Designing Semantic Game Worlds

Jassin Kessing
jassinkessing@gmail.com

Tim Tutenel
tim.tutenel@gmail.com

Rafael Bidarra
r.bidarra@tudelft.nl

Computer Graphics and Visualization Group
Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands

ABSTRACT

Current game worlds often fall short in providing consistency between the visual representation of the world and the way it feels, behaves, and reacts. This problem partly originates from the goal-oriented and cost-effective nature of the game development process, which mostly favors ad hoc solutions for one particular game, rather than investing in concepts like reusability and emergent gameplay. In broader terms, we observe that game worlds miss *semantics*, and we argue that its deployment has the potential to bring about the consistency missing in their content. Therefore, we present a novel approach aimed at enriching virtual entities in game worlds with information about their roles, how they relate to others, and how they can affect and interact with players, NPCs, and with each other. We discuss several requirements to achieve these goals, and introduce a semantic model to represent game worlds. In order to support and validate this model, we have developed *Entika*, a framework to facilitate the deployment of semantics during game development, as well as its maintenance during run-time. Furthermore, we briefly discuss several applications that demonstrate the power of this semantic model for game worlds. After careful evaluation of our semantic game world model and framework, we conclude that a semantically rich world representation can substantially assist designers in creating much more consistent game worlds.

Keywords

game worlds, semantics, object interaction

1. INTRODUCTION

The visual quality of game worlds increased massively in the last three decades. Representations evolved from pixelated two dimensional drawings to stunning, beautifully lit, and highly detailed three dimensional visualizations. The closer game worlds depict reality, the more noticeable it is for gamers when objects do not behave accordingly. And the better graphics get, the bigger this gap between visual and behavioral realism becomes. For example, objects not strictly necessary for gameplay purposes often remain dead and players are unable to interact with them.

Moreover, the problem of this lack of coherence between the visual representation of the world and the way it feels, behaves or reacts, hinders the immersion when playing a game. More specifically it breaks spatial immersion, which Björk and Holopainen [6] explained as the experience when playing in a perceptually convincing game world, i.e. when it both *looks* and *feels* real.

In the fields of linguistics, computer science and psychology, semantics is the study of meaning in communication. When focusing on virtual environments for computer games, we call semantics 'all information conveying the meaning of a virtual world and its entities' [33]. In this work, we will investigate whether semantics can be applied to game worlds to describe the actual meaning of virtual world objects, therefore going beyond their mere visual representation. In doing so, we want to get a step closer in bridging the gap between the look and the feel of game worlds.

To better understand why this gap is present in many games, it is important to take a look at how a game engine is typically structured. A game engine is an integrated software system of many different components. These components can be 2D or 3D rendering engines, physics engines, sound engines, scripting interfaces, artificial intelligence (AI) components, and many others. Developers often assemble a game engine as a patchwork of new components, components created for previous games and middleware components providing ready-to-use solutions for particular elements of game development. Next to the many technical issues involved in integrating all these different components, it is a challenging task to create and maintain the coherence between all the different representations of an object in the game world.

These objects are represented in the first place as a set of *properties* used for both development and gameplay purposes. One can think, for example, of a unique ID, the maximum amount of gun ammo, or the price of a shop item. In addition, an object is *presented* in the game, whether that is visible by means of a geometric model, a particle system, or an icon, or audible through the use of sounds and music. Increasingly often, some *physics* information is present as well, like a mass value and a bounding box, which can be used by a physics engine to properly apply physics (e.g. gravity) and detect collisions. Finally, a game object can show some basic *behavior*, usually defined by scripts that, for example, can prescribe how to move or how to cause damage to the avatar of the player. Although presentation, physics, and behavior make use of some properties, they are usually stand-alone, and only defined for their particular purpose. There is, therefore, a general lack of coherence between the properties used by all different game engine components. This lack of coherence could be solved by developers by investing a lot of time and money in properly blending the components they currently have in use, but this is not a structural, long-term solution.

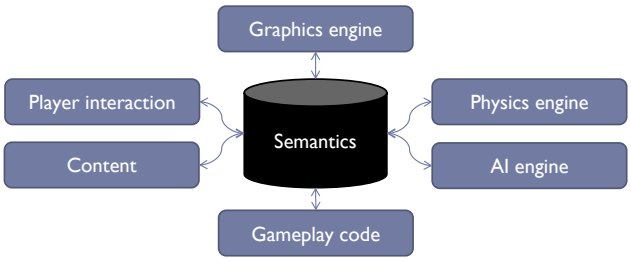

Figure 1: Representing semantics as the centralized object representation communicating with all game components.

Moreover, this coherence is nowadays more difficult to maintain, as the popularity of exploration and sandbox-style games such as *Grand Theft Auto IV* [32] is pushing the demand for bigger, more detailed game worlds. Because it would be too expensive and time-consuming to manually create every detail in these game worlds, the importance of procedural modeling, which can automatically generate parts of game worlds, has significantly increased.

The Sims [23], featuring a complex world with interactive objects and dynamic characters, shows how possible object interactions were specifically built-in for the purposes of that entertainment series [10]. Interactivity is perhaps the most unique element of games compared to other art forms. However, it makes it all the more difficult for their creators to deliver a spatially immersive, consistent, and believable product. It is one thing to design, or generate, a coherent game world, it is yet another to keep it coherent. A game world is not only changing and responding to a player's or agent's actions but also to objects and other entities in the game world. Maintaining a consistently realistic game world is a major challenge for any game developer.

We argue that what is truly missing here is a *glue* to keep everything together: glue to *combine* the different components of a game engine, and more specifically the different representations of the objects used by these components; glue to *integrate* different procedural generation techniques; and glue to *maintain* the coherence when the game world is evolving over time. It seems that an integrated semantic representation is necessary to be applied in the context of game worlds. This approach is shown in Fig. 1.

