Understanding Everything NPCs Can Do

Metrics for Action Similarity in Non-Player Characters

J. Timothy Balint
Delft University of Technology
j.t.balint@tudelft.nl

Jan M. Allbeck
George Mason University
jallbeck@gmu.edu

Rafael Bidarra
Delft University of Technology
R.Bidarra@tudelft.nl

ABSTRACT

Non-Player Characters (NPCs) have actions that allow them to reason about what they can do in a game and how they can do it. The background information about what they can do are the *components* of the action, and how they can do it is the *form* or *shape* of an action, which may be built up from several sub-actions. The components and shape of an action must be fully defined in a game, which can be tedious when several similar actions are needed. Furthermore, as the number of nuanced actions grows, more pressure is placed on an already constrained reasoning system. By discovering the similarities between actions an NPC can do, a given action set can be intelligently organized with similar components being generalized using an action taxonomy. To understand the similarity between actions we have developed measures of action similarity based on their constituent components and form. From this, we discover a metric to determine the generalization ability of an organization strategy for an NPC action set. We examine the use of our measures on a previously developed action set to show the nuances between those actions. Lastly, we find that intelligently organizing actions has a positive effect on virtual character reasoning abilities.

CCS CONCEPTS

•**Computing methodologies** → *Knowledge representation and reasoning;*

KEYWORDS

Artificial Intelligence in Games, Reasoning, Behavior Organization

ACM Reference format:
J. Timothy Balint, Jan M. Allbeck, and Rafael Bidarra. 2018. Understanding Everything NPCs Can Do. In *Proceedings of FDG, Malmö, Sweden, August 7-10, 2018 (FDG'18),* 10 pages.
DOI: 10.1145/3235765.3235776

1 INTRODUCTION

Non-Player Characters (NPCs) have become ubiquitous in games and simulations [15], with control of these characters performed through actions [22, 34]. These actions allow for greater control over virtual characters and provide them with the ability to utilize techniques that create more emergent and immersive environments. However, the term *actions* has several different meanings in games and we use the following definitions:

- *Primitive Actions* are the simplest form of an action. They are atomic entities with few commands on how they affect the world.
- Extra knowledge such as the objects that participate in the action (such as weapons or other characters) are *components* of the action [5, 10, 11, 19, 31]. Components can also include the start and end states of the world, known as conditions and assertions.
- Multiple smaller actions (sub-actions) can be chained together to create a *behavior*. Tree behaviors, such as Hierarchical Task Networks (HTNs) or Behavior Trees (BTs) are the most widely used form [5, 13, 18, 24, 29, 30]. The behavior itself is a component of an action. This gives the general definition of an *action* as a rich representation of knowledge and function.
- The collection of all actions a character can perform is called their *abilities*.

Actions have many roles in games from the obvious NPC control [24, 39] and narrative generation [28], to describing what is offered by a game world [19, 32], and even for game world specification [37, 41]. Because actions consist of components such as the behavior and their conditions to begin, there has been more focus on the components of the actions themselves. From the beginning [11], character planning has focused on the components of actions (namely their conditions and assertions). The behavior component of actions, useful for NPC control, has received a great deal of attention, specifically in their creation [8, 23, 29, 39]. However, behaviors by themselves only contain the form and must be encapsulated in an action with other components to be easily accessible by agent reasoning systems. This has led to representations of actions to be proposed in order to expand on their components [5, 31, 40]. These proposals have seen the need for encapsulating behaviors with other knowledge to assist NPCs in reasoning about what they are doing. However, there has not been a through exploration into comparisons that use the total knowledge of the actions. Understanding similarities between actions allows both game characters and designers to understand where nuanced differences exist between those authored actions.

More varied actions for NPCs requires a simulation author to fully define each of these actions, which becomes prohibitive as the number of actions and types of components (the character's *domain*) grows. It also requires more computation time for any reasoning abilities of characters. As more actions are added to the character's abilities, they need to be reasoned over to determine if that action should be performed. For example, if a character can *walk* to a given location, and has a destination to travel to, then the character should easily choose to perform that action. However, if the NPC can *Walk*, *Run*, *Ski*, *Shoot*, or *Sit*, some of which will cause the character to

FDG'18, Malmö, Sweden

travel to a given destination, the NPC must reason over the entire set to choose the action it should perform. Given that behavior planning is expensive in an already constrained simulation step budget [12], providing more actions causes the character's decision making to be computationally infeasible. However, by intelligently pre-computing similarities between actions, reasoning systems can infer knowledge, reducing the total amount a character must reason over. This means that if similarities exist between actions in a character's abilities, then organizing and generalizing those actions, such as into an IsA (concept generalization-specification) taxonomy, would intelligently contain the character's abilities and save reasoning time.

Simulation authors should have the ability to create more subtly varied actions and the nuances of those actions should have a minimal impact on NPC reasoning performance. Therefore, we propose that the abilities of NPC characters be arranged into taxonomies based on the similarities and overlap of those actions.. The main contributions of this work are:

- A method to compute similarity between game character actions. Specifically, we describe metrics to determine similarity between actions by both their components and behavior.
- Measures to compare taxonomies based on the needs of the NPC.
- Evidence that pre-computing similarities between actions and representing that as a taxonomy allows the characters to reason more quickly about their abilities. This leads to an application-based approach to creating NPC taxonomies.

This work is organized in the following manner: Section 2 describes other attempts to design metrics and taxonomies for game characters, as well as examines common features between different action representations. Section 3 discusses how a game world can be organized, while Section 4 describes metrics to compare action at a component level (Section 4.1), behavior level (Section 4.2), and ability level (Section 4.3). Finally, Section 5 shows the efficacy of our method, examining both the behavior and cost of different ways of organizing an action taxonomy.

2 RELATED WORK

Organizing actions into a taxonomy is not a new concept in itself and has been examined for many years for both game characters and other AI fields such as robotics. The work of Bindiganavale et al. [5] has a taxonomy of actions as part of an overall ontology (called the *Actionary*) for virtual agents. This work provides a high level specification for the *Actionary*, but does not provide a specific taxonomy. Similarly, the work of Badawi and Donikian [1] provides generalizations for virtual agent actions in respect to acting sites on objects (the actions that would accompany Smart Objects [14]). Tenorth and Beetz [33] designed an ontology for use with robotic movement, including a taxonomy for action classes. These classes were designed with different components of a robot (i.e, *closing a gripper* is a child of *closing something*). The development or removal of components would therefore require a new taxonomy, which should be expected. Finally, Balint and Allbeck [3] designed an action taxonomy based off of WordNet Hypernyms. Considering all those proposals, organizing actions into hierarchies has seen limited success for both NPCs and robotics. The previous work has taxonomies that are either underspecified [5] with no organizational guidance or over-specified [1, 3, 33] for a given project. Over-specification is due to taxonomies being proposed, which meet the particular needs of a project but have not been able to be generalized into a broader context. Instead of proposing a specific ordering of actions, we take a general approach based on similarity of actions, leaving the actual organization to be determined based on the end use of the actions. This provides a process to creating a taxonomy while not restricting the taxonomy to a given project.

