

Embedding Information into Game Worlds to Improve Interactive Intelligence

G. Michael Youngblood, Frederick W.P. Heckel, D. Hunter Hale,
and Priyesh N. Dixit

Abstract Current game worlds are visually rich but information poor – particularly poor from the artificial intelligence (AI) point of view. Where the player sees a rich visual representation of 3D objects, internally these are just very sparsely described collections of points in space. Tools for advanced world creation, character modeling, animation, and advancements in computer graphics have brought us into the age of near photo-realistic interaction; however, these interactions are still very limited in comparison to the real world, and the information is presented overwhelmingly for the player, packaged for the Graphics Processing Unit (GPU) with little reflection or structure suitable for use by AI systems. This problem of a lack of rich information suitable for consumption by the game AI often limits the true potential for deeper levels of interaction that are becoming more in-demand by game players. This chapter presents a number of tools and techniques, which are being used to improve the embedded information contained in immersive game worlds. Symbolic annotation of the environmental elements, advanced spatial decomposition, calculating the information value of the surfaces in an interactive environment, and visual analysis form the core tools and information generators of our Common Games Understanding and Learning (CGUL) Toolkit. Using these tools to incorporate information into the game design and development process can help create information-rich interactive worlds. AI developers can work with these environmental information elements to improve non-player character (NPC) interactions both with the player and the environment, enhancing interaction, and leading to new possibilities such as meaningful in-game learning and character portability. Case studies from two different projects using these techniques provide some additional insight and reference as to how these techniques have been incorporated into current game AI and research.

G.M. Youngblood (✉)
The University of North Carolina at Charlotte, 9201 University City Blvd.,
Charlotte, NC 28223, USA
e-mail: youngbld@uncc.edu

1 Worlds of Visual Richness and Information Poverty

Creating virtual worlds for games, simulations, interactions, observations, or any of the varied applications of these constructs is becoming a common task for the masses. This type of creation activity was once the purview of skilled modelers and trained digital artisans, but since the turn of the twenty-first century the barrier to entry has been significantly lowered. Creating virtual worlds is more accessible now than in times past, but even more amazing is that the graphical fidelity has increased while making this task easier. It has been the creation of tools that has facilitated this change—tools with different specialties for each part of the virtual world creation pipeline.

The first step of the pipeline is modeling, which is the creation of three-dimensional (3D) virtual objects in a tool that allows the combination and contortion of primitive shapes into complex models. Models defined by dozens to millions of vertices can be created using modeling tools such as Autodesk's Maya (www.autodesk.com/maya), 3D StudioMax (www.autodesk.com/3dsmax), or Softimage (www.autodesk.com/softimage); NewTek's Lightwave 3D (www.newtek.com/lightwave); the open source Blender (www.blender.org), or the freely available Google SketchUp (sketchup.google.com). These tools provide a means for modeling, animating (key framing and interpolating motion between frames usually based on articulated skeletons associated within the models), and rendering (the final target visual representation). In this chapter, we will talk about the modeling and animation aspects, but will assume that rendering will be done in the target interactive environment (i.e., the video game or simulation). We focus our terminology on polygonal modeling, but other types of modeling (e.g., NURBS) have similar properties and the reader should apply the same ideas regardless of the underlying modeling paradigm.

Modeling includes more elements than just the arrangements of vertices in space to create the desired geometry of the object being modeled. It also includes more than just static geometry (e.g., buildings and terrain) as many elements are specifically created to be dynamic (e.g., doors in buildings, characters, vehicles, and the ever-popular gun turrets). Models are textured with colors or imagery (i.e., 2D art) which is created separately and then *projected* upon the surface of the model. The base texture, referred to as the *texture*, is defined as the 2D pixel array containing the colors to be applied to the target model by a projection (e.g., spherical, cylindrical, or UV coordinates that map to specific vertices of the geometry). Other types of texture *maps* are also applied using the same projection. These include normal maps (a.k.a., bump maps) that define the surface height details by defining the normals, or slopes, of the surface allowing for light and shadow to be rendered appropriately giving depth to the base texture (height maps can also be used), specular maps that define the shininess of the base texture at specific locations, and diffuse maps that define how light reflects both in color and intensity. Other types of maps exist such as hit maps used to determine locations of damage, maps to aid in rendering infrared images showing the heat signatures of modeled objects, and other maps useful for representing various phenomena on the surface of the model. Textures and

texture maps are typically created with 2D image tools such as Adobe Photoshop (www.adobe.com/photoshop) and/or The Gimp (www.gimp.org), but are increasingly being made through procedural tools such as Allegorithmic's MapZone (www.mapzoneeditor.com). The model is usually saved as a mesh containing the vertex and edge information along with the texture and texture map file names and their projections onto the model. Thus, a model is saved in the model format along with the set of textures applied to that model (multiple files).

Dynamic models move and usually have one or more articulated parts. The modeling tools often allow the user to specify the skeletal structures of objects (any objects, not just simulated organic creatures) and define how the joints operate. Using a timeline, the user can create key framed poses to animate along the defined degrees of freedom. Forward kinematics is used to calculate the positions of the object between the key frames. Location and orientation in space can also be set in addition to the skeletal pose. Another type of texture map is used to define the weights of the mesh shell around the model skeleton specifying how it deforms with movement. The *animator* usually names each animation sequence as a set of frame numbers, which are typically saved along with the model in the model file. This process can also be performed with mesh vertex movement over time (i.e., vertex animation instead of skeletal animation).

The modeling tool can also make *scenes* that consist of several models arranged in space. However, these scenes are typically used for rendering and not for virtual world creation for games or simulation (although they could be used for this task) using the tool's renderer for movie or image generation. A game engine-specific *level editor* is typically used to assemble components and models into a world for interaction. Tools such as Ambiera's Irredit (www.ambiera.com/irredit) for the Irrlicht 3D Engine (irrlicht.sourceforge.net); QuArK (quark.planetquake.gamespy.com) for about 45 popular games using the Source (source.valvesoftware.com), Id Tech (www.idsoftware.com/business/technology), and other engines, as well as other editors often packaged with the PC version of a game that are used to create the immersive virtual worlds generated by those engines. Beyond just the arrangement of models in space, these level editors usually allow for some geometry creation as well as setting up light, dynamic elements (e.g., doors), and spatial logic (e.g., an area trigger that will open a gate when the player enters). Engine-focused level editors consume models from a modeler and allow the designer to assemble the interactive virtual world. The accessibility of these tools has inspired many individuals to create content and even whole games. The Mod(ification) Community consists of thousands of people worldwide who as a hobby make new and interesting virtual worlds for others to experience. Even online in persistent worlds such as Second Life (secondlife.com) user-generated content including buildings, clothing, and just about anything imaginable is available and is being created by the user, not the developer.

