A Semantic Navigation Model for Video Games

Leonard van Driel and Rafael Bidarra

Delft University of Technology,
Mekelweg 4, 2628CD Delft, Netherlands
leonardvandriel@gmail.com, r.bidarra@ewi.tudelft.nl
http://graphics.tudelft.nl/

Abstract. Navigational performance of artificial intelligence (AI) characters in computer games is gaining an increasingly important role in the perception of their behavior. While recent games successfully solve some complex navigation problems, there is little known or documented on the underlying approaches, often resembling a primitive conglomerate of ad-hoc algorithms for specific situations.

In this paper we develop a generic navigation model which we call semantics-based, because it enables AI to incorporate a wide variety of navigation information into the planning of a character's behavior, including raw geometry, strategic objects, and locomotion abilities. The role of semantics in this domain is highlighted and the presentation of the main components of this semantic navigation model is illustrated with a variety of examples.

The semantic navigation model has been validated and implemented within a navigation system for Unreal Engine 3 that requires little level designer intervention, has a rich interface for AI programmers, and can be extended with other types of semantic information. It is concluded that using this navigation model delivers more natural paths, requires fewer world annotation, and supports dynamic re-planning.

Keywords: navigation, game semantics, game AI.

1 Introduction

Successful improvements in current video games have not been evenly distributed over the development areas involved. So far, the ongoing trend of enhancing visual realism has clearly dominated [2]. In contrast, there are relatively few innovative applications of artificial intelligence (AI) techniques in these games, and mostly we have seen little improvement over the last decades. It is true that a small fraction of video games have shown significant sophistication when compared to the other games at their time. While other game development areas have pushed rapidly forward, using the most state-of-the-art technologies, game AI has been said to lag at least 25 years behind the academic level of research [1]. No wonder, therefore, that in recent years there has been an increasing incentive to help the game industry focus on the challenges of improving game AI [7].

One of the areas in which game AI has been lagging concerns the navigational capabilities of so-called AI- or non-player characters (NPC). As a part of the game AI, navigation is typically concerned with spatial search and path execution (how do you go from A to B?), and it is clearly distinct from the higher-level decision making process (where to go?, why should you go to B?). One of the problems faced by navigation in current games is that it has too often been tightly built up around the A* algorithm, which has stimulated the development of graph-based navigation algorithms and released an abundance of A* variants to run optimally under specific resource constraints. Still A* is only an algorithm and by far not a general solution to navigation [6]. As the requirements on navigation become more and more complex, devising suitable heuristics for A* turns very difficult, leaving us with Dijkstra's algorithm, which isn't but a good exercise on dynamic programming. We believe that although A* or Dijkstra may be good solutions for a large number of isolated problems, they should not be the starting point of the development of a navigation system.

In our view, navigation is about understanding the direct relation between the world and a moving character. This implies a significant shift in focus, from the field of low level search, typically associated with the A* algorithm, to what we call a semantic perspective, in order to capture and understand the environment from the point of view of movement and positioning. In the context of this paper, we define navigation as "the act of guiding a body through space-time". In practical terms, navigation deals with the prediction and execution of motion. In order to predict the outcome of motion, one has to know the navigation space to look for feasible or optimal paths. Without prediction, navigation would be based on reflexes only, which is generally referred to as steering. In order to execute motion, one has to make actual moves based on the planned motion and react to discrepancies between the internal representation and the world.

The main advantages of our semantic navigation model can be summarized as follows:

- it offers less restrictions to the design process, therefore allowing more complex game AI and a potentially richer and more challenging game play;
- it is based on fairly invariant concepts, which are valid through a variety of domains and genres providing better reusability;
- from a game development point of view, it provides a better link between disciplines, whereby both level designers and AI programmers share the same concepts and can therefore enable them to work together more closely.

In this paper, we first outline the basics of our semantic approach (Section 2), proceed with the development of the navigation model (Section 3), and then describe how we have implemented this in a practical setting, together with the results we obtained (Section 4) and end up drawing some conclusions (Section 5).