In addition to context specific ways to organize actions, there have been a few attempts to compare and contrast them. De Silva and Padgham [9] showed that HTNs are analogous in structure to another tree-based structure (Plan Trees). Botea et al. [6] used other domain knowledge to extract generalizable information from actions. Their work focuses on creating new actions by finding similarities between actions. Finally, Sagredo-Olivenza et al. [29] learned by demonstration how to modularize BTs. They used the *Levenshtein distance* to determine similarity between two actions. We generalize this method by providing multiple metrics that can compare the total structure (which can be represented as trees). Furthermore, unlike the previous methods which only find matches to generalize, our method provides a cost function that allows game characters to determine non-exact relations between their abilities.

Most closely related to this work is Kapadia et al. [16] and Sagredo-Olivenza et al. [29]. Kapadia et al. evaluates the control cost of a single BT over multiple ways of representing that same plan. This method only judges a node if it is connecting other nodes together (i.e, it does not look at the leaf primitive actions) and is not concerned with comparing different actions in the same set. Sagredo-Olivenza et al. compare learned behavior trees using the *Levenstein distance* on a sequence of action nodes. Therefore, they ignore differences in the actions, as well as the different ways that actions can be connected (i.e., the focus of Kapadia et al.). Our methods and metrics provide a more in-depth analysis of actions, and can be used to determine similarities between actions in a set. Furthermore, we examine the whole set of actions (the NPC's abilities) with the insight that there are similarities between them. This allows us to generalize components of actions, pre-computing their similarities so that a reasoning system does not need to compute them every time they are compared.

3 TAXONOMIES IN GAME WORLDS

Before describing metrics to organize and judge the organization of actions we first define taxonomies, both for actions and for objects. This lays a foundation for the structure of organizing agent abilities separate from their name (l is this work) as well as differences in how action components can be compared. Taxonomies are generalization (*IsA*) relationships with a bi-directional connection so that parents know their children and children know their parents[1]. For our work, we assume each child has only one parent. The advantage of this is that components on the parent can be more easily propagated down to the children. This reduces the total amount of distinct information in the system, which we show in Section 5 has a positive effect on game character reasoning abilities.

[1]Throughout this work, we use p to refer to the parent in a taxonomy.

The concept of an object taxonomy is not new, and has been used in previous work [5, 20, 21]. These object taxonomies classify the game world into different categories, with the implicit understanding that each category is given a different purpose in that world. Our taxonomy is concerned with the *semantics* [35] of the objects, and treats the graphical model as a semantic of the object. An example object taxonomy can be seen in Figure 1, where types of objects have semantic components (such as *Posture* or *Model*), which would be used by the agent reasoning system.

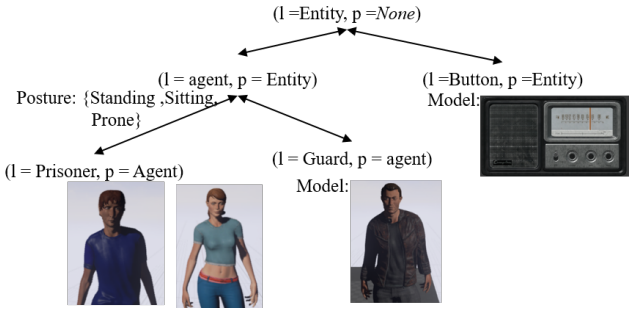


Figure 1: A sample taxonomy of five objects built from four graphical models.

Reflexively, actions can also be organized into generalizations, such as the grouping in Figure 2. This is different from organizing actions into a *plan tree* structure (such as a BT or HTN) in that the child actions are not needed to run the parent action. Each of the actions in Figure 2 can be run independently by a character, meaning each has its primitive or complex structure (shown as each action having its own *Behavior* in the figure). Similar to object taxonomies, action taxonomies can generalize components (such as the required objects) onto their parent, meaning that a character should not need to reason over all actions if it is only requiring a component of the action.

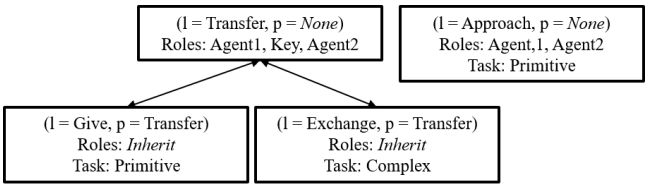


Figure 2: A sample taxonomy of four actions created from Shoulson et al. [31].

Oftentimes, object types in games are clustered into a single type, and abilities are not grouped at all. This hinders the ability to reason between disjoint sets, as known similarity becomes a binary property (it is either in the group or not). While our metrics (Section 4.1 and Section 4.2) are not dependent on objects and actions being organized into a taxonomy, doing so allows generalizations that can further differentiate and compare actions. Furthermore, it allows knowledge about the objects and actions to be extracted to higher levels, meaning the NPC does not need to consider each object or action individually, as they can assume the children inherit the components of their parents (shown in Section 4.3).

4 SIMILARITY IN NPC ACTIONS

As stated previously, actions are a rich representation that contain not just the *what*, but the *how* and *why* of character control. This means comparisons between actions can be performed on many levels. We use two levels, the components and the behavior of an action. Component level analysis examines the differences in the set of those components attached to an action (such as the objects used in the action). Behavior analysis further examines the shape of the steps in a complex behavior.

4.1 Component Based Similarity

Actions can be compared and contrasted by their aforementioned components. We treat components as unordered sets and each component of the action is orthogonal. In Figure 3a, one component of the action *Approach* is its objects. When our metric is comparing the objects on *Approach* (comparison between objects is assumed in the examples for the rest of the section) it is only concerned with there being two *agents*, and not the difference in how those two *agents* act. Those differences may appear in other components, such as the conditions or assertions, and would be noted in that (orthogonal) component. We also differentiate between actions and action templates. Templates are more general representations of an action, and thus their components are also more general. Figure 3c shows the template for *Approach*, which is constrained in Figure 3a and Figure 3b.

