Semi-Automatic Task Recognition for Interactive Narratives with EAT & RUN

Jeff Orkin
MIT Media Laboratory
75 Amherst Street
Cambridge, MA 02139
jorkin@media.mit.edu

Tynan Smith
MIT Media Laboratory
75 Amherst Street
Cambridge, MA 02139
tssmith@mit.edu

Hilke Reckman
MIT Media Laboratory
75 Amherst Street
Cambridge, MA 02139
reckman@media.mit.edu

Deb Roy
MIT Media Laboratory
75 Amherst Street
Cambridge, MA 02139
dkroy@media.mit.edu

ABSTRACT
Mining data from online games provides a potential alternative to programming behavior and dialogue for characters in interactive narratives by hand. Human annotation of course-grained tasks can provide explanations that make the data more useful to an AI system, however human labor is expensive. We describe a semi-automatic methodology for recognizing tasks in gameplay traces, including an annotation tool for non-experts, and a runtime algorithm. Our results show that this methodology works well with a large corpus from one game, and suggests the possibility of refactoring the development process for interactive narratives.

Categories and Subject Descriptors
I.2.7 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods – *frames and scripts.*

General Terms
Algorithms, Experimentation.

Keywords
Story understanding, authoring tools.

1. INTRODUCTION
Current approaches to authoring interactive narratives are labor intensive, and require technical skills and understanding of AI systems. Creating characters who can interact with each other, and possibly with humans, within the context of a narrative often requires designers to encode behavior and dialogue into some kind of hierarchical representation [3, 22]. While these representations are intuitive to engineers, designers do not necessarily have an engineering background, making current approaches inaccessible. Meanwhile, the popularity of multiplayer games is skyrocketing due to the ubiquity of broadband, and the growing number of online communities and services like Xbox Live. The combination of cheap storage, fast connections, and lots of players opens an opportunity for a new approach to authoring content, by recording human-human interactions, and annotating these gameplay traces such that characters can exploit them at runtime. Human annotations explain to the AI system the intent behind recurring patterns of physical and/or linguistic behavior.

This paper describes a methodology for annotating *tasks* in gameplay traces, where a task refers to a sequence of actions or utterances recognizable as a coherent, intentional unit by humans. The methodology includes an annotation tool designed for non-experts, and an algorithm for automatically recognizing tasks based on human-annotated examples. Designers use the Environment for Annotating Tasks (EAT) to produce data to train RUN (not an acronym), a simple yet effective algorithm for recognizing tasks at runtime. RUN can also be used to preprocess a corpus of thousands of recorded gameplay traces.

We have collected a corpus of 9,882 gameplay traces generated from over 15,000 people playing the roles of customers and waitresses in a virtual restaurant. Our long-term goal is to develop a case-based planning system that can exploit this corpus to power the behavior of AI-controlled actors who can interact in a convincingly human-like way. Case-based planning [10] refers to a system that utilizes examples from similar episodes in the past to plan actions in the current situation. We have previously described the first iteration of our case-based planning system, which automatically learns to imitate humans based on recurring patterns of behavior [17]. In this paper we describe how annotating and recognizing tasks will address issues in our first iteration, and improve the future iteration of the planning system.

Influenced by Schank's conception the *restaurant script* -- an abstract representation of typical restaurant behavior that enables a machine to understand stories about restaurants [20] -- we believe that generating convincing behavior and dialogue for AI-controlled customers and waitresses requires a representation of typical restaurant behavior that gives context to observed actions and utterances. In our case, this representation will be learned from thousands of gameplay traces, rather than hand-crafted, with tasks as a critical representational building block. EAT & RUN provide us with the means to recognize tasks in gameplay traces, as a first step toward learning the restaurant script. The rest of this paper describes the motivation for introducing tasks, the design of the EAT interface, the specifics of the RUN algorithm, our task recognition evaluation results, and discussion of related work.

2. THE RESTAURANT GAME
The Restaurant Game is an online game where humans are anonymously paired to play the roles of customers and waitresses in a virtual restaurant. Players can chat with open-ended typed text, move around the 3D environment, and manipulate 47 types

INT3 2010, June 18, Monterey, CA, USA.

of objects through a point-and-click interface. Every object provides the same interaction options: pick up, put down, give, inspect, sit on, eat, and touch. Objects respond to these actions in different ways -- food diminishes bite by bite when eaten, while eating a chair makes a crunch sound, but does not change the shape of the chair. The chef and bartender are hard-coded to produce food items based on keywords in chat text. A game takes about 10-15 minutes to play, and a typical game consists of 84 physical actions and 40 utterances. Everything players say and do is logged in time-coded text files on our servers. Player interactions vary greatly, ranging from dramatizations of what one would expect to witness in a restaurant, to games where players fill the restaurant with cherry pies. While many players do misbehave, we have demonstrated that enough people do engage in common behavior that it is possible for an automatic system to learn statistical models of typical behavior and language that correlate highly with human judgment of typicality. Details about data collection, analysis, and our first iteration of the planning system are available in previous publications [16, 17].

3. TASK HIERARCHIES & ANNOTATION

In this section, we motivate introducing a task-based representation into our planning system, discuss current applications of task hierarchies to interactive narratives, and explore ways to annotate tasks in recorded gameplay data.