2 Game Semantics

Semantics is generally defined as the study of meaning and relations among signs. In our present scope, game semantics is concerned with the structure and meaning of game elements, such as level geometry and player characters [9].

2.1 Navigation and the Current Lack of Semantic Structure

When designing game AI or a navigation system, explicit semantics become important because game mechanics needs to be explained, e.g. how should an AI character respond to a constantly changing world [4]. While human players can be very flexible in dealing with game semantics, an artificial player requires a designer that is well aware of this semantics in order to make the AI understand the game. This becomes apparent with some examples that show the link between semantics and daily game development. Here we focus on cover finding, because it is a very important and complex game play element in many modern games.

When a level designer creates the world geometry, he creates floors, walls, obstacles, and ledges based on specific interaction options between players and the environment. For example, a rock can be placed in such a way that it provides excellent cover close to the enemy base, so that an experienced human player will be able to recognize its tactical significance, while an AI character's cover finding algorithm might miss it. Here the understanding of the designer is not shared by the AI because it lacks a structure to contain such information.

Conversely, in that same level setting, we can imagine the level designer marking positions that are suitable for taking cover. This does allow the designer to accurately model the AI behavior for planning covered paths or escaping from a line of fire. However, it also reduces the AI's understanding of a covered area to a set of marks. Here, the designer's understanding of cover is explained to the AI in only a restricted form. This AI will be limited to the designers ability to spot good covered positions. Even worse, such an AI character will be completely exposed if the level geometry is changed afterward.

As a final example, imagine a player and an AI character having to cooperate during a fire fight and, say, both have a proper understanding of being exposed or in cover. At some moment, the latter decides to take cover behind a nearby rock. However, by arrival it fails to take actual cover, because the player was already standing there. The AI does not understand how its teammate's presence conforms to its understanding of cover, and it is now both fully exposed and blocking its teammate's line of fire.

What these examples have in common, is a lack of semantic structure due to a poor distribution of understanding among game development domains like level design, game play, and AI.

2.2 Our Approach

The semantics of game elements deals with the multiple domains of game development and how different building blocks of the game are understood in each domain. Here *understanding* is the relation between meaningful information and the domain where it is interpreted. Because games cope with a limited storage space, we can assume that all internal data is meaningful in some way in some domain. This includes cover markers that are meaningful to the AI, or texture data that is meaningful to the rendering pipeline, but also the game's storyline, which is mostly meaningful to the player, but often not to the AI.

An important element of a semantic model is the *concept*, which groups similar understanding of different data or in different domains. Domains can be level design, game play programming, but also the AI subsystem. For example, players base their understanding on the video output and AI on game data, but they both can have the same understanding of a bridge to cross a river. Here the bridge is the concept of an accessible surface for crossing a river.

Semantic navigation is an approach to navigation with a focus on the semantic model of the navigation AI. The key in creating a semantic navigation model lies in the identification of relevant *concepts* in the game. Concepts allow us to define navigation problems and solutions in a *domain-invariant* way. The latter is very important, because it allows us to design a navigation system without the restrictions of tools, languages and hardware resources.

As an example, we will look at how we define a spatial area over different domains. AI systems in general are very single-position oriented, mostly because a single point in 3D space is a very compact and unambiguous representation. Therefore a path often consists of a sequence of way-points starting at position A and ending at B. Instead, a level designer will rarely think of a single spot, but more of an area that is bound by all kinds of criteria like visibility, distance, and even shape. While the programmer and the designer have different *representations*, they both indicate the same area. This calls for the concept we will later on refer to as *place*. With it, instead of trying to explain each other how they see it, both can issue a place in their own language, while it is up to the implementation of place to combine these different representations.