It is important to wonder why such a powerful idea of a single semantic representation for all game components is not yet used today. Up till now, most game developers only add object and world data in an ad-hoc way, specific to their current game and specific for the component that requires it. An obvious reason is the extra work it involves. It clearly involves more efforts of the designers to add this generic semantic information discussed above. It is therefore important that defining semantics is not too time-consuming, and that such specifications are reusable between different development projects. The reusable nature of such a specification should largely compensate for the necessary extra efforts.

Therefore, in this article we present a novel approach to enrich virtual entities in game worlds with semantics, henceforth called *semantic game worlds*, in order to improve the generation and consistency of those worlds. We prefer to use the term *entities* instead of *objects*, because it encompasses everything that can be inside the world, including *substances* like wood and water, *living entities* like animals and humans (possibly the player), and *abstract entities* like 'a company' or 'an enemy'.

In the following sections, we present the main characteristics of semantic game worlds, and discuss several techniques that we developed to design and use them. After an overview of related work in the next section, we will introduce a semantic model for game worlds in Section 3. Entities in this model will be analyzed and described in more detail in Section 4. The *Entika* framework to support this model is introduced in Section 5. Section 6 continues with a showcase of several applications that could benefit from semantics, after which an evaluation of the model and framework is given in Section 7. Finally, our main conclusions can be found in Section 8.

2. RELATED WORK

Adding semantics to entities in game worlds is no easy task, as there is a serious lack of tools to specify it in current game development environments. Although the role of semantics in virtual environments is receiving increasing attention, so far not much research has been done on adding semantics to game worlds [33], let alone with the purpose of making objects more functional or improving the overall gameplay.

Research in artificial intelligence proposed the notion of *ontologies*, due to the lack of shareable and reusable knowledge bases [12]. In the context of this research, ontologies define the *meaning* of objects and the *relations* between them. Important relationships are *generalization* and *inheritance*, where classes are related, and each subclass inherits the features of its superclass [15]. The class 'car', for example, has 'vehicle' as its parent. Another important relation is *instantiation*, which relates a class with each of the individuals that constitute it, e.g. A 'Ferrari' is an instance of 'car'.

In 1975, Minsky introduced the concept of frame-based knowledge representation [26]. A *frame* is a representation of a stereotyped situation, e.g. 'being in a living room', which contains related information, such as the expectations of what might happen next. By interconnecting related frames into frame-systems, transformations between frames can be defined in order to represent, among others, actions, changes, and cause-effect relations. Frames themselves are organized in a hierarchy, and contain slots to specify particular (attribute) values. Inheritance is used to let sub-frames have access to attributes of their super-frames [28]. The notion of frames gained a widespread facilitation in the research fields of philosophy and artificial intelligence.

A decade after the introduction of frames, Douglas Lenat started the *Cyc* project [22][21]. He attempted to create a global ontology and a large knowledge base to represent common sense knowledge. Over the years, this system, of which parts were made publicly accessible through *OpenCyc*, has been expanded with thousands of concepts, facts, terms, and millions of human-defined relations between them. All assertions are formulated in *CycL*, a frame-based language, and the *CycL constraint Language*, a more powerful, but slower predicate calculus to express more complex knowledge.

Smart objects [19] were a successful proposal for adding semantics to virtual objects, dealing with many of the possible user interactions in a virtual environment. Noticeably, smart objects were primarily devised for manipulation, animation, and planning purposes. An example is an artificial agent that can open a door by moving its hand to the door knob, using the correct hand posture, and turning the knob. To model the interaction steps in agent-object interaction, a framework has been presented to associate smart objects with *user slots* [30]. An agent can only use an object when he has obtained a free user slot for it, after which specific *usage steps* can be consulted. Although smart objects are powerful for interaction purposes, they lack the information of what they are actually useful for. Gutiérrez et al. [13] and Ibanez-Martinez et al. [16] have both proposed other object representation models.

The *Nintendo DS* game *Scribblenauts* [1] is an interesting example of how frame-based knowledge representation, semantics,

and ontologies have been used in the game industry. In the game, players are placed inside an obstacle-filled level and are given a goal, e.g. reach the exit, or collect stars. This can be achieved by making clever use of objects, all behaving in a way one expects. By extracting information from dictionaries and encyclopedias, the developers created a large database of objects, mapped into a hierarchy of classes, and each class having a set of properties, such as physical characteristics and interaction possibilities.

Building upon the related work discussed above, we have described how semantic behavior of game objects can be specified by using *services*, and how this is integrated in the three main phases of the game object design process [20]. In a *specification phase*, generic object classes can be specified (including their attributes and services), after which the *customization phase* allows a selection of these classes to be customized into a concrete game project setting. Finally, instances of those objects can be placed in a game world in the *instantiation phase*. In this article we will put this approach in a broader context, starting with the development of a semantic model for game worlds, introduced in the next section.

3. REQUIREMENTS FOR A SEMANTIC WORLD MODEL

From examining the current limitations of game worlds and studying the related work, we have identified several possible improvements to the design and quality of game worlds. This section derives some requirements for the specification of game world semantics for both the design phase and the run-time phase. These requirements need to make sure the proposed semantics specification will help provide those improvements.

When trying to apply research on semantics to virtual worlds, we notice that much of the important information to store in a semantic representation of a game world is already present in some form or another. Bits and pieces of information are scattered throughout the different components of the game engine. Models are linked to scripts that describe their behavior upon interaction, the AI component stores information about agent behavior, the textures of a model often hold a clue to what materials the model is made of, etc. More often than not, there is no real cohesion between the data in these different components. A wooden table might have a wood-like texture attached to it, but the physics engine will likely have no way of using that information to actually treat that object as made of wood. This means that designers and programmers need to define repetitious data and, moreover, this could lead to inconsistencies in the gameplay that break the player's immersion.

