Abstract
Just as completed jigsaw puzzles are the product of the correct union of its pieces, stories are the result of the organized assembly of the experiences of its participants. This paper introduces a novel approach to reconstruct and generate complete narratives, from the analysis and processing of the memories and experiences of its participating actors. We propose a framework capable of representing and persisting each virtual agent’s perceptive memories and experiences. For each agent, said information is expressed as a collection of behaviors, interactions, and perceived events. Once collected, each agent’s memory is treated as a partial narrative. The architecture employs novel merging, validation, and extrapolation mechanisms to generate coherent and complete narratives as the product of the organized intersection of the partial narratives. In addition, alternative narratives can be generated by altering past behaviors and events and constraints. We demonstrate the efficacy of this implementation by constructing consistent narratives based on the extrapolation of data generated from running pre-scripted virtual simulations. It is worth noting that despite of the fact that we test the system in a controlled environment, the former does not have access to any information besides its main expected input, the participants memories.

1 Introduction
Not only have many traditional industries immensely benefited from advances in digital simulations, but others have now emerged solely because they are capable of representing realities with controlled computational environments. These are as diverse as their end products. Some examples are architecture, construction, engineering, visual arts and the video game industry, among others.

This demand results in the need of creating highly capable software solutions which allow designers and developers to bring stories to life in animated complex simulations. In many instances, even more important than being able to visually represent these models, is to extract or synthesize meaningful, compelling stories. In this paper, we seek to analyze these narratives from the perspective of each individual actor in the story, and develop mechanisms to create a holistic reconstruction of the story from the memories of a subset of its participating actors. There are many application fields which will benefit from the ability to recreate previous events from its participant’s recollections. For example, police investigations could precisely reconstruct scenes and gather facts based on witness narratives, a sports game could generate an entire play based on the memories of its spectators, and news stories could be assembled from disparate observer viewpoints.

Another advantage of a such a system is the capability of generating multiple plausible alternative narratives, or what-if scenarios, by modifying a simulation’s parameters such as behaviors, actors and events, among others. The use
cases for such a technology are only limited by the needs and the ability of the users.

We call the proposed framework an Event-Based Agent Framework (EBAF). It allows actors to observe its surroundings and record the events they perceive. This information is used to reconstruct events and is based on the natural principle of how events are perceived by its actors. We tackle the challenge of reconstructing narratives based on partial, incomplete, and possibly inconsistent recollections. For instance, agents might not be aware of the motive and facts behind certain events, if they weren't involved. These aspects are left for the systems mechanisms to process, analyze and ultimately rebuild.

In summary, EBAF algorithms extract each agents autobiographic memories that are captured throughout the simulation, and generates plausible, complete, and consistent narratives. Finally, we evaluate the robustness of EBAF in a pre-scripted, story-driven virtual environment.

2 Related Work

Generating coherent, meaningful narratives is a challenging problem. Many researchers [1], are tackling this with different approaches. Some main issues are identified as constructing a plot, creating believable characters, and managing an interactive storytelling environment. Much work can be found on manual authoring techniques for the narrative generation. An example of is the model mentioned by [2]. This instance describes predefined behaviors, which means that even small changes to the narrative result in monolithic work, proportional to the complexity of the stories. One approach, proposed in [3], involves learning scripted narratives from knowledge obtained via crowd-sourcing. The work in [4] proposes a framework which uses directed story graphs for authoring massive believable narratives with very little user interactions. In story graphs, the user interactions are facilitated in the form of choices at key points in the narrative. Of course, the complexity of such a graph grows, as the choices and paths increase in the narrative. The work in [5] presents Parameterized Behavior Trees (PBT’s) to improve the modularity and reusability of digital narratives. This helps in reusing similar story arcs to generate more complex narratives. The work in [6] proposes the use of computational intelligence to help complete inconsistent stories, using Partial Order Planning [7]. Coherence is an essential aspect of narratives. A solution to the problem of maintaining consistency is addressed in [8], which discusses a novel intent-driven planning technique to generative coherent narratives.


