propria nicchia di interferenza nel sistema-game sfruttando le risorse che il sottosistema-macchina gli mette a disposizione. Così il sistema di vita artificiale del gioco, l'organismo algoritmico che simula comportamenti “prevedibilmente imprevedibili” ma è incapace di adattarsi all'imprevisto *vero*, precipita e muore, ed è il game over.