A centralized knowledge base of a semantic game world representation is therefore the basis for many important benefits. This semantic representation should be a consistent source of information that needs to be accessible by all components of the game engine and that is understandable by both man and machine. A semantic model fit to be used for game worlds should, at least, allow designers to express all of the following aspects of game worlds:

- What a geometric model actually represents: what type of object it represents, what classes it belongs to.
- The essential physical or other characteristics of the objects in the game world, e.g. the damage that can be dealt by a sword, what matter it is made of, how it looks, how it sounds.
- The way a player (and other characters) can interact with the game world and its objects.
- How objects relate to each other: how they are placed relative to each other, what their dependencies are, ownerships etc.
- How the objects behave over time, possibly influenced by other objects in the world.

This information will make it possible to have the different game components to gather information from a single, centralized and consistent knowledge base, but it will also allow people working on the game, whether they are artists or programmers, to have a better understanding of the game world they are working on and to have a more expressive language at their disposal when communicating with the machines they work with, e.g. to more easily express their intent when creating procedural content generation algorithms.

In addition, there are requirements to which a semantic model for game world should adhere to:

- Inclusion of semantics should have a low impact on the design pipeline; reusablity is the key in achieving this goal, thereby reducing, not increasing, design efforts.
- The semantic model should provide a wide expressive range to designers; the model shoud allow designers to express their full intent without any limitations.
- Semantics should further enable procedural generation; this allows designers to combine and integrate multiple existing techniques to generate a consistent and coherent whole.
- Designers should be able to define physically sound game worlds; it should be possible to mathematically express dependencies between object characteristics. Note that this does not mean that all game worlds need to adhere to real world physical laws.
- Designers need to be able to approach game worlds and objects from different perspectives, such as shape, texture, composition, function, or behavior.
- The semantic model should provide a consistent way to define interaction with game worlds.
- The world should be kept semantically consistent throughout the whole game; maintaining consistency helps immersion.
- Semantically modeled game worlds should enable emergent gameplay.

In the next section, we will discuss the specification of semantics for entities while meeting the requirements listed above.

4. SEMANTIC ENTITIES

This section proposes our specification model for game world semantics. This model can be used as a ruleset to build ontologies, or alter existing ones, specifically to be used in the context of game worlds. In this model, we define some concepts, both based on real world subdivisions of objects and based on common elements often found in games. Rules and constraints between these concepts are set and explained. We will analyze entity classes in our model and describe which semantics can be specified to support them.

4.1 Entities

An *entity* is the most important concept in our model of semantic game worlds, and is defined as 'that which is perceived or known to have a distinct existence'. To expand the semantics of entities, and easily distinguish one entity from another, we enrich them with several general concepts. First of all, similar to frame-based knowledge representation, there are *attributes*, defined as 'characteristics of an entity'. One can think of attributes like 'mass', 'edibility',

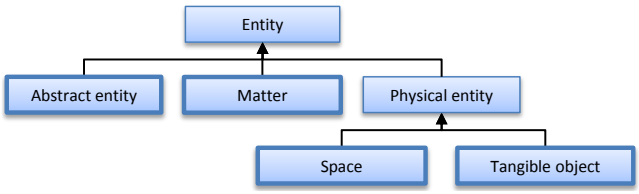

Figure 2: An overview of entities.

'comfort', 'color', and 'health'. Attributes are paired with values, and in order to express those values more clearly, it is convenient to define the notions of *unit category* and *unit*. A unit is any division of quantity accepted as a standard of measurement, while a unit category denotes those units that are derived from the same base unit. The value of 'mass', for example, can be expressed with a 'weight unit', e.g. 'gram', 'pound', or 'ounce'. In addition, we expand entities with *state groups*. A state group is a collection of *states*, of which only one can be active at a given time. An advanced coffee machine, for example, can either be 'on' or 'off', and at the same time, be in 'cappuccino' or 'espresso' mode.

As mentioned before, we have split up entity classes into more specific classes, in order to define more detailed semantics. A hierarchical overview of this is given in Fig. 2. First, we have defined *abstract entities*, as opposed to *physical entities*. Abstract entities do not have a physical representation in the world, even though they are present in some other way. Examples are companies, or, more game-oriented, factions or guilds. In contrast, physical entities are present in the world, have a particular position, and are usually subject to physics.

Probably the most notable physical entities in games are the *tangible objects*, like mechanically produced swords and ovens, and natural objects such as trees and ice cubes. Tangible objects are made up of a type of *matter* (described in the next subsection) that is shaped in a particular form, and thus having a particular quantity. A tangible object is either a separate object, or a compound object that consists of two or more tangible objects called *parts*. A table, for example, can be described as a compound object that consists of a table top (a separate object made of wood), and several legs (separate objects made of steel) underneath. The parts or matter a tangible object is made of, can change its behavior drastically. Without an engine, for instance, a car cannot drive. Besides being used as parts, tangible objects can be used to *cover* other tangible objects, as well bringing along different behavior. A coat, for example, warms the object it covers. One more property of tangible objects is that they can be physically *connected* to other objects, without being a part of it. Electrical devices are the perfect example for this, as they are usually connected to a power outlet with a cable.

In many games, inventories are present in which players can store their collectibles. There are numerous examples in role-playing games, where chests can be opened, after which the items in the chest's inventory can be looted and put in the player's (usually non-physical) inventory. This notion should clearly fit in the above semantic descriptions. Inventories themselves, however, are 'nothing': they are empty spaces, being able to hold items (tangible objects) or store matter. This brought us to the concept of *spaces*: bounded regions for which clear semantics can be specified. They have an extent in the world, even though their shapes are not clearly defined. With spaces, much more possibilities open up. Besides having inventories of chests (possibly filled with coins) and bottles (possibly filled with water), for example, a pocket in a jacket is a possible space as well, just like a parking space. With spaces, one can even define areas with particular semantics, such as no-walking zones, checkpoints (for racing games or saving purposes), or areas that trigger a booby trap.

4.2 Matter

We stated that tangible objects consist of matter. By analyzing matter, it is possible to define some interesting semantics. Instead of starting from an engineering point of view, we will look through the eyes of a physicist. Put simply, anything that has mass and occupies space is called *matter* [27]. Matter consists of chemical *elements* like hydrogen and oxygen, which are represented in a periodic table, and are assigned an atomic number and symbol. By combining chemical elements, matter is formed. Depending on a set of physical conditions, it has a particular state, represented by a gas, liquid, solid, or plasma form. In-game, visualization of matter depends on this state, varying from 3D models to particle/fluid systems.