The problem with all of these tools in the games and simulation content creation pipeline spanning texture, texture map, model, animation, and level creation is the incorporation of information elements with each of the products. Creating

virtual worlds for people has been the primary task for which these tools have been created, and they do a tremendous job at providing visually rich worlds for humans. However, humans are not the only elements interacting in these worlds as more and more intelligent characters are acting as friends, foes, proxies, and other integral roles within these virtual worlds. In games, particularly single player games, the non-player characters (NPCs) are the major actors in creating the drama and interactions to make it a game. Even in multi-player games, NPCs (also known as bots) round out teams and increase the immersion and action in the first- and third-person player genres. In the strategy genres (i.e., real-time strategy, turn-based strategy), the NPCs take and perform actions often at a high level of description from the players – making everything happen in the game. The core elements of interactivity in these environments are currently very controlled and explicitly designed and implemented, but the demand for increased interactivity and deeper immersion is on the rise. One mechanism to increase the fidelity of interaction in virtual worlds is to improve the intelligence and behavior spectrum for the NPCs [i.e., improve the artificial intelligence (AI) driving the game characters]. The first place to start is to give them more information about the world in which they are interacting. We must move from our current information-poor worlds to information-rich worlds, and since the user is becoming a key contributor, we must provide them with the ability to improve the information of their contributions as well.

The current problem of perception is illustrated in Fig. 1. All of the models, textures, and the arrangement rendered by the engine display objects recognizable to a human, but to the machine they appear as a set of points in space with some connectivity and set relationships. The machine cannot easily determine what it is perceiving is a house or a tree or a dog or a spaceship or anything really. This is the same problem that machine vision researchers are trying to solve – recognizing objects from images. However, in virtual worlds we do not have to solve this problem, all we need to do is to incorporate some knowledge engineering tasks into the content creation pipeline, namely, we need to have the creator add information elements that are attached and persist with the models, textures, and worlds.

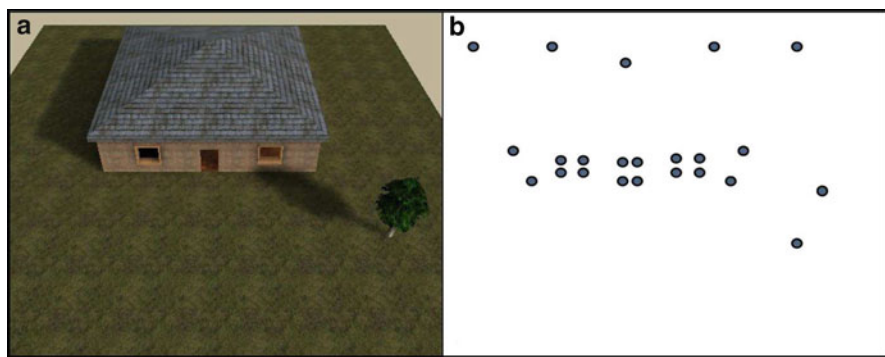


Fig. 1 In a virtual world, the human easily perceives the items in (a) as a house and a tree, but to the machine the world appears more like (b), as a collection of points in space

For example, when the modeler makes an oak tree they need to annotate it as an *oak tree*, or perhaps more specifically a *post oak tree from Texas*. The texture and texture maps could also contain this type of symbolic information, which could be consumed by intelligent entities in the virtual world.

Our focus is on techniques and tools that can be used to add information to elements of a virtual world targeted for games and simulations, but this work can certainly be more broadly impacting. In the sections ahead, we present tools and techniques for embedding information into virtual world objects (both static and dynamic) and their surfaces that lead to improved knowledge for consuming and interacting with elements of these environments with the goal that artists from professionals to hobbyists can incorporate these into their virtual world creation pipeline.

1.1 The Need for an Information-Rich World

Our own research work is in the field of game artificial intelligence (game AI) where we are in pursuit of human-level intelligence at both the character level and the system level. There is much work to do in this area, but we are strong supporters of the idea that interactive computer games are an opportunity area for exploring techniques and theories leading to human-level AI [14, 16]. One of the key problems in our current information-poor virtual worlds is that the playing field is clearly not level for AI characters (NPCs), especially when competing with humans in the same space. Hence, in practice game AI is often implemented with a level of cheating such as using perfect information (i.e., complete world knowledge) and unseen perception-aiding techniques (e.g., breadcrumb trails to follow the motion of characters). The first step is evening perception, which will allow artificial characters similar sensory perception of the world that the human players have. The human brain is a tremendously good pattern matcher to known objects; so most players are entering into these virtual worlds with 2–80+ years of real world experience. This is a little hard to compete with from the game AI character perspective, especially since the field of AI is just a little over 50 years old itself. However, what is really needed is to at least present similar information to both the human and non-human players, in this case, presenting object information and spatial relationships in a more readily processed manner to the NPCs – the same information the human gets with their eyes and from object recognition with the knowledge they acquired over time. We can then leverage the intelligence of the game AI designer/developer to encode proper responses to perception.

An interesting trend over the past few years has been the emergence of 3D object warehouses that offer downloadable content for free (sketchup.google.com/3dwarehouse) or purchase (turbosquid.com). Sites such as these make tens of thousands of models available for use in virtual worlds, rendered scenes, and any use imaginable. The vast majority of the models made are created by freelance, hobbyist, and even amateur modelers (many are students learning this art form).

It is commonplace to just buy off-the-shelf models to use in projects; however, this continues to propagate and even worsen the problem of information poverty. Given that embedding information into these creations is currently not a standard practice or even a step thought about in the process, there is now a massive, growing number of potential virtual world elements that have no information annotations. Furthermore, once the models are sold as a product, contact with the original creator diminishes and so does the full meaning and inspiration information elements associated with the creation and the creation process.

From a character AI development standpoint, information-rich elements assembled as the worlds are created can flow their specific information into tools for behavior development and be presented in-game to the AI character as perceptions of those objects. This is the key to helping create better AI-driven characters, providing a flow of meaningful information to better perceive the world objects, determining how to apply actions from the repertoire of available actions of the AI characters, and gaining some understanding from the elements in their world. We all make better decisions when better informed.

Acquiring knowledge over time and being able to apply previous knowledge from old problems toward solving new problems is the process of learning. Having more information-rich worlds will also allow NPCs the ability to apply machine learning techniques over observed sample data from observing other players (human and artificial) as well as itself. Being able to expand the knowledge of characters and allowing them to adapt over time is an exciting potential for interactive characters, which could increase the immersive experiences of virtual worlds.