ADAPT [13] is an agent architecture for designing and authoring functional, purposeful human characters in a rich virtual environment. It is a versatile platform to implement smart characters with rich interaction components. The work in [14] describes a framework for mitigating individual agent complexity while retaining agent diversity. The work of [2] provides a study for creating believable agents. Although the framework in [15] can be used to quickly generate complex and believable narratives, the memory requirements increase exponentially as the number of possible actions increases. Boloni presents a cognitive architecture [16] where agent biographic memories are stored as concept overlays, and reasoning in done using statistical approaches. Very few attempts have been made to introduce a memory model in virtual agents, which can also be used to for narrative discourse. This is the main focus of this paper.

3 Framework Overview

Figure 3 presents an overview of the framework.

Narrative Module. At the system level, we use an event-centric architecture similar to [10] for authoring pre-scripted narratives. Parameterized Behavior Trees [17, 11] are used as the logical representation for events between actors in a story. When an event executes, it broadcasts information to all actors in the story, which
includes details of event execution, and how it updates the states of participating actors.

**Message Bus.** This component is responsible for broadcasting an event’s details message to all the simulation’s participating agents. All the messages passed to the bus will eventually be sent to all agents. Although these messages are passed in real-time during each simulation cycle, they will be pushed into a data structure. Once a cycle is completed, all the messages in this queue are published. The bus is cleared at the start of every cycle.

**Story and Actor State.** Throughout the execution of a story, each smart object will have a specific state at any given point in time. System state keeps track of all the object states in the simulation. Starting with the initial state of the system, the System State keeps updating the object states after every event execution. So, if queried, the System State module returns the state of an object at that point of time in the story.

**Agent Architecture.** The agent model simulates perception using a simple vision model, memory as a collection of events it perceived, and spatio-temporal reasoning of past events. Although sensory information can be captured via different channels, this implementation just considers visual stimuli. When an observer receives an event from the Message Bus, it verifies whether that event is related to the objects in its field of view. If it is, the observer module gets the state of the object from the System State and creates a memory with the object state, perception data and event notification. These memories are stored in the agent’s memory module.

### 4 Memory Representation

A Memory Event \((m)\) is composed of:

- **memoryName** - This is a three-tuple construct of the form: \((actor_1, actionName, actor_2)\). For example, John opens Door. Any action (or affordance) is indexed by its unique token ‘actionName’ identifier. Smart objects also have a unique identifier, ‘name’.

- **type** - A memory event can either be a start event or an end event denoted by \(m_{start}\) and \(m_{end}\) respectively. By default, start events only have \(startTime\) and end events only have \(endTime\).

- **startTime** and **endTime** - Each event’s start and finish timestamps.

- **actorOneState\((S_1)\)** - The state of the first actor, represented as a set of conditions on its state \(\{\phi\}\).
Figure 3: Overview of Narrative Reconstruction process

- actorTwoState(S2) - Just as above but for the second actor in the event.
- perceptionData - Visual sensory data recorded by agent perception. For example, it records the spatial information of the objects in the memory event at the time of recording.

Affordances are the main execution units of behaviors. These are represented as the leaves of Behavior Trees, and every event is made out of behaviors, so ultimately, every event is a behavior tree at the system level. Hence, the partial memories of each individual actor can be represented as a Parameterized Behavior tree with missing, or incomplete leaf nodes.

5 Memory Generation

Every agent in the scene is capable of perceiving and to persist its own experiences. Agent perception constantly checks the objects in its Field of View (FOV) and waits for the Message Bus to deliver updates on the states of these objects. Whenever an event notification appears, the agent will only update those which it perceives (see Figure 2). When this occurs, the agent creates a memory object by populating the spatial information and getting the object state from System State module. All the memories of an agent are stored in its biographic memory (Mobs). This process is summarized in Algorithm 1.