We have split up matter in four different types. The first type are *substances*, which either consist of pure elements, or a combination of multiple elements through chemical reactions. Water and sugar are two examples, as they cannot be broken apart without chemistry. For the second type, we go one step further, by blending substances together -physically, not chemically- in order to form a *mixture* [11]. Tungsten steel is an example of a mixture. When two substances, possibly having their own semantic behavior, are mixed together, they make place for a new mixture, possibly having its own unique behavior. In role-playing games, for example, this might be useful for the creation of potions, where different ingredients mixed together can result in a potion with its own specific effect(s). This already happens in the game *The Elder Scrolls V: Skyrim* [5]. *Compounds* are similar to mixtures, but these are combined chemically, like seawater, nicotine, and shampoo. Finally, we have included *materials*, which are either raw or semi-finished (the result of mechanically processing raw materials). Some examples are cotton and ore (raw), and steel (semi-finished). This semantics can be very useful for games that deal with resources, which are usually building simulations or strategy games like *Heroes of Might and Magic* [29] and *Warcraft* [7], or the more recent mobile phone app *Alchemy* [3].

By defining matter, tangible objects will inherit semantics in the form of physical and chemical attributes, allowing a wooden table top to burn, for example, in case one has defined that the material 'wood' is flammable. In addition to this, we allow each tangible object to have one or more *layers*, made of a type of matter. Paint and anti-oxidation are two examples, the latter changing the semantics of steel when applied to it. Insect repellent is another example.

4.3 Relations

Thus far, we have described various semantics of indivual entities. Now, we would like to introduce a few more concepts that relate entities to each other.

First, a *family* is any number of entities that satisfy a set of conditions. Given all defined concepts above, one might include all red and non-rotten apples with a mass of at least 100 grams, or all small wooden tables. Instead of being derived based on conditions, one could also manually combine related entities into a *group*. An example of a group is a farm, which may contain a farmhouse, a field, and a cow.

A *predicate* is similar to adverbs in natural language; examples are 'big', 'beautiful', and 'tasty'. A unique aspect of predicates is that they cannot be defined without the point of view of another entity, and conditions on attributes or states. Humans may find

a building 'tall' when its height is greater than a particular value, while a giant may consider this 'small', having other conditions for the predicate 'tall' and 'small'.

Finally, one can think of more relations, such as family relations ('Jane is the mother of John'), alliances between countries, ownership relations ('a person owns a book after buying it'), or placement relations ('a table should be placed on the floor'). To allow the specification of these relations, we have defined a *relationship type*. For each relationship type, *relationships* can be defined between a source and target entity.

4.4 Services

Besides describing the physical properties of entities, semantics can also be used to specify their behavior. This not only allows designers to create a dynamic game world in which entities undergo global changes, but it also enables them to let players interact with entities in a way they expect.

The notions introduced in the previous subsections give us a foundation for the definition of *services*, first mentioned in [20]. In the real world, entities have particular functions and provide services, which should also be the case for entities in a virtual world; for example, a jacket has the service of providing warmth to the person wearing it. As such, services are a very powerful way to express semantics in game worlds. We define a service as 'the capacity of an entity to perform an action within a context'. We will split up this definition and discuss each part in more detail.

Because we want to specify the behavior of each type of entity (whether it is a book that provides knowledge, a plane that is able to fly, or a fridge that keeps beverages cool), we define services for entities, both abstract and physical. We want to define what they can do, what their behavior is, and how they react to others; hence the term 'capacity'. The process that is performed by an entity is described as an *action*. Actions can be seen as abstract descriptions, defining what happens; they are like verbs in natural language: play, boil, eat, shoot, etc. Related to this term, we will also use the notion of *event*. This is an action that is accompanied by an *actor*, and possibly a *target* as well. Some examples are the following (with the last one having a target): a kid can play, water can boil, guns can shoot, and humans can eat food. The generalization ontology that we have included in our model plays an important role here. For example, if a gun can shoot, inheritance will make sure that pistols and machine guns, both children of the 'gun' class, can shoot too. One more example, where humans need a key to unlock a door, introduces another aspect of events: *artifacts*, which are tangible objects that are used to successfully perform the action.

Within our definition of a service, we used the notion of *context*. This context describes all the *conditions* that should be satisfied in order to trigger an event. There are many types of conditions, a spatial condition being one of them: if the player is in the vicinity of an enemy, it will attack him; only buildings that are within a distance of 5 km will be polluted by a factory; an alarm will be triggered when there is movement within 10 meters. Conditions can also be based on the above concepts: an oven should be on before it heats up the things inside it (states); when the level of health has reached zero, the player will die (attributes); a car will only drive with an engine (parts); a fish should be in water in order to live (matter and spaces); a computer should be connected to a power cable (connections); an object made of wood can break (matter); kids will only eat food which they think is tasty (predicates); a key will only open one specific door (the one with which it has a relationship).

Even though conditions may be satisfied, if nothing happens, there is no real purpose to define an event. Therefore, events have at least one *effect*, which we have split up in either of the following:

- *Reaction*: A reaction is another action that is performed by an entity (thus an event itself), possibly, but not necessarily, the actor or target of the original action. This action-reaction principle can be seen in many situations: when a button is pressed, an elevator appears; if an elf attacks an orc, the orc will fight back; in case the player enters a room, a trap will be triggered, etc.
- *Change*: There are many changes one can think of, most of them being based on the concepts described in Section 4, with changes on attribute values or states the most common ones. Given are several examples: an oven will increase the value of the 'temperature' attribute of the objects that are inside it; a locked door will change to the 'unlocked' state when it is unlocked; when eaten, a cookie will decrease the hunger level of the one that ate it; if a jacket is worn, it will keep the wearer warm; while running, a human will become fatigued.
- *Creation/deletion*: Real-time strategy games often show the creation of new military units from barracks, usually only possible when particular resources are supplied. A bomb will delete itself from the game world when it explodes, just like a cookie when it is eaten.
- *Transfer*: Vending machines, such as the ones in *BioShock* [17], can supply weapons to humans (e.g. the player). Before this transfer takes place, the human should have transferred a coin to the machine.
- *Transformation*: In a game like *The Sims* [23], children will eventually grow up; they are transformed into adults. Although they will keep their attributes (like age and preferences), being an adult opens up new interaction possibilities and behavior.
- *Relationship establishment*: This effect will establish a relationship between two entities. For example, when buying a new telephone, a relationship is established between the phone and a number, and another relation between the phone and the buyer.