The sections moving forward are being written as a way to educate the current and new masses of modelers and tool builders with regard to elements important for game AI, providing insight through a particular lens for fellow game AI practitioners, and establishing a way to move toward common approaches of encapsulating information elements for interactive (game) AI in virtual worlds. This is also a blueprint for the interactive AI researcher and developer to better understand and help guide the incorporation and usage of information-tagged elements. Beyond just information tagging, we will also discuss ways in which the information could be organized and how additional information can be processed by leveraging information-tagged elements.

1.2 Overview

This chapter presents a cohesive and integrated view of our key approaches, ideas, and work in adding, generating, and using information in interactive virtual worlds. We will begin with basic information tagging to objects, present a method for organizing information, discuss the generation of knowledge from processing world information, and then provide some insight into research projects that have incorporated these techniques and ideas. Throughout the discussion key principles will be highlighted to allow for easy incorporation into one's own workflow.

2 Embedding Information

The fortunate byproduct of the the adoption of standards such as XML is that ASCII-based, readable formats are in vogue.

This section will present a number of ideas with references to techniques and tools that can be embedded into the comments or extensible sections of many file formats. Keeping information as close to the original element as possible is preferred, but care should be taken to make sure that the added information is preserved when editing the source.

Key Principle: Prefer embedding information as comments or extensions in the source files over creating separate information files.

2.1 Information at the Object Level

The lowest level of embedding information should be at the object level. Each model should be tagged with information that describes what it is (i.e., nouns) and the properties of the object (i.e., adjectives). As each object also provides some set of actionable properties (i.e., a verb can be applied to it), an affordance that defines that property between the object and an actor (e.g., AI character) should also be added. Affordances are often described merely as the action(s) that can be applied (e.g., an object an actor can sit upon is described as having the affordance “sit”). The term *affordance* was introduced by Gibson to describe all “action possibilities” available in the environment [6]. For example, if there is an unoccupied chair sitting upright in the world, there is also an opportunity for sitting in the chair. In game AI, this can be used to sidestep the symbol grounding problem, at least in virtual environments; rather than define the quality of *chairness*, we merely require that anything that can be used as a chair should have a “sit” affordance defined.

Affordances can be directly added during creation or level design to objects in games to provide information about what sorts of actions can be taken using the objects to which they are attached. Affordances may also exist separate from objects in the environment. If a character possesses a hang glider, for example, it will state that it can be used as a tool to enable flight, but that it requires a launch point. A launch point affordance could then be placed at spot on a cliff edge. By the level designer placing this information in the world, the character AI needs no additional knowledge about launch points, or even the difference between a hang glider or a paraglider. Affordances have been successfully used in several games and simulations, including the popular Sims series from Electronic Arts [4].

Formally, we can define the information to be stored in each object model as the 3-tuple (ν, α, Δ) where ν is the set of all describing nouns (descriptions), α is the set of all describing adjectives (attributes), and Δ (change) is the set of all affordances. Note that applying an action to an object may change the state of an object (e.g., change its location or potentially damage it). An embedded synonyms list

Fig. 2 Objects tagged with Cyc constants (identifiable by the hash-dollar prefix) in a virtual world. Note how more perceivable the world is from an information view. Note that not all objects are labeled in this figure, but from what is, the character AI could start to reason about this world. Canine AI could head toward the tree, character AI could head toward the door and enter the house, and so forth – a world of many possibilities



could also be kept for each description to facilitate better understanding and faster searching for matching concepts or actions, or a lookup using a tool such as WordNet (wordnet.princeton.edu) could be used. Foreign language equivalents could also be embedded or looked up using commonly available translators.

The descriptive tuple elements can be filled out using the standard words and grammar of the person providing the tags (preferably the creator of the object) or using descriptive elements from a common sense knowledge base and a relational ontology such as those provided by ConceptNet (csc.media.mit.edu/conceptnet) or Research Cyc (research.cyc.com). An example of Cyc constants (nouns) tagged to object models in a virtual world can be seen in Fig. 2.

Some objects are more complex to model than others, and it is commonplace to model complex objects as a series of connected parts – particularly objects such as articulated parts (e.g., the turret of a tank is typically modeled separately from the body). Each part should contain its own set of annotations as well as the whole object containing a traceable reference to the information of the constituent components. In many game/simulation engines, the model component names can be retrieved and associated with the information tagging elements, but creators should be careful to ensure that the names match both in the information tags and the model part names.

Key Principle: Embed description, attribute, and affordance information within each object part and the whole object.

Another factor in creating virtual environments and objects is the way in which the world elements are built, particularly structures that can contain other objects. Modeling tools typically allow the creation of objects such as extruded rectangles and then allow the user to remove a portion of it as illustrated in Fig. 3a, b. If the model was just for visualization, this is perfectly acceptable; however, when the model is used as an interactive element or an element that will be perceived by game AI, it can cause problems. If there was a window in the void, it would appear to be placed in the middle of a wall with a section cutout, and this causes an

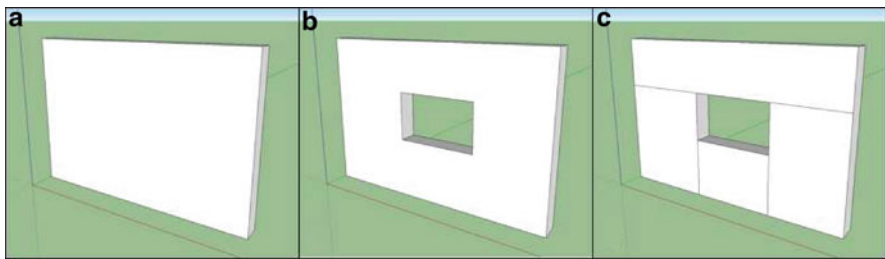


Fig. 3 Using a modern modeling tool (Google SketchUp in this case), a user can easily create a 3D object such as the wall in (a), but when creating a window as in (b) the tools can cause problems from an information standpoint. Panel (b) shows an extruded rectangle with a cutout missing extruded rectangle making the hole, which can be difficult to reason about spatially especially if a window occupied the missing cutout. From a game AI and information tagging perspective, it would be better to model each section of the wall independently with regard to potential physics engine forces as shown in panel (c)

unnecessarily complex spatial arrangement that may have to be reasoned over by an NPC. It also causes problems for information tagging as there is a void area to negate the information and then replace with potentially another object's information (e.g., a window placed in the void). A more simple approach would be to model the wall and void as shown in Fig. 3c, which allows each segment to stand on its own with its own properties without any complex spatial arrangement or potential overlap in void/occupied space from multiple models if a window was installed. This arrangement also shows a structure that could perform more consistently to a real structure under the influence of forces from a physics engine – something that should also be taken into consideration.

Key Principle: Keep world geometry simple and avoid the use of object cutouts.