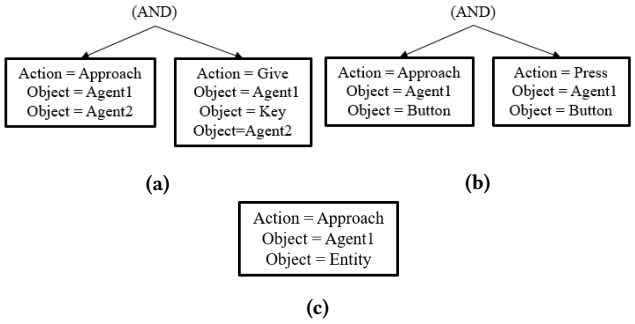


Figure 3: Three behaviors, (a)*Transfer*, (b)*SoundAlarm*, and (c)*Approach* with the associated sub-actions and object components.

To calculate the total costs of the components between two actions, we calculate the symmetric difference using Equation 1; where the actions are represented by **A** and the component set of that action is represented by the function C. The symmetric difference is a communicative measure that determines outliers between the two sets. It does not consider order of the components, but focuses on "what" is part of the action. By not considering the order, actions that are symmetric (active and passive in verbs) are considered equivalent. Furthermore, the symmetric difference means that overlapping components (objects) between actions are a cost of zero. Specifically for objects used in actions, the object taxonomy of Figure 1 provides multiple ways that objects can overlap. For example, overlap between two objects can mean an exact match (the *agents* matching each other in Figure 3) or through

generalizations. For generalization matches, parents in one action can match to children in the other. The *Approach* sub-behavior used in Figure 3a does not directly overlap the second object in Figure 3b and therefore is not a direct match. When comparing the two with the general interpretation of Figure 3c, both *Agent2* and *Button* match to *Entity* based on the object taxonomy. This allows our measure to consider possible connections, instead of strict object-object connections.

$$D(\mathbf{A}_i, \mathbf{A}_j) = \textit{symmetric difference}(C(\mathbf{A}_i), C(\mathbf{A}_j)) \tag{1}$$

4.2 Behavior Based Similarity

In addition to the components of an action, we can compare the shape of them by examining the behavior. For comparison of behaviors, a consistent representation is needed, one that allows all components to be compared. HTNs and BT are tress, as seen in Figure 3, and are more specifically actions connected by conjunctives such as *AND* (*SEQUENCER*) or *OR* (*SELECTOR*). These sub-actions themselves have shape, and so it is possible to expand the behavior of those actions as well. To factor for this we propose a metric that can be used with both expanded and unexpanded sub-behaviors, that is, can solve this problem by solving the smaller sub-problems of sub-action similarity. This leads itself to computing an edit distance [4] between two behaviors. The edit distance between two trees describes the cost to convert one tree into another. For behaviors, this is the cost of changing actions and connections to transform one behavior into another. This means that both the constituent sub-actions as well as the shape (how the sub-actions are connected together) contribute to the total cost, and therefore the total similarity, between two actions. The edit distance (more specifically, the *Levenshtein distance*) has been used by Sagredo-Olivenza et al. [29] to compare learned actions. However, they only used the sequence of actions (a sequence of names) in their comparison without considering connections. The more general edit distance can capture subtleties apparent in the connections.

One aspect of the edit distance is that each node (connection and sub-action) is compared between the two behaviors and a cost matrix is computed. The individual costs is computed between actions (organization and components) and connections. If actions are already organized in a taxonomy, we can exploit where actions are in the taxonomy as part of the total cost, using a similarity metric like Wu-Palmer Similarity [38] seen in Equation 2. In this equation, **lcs** is the *Least Common Sub-sumer*, measured as the closest parent between $\mathbf{A}_i$ and $\mathbf{A}_j$ in a taxonomy. For example, in Figure 2, the cost of *Exchange* and *Give* is $\frac{2}{4}$, due to the common parent of *Transfer* (where the depth of *transfer* starts at 1). One issue with this is when a connection does not exist. In Figure 2, there is no connection between *Approach* and *Give*. Therefore, a false root should be used to connect all disjoint ability sets. This makes the cost between *Approach* and *Give* $\frac{0}{2}$ (with a false root depth of 0). Furthermore, the cost between two components of each behavior is an important part of the equation, and for that we use Equation 1. To keep the *symmetric difference* from dominating the total cost, we normalize the *symmetric difference* with the union of the two component sets. Each sub-equation is then averaged together to obtain a total cost between two sub-actions.

$$D(\mathbf{A}_i, \mathbf{A}_j) = \frac{2 * depth(\mathbf{lcs})}{depth(\mathbf{A}_i) + depth(\mathbf{A}_j)} \tag{2}$$

Specifically for more complex behaviors, the type of connection in addition to the number of children and their components effects the total comparison. That is, the behavior of *SEQUENCE* and *SELECTOR* nodes are different, and this difference must be taken into account for the total edit distance. Therefore, we associate a binary cost when two connectors are compared (seen in Equation 3), such that the cost is zero if they are the same connector (Figure 3a and Figure 3b) and 1 otherwise. The cost comparison between an action and connection is always 1.

$$D(\mathbf{A}_i, \mathbf{A}_j) = \begin{cases} 0 & if\ \mathbf{A}_i = \mathbf{A}_j \\ 1 & else \end{cases} \tag{3}$$

The end result of the edit distance is the minimum effort to transform one behavior into another. At this stage, our measure provides the cost of changing behaviors, where lower costs mean more similar objects. This may be useful for re-planning systems. However, to compute similarity between behaviors, we perform **L2** normalization between the total set of behaviors. The normalized cost is then subtracted from 1 to provide an overall similarity comparison between behaviors.

4.3 Action Generalization and Ability Comparison

In addition to metrics for comparing actions, it is possible to examine entire action sets (that is, action taxonomies or partial taxonomies) in order to make comparisons on the effectiveness of the organization. Part of this requires generalizing components of the action on the taxonomy by pre-computing the similarities and storing them for further use. Furthermore, by understanding how much a taxonomy can generalize an action set, it can provide a better understanding as to how well that system can be used in an action selection mechanism based on the given component.

To generalize information in a taxonomy, we use a two-pass technique, with a high level overview in Figure 4. The first pass starts with the deepest interior actions in the taxonomy (the parent's of the leaf actions). When part of a component is the same for all children, that component is also attached to the more general action. This is propagated to the roots of the taxonomy, seen in Figure 4b. We then do a second pass to remove extraneous information from the children actions, with the final result seen in Figure 4c. We implement a two-pass solution due to the generality of organizing actions.