Different events can be defined for one particular type of action, having different actors, targets, contexts, and effects. Take the action 'eat', for example. If a dog eats a dog snack, that will decrease its level of hunger, and make it happy. For humans, the effects will likely be different, so it may be wise to define another event for a human. Furthermore, events and effects should be accompanied by a notion of *time*, indicating how long an event should be executed and how long an effect should last. Some events are discrete and will only occur one or a few more times (e.g. flipping a switch), while others are continuous for a fixed amount of time, or everlasting (e.g. getting hungry). The same applies to effects: applying a bandage might immediately heal the target, while drinking a potion might heal the target gradually over some time.

5. SEMANTIC FRAMEWORK

To validate the approach presented in the previous sections, we have developed *Entika*, a framework that enables game developer teams to easily declare semantics for their virtual world. *Entika* consists of two different modules, and their combined use makes a very convenient way to design a semantic game world. First, in order to enable game designers to quickly specify semantic entities, we have created the *Semantics Editor*. Second, to relieve game programmers from handling all the semantics during run-time of

a game, we have developed the *Semantics Engine*. Any design changes with the Semantics Editor will automatically be picked up by the Semantics Engine, thus allowing developers to easily refine and test the semantics in their game. Both modules will be discussed in more detail.

5.1 Semantics Editor

The Semantics Editor makes it possible for game designers to edit libraries of entity classes and all other concepts of Section 4. The idea behind this editor is to specify new classes (including tangible objects, attributes, and actions), and modify or remove existing ones. Each concept is represented by its own library. To do so, the editor provides the user with an overview of the available libraries, and the classes that populate them. Each class can be specified in great detail, meaning that its semantic information can be fine-tuned at will, ranging from its name and description, to relations between that class and other classes. For example, a tangible object can be equipped with attributes, but it is also possible to define the matter of which it is made, what its relationships with other objects are, or which actions it can perform. Because of the implementation of the generalization ontology, derived classes will inherit the semantic information (including attributes and events) from their parents, although specific values can be overruled if necessary. When, for example, the 'physical object' class is assigned the 'mass' attribute, each underlying child class will inherit this attribute, but the specific mass value can be modified for each one of them.

In addition to generic and reusable classes, game-specific objects can be created by basing them on a tangible object. Besides customization of inherited semantics, game objects can be further customized with references to e.g. geometric models, textures, and audio files. For each of these extra properties, conditions can be specified to indicate when they have to be used. For example, only when a radio is in the 'on' state, it should play a particular music file.

Fig. 3 shows a screenshot of the Semantics Editor. On the left, all libraries are displayed; on the right, some of the semantics of the selected 'human being' class are shown. The user is able to modify its names and description, and observe a list with its parents and children (from the generalization ontology). Because of inheritance, the human being has a 'health' and 'mass' attribute, defined at one of its parents. In addition, the human has an attribute of its own, 'hunger', having a default value of 10, and ranging from 0 to 100. Usability was one of the aspects that was aimed for when developing the editor, which has been achieved by providing a clear and distinctive overview of all information, the possibility to hide unwanted information, and providing user-friendly ways to quickly specify semantic information. Actual storage of the libraries is done in databases, making the semantic information easily and quickly accessible to game programmers, even without the editor.

5.2 Semantics Engine

Analogously to what a physics engine does with in-game physics, the Semantics Engine maintains the semantic consistency of the world during run-time of a game. After creating entities in the Semantics Editor, instances of them can be placed in a game world. By making use of the Semantics Engine, game programmers do not have to implement the execution of semantic behavior of these instances, as the engine is charged with this handling.

The engine has several main features. First, it maintains all the game worlds that have been created, and all the instances that have been placed inside them. During run-time of a game, the engine

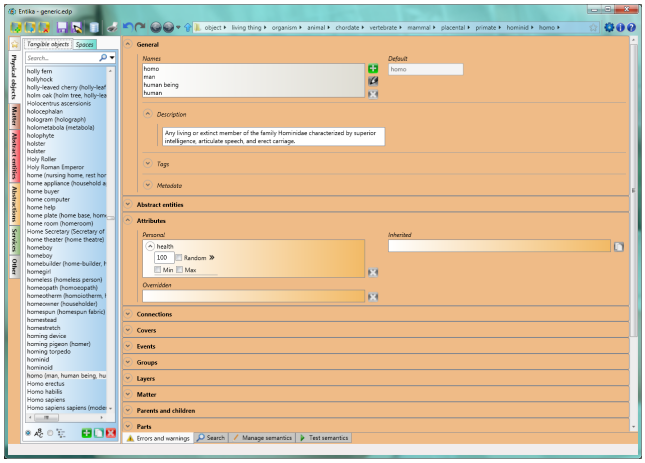

Figure 3: A screenshot of Entika's Semantics Editor.

updates all instances, and checks whether they have any active services. If so, when the requirements have been satisfied, events are executed, and the corresponding effects are applied. This means that attribute values can be changed continually, physical object instances can be moved from one space to another, substance instances can be mixed together, and new instances can be created, among many other possibilities. In turn, these changes could result in the activation of an inactive service, of which the effects will be applied from then on. For example, if someone uses the 'turn on' action on an oven, it will get in the 'on' state, making it heat up its inventory items as long as it stays on.