Remember that there are two types of objects in interactive worlds: static elements that do not change state and dynamic elements that change state in many degrees of freedom. The principles described to this point apply to both types of models, but dynamic objects should also contain information pertaining to their state changes and degrees of freedom to provide information to intelligent entities. Human players can easily recognize a door in a building or house and have an intuitive feel for the actions and behaviors it will have under dynamic conditions; however, NPCs have no idea – How far does the door open? in what direction? and so forth. Dynamic information can be represented by a set of variables, their ranges, and actions that can be applied (affordances). Since physics engines are used in modern games and simulations, all world objects should have a defined mass (and distribution if not uniform), weight, and center of gravity. This information can assist in reasoning about these objects and how they will behave when actions are applied to them.

When creating world objects, simplifying bounding geometry should be provided to the game/simulation engine for use in collision detection and other calculations. Default settings in most modeling tools will provide bounding boxes over the

whole object or the key grouped parts, but oftentimes these boxes are very rough simplifying boundaries. Ill-fitting bounding geometry can cause problems with not allowing objects close enough, creating odd force fields around world elements, or in other cases allowing objects to pass through supposedly solid objects. Content generators need to include good bounding geometry for their creations.

Reuse from prior projects, off-the-shelf purchases, and using open work are core elements in modern game development. As these libraries of models, textures, and other game-ready components are being created, it is important that *all* of the pertinent information be added to these objects. Embedding the information described in this chapter can greatly facilitate use of created objects into games and simulations because they lend themselves to supporting game AI and even game physics. Modelers at all levels need to think beyond just the visual aspects and be sure to provide *information complete* objects. This notion of *information completeness* is an important concept and may be a deciding factor in whether or not a modeled object can be used in a game or simulation, especially as game AI increases in complexity – in fact, making more advanced game AI depends on this in many cases.

In the next sections, we will discuss a number of techniques that have tools associated with them that were developed from researched techniques discovered by the University of North Carolina at Charlotte's Game Intelligence Group (gameintelligencegroup.org). If the reader is interested in any of these tools or learning more about these techniques, please refer to our website or contact us directly. Many of these tools, while not available to the public, are shared with fellow academics and can be licensed to industry.

2.2 Creating Places

Creating virtual worlds or levels by assembling and placing models in the desired configuration is only the beginning of the work needed to make a viable interactive environment. Even with *information complete* models, which include proper bounding boxes for collision detection and resolution, we only have an environment that player-controlled characters can explore. Character AI could carefully stumble around exploring and creating a map in its memory, but this would take time and be rather inefficient. What is needed is a way to represent and understand the free space between objects in the virtual world to support spatial planning and pathfinding.

There are a number of representations and methods for decomposing the navigable free space that exists between and inside virtual world objects. These range from simple graph-connected, manually placed way points to advanced automatic decomposition techniques [10]. While way points have their uses, automated spatial decomposition methods can provide navigation meshes (navmeshes) that provide higher coverage of the navigable spaces and a multi-modal method allowing for reasoning at both a local detailed level and a topological view. Many techniques based on Graphics Processing Unit (GPU)-like decompositions, vertex connectivity, and Delaunay triangulation decompose the environments into lots of low order

shapes (polygons in 2D and polyhedrons in 3D). Low order shapes tend to have many triangles or pyramids converging at common vertices that make localization near those points difficult due to character footprint.

Human navigation is not based on thinking about space as moving from one precise point to another and does not have convergence difficulties in the corners. We tend to view the world as a series of connected places. This place-centric notion can be very helpful when creating navigating character AI because we can match the navmesh representation to our own intuitive view of the world – one in which we can more naturally understand and more intuitively specify directions to the character AI. Higher order decompositions such as those we have developed, which are based on polygon (2D) and polyhedron (3D) seeding and growth [7–10], can provide complete world coverage. Decomposing all open spaces with regions that touch each other allows us to create a topological map of the region connectivity, while also providing a region breakdown that more closely represents a series of connected places. Figure 4 illustrates some of the basic methods of decomposing navigable space into a navmesh. We advocate using our Planar Adaptive Space-Filling Volumes (PASFV) algorithm (2D) and Volumetric Adaptive Space-Filling Volumes (VASFV) algorithm (3D) to create places and the associated navmeshes within a created world. Navmeshes allow for more efficient path planning using search over the topological graph and easier pathfinding by reducing local navigation reasoning to smaller, more manageable places. The decomposition of space can also be leveraged to reduce reasoning complexity by compartmentalizing information into the places where the objects exist. This notion of *information compartmentalization* is useful in that it reduces the number of objects and information elements an AI entity has to reason over to those in its immediate place and possibly the adjacent places.

Information complete models can aid in decomposition methods by biasing the algorithms toward common elements such as areas covered by grass, sidewalk, or road. In this manner, the areas covered by like objects and materials can be cohesively decomposed making it easier for character AI to “walk on the sidewalk” or to “play in the grass.” This generated information now added to the world can also be organized hierarchically to facilitate improved efficiency in spatial planning or

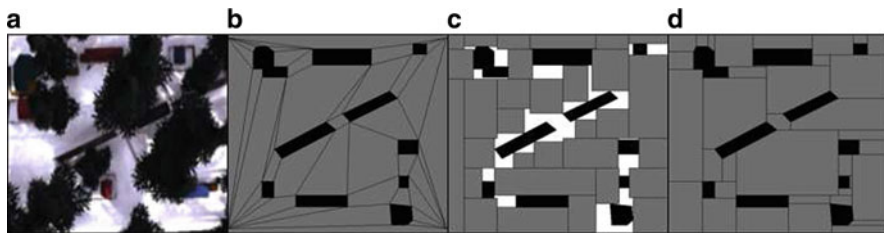


Fig. 4 (a) A game level from a Quake 3 Mod (Twin Lakes from Urban Terror by Silicon Ice Development on id Software’s idTech 3 Engine). (b) This level decomposed with the Hertel–Mehlhorn algorithm [15], a triangle-based vertex-connecting technique. (c) Same level decomposed with the Adaptive Space-Filling Volumes algorithm [19]. (d) Same level decomposed with the Planar Adaptive Space-Filling Volumes (PASFV) algorithm [9]

other relevant tasks. In addition, it makes it easier to pre-compute and store actions within those places as is a common technique in game AI for increasing speed and efficiency.

Key Principle: Leverage knowledge generation techniques such as spatial decomposition to help provide deeper world knowledge in an intuitive, useful, and efficient organization.

2.3 Information Services

All of the information we have embedded, then generated, and finally organized is typically collected and then presented to requesting components of the game engine. The common method for delivery of this type of service is through a code library that provides the desired functionality in linked code, since games are typically created with compiled languages. However, it could also be provided through interprocess communication (IPC) mechanisms with defined protocols – we have provided services using shared memory in a number of game environments as well as with code libraries. We have found it helpful to write processing elements that collect and organize all of the information from the source materials into two files, one for static information that never changes and one for dynamic information that has to work with the game engine more closely. The part of our Common Games Understanding and Learning (CGUL) Toolkit that provides the static information is called Static Spatial Perception Service (SSPS), and the part that provides the dynamic information is called Dynamic Information Augmentation Service (DIAS).