3.1 Motivation for Task-Based Representation

Our planning system selects actions and utterances for a character by searching for a gameplay trace that best matches the observed interaction history, and then imitating what a human did in that situation. The first iteration of this system can only compare histories based on fine-grained surface similarities of action and utterance sequences. Without the courser-grained context provided by tasks, we see two issues: (1) cycles (e.g. infinite loops of serving drinks) and (2) non-sequiturs (e.g. answering questions that were never asked).

As a gameplay session progresses, and the interaction history grows, it becomes less likely that the system will find a gameplay trace that matches the history exactly. Our current solution only compares recent history (actions since the last dialogue ended), but this eliminates long-term memory, and leads to amnesic cycles of repeated behavior. Cycles also result from imitating games where players are misbehaving, and truly engaging in cycles. It is trivial to ignore abnormal actions (e.g. eating trash. sitting on lobsters), but without some higher-level structure to give context, it is difficult to recognize that ordering many items from the chef to pile on the counter is different from ordering multiple items to serve to a customer. Without structure, it is difficult to recognize typical actions and utterances in atypical contexts.

Non-sequiturs are a side-effect of characters periodically switching which gameplay traces they are following (out of the corpus of thousands) in order to better match recent history. Upon retrieval of a new trace, the character advances a pointer to the action after the matched portion and blindly imitates whatever the human did next, which leads to non-sequiturs when the human in the trace was responding to something that happened prior to the match point. For example, in one run we observe this sequence: a customer asks for a menu, the waitress picks up a menu from the podium and brings it to the table, the customer spontaneously says "Florida" (responding to "where are you from?" asked before the match point).

Introducing a task-based representation will allow the system to retrieve gameplay traces by matching at a courser-grained level, where it is more likely to find traces that match a much longer span, or even the entire history, restoring long-term memory and eliminating cycles. Tasks will provide context for actions and utterances, allowing characters to ignore actions within tasks that began prior to the match point, and to filter out actions occurring in unusual contexts. Tasks will also allow characters to disentangle behaviors that occur in parallel, and do not require a response from one another. The waitress can ignore the customer taking bites of his meal while she is cleaning tables because, while the actions are interleaved temporally, they belong to separate tasks that overlap in time.

3.2 Task Hierarchies in Games & Narratives

A number of interactive narrative systems have been developed based on variants of Hierarchical Task Network (HTN) planning systems [3, 22]. HTNs provide an intuitive means of specifying an overarching compound task, which can be decomposed recursively into sub-tasks, until reaching primitives that can be executed in the game engine. A task may coordinate multiple characters interacting with one another, and sub-tasks may execute sequentially or in parallel. Young has noted that this hierarchical structure is amenable to storytelling, where tasks represent recurring patterns of action which can be re-used within the narrative. Dramatic beats in Façade can also be described as hierarchical plans, implemented in a reactive planning language called ABL [13]. In all of these systems, plans are encoded by hand. Mateas comments that the authorial burden for hand-programming plans for Façade was high, and future research should focus on ways to generate story pieces that decrease authorial effort, possibly by exploring case-based generation techniques.

Aside from the authorial burden, there is the issue of coverage. It would be simple enough to hand-craft a plan to specify typical restaurant behavior, where a customer sits, looks at a menu, orders, eats, pays, and leaves. In our gameplay traces, we see many variations from the norm, such as customers who order an appetizer before looking at the menu, get food to-go without ever sitting down, dine & dash (without paying), or steal the cash register. Schank describes the early days of Legal Sea Foods, where people had to pay for their food before eating [20]. There are also nuances due to cultural differences. It is unlikely that a designer will account for every possible variation when hand-crafting a plan, leading to failure when the system encounters deviations from the norm. Generating a hierarchical plan from thousands of human examples has the potential to provide better coverage.

3.3 Annotating Tasks in Gameplay Traces

An alternative to authoring a hierarchical plan by hand is to extract it from an annotated gameplay trace, as Ram and colleagues have demonstrated in the domain of case-based planning for strategy games [19]. After each gameplay session, a human annotates a log file by labeling each action with the task

the player was trying to accomplish. For example, an Attack action might be annotated with KillUnit and WinGame. Applying Allen's temporal reasoning framework [1], tasks are determined to occur in sequence, in parallel, or as sub-tasks of one another. If one task occurs during another, it is assumed to be a sub-task, otherwise if tasks partially overlap temporally they are assumed to be pursued in parallel. Ultimately, all tasks are sub-tasks of the overarching WinGame task. These temporal relations inform the extraction of a hierarchical plan for playing the game. Annotating just a few games has been shown to produce an AI-controlled player capable of beating the game's built-in AI.

3.4 Annotating Tasks in Interactive Narratives

Our annotation scheme is similar to Ram's with respect to supporting tasks that occur in parallel temporally, but we do not assume anything about hierarchical structure from the annotations. In scrutinizing our data, we find that the characteristics of social interaction between two players in a restaurant are quite different from the behavior of a strategy game player. Dramatic improvisation between humans in an open-ended scenario is highly varied and fluid, exhibits many instances where one task occurs during another but is not a sub-task, and may not all contribute to a single overarching task -- there is no notion of *winning* in *The Restaurant Game*. Gervás has noted that stories rarely have one single end point where the goal of the story can be said to be achieved, and many stories have no identifiable goal at all [6].