We propose to define a semantic model based on a set of concepts and representations. A concept is defined by the relations it has with other concepts, like how one concept restricts or modifies another. A concept has a representation for each domain in which it has a relevant meaning. We will refer to the correspondence between representations of one concept in different domains as the *translation* between these representations. It is these translations that make the semantic model consistent, because it is the translation that will eventually make a concept valuable; see a schematic example of this in Figure 1.

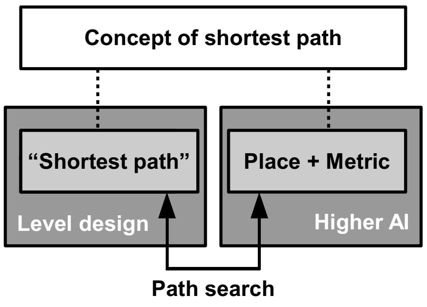

Fig. 1. Schematic representation of the relation between concept (white), domain (dark gray), representation (light gray), and translation (arrow) for "the shortest path"

Imagine, for example, an AI character that wants to move to a covered position because of an enemy approaching. The domains here are the behavior system and a locomotion system. Because the enemy is spotted, the behavior decides to go to a safe location because it wants to maximize chances of survival. The locomotion knows that a set of way-points leading to a position of low visibility is safe. It is the fact that we are able to *translate* safety in the domain of behavior to the domain of locomotion, that makes the concept of safety *valuable*. Generally such a translation exists and will probably require a way-point graph, cover evaluation, and a search algorithm.

Whether a translation can be implemented into a navigation system depends on the design and system limitations. This means that a consistent semantic model does not necessarily result in a feasible implementation model. We will refer to the feasibility or quality of such a translation as the *alignment* of two representations. In the previous example, if both representations of safety are well aligned, the locomotion system will offer the behavior system a path that does maximize the chances of survival.

Defining a general concept in the programming language of a large domain can be a complicated step, but it can be overcome by sub-dividing concepts and domains into more specific concepts for smaller domains. While a more specific concept is less relevant in general, it will have more concrete representations in a set of sub-domains. This approach leads to a concept hierarchy of general and specific concepts, which meets both a wide range of domains and the specific nature of all sub-domains. As we move down the hierarchy, the definition of concepts becomes more implementation specific. This process of exchanging generality for ease of representation, will be referred to as *mixing*. Take, for example, the concept of a path, that an AI character can use to move from one point to another. Instead of trying to directly represent a path, we first introduce the concept of a way-point, which is a position on the path. Using the concept of way-points, we can easily implement a path by moving in a straight line from way-point to way-point. Here the general path concept has been *mixed* with the domain specific use of point locations in programming languages.

The implementation of a navigation system that is based on a semantic model, is in many ways the same as one without it. We still need module interfaces, search algorithms, data models, and so on. However, by implementing the former we also create a powerful language that has been explicitly defined through concepts. Such a language can be used to define complex relations and even new concepts, without having a direct impact on the implementation. We not only define these rules of the game implicitly in programming code, but also explicitly in the semantic model, allowing them to be understood by the AI.

The semantic model provides structure to the implementation in different ways depending on the extensiveness of the concept *hierarchy*. This hierarchy can be used as a basis for the class hierarchy of an object-oriented software design. The representations of concepts can be used to derive data models for our classes. The meaning of a representation can be implemented by the functions that operate on this data. This allows us to group functionality based on its role in the semantic model and on the domain it provides meaning for.

A part of the implementation will deal with the translation of representations. This will determine the alignment of the data. For example, for good alignment of the path concept, we should either have a static world geometry, or regularly update the navigation grid. Clearly the alignment and the frequency of this updating are related.

3 Semantic Navigation Model

Navigation is a fundamental AI component in the action-adventure games genre. It supports many AI components and plays a principal role in connecting the general AI to the game world. In general, navigation deals with the problem of guiding objects, more specifically finding paths.

Although the area of navigation originates in search algorithms and system design, we choose to approach it from an abstract, conceptual level, before we advance towards its design and implementation level. Because navigation is not a goal by itself, we start by identifying the domains that define the semantics for our model. Next we group all representations in the domain set into concepts that will form the basis of our navigation model.