SSPS (pronounced “sips”) at its core provides access to the navigation mesh and all compartmentalized object information. The mesh itself is represented in two ways: as a collection of geometric entities and also as a directed graph. The geometric representation is useful for local place navigation and storage of compartmentalized information, while the graph includes search functionality (A* search for CGUL SSPS) to allow global navigation. When SSPS loads, it reads the navigation mesh and all object information from an XML file. It then constructs a default graph to plan through a defined size character bounding box. When a character is instantiated, the SSPS service can generate a new graph based on the actual size of the character bounding box.

The main SSPS query is *findPoint(location, hint)*. This query searches for *location* in the navigation mesh, returning the unique ID of the place in which it is contained, or UNDEFINED if it is not contained in any place. The *hint* provides a starting point for the search; if provided, this place will be checked first, followed by all of its adjacent places. The search continues, breadth-first, until the point is found or all places have been checked.

Search-based (A*) path planning is provided by the *findPath(start, end, start_hint, end_hint)* query. This query returns a *list* of place IDs. If no path is found, the returned list is empty; otherwise, there is a path through the navigation

mesh from the starting point to the end point that passes through each of the places in order. Character AI can use this to traverse the maps moving from place to place. Navigation between places involves moving from *gateway* to *gateway* within the place by local navigation. *Gateways* define the connection planes between two places. Local navigation can use simple centroid movement with obstacle avoidance, probabilistic roadmaps, occupancy grid navigation, or any reasonable method. Due to the smaller size of a place in comparison to the world, local navigation is usually fast even using heavy-weight techniques.

Several other SSPS queries exist to support other needs. For example, *generateRandomValidPoint(region, ground)* will generate a random point, guaranteed to be in a free space (navigable) region. The region reference is filled with the ID of the region in which the new point is contained. The ground parameter is a boolean value specifying whether the point should be on the ground; if it is false, the generated point will have a z (height) value between the region's z_{min} and z_{max} values. If a point is believed to be within a positive space (non-navigable) region, the *findPointInObject(location)* query will search all positive space regions for the given location.

The primary computation performed by SSPS is the point in polyhedron calculation, which tends to be very fast once localized, given that the character movement is not too rapid or that they cannot teleport across the world at random. As the character is localized, information about all objects (hopefully information complete objects) in the current place is provided and other information is available by query (e.g., information about adjacent places). Thus, SSPS provides a constant, detailed perception of all static objects in the virtual world to interested entities in the game/simulation engine.

DIAS is a similar service to SSPS, but it only provides additional information about dynamic objects. The core game/simulation object usually maintains all dynamic information since it controls their behavior and state in accordance with their design and the engine design paradigm. However, to convey model and other external information (e.g., affordances), DIAS serves to augment the internally tracked dynamic information for the object. It can also be used to help interpret specific engine implementation states to a more common state model to present to the game AI.

Key Principle: Utilize services to provide the *information complete* data to key game/simulation engine components and facilitate the use of more advanced AI techniques to improve both character and system intelligence.

2.4 Virtual World Dynamics

The environments of modern games and simulations are ever changing. As both players and characters interact with each other and the objects and environments of the virtual world, context changes often govern new modalities of play and interaction. As more *information complete* elements comprise these virtual worlds,

the dynamic nature and potential only increases. We discuss two primary notions of handling virtual world dynamics to support game AI, namely *probabilistic affordances* and *influence points*.

2.4.1 Probabilistic Affordances

The basic concept of affordances is useful in itself as we previously discussed, but it does not address differences in characters, or how world state may affect affordances. We approach these issues with the idea of *probabilistic* affordances. *Probabilistic affordances* may present different actions depending on the current context of the interaction.

Our approach is to create affordances that use information about current world state and the character AI to determine the currently available set of actions. A probabilistic affordance is a state machine with an input set composed of environment and character attributes, an output set of actions, and an output set of expected outcomes. Environment attributes include information about the local place: what characters are present, their relationships, and the state of any doors, objects, or other devices in the place. Character health, inventory, action model, and history of interaction with related affordances are part of the character attributes. The character action model in this case refers to the actions that the character is capable of performing. In addition, character attributes may include physical or personality characteristics. Physical characteristics may be role-playing game (RPG) style attributes such as strength, dexterity, and wisdom. Personality characteristics define how characters will react in social settings and may be a character's alignment in an RPG or a set of Hofstede parameters in a cultural simulation.

The actions output by the affordance are from the character's action model and include how the object or world location the affordance is tied to should be used. For example, the action output for pulling a lever would include three pieces of information: the *pull* action, the lever object as the *target* for the pull action, and finally the location the character should be standing in to pull the lever. For a sword on the floor, the character would see the *pick up* and *attack* actions, with the sword as the *target* for pick up, and the sword as the *tool* for attack. The pick up action would include the location of the sword, but the attack action would not provide a location.

Finally, the expected outcome distribution provides the character a notion of the expected change in world state after the action has been executed. This is a probability distribution to allow the specification of outcomes that are uncertain. In the previous examples, the lever pull action would have two probabilities: the linked door could open with a 90% probability, the rusty chain that raises the door could break with a 7% probability, or the lever itself could break with a 3% probability. For the sword, the pick up action would place the sword into the character's inventory with a 95% probability, or the character could trip and fall with a 5% probability. Until the character picks up the sword, the attack action will have no probability of success, but once the character has the sword, it will provide probabilities based on the characteristics of the sword.

Note that the affordance input requires a history of interaction with related affordances. This allows additional interesting scenarios. Imagine that our hero has just entered a dungeon room, and in this room there are four levers, three doors, and a trap door leading to the pit of infinite doom. The affordances on the levers have outcomes of opening the door to the exit, the treasure, or a peckish dire badger. In addition, they have a probability of triggering the trap door to the pit of infinite doom. As our hero pulls the levers and observes the results, the probabilities should adjust to reflect his experiences so far. Including the history of how the hero has interacted with each of the levers with the query allows the affordance probabilities to adjust as he tries each lever. If the first lever opens the exit, the probabilities are now 33% finding the treasure, 33% encountering the monster, and 33% certain death. Deciding that this is an acceptable risk, the hero chooses to open one more door. His second attempt releases the dangerously hungry dire badger. After defeating the monster, the hero can decide that his threshold for danger has been surpassed, and he is not willing to risk the 50% chance of falling in the pit for the chance to retrieve the treasure. If our hero fails and falls into the pit of infinite doom, the history of his interactions with this set of affordances is carried with him to his next life. So long as the history of interactions is maintained, no other changes need to be made to the hero's controller to allow him to make the correct choices when he next enters this room. In effect, learning occurs, but without the need to explicitly model a learning process for the character. This updation also applies to interactions with players and how they change the environment.