In one game we observe a waitress taking an order from a customer while cleaning a table. Taking an order is not a sub-task of cleaning the table. In another game we observe a customer asking the waitress where she is from, while he continues eating his meal. This behavior contributes to a higher-level task of socializing, and is not a sub-task of eating. Eating and socializing are in different *realms* of thought [15], and do not belong to the same hierarchy. Trying to fit all of these tasks into one over-arching task is awkward. In addition, characters' goals may conflict – cleaning the table and taking an order may be sub-tasks of the waitress rushing the customer out, to make more money by seating someone else, however the customer's temporally overlapping behavior of drinking wine may be a sub-task of enjoying a leisurely meal.

This is not to say that hierarchies are not useful for representing interactive narratives for a case-based planning system. Grouping tasks into courser-grained tasks (e.g. subsuming tasks for greeting, seating, and giving the customer a menu into a task for commencing dining) will help retrieve a maximal number of matching gameplay traces by abstracting away subtle variations, and will give context to screen out unusual behavior. We take three positions with regard to task hierarchies for interactive narratives: (1) Task hierarchies should not be forced to collapse into a single overarching task. Multiple hierarchies may co-exist. (2) Task hierarchies should not be assumed based on annotations at the lowest level. Instead, tasks produced at one level can serve as input for task annotation of the next level of the hierarchy. (3) The flexibility of the hierarchy depends on the number of annotated gameplay traces, the more the better.

4. ANNOTATING TASKS WITH EAT

The Environment for Annotating Tasks (EAT) is a browser-based tool (developed in Adobe FLEX and Action Script 3) designed to allow non-experts to intuitively and efficiently annotate tasks in gameplay traces. The interface displays a game as a horizontally scrolling timeline, where each node represents an action or utterance. We omit actions for physical movement from the timeline -- it is safe to assume if the waitress picks up pie from the counter and puts it down on a table, she moved from the counter to the table. Annotators use the mouse to enclose nodes in boxes representing tasks, and then select a task label from a list. In expert mode, the annotator can add to the list of labels while annotating. Annotation is complete when every node has been enclosed in a task. Each node can only be enclosed in one task. In cases of ambiguity, the annotator chooses one or the other. Annotators can move nodes up or down into separate *bands* to accommodate cases where tasks overlap temporally, and actions and utterances from multiple tasks are interleaved. Figure 1 shows a timeline before and after annotation.

It took one of the authors 20 hours to annotate 100 games (randomly selected from the corpus), each with an average of 125 nodes. The list of task labels was generated during this initial annotation process, to cover the range of tasks typically observed in restaurant interaction. There are 28 task labels, plus OTHER for tasks that do not fit any of the labels in the list. Examples include W_DEPOSITS_BILL, W_SERVES_FOOD, W_CLEANS_TABLE, C_DRINKS, C_GETS_SEATED, C_EATS, C_COMPLAINS_BILL, SHARE_NAMES. While the optional prefix denotes customer (C) or waitress (W), labels are intentionally ambiguous as to who carries out the action. For instance, C_GETS_SEATED can be applied to a sequence where the customer seats himself, or a sequence where the waitress shows him to a table, as long as the end result is that the customer is sitting at a table. Any task can include a mixture of actions and utterances, carried out by either actor or both. OTHER is applied to sequences of mischievous behavior (e.g. eating flowers, sitting on the stove, stealing the register), atypical behavior (e.g. waitresses eating meals while on the job), or unintelligible dialogue. Recognizing tasks for misbehavior is of interest, but remains for future work. However, providing characters with a model of typical behavior enables them to disregard off-script behavior, which is a significant step toward more robust interactions.

Five additional annotators who were not involved with the development of EAT annotated a 10 game subset of the previously annotated games to evaluate inter-annotator agreement. Two of the annotators work in our lab. We had no face-to-face contact with the other three, who were given instructions via a tutorial web page. The annotators applied labels from the list supplied by the expert annotator. Annotating 10 games took three hours, and the annotators reported that the task became easier and faster after annotating the first few games. We computed an average kappa coefficient between the five annotators and the expert of 0.81. The annotators from our lab scored high agreement (0.8 and 0.9), but the other annotators still achieved substantial agreement (0.75, 0.7, and 0.89). Kappa between 0.61 and 0.80 is considered substantial agreement [12]. Our results suggest that this annotation task is accessible to non-experts with minimal investment in training.