3.1 Domains

Our design of a semantic navigation model starts by investigating the domains that relate to navigation, because it is within these domains that the initial need for navigation arises. This investigation is based on a simplified view on today's game development process [5][10]. Our choice of domains covers the key domains relevant to ensure a wide applicability, rather than attempting to fit all development process details into one single model.

We will deal with two classes of domains, namely game development and system components. Within *game development*, there are two domains particularly interested in navigation: the level designer and the AI programmer.

The *level designer* creates the level geometry and places a variety of special objects and markers in this environment. From his perspective, it is important to understand how the design affects the behavior of an AI character, limiting its behavior or creating opportunities. A level designer, therefore, requires a visual representations of concepts that provide feedback during the design process.

The *AI programmer*, in turn, designs the behavior of the AI using a programming language. From his perspective, concepts will mostly be represented by a single name referring to, for example, an object-oriented class. For the AI programmer, it is important that concepts refer to real-life behavior and navigation. The AI programmer will further rely on graphical representations for both debugging purposes and the visual representation used by the level designer.

In the class of *system components*, we distinguish the domains of higher AI and motion control. The *higher AI* is a general decision making component, which deals with decisions that are unrelated to navigation, like strategy, intuition, and emotion. The higher AI uses implicit descriptions of a path, e.g. by specifying the start and end positions. It is mostly concerned with understanding the link between its goals and such specifications. In order to plan its actions, the higher AI must also understand the cost of trying to reach these goals.

The *motion control* component coordinates a character's movement in terms of animation and physics simulation. Therefore, it has to understand how a route can be traversed based on some explicit definition like a sequence of way-points. While motion control has a subjective role compared to the higher AI, it does take part in the feedback from the world to the higher AI.

3.2 Concepts

The initial process of identifying representations and shaping concepts has been performed in previous research [3], which was geared towards the development of an action-adventure game. Based on that experience, we now derive our semantic model in a top-down fashion, starting at the top of its concept hierarchy with the trivial concept of *navigation*. From there, we look for implementation invariant concepts that describe navigation across all relevant domains.

Path and space. The first sub-division in the concept hierarchy is one rather abstract, as we define the concept of navigation based on the sub-concepts of path and space. A *path* is a way for a character to move about in the game world. As the higher AI formulates goals, it needs a way of expressing them in navigational terms.

The navigation *space* is the collection of all possible paths to navigate, including all traversible surface, open doors, and accessible switches. The explicit definition of this collection is important to test path existence and to quantify path quality. We define a navigation problem as an optimization problem over the navigation space (e.g. the safest path away from danger).

World and ability. The game world is the collection of all game elements, including level geometry, movable objects, and characters. The player is presented with a representation of this world, 'dressed' in pretty textures, particle effects, etc. For the purpose of navigation, one has to 'look through' this dressing and see the elements that matter to navigation, i.e. the navigation space. Although conceptualizing the game world is a simple step, it is important to group all views on this same world into a single concept. The concept of space describes the game world from the perspective of an AI character. This world is primarily created by the level designer, who shapes the geometry and places obstacles and passages, and it is generic in the way it will be used by both human players and AI. It is therefore the perspective of the AI character that filters the world to this concept of space.

We also introduce the concept of ability, from an AI's perspective, defined as a means of a character to change itself and the world for navigation purposes. The most common ability is walking, which changes the position of a character. An

AI character becomes more versatile by adding navigation abilities like jumping, opening doors or pushing objects. While the game world remains the same, an increase of abilities makes the AI perceive a richer world with more paths.

Place and metric. In order for an AI character to move, it has to somehow specify its intentions or desired state, for example the desire to be in a safe location. Although it might not be a specific location that the AI has in mind, there must be some way of anchoring the navigation to the world geometry. Therefore we introduce the concept of place, which is a sub-set of space. A *place* can be a single explicit location, but also a disjoint area.