Note that even though the Semantics Engine can create or remove instances during its updates, this does not automatically mean that their required audio-visual content is actually added to or removed from the game itself. It is still up to the game programmers to handle this. However, the engine does give a notification when this should happen. Because of this portable behavior, the engine can be used in many games.

The engine also offers game programmers several useful tools to improve in-game object interaction. An example is requesting what actions are useful to perform on a certain instance, which the programmer can use for the graphical user interface of a game, or whatever way he chooses to let the player decide what action he wants to perform. For example, when a player selects a door, the engine might (dependent on the defined semantics) propose the action 'close' when the door is in the 'open' state, while it might suggest 'open' and 'smash' when it's 'closed'.

Because of the Semantics Engine, instances are constantly updated, resulting in a lot of dynamics, assuming any semantics have been defined for them. This leads to adaptive game worlds that change over time, forcing the player to adapt as well and think about the results of an action, but also allowing him to think creatively to accomplish something. This is a great improvement with respect to many games that are currently available. However, it leads to a downside if the addition of semantics to games results in too much extra memory overhead, or if processing the handling of services requires too much time. In order to measure the performance of the current Semantics Engine, we have created three simple test cases. To provide reliable and representative results, we have decided not to use a world with different entities, but a world with instances that are all based on the same entity. We have upscaled everything to the extreme -the execution time of single events is less than a millisecond- to check whether the engine is

Table 1: The update time (in ms) of Entika's Semantics Engine

Nr. of instances	10	100	1,000	10,000	100,000
Test case 1	0	0	5	58	632
Test case 2	0	0	2	30	351

capable of handling a huge amount of semantic instances.

1. In test case 1, humans are becoming hungry, by having a service that has an effect on their 'hunger' attribute each second.
2. For test case 2, we let weapon factories produce one new sword per second.

The update times of the engine for 10, 100, 1,000, 10,000, and 100,000 instances have been measured, and are shown in Table 1. For all test cases, we have used an Intel Core i7-870 2.93 GHz PC with 8 GB RAM. For test cases 1 and 2, having 1,000 instances and services requests more of the engine than 10 instances, but performance is still very well. Even with 10,000 instances, the engine will not slow down a game that much. The extreme scenario of 100,000 instances shows that the engine needs to be optimized, which was not our primary focus during this research. It is clear that the engine is not yet capable of handling the events of thousands of instances at a time, although these extreme scenarios are unlikely to appear in real game worlds.

6. APPLICATIONS

Semantics is a powerful means for game development, and can be applied to several domains. In this section, we will shortly discuss some of the possible applications that are built using our semantic model.

The field of procedural content generation can benefit in at least three ways. With the concept of relationships, as described in Section 4.3, placement relationships can be defined between physical objects. These relationships can then be used for *semantic layout solving*. The layout solving approach we have developed uses step-by-step procedures to add objects to scenes in a valid and logical way. In this manner, an entire scene layout can be generated automatically. The defined placement relationships decide what is a valid location for an object. Every physical object can have a number of placement relationships defined that describe where and how instances of that physical object should be positioned in relation to instances of other physical objects. A more in-depth description about the use of our model in semantic layout solving can be found in [33] and [35].

The integration of procedural content generation techniques to form complete buildings is still largely unexplored, limiting their application to open game worlds. In [36], we propose an approach that integrates existing procedural techniques to generate such buildings. With minimal extensions, individual techniques can be coordinated to create buildings with consistently inter-related exteriors and interiors, as in the real world. The solution, making use of the semantics of building elements, consists of a framework where various procedural techniques communicate with a moderator, which is responsible for negotiating the placement of each element.

Once game worlds are finished, they are usually static and therefore valid in the context and the situation for which they were built. But a game might need the same world under different circumstances. Instead of forcing designers to manually rebuild every element of the game world all over again, we proposed the concept of using *procedural filters* [34]. Procedural filters provide a layer of customization that can be applied to a finished game world or the objects therein. These filters will not structurally change the world, but add or change its finishing based on a new context. For example, we might want the same game world in springtime for one level of the game, but also in wintertime for another level. Filters use building blocks to alter the visual appearance of objects in the game world. A graph built up of these building blocks represents the procedure that is to be followed by the filter. By using the semantic information available in the scenes, the filters can more easily fine-tune the appearance to the specific circumstances of the game world and of the objects themselves.

Finally, with the addition of a semantic layer to game worlds, agent behavior can be improved by letting them make use of semantic entities. *Entika* supports the use of agents, by providing several methods to search through the instances in a game world. In case an agent is looking for a specific entity, the Semantics Engine can provide information about all instances of that entity, so the agent can find its way towards them and interact with them. For this, planning and pathfinding techniques are typically required, such as *drives* and *backward chaining*; see for example [2], [4], and [25]. Instead of making an agent aware of everything in the game world, future work may include agents that have their own *worldview*, in which learning patterns are applied to let an agent update its knowledge about the whereabouts of others when he comes across them. Furthermore, future work might focus on *roles* and *tasks*, and familiar concepts from the BDI agent model: *beliefs*, *desires*, and *intentions* [8][31].

7. EVALUATION

In this section, we discuss the evaluation results of our semantic model. We performed a number of interviews with game developers, both designers and programmers, and talked about our main problem statement -the lacking object behavior as opposed to the increase in graphical realism- and the proposed model. Interviewees also had the opportunity to try out the Entika proof of concept editor by creating a scenario we prepared for them, based on some notions from basic city building game.

The interviewees confirmed they do not see any significant increase in realism (or higher detail) in current object behavior and interaction. However, many of them did mention a huge increase in destructability: many games now allow all (or many) objects in the game world to be destroyed upon explosions or gunshots. However, when linked to the proposed model, a concern was raised about the destruction of objects, leading back to theories about existence and the philosophical question of what makes an entity that entity. Consider a television that is cut into pieces. Is it still a television? Does is still provide its usual services? But what about a piece of paper? Even after cutting it, it is still writable. And a candle? It will only remain a candle as long as it has not burned up. With more realism in games, and more possibilities to destroy (objects in) the environment, interviewees deemed it necessary to study how partially destroyed objects get along with semantics, which we do not deal with in our current approach. We do make use of the *aggregation* ontology [15], though, where parts are related to the entire assembly ('a wheel is part of a car'). This ontology might be a step in the right direction for total destruction.