Our generalization method, seen in Figure 4, processes the taxonomy and generalizes as much information as possible. While Figure 4a begins with the whole taxonomy, a correct taxonomy can be used with any number of component properties to arrive at Figure 4c. However, the method shown in Figure 4 does not compare organization strategies, but only generalizes the taxonomy it creates. To see the change, we compare the taxonomy before generalization (Figure 4a) with the end result (Figure 4c) using Equation 4. In Equation 4, **ACT** refers to the entire action set, with $\mathbf{ACT}_B$ being

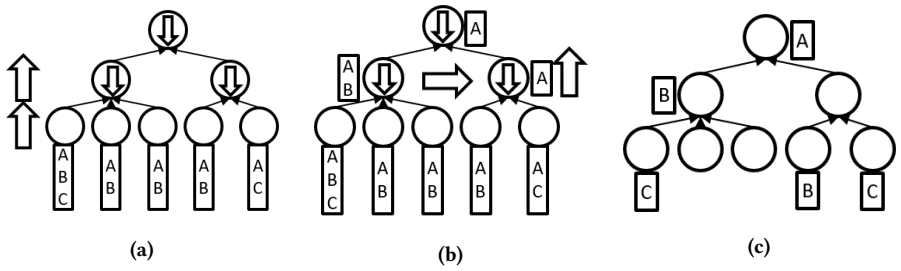


Figure 4: A high level over-view of generalizing data on the taxonomy. Our method (a) first finds all component properties that are the same between children, traveling up the taxonomy. It then (b) removes all properties from the children that are also on the parent, starting with the lowest level. The end generalized result can be seen in (c).

the starting taxonomy and $\mathbf{ACT}_F$ being the final taxonomy.

$$\Delta = 1 - \frac{|C(\mathbf{ACT}_B)| - |C(\mathbf{ACT}_F)|}{|C(\mathbf{ACT}_B)|} \tag{4}$$

While Equation 4 is useful in determining the amount of redundancy in an action taxonomy, there are several occasions in which it is too optimistic a measure. As Figure 4 shows, reduction of components only moves partial information around and may not be applicable in situations that require the entire component to be moved. In those instances, it is preferable to use a binary (per-action instead of per-component) reduction, seen in Equation 6, where C is replaced by Equation 5. As binary reduction determines the change in the number of actions with a component, it provide a more realistic approximation of how the taxonomy is structured and how much information is unnecessary for actual processing.

$$BIN(\mathbf{ACT}) = act, \forall act \in \mathbf{ACT} \text{ if } C(act) > 1 \tag{5}$$

$$\Delta = 1 - \frac{|BIN(\mathbf{ACT}_B)| - |BIN(\mathbf{ACT}_F)|}{|BIN(\mathbf{ACT}_B)|} \tag{6}$$

Both of these reduction measures hinge on the usability of a component by an NPC. As more components and actions are added, the size of the considered action set has an effect on the computational costs of the character's reasoning tools. Also, the considered action set may be a subset of the total action set, in that actions appear the same to the reasoning mechanism. If we know that a specific component of the action (i.e objects or conditions) is used by the reasoning system, then we can determine if one taxonomy is more appropriately structured than another.

5 EXPERIMENTATION

To show the efficacy of our metrics on organizing NPC abilities, we examine the effects our metrics have on determining the similarity of NPC actions as well as game characters' reasoning abilities[2]. Our action set is based on the events of Shoulson et al. [31], seen in Table 1. We automatically generate generalizations for objects using WordNet [36] and the method of Pelkey and Allbeck [27]. This creates a total of 21 actions and 31 objects. The action set is split between seven actions with complex behaviors (actions that have more than one sub-action) and the rest are primitive behaviors. The average number of object used in each action is four with a standard deviation of two.

[2]The experiments presented here a part of a larger subset found in J. Timothy Balint's dissertation [2]

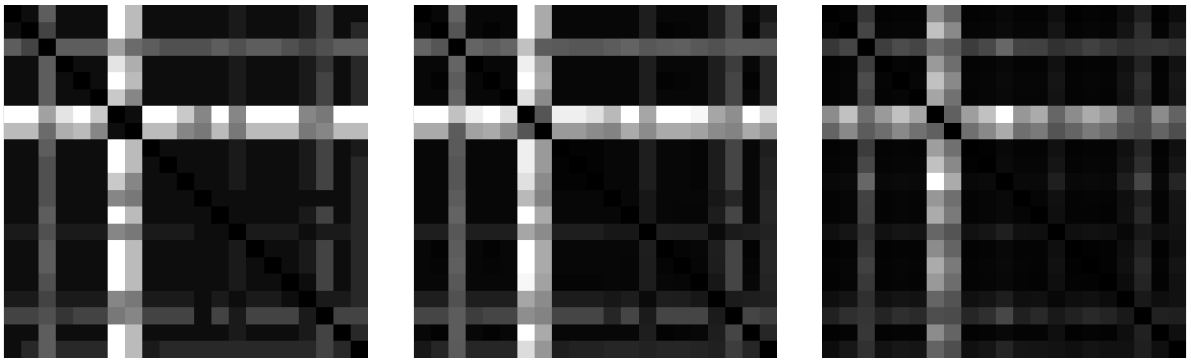
Our similarity metrics are designed to expose the nuances inherent in a game character's action set. Those similarities are stored to ease the computational burden that appears when using a character reasoning system. Therefore, we first show the difference using the metrics of Section 4 have on the similarity between actions. We then explore the effects different taxonomical groupings have on character reasoning abilities. Finally, we ground our work in a game example, showing what similar actions mean in a game.

Action Name	Type of Behavior	# of Objects
Hide	Primitive	2
Lock	Primitive	2
EscapeCell	Complex	3
Press	Primitive	2
Guard	Primitive	2
Trap	Primitive	2
TrapGuardsAlarm	Complex	9
TrapGuards	Complex	9
Draw	Primitive	2
Daze	Primitive	2
Call	Primitive	2
Approach	Primitive	2
Give	Primitive	3
Exchange	Complex	3
Open	Primitive	2
Unlock	Primitive	3
Take	Primitive	3
StealKey	Complex	3
SoundAlarm	Complex	7
Close	Primitive	2
DistractGuard	Complex	7

Table 1: Actions from Shoulson et al. [31]

5.1 Semantic Relations from Action components

To show what can be understood from the syntax of actions, using the edit distance and equations of Section 4, we examine the similarity of different actions and how the similarity changes with parameterization. To examine the different effects our metrics have,



(a) (b) (c)