A navigation solution is always restricted to some part of the environment, for example, related to the current or future position of a character. The place concept defines such a part of the environment, ranging from a single point to a large or scattered area. Using places, we can restrict the solution set by defining where the path begins and ends, and restricting the path as a whole.

The concept of place offers the AI character a large variety of paths and possibilities, through which it can roam freely using its abilities. However, when the AI has a clear goal, it typically searches for a single, optimal path. In order to evaluate places and paths, we introduce the concept of metric. A *metric* is a function that maps a place or path onto a real value. This makes places and paths comparable and allows for a definition of optimality.

In order for the higher AI to specify what navigation can do for its goals, it has to express them in terms of a metric. For example, to find the shortest path, a metric of distance is needed. In practice, these goals require much more complex metrics which measure not only distance, but also cover, visibility, and health change. Such metrics can be achieved by combining several primitive metrics into a single real value, for example using linear combinations or thresholds.

In the domains of level design and AI programming, the visualization of a metric is most important. For a level designer, it is not only important to see the possible paths to navigate, but also which ones will be preferred by the AI, and under which circunstances. For example when placing objects that provide cover, it is important to see how the AI will evaluate these places.

4 Practice

The semantic model presented in the previous section offers a generic approach to navigation in video games, transparently dealing with semantics on various domains. In this section, we develop a further refinement of our semantic model toward an actual implementation based on more specific system requirements, and provide the link between concepts and software. Finally, we briefly evaluate our semantic approach and reflect on the advantages of a semantic model for navigation.

4.1 Refining the Model

Our generic approach allowed us to think of navigation without the restrictions of system design, implementation, and hardware. It is the platform on which video games run that enforces these restrictions, not games or AI itself. Therefore, in order to progress in the areas of games and AI, it is important to meet these restrictions in the end, instead of starting with them.

Moving from a generic semantic model to a navigation system implementation, we introduce new concepts that gradually mix generic concepts with implementation concepts. Here we respect the domains of game developers and system components, while we express their representations in a programming language. It is important here, not to confuse the system components with the actual implementation.

World and ability. Our characters move by walking or jumping, and it is thus the surface of the level geometry that is of interest to navigation. To represent both the geometry and the topology of this surface in the higher AI domain, we take a cell-decomposition approach to construct a graph of spatial nodes that can cover the curved plane of the level geometry, which is similar to a navigation mesh [8]. Here the graph captures the topology, by defining links between locations where a set of paths can be traversed. Each node is accompanied with a 2D convex area that locally represents the geometry. We will refer to this graph representation as the *mesh.*

To translate between the geometric world representation and the mesh representation, we use collision traces to sample space. First the level geometry is decomposed into 2D areas, and convex sub-areas, called tiles, are traced out. Next the edges adjacent tiles are connected though links, which are also constructed based on collision traces. Because our collision traces are relatively cheap, we were able to implement a real-time translation, which is able to cope with slow but complex changes in the level geometry.

An ability is represented by the higher AI as a set of rules to test a subspace for applicability. For example, walking has rules about the steepness, the floor and the width of the collision cylinder that can be placed on top of the floor. Besides walking, we also introduced the abilities of crouching, jumping, and walking under all angles; see Figure 2. These are all simple variations on the rules of walking.

Abilities are used to identify convex areas and links for the mesh. This is a relation between ability and world, which allows a character specific view on the world. Each application of an ability in a specific location is an instantiation of this ability, to which we will refer as an *action.* An action is represented by an ability and a specification of where and how it is performed. This leads us to another representation of the mesh, namely as a set of related actions. This representation is of interest in the domain of motion control, because it represents all possible moves a character can make in a certain location, assuming the piecewise independence of motion steps.

Place and metric. We restrict the concept of place to the representations of a single location, a mesh element, or a set of places (on which one can perform the operations union, intersection or difference). Although restrictive, this notion is sufficiently rich for navigation while avoiding complex geometric evaluation.