Some actions are only possible with the assistance of others. Affordances as discussed so far only support a single character; one character queries the affordance for the details of the actions that are possible, and then can attempt them alone. We extend the notion of probabilistic affordances to provide multiple *slots*. A slot is an action that one character can take to activate the affordance, and multiple slots allow multiple characters to work together to achieve some action not possible with a single character. Each slot is either required or optional. Required slots must be filled for the action to be successful; optional slots may enable it to complete faster or provide a better chance of success for results that are uncertain.

To return to the levers example, a multiple character affordance might be required to pull a particularly rusty lever. This lever has three action slots, two of which are required and the third is optional. One character does not have the strength to pull this lever, but two can. However, if only two characters attempt to pull the lever, it only has a 60% chance of working. A third character can join and improve the chance of the lever pull opening the door. Thus, this can help facilitate collaboration between not only NPCs but also players and NPCs.

Probabilistic affordances start with the artist annotated affordances for an object and are then extended by the game/level designer into the probabilistic affordance framework, which takes into account the multiple contexts and requirements for using and interacting with virtual world objects. This also puts a significant portion of the prerequisite information for intelligent use of an object with the object in the place it resides. This reduces the burden for remembering/storing all world interactions with the character and distributes it throughout the virtual world to be retrieved at the proper time under the proper context.

Key Principle: Probabilistic affordances should be added in the annotation phase of modeling and/or level design, but can update through interaction providing the ability for game AI to change with virtual world dynamics.

2.4.2 Influence Points

Influence points provide a technique to add information to the game world at run-time so that characters can take advantage of new information that was not available at the time the world was created. The primary use of influence points in our research has been to allow the addition of tactical information to the environment by enabling characters to use the world as a blackboard for communicating observed conditions to teammates. Influence points are similar to the idea of influence maps, but take advantage of navigation mesh decompositions – which we highly encourage as a spatial navigation structure as previously discussed. Individual influences are set in a specific place instead of using a high-resolution grid map. Because a place includes an area that would be covered by a much larger number of map cells, this allows much greater efficiency than influence maps. The influence of an entity on a map may not be limited to a single region – in these cases, additional child influence points are added to neighboring regions. To demonstrate the idea, we present an example of how characters can use influence points to further enrich the world with information.

A character is attempting to make it to a goal, the gold mine, to fetch resources as shown in Fig. 5. The character starts at its home base and has a choice between two paths. The longer path is defended by a friendly turret. The other path is short, but unknown. The character travels the short path and is ambushed by enemy forces. After retreating from the battle, it updates the navigation mesh by adding an enemy influence to the area where it was attacked.

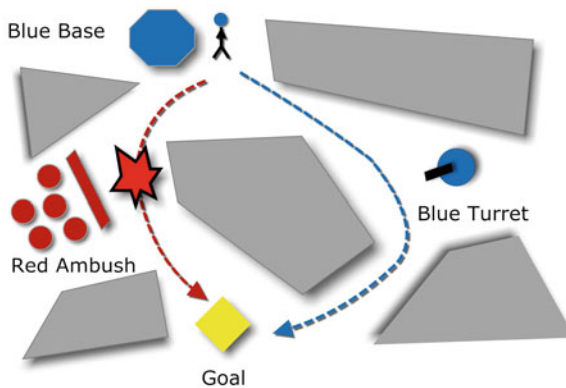


Fig. 5 Example world in which influence mapping can provide useful tactical information

Now the path passes through an area where the enemy is powerful. The longer path, on the other hand, passes through an area that is well protected by a defensive turret. After running its pathfinding algorithm a second time, the character finds that the least expensive path (incorporating chance of injury as a cost) is the longer path that passes the defensive turret. The character now follows the longer path and makes it to the gold mine successfully, returning valuable resources to home base. Other characters can now use this information as well, as it is maintained in the world representation rather than in the initial character's internal memory. The point strength of the influence decays over time to represent uncertainty about changes in the world, but it can be updated after creation.

In this example, if a grid-based technique is used, the character must spend a nontrivial amount of time inserting influence into the map, and potentially updating the large number of cells affected by the enemy and ally forces. If a navigation mesh is used, however, the insertion and update steps could be far less expensive; insertion is $O(n)$ in the number of regions in the navigation mesh, and updates to influence values can be made in constant time for decreasing values [or $O(n)$ for increasing values] [12].

While our example shows the influence points used primarily as a tactical tool, they can also be used to provide other types of information. Characters can label the world with information about events that have occurred or mark a region as a target for a particular activity. For example, if a character makes a mess in a virtual kitchen, it could place a "dirty" influence in the room. This would mark the kitchen as a target for another character to clean, or get into an argument with its virtual roommate/spouse about housekeeping responsibilities.

Key Principle: Dynamically embedding information into the environment at run-time can greatly improve character intelligence by adjusting to changes within the game/simulation.

2.5 Information from Interaction Observation

Running analytics on every aspect of a game including all player and NPC decisions and actions, as well as tracking spatial movements to understand level utilization and overall performance, is a popular focus in today's game industry – especially in online games where keeping interest high keeps paid subscribers engaged. The key to this is logging. Log everything useful that does not affect game play by slowing down the system, and with user permission collect these data to improve game play and the user experience.

Key Principle: Log and collect as much relevant data from game play and interaction as possible without impacting game performance.

Collecting data is just the first step, what is interesting is what additional information can be learned from that data. In many cases, there is so much data that it can

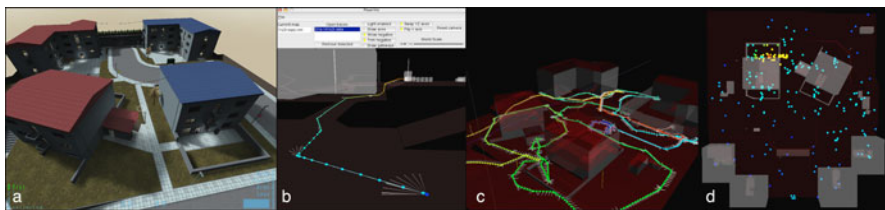


Fig. 6 Example analytics derived from logged data collected from the (a) Urban Combat Testbed and visualized using (b) PlayerViz showing the player traces and interactions over time for (c) an entire play session. This can be further analyzed through visual data mining, developing math models to automatically create (d) heat-maps showing the commonality of behaviors in spatial locations across a number of players/characters

become unwieldy to work with and gain knowledge. Visualizing the data and using an interactive visualization process to manipulate, isolate, enhance, and frame the data, such as is provided by our PlayerViz tool (shown in Fig. 6b), can be useful for performing visual data mining to find patterns in the data. Math models can be built around discovered regular patterns and used to generate other visualizations such as heat-maps (shown in Fig. 6d) of phenomena occurrence in the virtual world [18,20]. Such analysis can be useful for understanding interaction problems, exploits, use of environmental elements, and problems with level design. The key unit of analysis we use is the *player trace* (shown in Fig. 6c), which captures player/character spatial motion colored over the passage of time, gaze direction, and any interactions (e.g., pushing a button, firing a weapon).