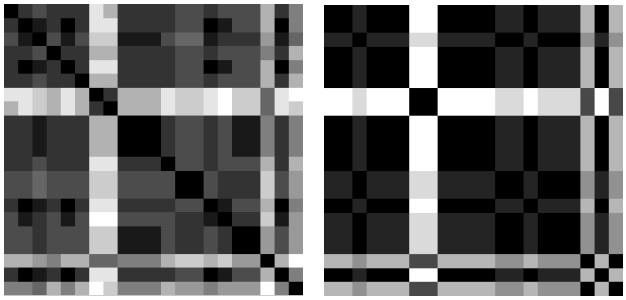
Figure 5: Similarity matrices for comparing the behaviors of actions. (a) The Levenstein Distance using only a string of action names. (b) Comparing the full shape of each behavior using component costs as well. (c) Comparing the shape of each behavior using component costs and Wu-Palmer similarity with a WordNet generated taxonomy for both actions and objects.

we compute similarity scores between all actions and display the result as a similarity matrix, seen in Figure 5(b-c). For comparison, we also compute a similarity score with only the action names used in the behaviors using the *Levenshtein distance* of Sagredo-Olivenza et al. [29] seen in Figure 5a. The atomic character for the Levenshtein distance is l, and the compared string is the list of all names. For all experiments, all actions are processed using the order of Table 1. For the similarity shown in Figure 5, darker shades indicate greater similarity between two actions.

From Figure 5, it can be seen that there are four very distinct actions in the set. These correspond to large, complex behaviors with many sub-actions, and it should be expected that any metric that compares actions would find those different from many of the primitive actions. Figure 5b and Figure 5c also show more nuanced bands in the lower right quadrant of the graphs, which are much less prominent than in Figure 5a, especially when compared to the diagonal (where each action is compared to itself). Figure 5b and Figure 5c include a per action component analysis and a more detailed shape analysis, with Figure 5c containing similarity based on an organization of actions using Equation 2 and object comparison that considers an object taxonomy. These two figures show that actions with similar components are considered more similar, which is something that Figure 5a does not. Furthermore, Figure 5b and Figure 5c display differences between the two most complex actions, which is not easily seen in Figure 5a. Having a full structure comparison and similarity of each component shows more subtle differences between actions.

Based on the above analysis, we examine similarity computed for only the objects (that is, no behavior or shape comparison) for the set of actions, with the resulting similarity matrix shown in Figure 6. In Figure 6, we examined two different strategies for calculating the semantic difference of objects used in actions, one with just the object names 6a, and one with a WordNet created object taxonomy 6b. Similar to Figure 5, darker shades represent more similar actions in Figure 6.

The similarity matrices of Figure 6 are much more varied than for the analysis with the behavior in Figure 5 more closely following the number of object comparison in Table 1. This is to be expected,



(a) (b)

Figure 6: Similarity scores of the objects of actions when comparing (a) only the objects used with (b) the objects and their generalizations.

as Figure 6 is not biased by large, complex behaviors. The stark bans from those complex behaviors still appear (as they use several different objects), but Figure 6 also shows differences between primitive actions, evident in the upper left quadrant of both figures in Figure 6. An advantage of using component analysis is that subtle differences between primitive actions are more visible. Using an object hierarchy shows similarity between seemingly dissimilar sets, evident in the darker squares in the lower right-hand corner between Figure 6a and Figure 6b. This is due to the overall similarity offered through the taxonomy, meaning that the type differences are not fully different.

5.2 Effects on Character Abilities

In this work we have discussed similarities between actions, which can be used to organize an NPC's abilities. To show the effects of organization, we examine an application-based approach to organizing a character's abilities. We have implemented two different virtual character reasoning algorithms: a location-object selection mechanism for crowds of background characters based on Normoyle et al. [25] and a condition-assertion based narrative generation system based on Kartal et al. [17]. We use these two methods to

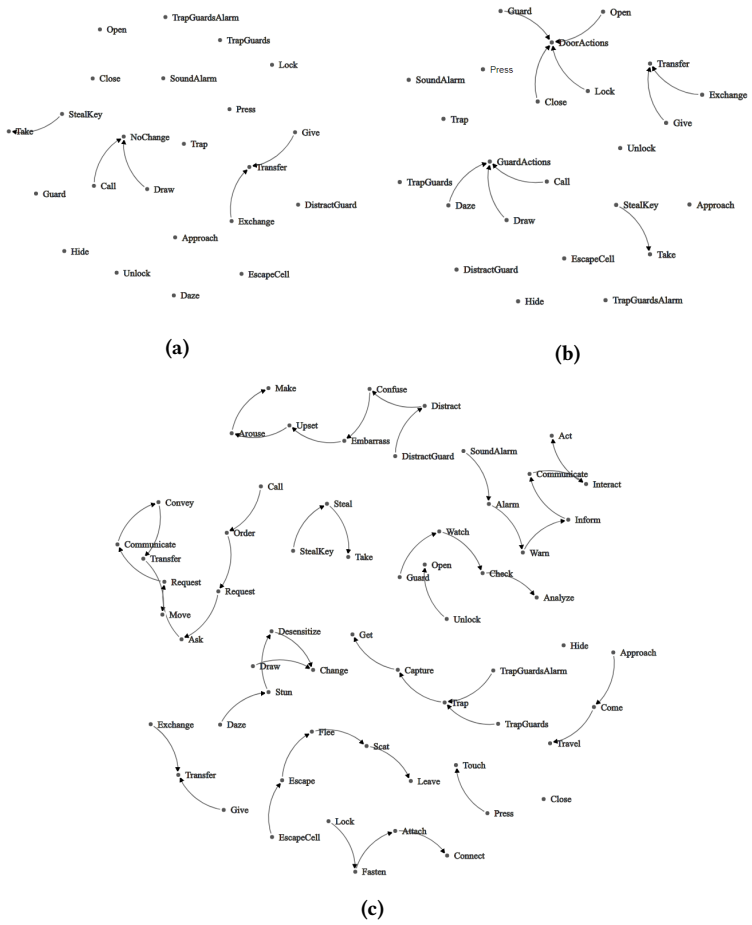


Figure 7: A visual representation of the three taxonomy organizations used in this study from the unorganized set in Table 1. (a) Actions organized to be condensed by condition-assertions. (b) Actions organized to be condensed by objects. (c) Action organized into a WordNet taxonomy using Balint and Allbeck [3].

display the changes in run-time that intelligently organizing action sets afford. The location-object selection mechanism is analogous to "what a crowd of characters can do in an area," while the narrative driven method is a more general planner. We use the same action set as in Section 5.1.