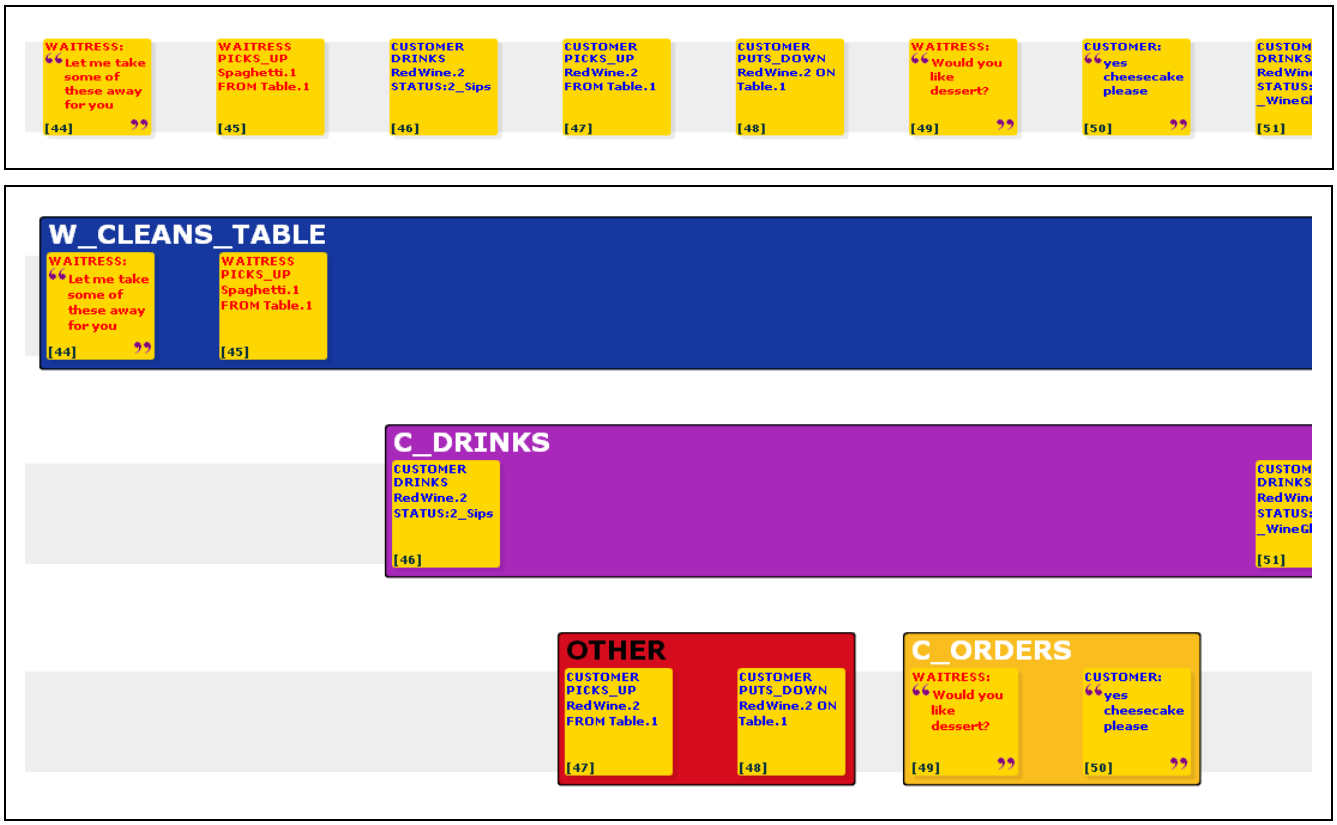

Figure 1. Gameplay trace in EAT before (top) and after (bottom) annotation.

We have annotated tasks for one level of a task hierarchy, but it is not hard to imagine repeating this process recursively, where tasks at one level become nodes for annotation at a higher-level. To support nesting tasks, which may overlap temporally, the interface needs to allow boxes that can stretch over multiple bands to enclose vertically adjacent tasks. Adjacent tasks can be stored as sequences of task start and end nodes. For example, discussing where players are from while eating can be represented: C_EATS.1.START, SHARE_GEOGRAPHY.2.START, SHARE_GEOGRAPHY.2.END, C_EATS.1.END.

Ultimately we want to annotate an entire corpus of 10,000 gameplay traces to guide the case-based planning system. The combination of EAT and RUN affords us two options, each with trade-offs. One option is for humans to hand-annotate the entire corpus online, the other is to use 100 annotated games to train RUN to automatically annotate the rest. The fact that EAT is browser-based and accessible to non-experts allows us to reformulate the difficult problem of authoring behavior into something akin to data-entry. Testing inter-annotator agreement with a larger population is required to draw conclusions about the accessibility of the tool in general, but our pilot test results are encouraging, pointing to the potential possibility of hiring an online workforce to annotate data through an outsourcing service such as Amazon's Mechanical Turk. Annotating 10,000 games will take about 2,000 hours – almost a man-year of work. However, unlike traditional approaches to authoring AI behaviors, where the labor cannot be easily distributed and parallelized, the mythical man-month [2] actually *does* apply here. Dividing the corpus among ten annotators would allow completion in about one month. Annotating gameplay traces automatically with RUN has the advantage of being fast and free, but will only recognize patterns for tasks observed in the 100 training games. Online annotation costs time and money, but can collect a much wider variety of examples, providing better coverage of the subtle nuances of human behavior. Either way, the RUN algorithm is still necessary to recognize tasks during interactions at runtime.