From the perspective of our problem statement, two interesting notions surfaced by an interviewee: *internal consistency* and the *domino effect*. With internal consistency, the interviewee meant that it is not always necessary to have a world that is consistent with the real world, although internal consistency is necessary. It is unacceptable to have two similar objects behave in a different way. However, this is still often the case: sometimes a particular object

is usable in one level, because it is associated to the story, while similar objects in other levels are no longer usable or behave differently. This would suggest that a centralized, consistent semantic representation of the game world and the objects could definitely increase immersion by helping developers to maintain this internal consistency. With the domino effect, the interviewee referred to being allowed to set up constructions of multiple objects that, when combined, can set off a huge chain of events where the effect of one object triggers an action in the next, like in *Little Big Planet* [24] or *The Incredible Machine* [18]. The interviewee thought it would be nice if a player could set up similar constructions in, e.g., a first person shooter to defeat an enemy in an ingenious, creative and original way. However, he also mentioned that sometimes it is more fun to keep it less complicated so the player can quickly see what objects can be interacted with, and which cannot. Nonetheless, it does seem that detailed object behavior would spark players' creative thinking. Although this does not cater to all players, it certainly suits certain playing styles and game genres.

Regarding this last comment, it should be stressed that a semantic game world isn't but a (powerful) means to serve the gameplay, and will never automatically make dispensable the creative work of designers. Care should be taken to avoid overloading entities with superfluous semantics, as semantics make virtual entities not only behave more as one expects, but more complexly as well, which could end up undermining the gameplay. Although this was not the case for *Scribblenauts*, the *Xbox 360* version of *Alone in the Dark* [9] supports this: the need to replace flashlight batteries once in a while received many complaints of gamers, resulting in the removal of this function from the later released *PlayStation 3* version. Game designers will always have to think carefully on which semantics are desired in a game, and how to achieve a good balance between a world with convincing behavior and good gameplay. Semantics combined with entities that appear in the background, or entities that are used by agents, for example, will result in a more convincing world in the eyes of players, without influencing them directly. For entities that are in reach of the player, it might often be wise to moderate their level of detail.

Along these lines, it also important to note that the semantic framework can work side by side traditional methods of gameplay programming. It is not necessary to employ the framework for all aspects of gameplay, since that would lead to far. Elements of gameplay that are vital for the game and are very complex, involving a lot of tweaking, are still better to be left to the traditional approach. A clear example, is the behavior of cars in racing games or other games where driving is an essential part of the game.

The reactions of the interviewees to the semantic world model were generally positive. They were unanimous that such a model could definitely be useful: it would make it easier to extend games, there would be opportunities for games with multiple storylines (branching story arches), it could be a solution to provide emergent gameplay, and it could make the development process easier. Without mentioning semantic layout solving, one interviewee saw possibilities in automating game world generation, and potential in the relationships between objects. There were, however, some concerns about performance and scalability of a potential implementation. Besides, it would make games more *unpredictable*, which makes debugging and testing much harder.

The ease of use of our proof of concept editor was still a concern, especially with interviewees with a less technical background. To be really practical and usable, a great deal of effort needs to be put into the editor's usability. The main challenge is to offer all the flexiblity and all the options available in the current editor, but representing them in such a way that users are not overwhelmed by all these options. From the interviews it became clear that the user (especially when learning to use Entika) needs some more guidance, comparable to wizards, or through the use of examples.

One interviewee saw some in-game advantage in the fact that it would allow easier creation of simple simulations. As an example he mentioned the game series *The Sims*. However, with more of these interaction possibilities, more in-game animations are required to preserve immersion. Although motion captured animations still result in more fluidity than other current techniques, capturing them for all possible interactions is unwanted. To circumvent this problem, the technique used for smart objects [19] might be applied. Similarly, the approach used in *Twig* [14], a procedural animation system that supports physical interactions between characters, might be helpful.

Finally, some interviewees immediately mentioned that specifying all semantic information would take up a lot of time. It is indeed an interesting question when and by whom the semantic libraries have to be populated. Although this is not the topic of our current research, several possibilities can be devised, from letting game developers design the libraries and define relations for each game (which isn't very practical), to letting them incrementally fill generic libraries (throughout several development projects), from which they would derive a specific library for each game. The latter has the clear advantage that in each subsequent project, more classes will be available. It is obvious that this does not force designers to reuse entities exactly as they are specified by others, or in other projects. New additions and relations can always be made and established. Furthermore, this means that designers do not have to limit themselves to traditional meanings and behavior of existing entities: designers that are willing to add original, unusual behavior to common objects are therefore not prohibited to do so.

8. CONCLUSIONS

Despite exuberant visuals, most current games considerably lack consistency between the visual representation of the world and the way it feels, behaves, and reacts. In this article, we argued that this is due to missing semantics, and because designing game worlds with semantics poses especially difficult challenges, including the inherent complexity of maintaining and upscaling all interactions among entities. We presented a solution to the problem in the form of *semantic game worlds*. We set up a sound model for these worlds by stating several requirements, and by splitting up the entities populating them into more specific classes with their own unique properties, such as attributes, matter, and services.

This approach has been implemented and validated by means of the integrated *Entika* framework which effectively supports a simple and intuitive definition of semantics. Among the numerous advantages of this approach, its Semantics Editor promotes reusability of previously specified entity semantics, and easily supports behavior customization as required by each specific game. Furthermore, specified semantics seamlessly blends with our Semantics Engine, charged with all semantics handling during the game.