Player traces that capture the view frustum of the player can also be used in conjunction with world geometry to track what surfaces the players viewed. Over a number of players, we can use a texture painting technique to radially and distance weigh observation from the virtual fovea point to the incident world geometry. Summation of the observations on the world geometry surfaces can provide a single value of the *information value* of that surface, and visual examination can present the *observation density* revealing the highest value location on that surface to place artifacts for interaction or desired observation. Conversely, this information can be used to identify good locations to hide artifacts in the virtual world. While this is a context-sensitive approach, it can be very useful to guide level design, art detail focus, and understand behavioral influence [3].

Data collection and analysis are a key part of modern game design and development. It can greatly assist in understanding how embedded virtual world information is used, misused, or simply ignored. It can guide attention resources in level design, information tagging, and many aspects of game creation toward issues of importance – improving the overall efficiency of the game creation process.

Key Principle: A wealth of additional information can be learned and used to improve game play and interactivity from logged data, and interactive visualization tools can significantly assist in this knowledge discovery.

3 Case Studies

The process of adding and deriving information to support game AI starts with model creation, it is integrated in level design, used in engine accessible services, dynamically expanded in-game, and augmented through additional discovery and analysis. The two projects in this section met project goals and were evaluated in their own unique ways; however, both depended on elements of information tagging and knowledge creation from the techniques presented here to be successful.

3.1 *The Transfer Learning Project*

The first year effort of the Defense Advanced Research Projects Agency (DARPA) Transfer Learning Program led by the ISLE Team (www.isle.org) used the University of Texas at Arlington's Urban Combat Testbed (a Quake3 total conversion mod) [21] to explore the mechanisms of learning in a source experience and applying that learned knowledge to a target experience where the difference between the source and target examined a particular type of learning (e.g., memorization, extrapolating, restructuring, abstracting, generalizing, and so forth). Sub-teams from Stanford University (ICARUS architecture agents), the University of Michigan (SOAR architecture agents), and the University of Texas at Austin (reinforcement learning agents) leveraged SSPS, DIAS, and an early version of PlayerViz to successfully show artificial transfer learning that in most cases exceeded that of humans under similar conditions [2]. This rare occurrence of using multiple academic, high-quality AI agent architectures in the same testbed was possible because of the information embedded and conveyed in the environmental objects.

Specifically, a navmesh similar to that shown in Fig. 4b was used for all agent navigation. World objects such as doors and crates were all annotated by Research Cyc constants and used to perform object and spatial reasoning using information provided in-game by SSPS and DIAS. The focus was primarily on solving spatial puzzles; so there was a heavy reliance on the information provided for the AI to solve problems by learning in one environment and then applying that solution to a similar problem, which sometimes came down to simple symbolic substitution (e.g., climb over create instead of a bush). However, none of this could have been accomplished without embedding information and providing dynamic updates.

3.2 *DASSIEs*

The DARPA-supported Dynamic Adaptable Super-Scalable Intelligent Entities (DASSIEs) Project addresses the problem of building an intuitive user interface for interactive character design. DASSIEs makes use of each of the elements described



Fig. 7 The DASSIEs Project BehaviorShop v1.0 overall behavior layer builder, which is a graphical subsumption/behavior-based control specification tool, is shown in panel (a). The specific behavior layer builder including a graphical, spatial selector for the target environment is shown in panel (b). The BehaviorShop specified character is executed by the BEHAVEngine in our FI3RST (FIrst and 3rd-person Realtime Simulation Testbed) environment is shown in panel (c)

in Sect. 2 to support the BehaviorShop interface and the BEHAVEngine AI engine in multiple target game/simulation environments. Figure 7 shows the DASSIEs components in action.

BEHAVEngine is an AI engine for implementing AI characters using hierarchical behavior-based subsumption controllers [13]. Information from the simulation environment is received by a perceptual model, which augments the raw world state with additional information. This includes querying affordances to determine available actions. Part of this process can make queries to the character’s working memory through the memory model. The memory model itself provides a store for character state information as well as a facility for making queries from relational ontologies (such as ConceptNet and ResearchCyc). Once the world state has been processed into *percepts*, this information flows into the subsumption controller. The subsumption controller makes decisions about what behaviors to run, and individual behaviors process the current set of percepts to generate appropriate actions. Behaviors may also query the perceptual and memory models, gaining access to working memory (probabilistic) affordance information, influence points, ontologies, and the SSPS/DIAS services. Finally, the behaviors send action requests to the action model, which can build commands for the simulation environment to execute actions, use affordances, and move through the world. The action model can also query the SSPS service to find paths through the environment.

BehaviorShop is a GUI for creating characters for BEHAVEngine [11]. It provides support for creating behavior for individuals and teams of characters in a modular fashion. Information about character behaviors is loaded from metadata provided by BEHAVEngine, while information about the simulation environment is received from the game/simulation environment from embedded information and information processed by the CGUL tools as described in Sect. 2. SSPS queries allow BehaviorShop to display maps of the environment, which allow users to more easily create spatial behaviors for characters. Object information can be queried from the game environment object database and includes information about affordances. Ontology information can also be used to specify the qualities of objects for use in the behaviors rather than binding specific object types or instances in the character definition.

DASSIEs highly leverages the work described in this chapter to make character behavior specification easy enough for the novice user to specify, while powerful enough to act intelligently in many roles in the game/simulation environment [5]. One of the key aspects of DASSIEs is that information flows from the world up. Having an information complete world with a built-in navigation mesh and structures for dynamic information storage and adjustment allows for BEHAVEngine to mostly just provide an architectural framework for behavior organization and firing. BEHAVEngine also provides a set of perception sensors, which convey much of the embedded information, and actuators, which provide a set of actions in which the characters can interact in the world. Many of the symbolic annotations from the world percepts are left ungrounded until specified for behavior inclusion in BehaviorShop. The base set of actions in BEHAVEngine is dictated by the affordance actions, but is often just scripted from a library of existing low-level actions and then grounded to the affordance language. BehaviorShop provides the character programming interface in a subsumption architecture, behavior-based control paradigm [1, 17], which starts off as an empty shell and is then populated with action and perception choices passed from BEHAVEngine and the simulation environment. Most of the behavior specification is done through building piecewise English sentences or graphical selection/drawing (e.g., patrol routes). In this manner, the rich world information comes through the AI engine to the programming/configuration environment. Meaning is given to the symbolic information in BehaviorShop and executed in the environment using BEHAVEngine.