5. THE RUN ALGORITHM

The RUN algorithm is a simple yet effective algorithm for recognizing tasks at runtime, based on a dictionary of examples extracted from annotated gameplay traces. RUN can also be used to automatically annotate tasks in a corpus of gameplay traces, as a preprocessing step to support the case-based planning system.

5.1 The Task Dictionary

We compile a dictionary of examples by extracting each instance of an annotated task from each gameplay trace. This provides many subtle variations of token sequences (where tokens are actions and utterances) sanctioned by humans as valid examples. While annotators see plain English text (e.g. WAITRESS PICKS_UP Bill.1 FROM Table.4, or "what would u like?"), internally actions and utterances are stored in a machine-comprehendible representation. Actions are stored as indices into a lexicon. We have previously described automatically learning an action lexicon from thousands of gameplay traces [16]. Our lexicon has over 7,000 unique actions, represented in a STRIPS-like format [4], with preconditions and effects. Actions are

context-sensitive and role-dependent (e.g. waitress picks up pie from counter). In the 100 annotated games, we observed 665 unique actions. Utterances are stored as dialogue acts, represented as {speech act, content, referent} triples. For example, "I'll have the steak" is represented as DIRECTIVE_BRING_FOOD. Elsewhere we have described a dialogue act classifier that can be trained to automatically label utterances [18]. For this study, we hand-annotated the utterances in the 100 games prior to annotating tasks, in order to evaluate the task recognition algorithm in isolation in a best-case scenario. We observed 312 unique dialogue act triples in the 100 annotated games. Tokens for utterances with dialogue act annotations containing OTHER are omitted from the extracted examples.

We do not attempt to generalize examples – RUN will only recognize tasks that match one of the examples exactly (in terms of action indices and dialogue act triples). After pruning examples that have only been observed in one gameplay trace, our dictionary has 264 unique examples, covering the 28 tasks labels. Each label has between 1 and 29 examples (mean 9.43, median 7), where an example consists of between 1 and 10 tokens (mean 2.73, median 3). There are 322 unique tokens in the dictionary after pruning.

The dictionary is stored in a *trie* (or suffix tree [9]), which stores each example as a branch from the root to a leaf node. If the first *n* tokens are identical in two example sequences, they will share the first *n* nodes from the root, and then split into separate branches. Testing whether some sequence matches (or is a substring of) any valid example is efficient with this data structure. When inserting examples into the trie, we append an END token, which allows us to detect valid examples that may be substrings of longer valid examples. The END nodes become the leaves of the trie, and these nodes store the task label describing the example that formed the corresponding branch.

5.2 Task Recognition & Sequence Alignment

Like a human annotator identifying tasks by separating actions and utterances into bands and grouping them into boxes, RUN is intended to disentangle tasks represented by sequentially interleaved tokens, which may be separated by arbitrarily long gaps. RUN accomplishes this by searching for sub-sequences in a gameplay trace that match examples in the task dictionary. On the surface, this pattern-matching problem seems similar to sequence alignment in computational biology [9], where algorithms exist to search for known protein sequences within much longer sequences. However, the performance requirements and characteristics of the data are very different, justifying development of a new, simpler algorithm.

Biological sequence alignment algorithms run offline, searching for one sequence at a time in a large database of sequences. Algorithms that guarantee optimal alignment employ dynamic programming solutions, which are expensive in terms of space and time. More efficient, heuristic search-based algorithms exist, which sacrifice optimality for speed, but are still offline algorithms that search for one sequence at a time, as well as being complex to implement. An interactive narrative system demands a fast, memory-efficient algorithm that runs online to detect tasks in real-time as a character receives new observations, and ideally is easy to implement. Optimal alignment is not as important as getting the gist of the situation by simultaneously recognizing examples of all possibly occurring tasks, while also filtering out irrelevant observations for behavior that is mischievous or does not require a character's response.

Due to the nature of the data, sequence alignment and task recognition are essentially different problems. Biological data has a small number of tokens which have no meaning on their own (e.g. A, T, G, and C in DNA). The tokens themselves are entirely ambiguous, but when ordered in different combinations give meaning to sequences. Our tokens (actions and utterances), on the other hand, are meaningful on their own, and very few of them are ambiguous in terms of task membership. Of the 322 unique tokens in the pruned dictionary, only 19 can be found in examples of more than one task. (With the exception that any token can be labeled as task OTHER, discussed more below). The most ambiguous tokens are associated with utterances like "yes" and "thanks." We did not make any special effort to minimize ambiguity of tokens to ensure tasks would be easily separable; but rather this is a natural consequence of humans grouping actions and utterances into coherent sequences, within some structured scenario. It is this same meaningful property of the tokens that leads to high inter-annotator agreement between humans in the data that generates the examples for the dictionary. Matching patterns within sequences of highly ambiguous biological tokens necessitates more computationally intense approaches than are required for recognizing more easily separable tasks.