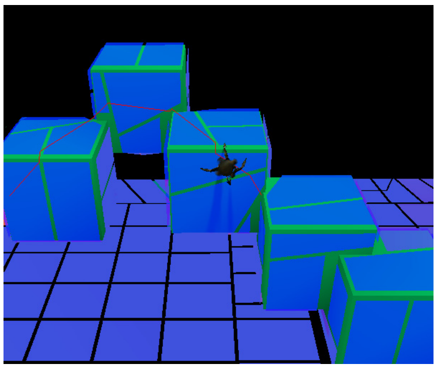

Fig. 2. Screen shot of a spider-like AI character traversing a path (red line) across connected cubes (green). The walkable areas (blue) of the mesh are based on the character's ability to walk under all angles.

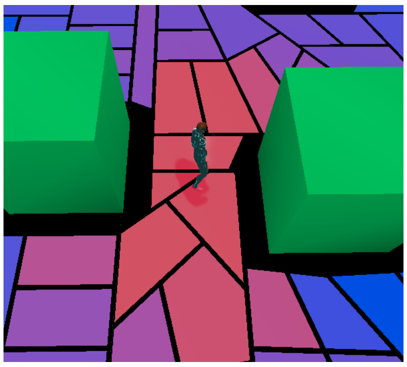

Fig. 3. Screen shot depicting the mesh nodes (red/blue) surrounding two cubes (green). The mesh is colored according to the vision of the AI character (gray) standing in the middle (red is high visibility).

Our approach offers metrics to evaluate e.g. distance, visibility, cover, and also operators on metrics. The distance metric evaluates a link by its Euclidean length and an area by its average diameter. To evaluate the visibility metric, we sample points on a link or in an area, and move a phantom character to the position of these samples to perform a set of ray casts from the eyes of an enemy character to the phantom body. Figure 3 shows a screen shot of the visual representation of the visibility metric.

Based on the visibility metric, we define cover by evaluating local visibility change, e.g. by assuming that positions with good cover are well hidden and close to other high visibility positions. To create more complex metrics, we use the operator metric to combine primitive metrics like distance and visibility. Here we found linear combination and maximum component to be most useful.

Places become a powerful tool for defining paths when we create places based on a metric. For example, an AI character that is under attack needs a covered position to recover. By providing a path that leads to a covered place, we have the AI character move to safety.

Path. The concept of path is the combining element of the navigation system. It is represented by the higher AI as a navigation problem based on a place and metric, and by the motion controller as a solution based on a sequence of actions. The translation from problem to solution is the search for a feasible path, which we approach by performing A* graph search on the mesh graph. The problem is defined by a start and end place, and a metric. The graph search start and end node are determined by evaluating start and end place. The area nodes and link edges weight are based on the metric. Finally a sequence of actions is derived by fitting an action to each area and link.

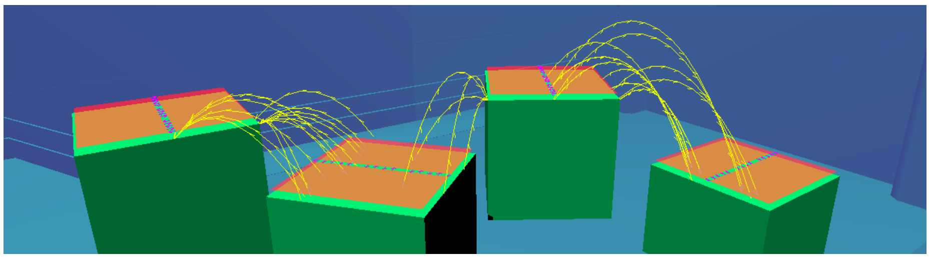

Fig. 4. While placing level components (green), the level designer is made constantly aware of the navigation space. Convex walkable areas (orange) are connected through walk links (purple) and jump links (yellow).