The key strength of DASSIEs and our approach to embedding information is that the subject matter expert (not a computer scientist) creating the models, environments, and context can effectively specify in a simple manner how they intend for those elements to be used in interaction or at a minimum how to be perceived. When propagated through a tool like BehaviorShop, they can then also more easily specify how the AI should behave in these environments. Thus, not only can we make smarter AI by providing more information, but we can also enlist the help of the non-programmer content creator to make better interactive AI.

4 Summary

Better AI in games and simulation comes at the expense of having more information in which to perceive, reason, and ultimately act upon. The current trend toward procedural and user-generated content as well as libraries of pre-made 3D assets unfortunately provides the means to continue to make information-poor virtual worlds. These environments are often difficult for AI to reason in because of the sparseness of information and lack of a designer/programmer to inject information. This chapter discusses the specific issues with the current trends in virtual world creation and provides some suggestive guidance on how to make and use information-rich worlds by embedding information into world objects and environments – building worlds with regard to the AI and physics engines as well as leveraging organizational tools

to provide intuitive paradigms for reasoning about space. We forward the notion of having information complete objects and virtual worlds that makes them more amenable to AI. Techniques for handling the dynamic nature of these objects and environments are presented through probabilistic affordances and influence points to provide a complete notion of virtual world specification and use with regard to the AI, making light of the importance of embedded information and providing methods to handle embedded dynamic information. The importance of logging and analysis in understanding and improving virtual world construction and embedded information to move toward efficient and information rich environments is discussed, and we end with two case examples that leverage information-rich environments to achieve their advanced AI goals.

Acknowledgements Thanks are due to the following people, beyond those who are coauthors, who helped inspire, guide, and contribute to this body of work: Dongkyu Choi, Nick Gorski, Larry Holder, Nikhil Ketkar, Bharat Kondeti, Tolga Konik, John Laird, Pat Langley, Yaxin Liu, Cynthia Matuszek, Maheswar Nallacharu, Billy Nolen, Michael Ross, and Ashish Singh. This material is based on research sponsored by the US DARPA. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the US Government.

References

1. Brooks, R.A.: A Robust Layered Control System For A Mobile Robot. In: IEEE Journal Of Robotics And Automation, vol. RA-2, pp. 14–23 (1986)
2. Cook, D.J., Holder, L.B., Youngblood, G.M.: Analysis of Human Transfer Learning Using a Real-Time Game Testbed. IEEE Transactions on Knowledge and Data Engineering (2007)
3. Dixit, P., Youngblood, G.M.: Optimal Information Placement in 3D Interactive Environments. In: Sandbox Symposium (2007)
4. Evans, R.: Implementation of Personality Traits in The Sims 3. Game Developer's Conference AI Summit (2009)
5. George Alexander, G. Michael Youngblood, Frederick W. P. Heckel, D. Hunter Hale, Nikhil S. Ketkar: Rapid Development of Intelligent Agents in First/third-person Training Simulations via Behavior-based Control. In: Proceedings of the 19th Behavior Representation in Modeling and Simulation Conference (BRIMS) (2010)
6. Gibson, J.J.: Perceiving, Acting, and Knowing, chap. The Theory of Affordances. Lawrence Erlbaum Associates (1977)
7. Hale, D., Youngblood, G., Ketkar, N.: Using Intelligent Agents to Build Navigation Meshes. In: Proceedings of the 23rd Florida Artificial Intelligence Research Society Conference (FLAIRS) (2010)
8. Hale, D.H., Youngblood, G.M.: Dynamic Updating of Navigation Meshes in Response to Changes in a Game World. In: Proceedings of the 22nd Florida Artificial Intelligence Research Society Conference (FLAIRS) (2009)
9. Hale, D.H., Youngblood, G.M.: Full 3D Spatial Decomposition for the Generation of Navigation Meshes. In: Proceedings of the Fifth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE) (2009)
10. Hale, D.H., Youngblood, G.M., Dixit, P.N.: Automatically-generated Convex Region Decomposition for Real-time Spatial Agent Navigation in Virtual Worlds. In: Proceedings of the Fourth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE) (2008)

11. Heckel, F.W.P., Youngblood, G.M., Hale, D.H.: BehaviorShop: An Intuitive Interface for Interactive Character Design. In: Proceedings of the Fifth Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE) (2009)
12. Heckel, F.W.P., Youngblood, G.M., Hale, D.H.: Influence Points for Tactical Information in Navigation Meshes. In: Proceedings of the 4th International Conference on Foundations of Digital Games (FDG), pp. 79–85. ACM (2009)
13. Heckel, F.W.P., Youngblood, G.M., Hale, D.H.: Making User-Defined Interactive Game Characters BEHAVE. In: Proceedings of the 22nd Florida Artificial Intelligence Research Society Conference (FLAIRS) (2009)
14. Heinze, C., Goss, S., Josefsson, T., Bennett, K., Waugh, S., Lloyd, I., Murray, G., Oldfield, J.: Interchanging Agents and Humans in Military Simulation. *AI Magazine* **23**(2), 37–47 (2002)
15. Hertel, S., Mehlhorn, K.: Fast Triangulation of the Plane with Respect to Simple Polygons. In: International Conference on Foundations of Computation Theory (1983)
16. Laird, J., van Lent, M.: Human-level ai's killer application: Interactive computer games. *AI Magazine* (22:15–25) (2001)
17. Mataric', M.J.: MIT Encyclopedia of Cognitive Science, chap. Behavior-Based Robotics, pp. 74–77. MIT Press (1999)
18. Priyesh N. Dixit, G. Michael Youngblood: Understanding Playtest Data through Visual Data Mining in Interactive 3D Environments. In: In the Proceedings of the 12th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia & Serious Games (CGAMES) (2008)
19. Tozour, P.: AI Game Programming Wisdom 2, chap. 2.1 Search Space Representations, pp. 85–102. Charles River Media (2004)
20. Youngblood, G.M., Dixit, P.: AI Game Programming Gems 7. Understanding Intelligence in Games Using Player Traces and Player Graphs. Charles River Media (2008)
21. Youngblood, G.M., Nolen, B., Ross, M., Holder, L.: Building Test Beds for AI with the Q3 Mod Base. In: Proceedings of the Second Artificial Intelligence for Interactive Digital Entertainment Conference (AIIDE) (2006)



<http://www.springer.com/978-1-4419-8187-5>

Artificial Intelligence for Computer Games

Gonzalo, M.; Gómez-Martín, M.A. (Eds.)

2011, XII, 200 p., Hardcover

ISBN: 978-1-4419-8187-5