Algorithm 1: Method to generate autobiographic memories of an agent

1 do
2   msgs = MsgBus → {msgs}
3   foreach msg ∈ msgs do
4     if Objmsg ∈ ObjsInFOV then
5       S1 = GetObjState(msg → {Obj1})
6       S2 = GetObjState(msg → {Obj2})
7       Mobs = Mobs ∪ memory(msg, SpatialInfo, S1, S2)
8   while MsgBus.hasMsgs()

6 Memory Reconstruction

Memory reconstruction is divided into three steps, as illustrated in Figure 3. First, the selected agent memories are merged, while accounting for temporal consistency and ordering of events. Then, the consistency of the generated narrative is validated. For any found inconsistency in the story arc (either through missing or incomplete event information), a partial order planner is used to extrapolate the missing parts of the narrative. These steps are discussed in detail below.

6.1 Merging Agent Memories

In this module, the memories from the selected agents are merged and the resulting start & end events are paired. For memories with no existing counterpart, the corresponding event is esti-
Guard 1

I looks at MallEntry1 (start)
Rob meets me (start)
Rob meets me (end)
Dan meets me (start)
Dan meets me (end)

Guard 2

I looks at MallEntry2 (end)
I looks at MallEntry2 (start)
Rob meets me (start)
Dan meets me (start)
Dan meets me (end)

Algorithm 2: Method to merge memories, pair events and estimate times

1. MergeAgentMemories()
2. CategorizeMemories()
3. PairStartAndEndEvents()
4. Sort $M_{start}$ by $m_{start} \rightarrow startTime$
5. Sort $M_{end}$ by $m_{end} \rightarrow endTime$

(a) All Memories

Guard1 looks at MallEntry1 (7.639292, -1)
Guard1 looks at MallEntry1 (-1, 7.705658)
Rob meets Guard1 (12.62267, -1)
Rob meets Guard1 (-1, 14.39522)
Dan meets Guard1 (-1, 15.35437)
Dan walks to MallEntry1 (15.40655, -1)
Dan meets Guard1 (-1, 15.35437)
Guard2 looks at MallEntry2 (7.639304, -1)
Guard2 looks at MallEntry2 (-1, 8.634283)

Categorized Start Events

Guard1 looks at MallEntry1 (7.639292, -1)
Rob meets Guard1 (12.62267, 14.39522)
Dan meets Guard1 (12.62268, 15.35437)
Dan walks to MallEntry1 (15.40655, -1)

Categorized End Events

Guard1 looks at MallEntry1 (7.639292, -1)
Rob meets Guard1 (12.62267, 14.39522)
Dan meets Guard1 (12.62268, 15.35437)
Guard2 looks at MallEntry2 (7.639304, -1)
Guard2 looks at MallEntry2 (-1, 8.634283)

(b) (c)

Figure 4: (a) shows the the auto-biographic memories. Some memories in (b) have no start or end, represented as -1. (c) shows the events after categorizing into start and end events with existing memory events paired with start & end times populated.

Having a respective start-event for every end-event is necessary as these pairs are used to validate the narrative, as described in Section 6.2. Algorithm 2 is used for merging, pairing & estimating memories. First, the memories of every selected agent is extracted and merged, and the resulting unique set of memories are stored in $M$ (MergeAgentMemories function of Algorithm 2). Once gathered, CategorizeMemories function sorts memories into start and end events ($M_{start}$ & $M_{end}$ respectively) based on the type of the memory event. Figure 4 shows the step-by-step process of how the extracted memories are categorized using an example.

The pairing of the events is done in two steps. If the end event for a start memory is already present in the current memory collection, the start & end times for the events are populated using their counterparts (See lines 4-7 of Algorithm 3). For memories whose counterparts do not exist in the agent memories, an estimate for the time taken for that event is calculated using Memory Time Estimation function. These estimations, are defined in the original atomic affordance definition. For example, GoTo action will have an estimation function described as $[distanceToTheLocation / agentVelocity]$. Also, an estimated memory ($m_{Est}$) is created for this, which will be used for Narrative Validation. Lines 9-18 of Algorithm 3 describe the implementation of the estimation, and Figure 5 shows an example of the estimated event. Finally, by topologically sorting the start events ($M_{start}$) we get the temporal structure of the narrative. In section 6.2 we check whether the narrative deduced from the agent memories is complete or consistent.