4.2 Results

A prototype navigation system based on our semantic navigation model has been built as an AI sub-system for the Unreal Engine 3. This prototype focuses on navigation for action-adventure games. The AI system in these games has to provide resistance in combat situations and assist in the story line. This includes on-surface navigation in a 3D world with strategic and scripted paths of a high quality. Navigation instructions were generated by a state-based system, and motion instructions were executed by a physics-based motion system.

Previously, designing a game environment did not only require the designer to place geometry, but also to iteratively add annotations for the AI followed by run-time testing of player and AI movement and interaction. By implementing real-time concept translation in our prototype, designers gain insight in the AI's abilities and understanding while shaping the game environment; see Figure 4. This allows for much less annotation and testing during the design phase.

While many concepts are already present in our engine, the explicit definition of these concepts allows a much sharper separation of scripted AI and motion control. The introduction of new concepts and the wrapping of the existing concepts made the AI system modular, so that it can be configured by simply adding and removing concepts, thus improving reusability.

In turn, navigation has been separated from the higher AI, improving robustness and making the higher AI less dependent on the low-level routines that perform the navigation. High-level AI behavior can be designed using high-level concepts that describe a characters behavior, giving programmers a natural understanding of the concepts, and supporting design of higher-level behaviors without being distracted by low-level issues.

5 Conclusion

In this paper we argued that most current navigation subsystems in games are too biased towards very particular applications, their design strongly suffering from contamination by implementation aspects as graph search, terrain specifics, etc. We have presented a hierarchically structured semantic model that helps defining rich and clear semantics in the design of a navigation system. This model provides a set of basic concepts that apply to generic navigation systems in action-adventure games. This approach leads to better aligned representations in the domains of game development and system components. We believe that the development of game AI for navigation can significantly profit from a semantic navigation model like the one we have described. This semantic navigation model has been validated and implemented within a navigation prototype system for Unreal Engine 3 that requires little level designer intervention, has a rich interface for AI programmers, and can be extended with other types of semantic information. It is concluded that using this navigation model delivers more natural paths, requires fewer world annotation, and supports dynamic re-planning. Our prototype system confirmed that game AI development based on a semantic model is perfectly compatible with the performance requirements related to limited CPU and memory usage.

References

1. Baekkelund, C.: Academic AI Research and Relations with the Games Industry. In: AI Game Programming Wisdom 3, Charles River Media (2006)
2. Buro, M., Furtak, T.: RTS Games as Test-Bed for Real-Time Research. In: Invited Paper at the Workshop on Game AI, pp. 481–484 (2003)
3. van Driel, L.: Semantic navigation in video games. MSc thesis, Delft University of Technology (2008)
4. Heckel, F.W.P., Youngblood, G.M., Hale, D.H.: Influence Points for Tactical Information in Navigation Meshes. In: Proceedings of Fourth International Conference on the Foundations of Digital Games, Port Canaveral, FL, April 26-30, pp. 79–85 (2009)
5. Mesdaghi, S., Stephens, N.: The 2005 Report of the IGDA's Artificial Intelligence Interface Standards Committee. In: IGDA (2005)
6. Reese, B., Stout, B.: Finding a Pathfinder. In: Proceedings of the AAAI Spring Symposium on Artificial Intelligence and Computer Games (1999)
7. Savage, F.: What are the Hard Problems in Game Development? In: Proceedings of the 3rd Annual Academic Days on Game Development in Computer Science Education, Miami, FL, 28 February-3 March (2008)
8. Tozour, P.: Building a Near-Optimal Navigation Mesh. In: AI Game Programming Wisdom, Charles River Media, pp. 171–185 (2002)
9. Tutenel, T., Bidarra, R., Smelik, R.M., de Kraker, K.J.: The role of semantics in games and simulations. ACM Computers in Entertainment 6(4), a57 (2008)
10. Yue, B., de Byl, P.: The state of the art in game AI standardisation. In: CyberGames 2006: Proceedings of the 2006 international conference on Game research and development, pp. 41–46 (2006)