We show the effect different organization strategies of actions into taxonomies have for two application-based components. To do so, we devise three different taxonomies, seen in Figure 7. One is an application agnostic *WordNet Hypernym* strategy (Figure 7c, generated using the method of Balint and Allbeck [3]). The other two are hand-built taxonomies based on component similarity. We use exact binary matching on objects (Figure 7b) and exact binary matching on the combination of conditions and assertions (Figure 7a). We also add a baseline comparison of having no generalizations, analogous to Table 1. We run our generalization algorithm on the different taxonomies for both components and display the results in Table 2. In Table 2, we show two measures for calculating the change in components, partial comparison (Equation 4), and binary comparison (Equation 6). Table 2 shows the contrast and similarities between organization strategies when measuring the movement of components in the taxonomy.

Δ Component Type	Figure 7c	Figure 7b	Figure 7a
Object (Equation 4)	90%	80%	93%
Condition-Assertion (Equation 4)	92%	80%	90%
Object (Equation 6)	90%	65%	95%
Condition-Assertion (Equation 6)	90%	100%	80%

Table 2: The change in components caused by the generalizations for our organizational strategy.

As can be seen from Table 2, using a taxonomy allows for components of an action set to be combined together, generalizing the actions. The change seen when organizations strategies match the component being generalized is greater than when the organization is mis-matched or agnostic. Non-binary redundancy is being

measured on the components and not the actions themselves. So, an object focused method may have actions that are exact in all of their siblings, but partial matches will also be compressed down.

Next, we show the effects of pre-computing similarities of actions on run-time using a location-object based action selection mechanism, Normolye et al. [25]. We compute the change probabilities for each action based on the objects in the system. To do so, we find all actions with objects attached to them (similar to our binary taxonomy measure) and consider this the action set the NPC crowd considers (as Normolye et al. is a crowd based reasoning system). Beginning conditions and end distributions between actions are randomized for each run. To better show the effects of similarity pre-computation, each action is assigned ten locations (so that the system is calculated on ten times as many actions). This test was run on an Intel Xeon 2.3GHz, eight core processor on a single thread. We run 1000 trials, reporting the average in Figure 8.

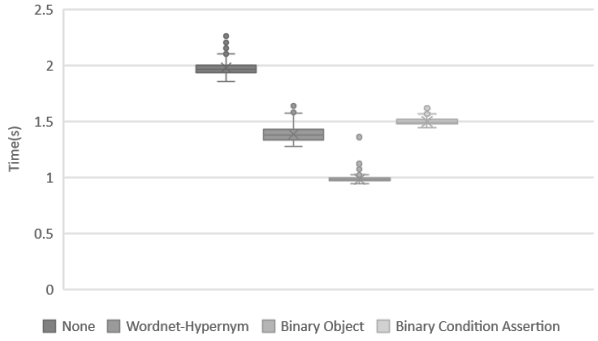


Figure 8: The average time to compute a distribution of crowd characters for a location-object based method vs. the organizational strategy.

What is not surprising from Figure 8 is that, as the total number of actions considered decreases due to pre-computed similarity, the average time to run also decreases. As the number of actions to consider decreases, the number of equations the convex optimization algorithm must solve also decreases, and so this result is expected. What is more interesting is that the standard deviation decreases dramatically as the compression increases. As the connectivity of the system is controlled by the overall number of considered actions, then reducing the total number of actions will reduce the total possible connectivity. Less possible variance in the connectivity will mean less variance overall. Figure 8 follows the effects shown for binary object compression in Table 2, showing that, for this method, the measure presented in Equation 4 is useful for quickly calculating if an organization is useful.

We also test the effect different organizations of the same action set have on condition-assertions. Note from Table 2 that the generalization ability of condition-assertions are much lower than objects for both component and binary generalization. Condition-assertions are much more varied than objects, and are therefore more difficult to compress. We use a narrative variation generator that searches for action sequences that match a desired goal (assertion) state. As we are only concerned with finding the goal states, we treat all narratives that match the goal state as viable, and do not judge the narratives themselves. Each test started with five random goals and was run on an Intel i5, four core processor on a single thread. We report the average runtime for nineteen different trials in Figure 9.

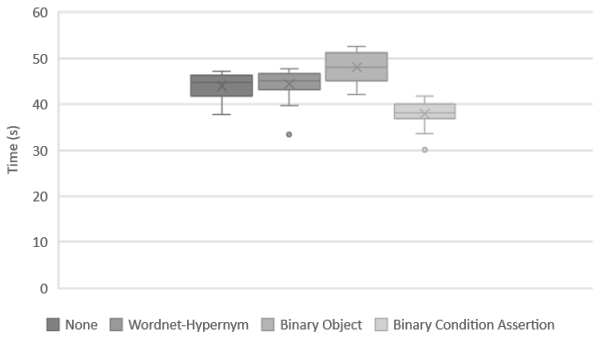


Figure 9: The average time to compute a narrative plan vs. the organizational strategy.

As can be seen from Table 2, the binary condition-assertion focused taxonomy (Figure 7a) is the best for condition-assertion reduction, whereas there is not as much difference for any partial generalization. Figure 9 shows that the condition-assertion reduction better reflects how the change effects the runtime for our narrative plan generator. This is because reduction on the search tree is affected by the **total** number of actions. With fewer considered actions, the chance of finding an action that matches the goal state through random simulation increase, as does the probability that the assertions being searched for are found. In reality, the search space is pruned through the use of the taxonomy, resulting in finding a viable narrative quicker. The action sets consisting of only the actions, the object-compressed taxonomy, and the WordNet hypernym set all performed similarly, and slower than the condition-assertion focused taxonomy. From Table 2, the binary objects are different for the binary condition-assertion set, but are similar to the others. Therefore, when choosing which compression measure to use on the taxonomies, how the reasoning method works should be taken into consideration.

5.3 Demonstration

So far, we have shown how NPC actions can be compared and how those comparisons can help organize actions into a taxonomy. Taxonomies can hold pre-computed knowledge useful in NPC reasoning systems and so we provide a small example to illustrate how that can manifest in characters. We use a purchased virtual environment featuring two virtual characters shown in in Figure 10. The virtual environment, which is modeled after a medieval style pub, contains over eighty object types. We have two characters in the environment, a male and female character. As part of the scenario, the female character must move the male character away from the bar in order to get to the exit. We do this as it is similar to the scenario in event-centric planning, in which a prisoner NPC draws over the *Guard* to move them away from his post. The reasoning

system for the female character is based on a condition-assertion planning system.