While biological sequence alignment algorithms are computational overkill for task recognition, the fact that they align one pair of sequences at a time offline does not address the unique challenges associated with simultaneously recognizing any example of any task type within observations of fluid, tangled human interaction between two actors. Each task may be represented by many different example sequences, some of which may be substrings of one another. A single game may contain multiple instances of the same task, and these tasks may be overlapping in time. A gameplay trace may contain degenerate uncompleted task fragments, which should be ignored (by labeling the fragment OTHER). If a customer asks for a menu, and the waitress does not respond, the task recognizer should not detect that a C_GETS_MENU task occurred (because in fact it did not – the customer never received a menu). Through the same process the recognizer should ignore typical actions and utterances occurring in atypical contexts, such as ordering many pies from the chef and stacking them on the counter. Only completed tasks that completely match human annotated examples of typical behavior should be recovered.

5.3 The Algorithm

RUN is a greedy algorithm that iterates over a sequence of tokens, adding each to an accumulating sequence in the Open or Closed list (closed sequences can continue to be extended). Accumulating tokens in multiple sequences in the Open and Closed lists is conceptually similar to what humans are doing with EAT, by separating actions and utterances into different bands to disentangle tasks. Once the entire input sequence has been processed, the sequences in the Closed list indicate the task labels to apply to spans of the original sequence (possibly including gaps within). A labeled sequence is guaranteed to match one of the examples in the task dictionary. Figure 2 presents one iteration of the algorithm in pseudo code, which can be run in a loop to

```
Dictionary d
SequenceList open, closed
SequenceTokenIndex iPos
Token tok

void processToken(d, open, closed,
   tok, iPos) {
   if(addToOpen(d, open, closed,
      tok, iPos)) return
   else if(startNew(d, open, closed,
      tok, iPos)) return
   else addToClosed(d, closed, tok, iPos) }

Boolean addToOpen(d, open, closed,
   tok, iPos){
   for each seq in open {
      append(seq, tok)
      if( !exists(d, seq) )
         remove(seq, tok), continue
      seq.end = iPos
      remove(open, seq)
      if( matchesWholeExample(d, seq) )
         seq.label = getLabel(d, seq)
         push_front(closed, seq)
      else push_front(open, seq)
      return true }
   return false }

Boolean startNew(d, open, closed,
   tok, iPos) {
   seq=tok, seq.start=iPos, seq.end=iPos
   if( !exists(d, seq) ) return false
   if( matchesWholeExample(d, seq) )
      seq.label = getLabel(d, seq)
      push_front(closed, seq)
   else push_front(open, seq)
   return true }

Boolean addToClosed(d, closed, tok, iPos) {
   for each seq in closed {
      append(seq, tok)
      if( !exists(d, seq) )
         remove(seq, tok), continue
      if( matchesWholeExample(d, seq) )
         seq.label = getLabel(d, seq)
         seq.end = iPos
      remove(closed, seq)
      push_front(closed, seq)
      return true }
   return false }

void appendOpenToClosed(d, open, closed) {
   for each o_seq in open {
      for each c_seq in closed {
         if( o_seq.start < c_seq.end )
            continue
         append( c_seq, o_seq )
         if( !exists(d, c_seq) ||
             !matchesWholeExample(d, c_seq))
            remove(c_seq, o_seq)
         else c_seq.end = o_seq.end
            c_seq.label = getLabel(d, c_seq)
} } }
```

Figure 2. Pseudo-code for the RUN algorithm.

process an entire gameplay trace, or run periodically as new observations arrive at runtime. Our implementation was written in about a page of Java in a few hours, and processes an average gameplay trace (125 tokens) in 0.015 seconds on a Pentium 4.

The Open and Closed lists begin empty. For each token in the input sequence, we first try to add the token to an accumulating sequence in the Open list. If that fails, we try to start a new sequence with the token. Finally, if all else fails, we try to extend a previously closed sequence. Any time an open sequence is found that exactly matches a sequence in the dictionary, the sequence is moved to the Closed list. In some cases, a new sequence is started and immediately closed, but may continue to be extended in the Closed list. As a sequence in the Closed list is extended, we keep track of the last end point that marked an exact match of a complete example. This end marker is important when finally applying task labels based on the Closed sequences, for ensuring all task instances exactly match human-sanctioned examples. Dictionary queries determine if sequences are potentially valid (meaning they are sub-sequences of examples), and if sequences represent an exact match of a complete example. Whenever we successfully add a token to a sequence, we move the sequence to the front of the Open or Closed list. This leads RUN to prefer more compact sequences by continuing to extend whatever was most recently extended.

Once the entire input stream has been processed, task labels can be applied by iterating over the sequences in the Closed list. There is some book keeping required (omitted from the pseudo code) to keep track of which action or utterance instances are associated

with each token. Prior to assigning task labels from the Closed list, we try to salvage fragments in the Open list, which would be otherwise discarded, by trying to append them to the sequences in the Closed list with `appendOpenToClosed`. This step can be performed once after processing an entire input stream, or after processing each new observation at runtime.

RUN can be applied hierarchically by running multiple instances of the algorithm in parallel, once for each level of the hierarchy. The output of one level of the hierarchy becomes input for the next. Nested temporally overlapping task instances can be encoded as a sequence of start and end points, as describe in Section 4.