In semantically rich game worlds, semantics influences the presentation, physics, and behavior of entities, and their awareness of other entities. We believe that enabling designers to create these worlds will be instrumental to achieve a more consistent world experience. Besides having entities that are visually convincing, more and better interaction can be accomplished, due to plausible and expectable behavior. This in turn is considered one of the key conditions to significantly improve gameplay. The approach presented here, giving designers the possibility to include convincing semantics, while keeping much control on the fine-tuning of their entities, is a valuable aid in that direction.

In the future, we would like to experiment with several extensions for *Entika*. One of them is the annotation of 3D models, to indicate where exactly a player or agent can interact with them. As this is dependent on geometric models that come in a wide variety of file formats, annotation might pose some difficulties to keep the framework generic. Another possible extension is procedural destruction, based on the matter of a tangible object. Improving agent behavior, including roles, tasks, beliefs, and desires, based on semantics, is yet another promising extension. Next to the aforementioned extensions, we will keep improving the interface of the Semantics Editor to achieve more usability, and optimizing the performance of the Semantics Engine, in order to furhter reduce its overhead in computationally expensive games. All together, we are confident this will improve semantic game worlds and facilitate their widespread deployment.

9. ACKNOWLEDGMENTS

This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO).

10. REFERENCES

[1] 5th Cell. Scribblenauts, 2009. Warner Bros. Interactive.

[2] T. Abaci and D. Thalmann. Planning with smart objects. In *The 13^{th} International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision: WSCG*, pages 25–28, 2005.

[3] Andrey Zaikin. Alchemy, 2011. Android market.

[4] R. Aylett, A. Horrobin, J. O'Hare, A. Osman, and M. Polshaw. Virtual teletubbies: reapplying a robot architecture to virtual agents. In *Proceedings of the Third Annual Conference on Autonomous Agents*, pages 338 – 339, 1999.

[5] Bethesda Game Studios. The Elder Scrolls V: Skyrim, 2011. Bethesda Softworks.

[6] S. Björk and J. Holopainen. *Patterns in game design (Game development series)*. Charles River Media, December 2004.

[7] Blizzard Entertainment. Warcraft: Orcs and Humans, 1994. Blizzard Entertainment.

[8] M. E. Bratman. *Intention, plans, and practical reason*. CSLI Publications, 1987.

[9] Eden Studios. Alone in the Dark, 2008. Atari.

[10] K. D. Forbus and W. Wright. Some notes on programming objects in the sims. 2001.

[11] V. Gold. *Compendium of chemical terminology*. International Union of Pure and Applied Chemistry (IUPAC), 1997.

[12] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5:199–220, 1993.

[13] M. Gutierrez, F. Vexo, and D. Thalmann. Semantics-based representation of virtual environments. *International Journal of Computer Applications in Technology*, 23(2/3/4):229 – 238, 2005.

[14] I. D. Horswill. Lightweight procedural animation with believable physical interactions. In *IEEE Transactions on Computational Intelligence and AI in Games*, volume 1, pages 39–49, 2009.

[15] M. N. Huhns and M. P. Singh. Agents on the web: ontologies for agents. *IEEE Internet Computing*, 1(6):81–83, 1997.

[16] J. Ibánez-Martínez and C. Delgado-Mata. A basic semantic common level for virtual environments. *International Journal of Virtual Reality*, 5(3):25–32, September 2006.

[17] Irrational Games. Bioshock, 2007. 2K Games.

[18] Jeff Tunnell Productions. The Incredible Machine, 1992. Dynamix.

[19] M. Kallmann and D. Thalmann. Modeling objects for interaction tasks. In *Proceedings of the Eurographics Workshop on Animation and Simulation*, pages 73–86, 1998.

[20] J. Kessing, T. Tutenel, and R. Bidarra. Services in game worlds: a semantic approach to improve object interaction. In *Proceedings of the International Conference on Entertainment Computing*, pages 276–281, 2009.

[21] D. B. Lenat. A large-scale investment in knowledge infrastructure. *Communications of the ACM*, 38(11):33–38, November 1995.

[22] D. B. Lenat and R. V. Guha. *Building large knowledge-based systems; representation and inference in the Cyc project*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989.

[23] Maxis. The Sims, 2000. Electronic Arts.

[24] Media Molecule. Little Big Planet, 2008. Sony Computer Entertainment.

[25] I. Millington and J. Funge. *Artificial intelligence for games*. Morgan Kaufmann Publishers, 2 edition, 2009.

[26] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, 1975.

[27] J. Mongillo. *Nanotechnology 101*. Greenwood Publishing, 2007.

[28] B. Nebel. Frame-based systems. In R. A. Wilson and F. C. Keil, editors, *The MIT Encyclopedia of the Cognitive Sciences*. The MIT Press, 1999.

[29] New World Computing. Heroes of Might and Magic, 1995. 3DO.

[30] C. Peters, S. Dobbyn, B. MacNamee, and C. O'Sullivan. Smart objects for attentive agents. In *Proceedings of the International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision*, 2003.

[31] A. S. Rao and M. P. Georgeff. Bdi agents: from theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*, pages 312–319, 1995.

[32] Rockstar North. Grand Theft Auto IV, 2008. Rockstar Games.

[33] T. Tutenel, R. Bidarra, R. M. Smelik, and K. J. de Kraker. The role of semantics in games and simulations. *ACM Computers in Entertainment*, 6:1–35, 2008.

[34] T. Tutenel, B. Bollen, R. van der Linden, M. Kraus, and R. Bidarra. Procedural filters for customization of virtual worlds. In *PCG '11: Proceedings of the 2011 Workshop on Procedural Content Generation in Games*, New York, NY, USA, 2011. ACM.

[35] T. Tutenel, R. M. Smelik, K. J. de Kraker, and R. Bidarra. Using semantics to improve the design of game worlds. In *AIIDE '09: Proceedings of the 5^{th} Conference on Artificial Intelligence and Interactive Digital Entertainment*, Stanford, CA, USA, October 2009.

[36] T. Tutenel, R. M. Smelik, R. Lopes, K. J. de Kraker, and R. Bidarra. Generating consistent buildings: a semantic approach for integrating procedural techniques. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):274–288, 2011.