(a)

(b)

Figure 10: A NPC character achieving the same effect using two different actions. (a) The female character uses the *Draw* action to get the male character's attention. (b) The female character uses the *Call* action to get the male character's attention.

Figure 7a is a condition-assertion binary taxonomy. This means actions with a parent are completely equal on that component. From this, there are two actions that attract the male character and are equivalent: *Draw* (Figure 10a and Table 3a) and *Call* (Figure 10b and Table 3b). It should be seen from Figure 7a that these two actions have the common parent, which means that these two actions can be used interchangeably based on what they require from the environment and what they accomplish. Simulating two separate plans, we see that both of these actions have the same conditions and effects, and therefore one can easily be replaced with the other. Under further inspection of their definitions in Table 3, the only difference in them is their name and primitive action used (in this case, a single animation). This is a common occurrence, where different actions accomplish the same task. Usually a simulation author would have to copy and change the definition from one task to another, but by intelligently organizing actions into a taxonomy, the condition-assertions (and objects) only have to be defined once. This assists not only the simulation author, but also the character reasoning system, as their single definition defines them as equivalent.

Name	Draw
Object	Guard
condition	finishedAction(self.id)→SUCCESS
Behavior	Draw

(a) Draw

Name	Call
Object	Guard
condition	finishedAction(self.id)→SUCCESS
Behavior	Call

(b) Call

Table 3: Action definitions for (a)Draw and (b)Call

6 CONCLUSIONS, LIMITATIONS, AND FUTURE WORK

In this work, we have proposed similarity metrics for NPC actions that afford a more complete understanding of the similarities between actions, especially in regards to primitive actions. Analyzing the components of actions provides a greater understanding between primitive actions, whereas the full edit distance shape analysis more easily shows differences between complex actions. Our metrics therefore show the nuances of actions that do not necessarily appear when only considering the behavior as a sequence. We also described how NPC reasoning can benefit from intelligent organization of their abilities. The utility of an organization strategy is dependent on the reasoning system. Pre-computed similarities that the reasoning system uses are more beneficial. Keeping this in mind will allow for a wider variety of nuanced actions that can be reasoned over in a similar, constrained budget.

Although intelligently organizing actions into taxonomies can speed up a character's reasoning abilities, the construction of such taxonomies is not a trivial task. While work such as Balint and Allbeck [3] do provide a method to create an action taxonomy, the generated taxonomy does not take into consideration the components of an action, instead electing to use a natural language knowledge base to connect actions together. Optimizing based on the authored meaning of an action, which we performed in Figure 7b and Figure 7a, still requires examining each action, which can be a tedious process for a simulation author to undertake. Finding an optimal organization of character abilities is difficult to achieve. Better taxonomies can be determined, but it may not be possible to find an optimal taxonomy.

NPCs are not the only users of actions in games. Players often follow a similar pattern, although their actions may be more primitive. Player modeling is an active area of research and examining player actions from in-game behaviors has received some attention in recent years, using frequent pattern mining [7] or game play traces [26]. A similarity metric based on the components and behavior of actions may provide new and different insight into grouping players together and is an exciting area of future research.

ACKNOWLEDGMENTS

We would like to thank Nasrin Noor Ahmad for interpreting the action set used in our experiments, as well as Aline Normoyle for

supplying the code of her stochastic algorithm. We would also like to thanks Jessica Randall, Shib Duman, and Dianna M. Balint for their suggestions and edits. This work is part of the programme Virtual E-Coaching and Storytelling Technology for Post-Traumatic Stress Disorder, which is financed by the Netherlands Organization for Scientific Research (pr. nr. 314-99-104).

REFERENCES

[1] Marwan Badawi and Stéphane Donikian. 2007. The generic description and management of interaction between autonomous agents and objects in an informed virtual environment. *Computer Animation and Virtual Worlds* 18, 4-5 (2007), 559–569.

[2] J. Timothy Balint. 2017. *Automated Extraction of Action Semantics for Embodied Virtual Agents using Textual Knowledge Bases.* Doctoral dissertation. George Mason University, Fairfax, Virginia.

[3] Tim Balint and Jan M. Allbeck. 2015. Automated Generation of Plausible Agent Object Interactions. In *Intelligent Virtual Agents*, Willem-Paul Brinkman, Joost Broekens, and Dirk Heylen (Eds.). Lecture Notes in Computer Science, Vol. 9238. Springer International Publishing, 295–309.

[4] Philip Bille. 2005. A survey on tree edit distance and related problems. *Theoretical computer science* 337, 1-3 (2005), 217–239.

[5] Rama Bindiganavale, William Schuler, Jan M. Allbeck, Norman I. Badler, Aravind K. Joshi, and Martha Palmer. 2000. Dynamically Altering Agent Behaviors Using Natural Language Instructions. In *Proceedings of the Fourth International Conference on Autonomous Agents.* ACM, New York, NY, USA, 293–300. http://doi.acm.org/10.1145/336595.337503

[6] Adi Botea, Markus Enzenberger, Martin Müller, and Jonathan Schaeffer. 2005. Macro-FF: Improving AI planning with automatically learned macro-operators. *Journal of Artificial Intelligence Research* 24 (2005), 581–621.

[7] Zhengxing Chen, Magy Seif El-Nasr, Alessandro Canossa, Jeremy Badler, Stefanie Tignor, and Randy Colvin. 2015. Modeling individual differences through frequent pattern mining on role-playing game actions. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference.* AAAI Press, 2–7.

[8] Michele Colledanchise, Ramviyas Parasuraman, and Petter Ögren. 2018. Learning of behavior trees for autonomous agents. *IEEE Transactions on Games* PrePrint, 99 (2018), 1–1.

[9] Lavindra De Silva and Lin Padgham. 2004. A comparison of BDI based real-time reasoning and HTN based planning. In *17th Australian Joint Conference on Artificial Intelligence.* Springer, Cairns, 1167–1173.

[10] Kutluhan Erol, James Hendler, and Dana S. Nau. 1995. *Semantics for Hierarchical Task-Network Planning.* Technical Report T.R. 95-9. University of Maryland,College Park,MD,20742, Computer Science Department,Institute for Systems Research,. 30 pages.

[11] Richard E Fikes and Nils J Nilsson. 1972. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence* 2, 3 (1972), 189–208.

[12] John Funge. 2000. Cognitive modeling for games and animation. *Commun. ACM* 43, 7 (2000), 40–48.

[13] Kyunglyul Hyun, Kyungho Lee, and Jehee Lee. 2016. Motion Grammars for Character Animation. *Computer Graphics Forum* 35 (2016), 103–113.