5.4 Task Recognition Evaluation

We evaluated RUN with a 10-fold cross-validation test of recognizing tasks in the 100 human annotated gameplay traces, where each fold was trained on 90 traces and tested on 10. Our results show that RUN works well on our dataset, achieving a 0.744 precision and 0.918 recall. Precision measures how often RUN and the human assigned the same task label to a token. Recall measures how many of the human-annotated task instances were recovered by RUN. The baselines for precision and recall are 0.362 and 0.160 respectively. While these results are encouraging, we need to evaluate RUN on a dataset of gameplay traces from another domain for comparison. However, such datasets are not readily available.

The fact that RUN scores higher recall than precision indicates that the system is doing a good job of getting the gist of the interaction, but is sometimes omitting tokens within a sequence. High recall will benefit the case-based planner, which needs to find gameplay traces with similar task histories. Task labels with low recall are primarily the result of sparse data. For instance, RUN has trouble recognizing C_COMPLAINS_BILL, because there are only 15 examples, and the examples are highly varied. Annotating more gameplay traces should improve recall of C_COMPLAINS_BILL, especially if annotators focus on games where customers complain about the bill. Figure 3 suggests that we need at least 40 training files to maximize recall, in general.

Naturally precision is affected by sparse data as well, but the biggest factor lowering precision is related to tokens for lowering precision is related to tokens for utterances that were omitted from examples in the dictionary due to dialogue act annotations containing OTHER, indicating that a human found them in some way unclassifiable. In these cases, RUN will automatically assign the task label OTHER and ignore these tokens completely. RUN also has problems when an example in one task label is a substring that begins an example in another label. For instance, we label the following sequence W_PREPARES_BILL: waitress asks if the customer wants anything else, he replies "no, just the check", she interacts with the register. We apply the same label if the customer replies "no", and the waitress say "I'll get your bill then", and uses the register. The problem is that in other instances we see a waitress ask if the customer wants anything else, and the customer replies "no," and continues eating his meal. These instances are labeled W_CHECKS_IN. This is clearly ambiguous to RUN (as it is for humans), and perhaps suggests a revision to our labeling protocol. However, while ambiguity like this lowers our evaluation statistics, it is often ultimately inconsequential – it does not really matter whether we consider the whole sequence

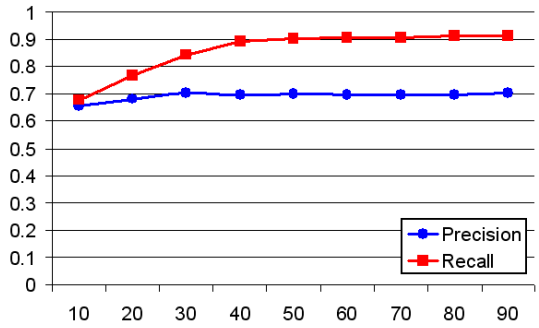

Figure 3. Effect of corpus size (x axis) on precision and recall

W_PREPARES_BILL, or W_CHECKS_IN followed by W_PREPARES_BILL.

One of our requirements for RUN is that it does a good job of filtering out tokens that do not contribute to typical restaurant behavior. Moving sequences from Open to Closed only when they exactly match a complete dictionary example is intended to filter out atypical behavior, even when composed of otherwise typical actions and utterances. We can evaluate success in filtering by looking at the recall for tokens labeled OTHER by a human. If we only use an Open list in RUN, and never move sequences to Closed, recall for OTHER is 0.594. The full implementation with Open and Closed achieves a 0.817 recall for OTHER, a 27% improvement over the naïve solution.

6. RELATED WORK

We relate our work to previous research on dialogue modeling and learning from human data, and highlight differences. Gorin et al [7] describe learning to route calls in response to the prompt "How may I help you?", by finding mutual information between routing decisions and n-grams in utterances. Satingh et al [21] developed a dialogue system that learns an optimal policy for a phone-based information system from interactions with human callers. Huang et al [11] trained chatbots by extracting title-reply pairs from online discussions. Our work differs by collecting data from humans situated in a (virtual) physical environment, where players dramatize an everyday scenario through a combination of (typed) dialogue and physical interaction, contributing to learning an interleaved model of actions and utterances representing a commonsense script of restaurant behavior.

McQuiggan and Lester [14] applied a similar methodology to ours (capturing demonstrations between humans in a game environment) to learn models of empathetic behavior. Their system learns to give emotional commentary through character gestures and pre-recorded audio clips. We are learning a model of collaborative behavior, including open-ended typed dialogue, where actors playing different roles depend on each other to dramatize a scenario constrained by learned social conventions. Gorniak and Roy [8] collected data, including speech, from pairs of players solving a puzzle in *Neverwinter Nights*, and constructed a plan grammar, which could be used to understand utterances between players. Similarly, Fleischman and Hovy [5] leveraged a task model of a game-based training exercise to understand natural language input. In these projects, hand-constructed models of the situation (plan grammar or task model) helped the system understand language. In contrast, we are training a system to automatically recognized tasks composed of utterances and

actions. While our data collection methodology is similar to previous work, we are working toward learning the structure of the situation from data, based on semi-automated annotation, rather than hand-crafting a plan grammar or task model. Learning the structure has the potential of producing a more robust model through a less laborious process.