[14] Marcelo Kallmann and Daniel Thalmann. 1999. Direct 3D Interaction with Smart Objects. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '99).* ACM, New York, NY, USA, 124–130.

[15] Mubbasir Kapadia, Nuria Pelechano, Jan Allbeck, and Norm Badler. 2015. *Virtual Crowds: Steps Toward Behavioral Realism.* Morgan & Claypool.

[16] Mubbasir Kapadia, Fabio Zünd, Jessica Falk, Marcel Marti, Robert W Sumner, and Markus Gross. 2015. Evaluating the authoring complexity of interactive narratives with interactive behaviour trees. *Foundations of Digital Games* (2015).

[17] Bilal Kartal, John Koenig, and Stephen J. Guy. 2014. User-driven Narrative Variation in Large Story Domains Using Monte Carlo Tree Search. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems (AAMAS '14).* International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC, 69–76.

[18] John Paul Kelly, Adi Botea, and Sven Koenig. 2008. Offline Planning with Hierarchical Task Networks in Video Games.. In *The 4th Artificial Intelligence and Interactive Digital Entertainment International Conference.* AAAI Press, Menlo Park CA, 60–65.

[19] Jassin Kessing, Tim Tutenel, and Rafael Bidarra. 2009. Services in game worlds: A semantic approach to improve object interaction. In *Entertainment Computing–ICEC 2009.* Lecture Notes in Computer Science, Vol. 5709. Springer, Berlin, Heidelberg, 276–281.

[20] Jassin Kessing, Tim Tutenel, and Rafael Bidarra. 2012. Designing semantic game worlds. In *Proceedings of the The third workshop on Procedural Content Generation in Games.* ACM, 2.

[21] Jean-Luc Lugrin and Marc Cavazza. 2007. Making Sense of Virtual Environments: Action Representation, Grounding and Common Sense. In *IUI.* ACM, New York, NY, USA, 225–234.

[22] Maurizio Mancini and Catherine Pelachaud. 2007. Dynamic Behavior Qualifiers for Conversational Agents. In *Intelligent Virtual Agents (Lecture Notes in Computer Science)*, Vol. 4722. Springer, Paris, France, 112–124.

[23] Negin Nejati, Tolga Könik, and Ugur Kuter. 2009. A Goal- and Dependency-directed Algorithm for Learning Hierarchical Task Networks. In *Proceedings of the Fifth International Conference on Knowledge Capture (K-CAP '09).* ACM, New York, NY, USA, 113–120.

[24] Xenija Neufeld, Sanaz Mostaghim, Dario Sancho-Pradel, and Sandy Brand. 2018. Building a Planner: A Survey of Planning Systems Used in Commercial Video Games. *IEEE Transactions on Games* Preprint (2018), 1–1.

[25] Aline Normoyle, Maxim Likhachev, and Alla Safonova. 2014. Stochastic Activity Authoring with Direct User Control. In *Proceedings of the 18th Meeting of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D '14).* ACM, New York, NY, USA, 31–38.

[26] Joseph C Osborn and Michael Mateas. 2014. A game-independent play trace dissimilarity metric.

[27] Cameron Pelkey and Jan M. Allbeck. 2014. Populating Virtual Semantic Environments. *Computer Animation and Virtual Worlds* 24, 3 (May 2014), 405–414.

[28] Mark O. Riedl and R. Michael Young. 2010. Narrative Planning: Balancing Plot and Character. *J. Artif. Int. Res.* 39, 1 (Sept. 2010), 217–268.

[29] Ismael Sagredo-Olivenza, Pedro Pablo Gómez-Martín, Marco Antonio Gómez-Martín, and Pedro Antonio González-Calero. 2018. Trained Behavior Trees: Programming by Demonstration to Support AI Game Designers. *IEEE Transactions on Games* PrePrint, 99 (2018), 1–1.

[30] Alexander Shoulson, Francisco M. Garcia, Matthew Jones, Robert Mead, and Norman I. Badler. 2011. Parameterizing Behavior Trees. In *Proceedings of the 4th International Conference on Motion in Games (MIG'11).* Springer-Verlag, Berlin, Heidelberg, 144–155.

[31] Alexander Shoulson, Max L Gilbert, Mubbasir Kapadia, and Norman I Badler. 2013. An event-centric planning approach for dynamic real-time narrative. In *Motion in Games.* ACM, Doublin, Ireland, 121–130.

[32] Alexander Shoulson, Mubbasir Kapadia, and Norman I Badler. 2013. Paste: A platform for adaptive storytelling with events. *Intelligent Narrative Technologies* 6 (2013), 216.

[33] Moritz Tenorth and Michael Beetz. 2012. A unified representation for reasoning about robot actions, processes, and their effects on objects. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on.* 1351–1358.

[34] Marcus Thiebaux, Stacy Marsella, Andrew N. Marshall, and Marcelo Kallmann. 2008. SmartBody: Behavior Realization for Embodied Conversational Agents. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1 (AAMAS '08).* International Foundation for Autonomous Agents and Multiagent Systems, Estoril, Portugal, 151–158.

[35] Tim Tutenel, Rafael Bidarra, Ruben M. Smelik, and Klaas Jan De Kraker. 2008. The Role of Semantics in Games and Simulations. *Comput. Entertain.* 6, 4 (Dec. 2008), 57:1–57:35.

[36] Princeton University. 2010. *About WordNet.* Technical Report. Princeton University. http://wordnet.princeton.edu

[37] Josep Valls-Vargas, Santiago Ontanón, and Jichen Zhu. 2013. Towards story-based content generation: From plot-points to maps. IEEE, 1–8.

[38] Zhibiao Wu and Martha Palmer. 1994. Verbs Semantics and Lexical Selection. In *Proceedings of the 32nd Annual Meeting on Association for Computational Linguistics (ACL '94).* Association for Computational Linguistics, Stroudsburg, PA, USA, 133–138.

[39] R Michael Young, Mark O Riedl, Mark Branly, Arnav Jhala, R J Martin, and C J Saretto. 2004. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development* 1, 1 (2004), 51–70.

[40] Richard Zhao and Duane Szafron. 2014. Virtual Character Behavior Architecture using Cyclic Scheduling.

[41] Alexander Zook, Stephen Lee-Urban, Mark O Riedl, Heather K Holden, Robert A Sottilare, and Keith W Brawner. 2012. Automated scenario generation: toward tailored and optimized military training in virtual environments. In *the International Conference on the Foundations of Digital Games (FDG '12).* ACM, Raleigh, North Carolina, 164–171.