7. CONCLUSIONS & FUTURE WORK

RUN performs well on our dataset, recalling almost 92% of the human annotated task instances. There are numerous possible improvements to RUN that may help us recall the remaining 8%, many of which are composed of sequences that end up discarded in the Open list. There may be ways to merge fragments in the Open list to recover additional valid examples. When RUN encounters ambiguity, we might benefit by exploiting frequency statistics of examples to determine which sequence to extend, rather than always selecting the most recently extended. Our results reflect the best-case scenario, with human annotated dialogue acts; we need to evaluate the impact of automatically classified utterances. We would also like to evaluate how well we can detect tasks representing common ways to misbehave.

The big question is how well this methodology generalizes to new scenarios. There are certainly other routinized everyday activities, but the more interesting question is whether latent script-like structures can be recovered from online re-enactments of less mundane scenarios (e.g. hold-ups, lovers' quarrels, interrogations, etc). We are building a new game to evaluate such generalization.

Our methodology relies on human annotation, and further evaluation of inter-annotator agreement is required to determine if strong results hold when online annotation is taken to an extreme, and is distributed among anonymous outsourcing workers. If results do hold, this hints at an exciting new potential approach, in which we can refactor the difficult problem of authoring content for interactive narratives by collecting content from humans interacting online, and outsourcing the labor involved in making these gameplay traces useful to an AI system. This democratization of the development process shifts the focus from engineering, to directing humans to dramatize the desired performances, and labeling these examples such that a designer can sculpt narrative possibilities from a desired subset of the data.

8. REFERENCES

[1] Allen F. G. 1983 Maintaining knowledge about temporal intervals. Communications of the ACM, 26(11):832–843.

[2] Brooks, F.P. 1975 The Mythical Man Month, Reading, Mass.: Addison-Wesley.

[3] Cavazza, M., Charles, F., and Mead, S. J. 2002 Sex, Lies and Videogames: an Interactive Storytelling Prototype. In Proc. of AAAI Symposium on Artificial Intelligence & Interactive Entertainment.

[4] Fikes, R.E. and Nilsson, N.J. 1971 STRIPS: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3-4), 189-208.

[5] Fleischman, M. and Hovy, E. 2006 Taking Advantage of the Situation: Non-Linguistic Context for Natural Language Interfaces to Interactive Virtual Environments. In Proc of Intelligent User Interfaces.

[6] Gervás, P., Lönneker-Rodman, B., Meister, J.C., and Peinado, F. 2006 Narrative Models: Narratology Meets Artificial Intelligence. In Proc. of Toward Computational Models of Literary Analysis, International Conference on Language Resources and Evaluation. Genoa, Italy.

[7] Gorin, A., Riccardi, G., and Wright, J. 1997 How may I help you? Speech Communication, Volume 23, Elsevier Science.

[8] Gorniak, P. and Roy, D. 2005 Speaking with your sidekick: Understanding situated speech in computer role playing games. In Proc. of Artificial Intelligence & Digital Entertainment.

[9] Gusfield, D. 1997 Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology. Cambridge University Press.

[10] Hammond, K. F. 1990 Case based planning: A framework for planning from experience. Cognitive Science, 14(3).

[11] Huang, J., Zhou, M., and Yang, D. 2007 Extracting chatbot knowledge from online discussion forums. In Proc. of the International Joint Conferences on Artificial Intelligence.

[12] Landis, J.R., and Koch, G.G. 1977 The measurement of observer agreement for categorical data. Biometrics, 33(1).

[13] Mateas, M. 2002 Interactive Drama, Art and Artificial Intelligence. Ph.D. Thesis. School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

[14] McQuiggan, S., and Lester, J. 2006 Learning empathy: A data-driven framework for modeling empathetic companion agents. In Proc. of Int. Joint Conference on Autonomous Agents & Multi-Agent Systems. Hakodate, Japan.

[15] Minsky, M. 1987. The Society of Mind, Simon and Schuster.

[16] Orkin, J. and Roy, D. 2007 The Restaurant Game: Learning social behavior and language from thousands of players online. Journal of Game Development, 3(1).

[17] Orkin, J. and Roy, D. 2009 Automatic Learning and Generation of Social Behavior from Collective Human Gameplay. In Proc the International Conference on Autonomous Agents and Multiagent Systems.

[18] Orkin, J. and Roy, D. 2010 Semi-Automated Dialogue Act Classification for Situated Social Agents in Games. In Proc of the AAMAS Agents for Games & Simulations Workshop.

[19] Ortanon, S., Mishra, K., Sugandh, N., Ram, A. 2007 Case-based planning and execution for real-time strategy games. In Proc. of the 7th Int. Conference on Case-Based Reasoning Research and Development.

[20] Schank, R.C., and Abelson, R.P. 1977 Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures. Lawrence Erlbaum Associates.

[21] Satingh, S., Litman, D., Kearns, M., and Walker, M. 2002 Optimizing Dialogue Management with Reinforcement Learning: Experiments with the NJFun System. Journal of Artificial Intelligence Research.

[22] Young, R. M. 2007 Story and Discourse: A Bipartite Model of Narrative Generation in Virtual Worlds, in Interaction Studies: Social Behaviour and Communication in Biological and Artificial Systems, 8, 2(32).