6.2 Narrative Validation

In this phase, we check whether the generated narrative is consistent. As part of this process, we identify causal links between memory events, which are used to determine the validity of the generate memory stream. A causal link between $m_1$, $m_2$ can be formally expressed as $CL(m_1,edge,m_2)$. The general execution flow of this phase is as follows:

1. Analyze Narrative Consistency
2. Determine possible start state of the narrative
3. Establish causal links between memory events
4. Identify Inconsistent Causal Links in the Narrative
Algorithm 3: Method to pair start and end events and populate start and end times. If the counter part of a memory does not exist, the time is estimated.

1 **PairStartAndEndEvents()**
2   foreach \( m_{\text{start}} \in M_{\text{start}} \) do
3       if \( m_{\text{start}} \rightarrow \text{endTime} == -1 \) then
4           \( m_{\text{end}} = \text{FindPairForMemory}(m_{\text{start}}) \)
5           if \( m_{\text{end}} \rightarrow \text{Exists()} \) then
6               \( m_{\text{start}} \rightarrow \text{endTime} = m_{\text{end}} \rightarrow \text{endTime} \)
7               \( m_{\text{end}} \rightarrow \text{startTime} = m_{\text{start}} \rightarrow \text{startTime} \)
8           else
9               timeForCompletion = \( \text{EstimateTimeForMem}(m_{\text{start}}) \)
10              \( m_{\text{start}} \rightarrow \text{endTime} = m_{\text{start}} \rightarrow \text{startTime} + \text{timeForCompletion} \)
11              \( m_{\text{Est}} = \text{EstimateEndEvent}(m_{\text{end}}) \)
12              \( M_{\text{end}} = M_{\text{end}} \cup \{m_{\text{Est}}\} \)
13   foreach \( m_{\text{end}} \in M_{\text{end}} \) do
14       if \( m_{\text{end}} \rightarrow \text{startTime} == -1 \) then
15           timeForCompletion = \( \text{EstimateTimeForMem}(m_{\text{end}}) \)
16           \( m_{\text{end}} \rightarrow \text{startTime} = m_{\text{end}} \rightarrow \text{endTime} - \text{timeForCompletion} \)
17           \( m_{\text{Est}} = \text{EstimateStartEvent}(m_{\text{end}}) \)
18           \( M_{\text{start}} = M_{\text{start}} \cup \{m_{\text{Est}}\} \)

(a)

<table>
<thead>
<tr>
<th>Categorized Start Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guard1 looks at MallEntry1 (7.69292, 7.03658)</td>
</tr>
<tr>
<td>Rob meets Guard1 (12.62687, 14.39522)</td>
</tr>
<tr>
<td>Dan meets Guard1 (12.62687, 15.35437)</td>
</tr>
<tr>
<td>Dan walks to MallEntry1 (15.40655, -1)</td>
</tr>
<tr>
<td>Rob meets Dan (26.09479, -1)</td>
</tr>
<tr>
<td>Guard2 looks at MallEntry2 (7.69304, 8.634283)</td>
</tr>
<tr>
<td>Rob meets Guard2 (34.95145, 46.2912)</td>
</tr>
</tbody>
</table>

(b)

<table>
<thead>
<tr>
<th>Categorized End Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guard1 looks at MallEntry1 (7.69292, 7.03658)</td>
</tr>
<tr>
<td>Rob meets Guard1 (12.62687, 14.39522)</td>
</tr>
<tr>
<td>Dan meets Guard1 (12.62687, 15.35437)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Estimated Start Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guard1 looks at MallEntry1 (7.69292, 7.03658)</td>
</tr>
<tr>
<td>Guard2 looks at MallEntry2 (7.69304, 8.634283)</td>
</tr>
<tr>
<td>Rob meets Guard1 (12.62687, 14.39522)</td>
</tr>
<tr>
<td>Dan meets Guard1 (12.62687, 15.35437)</td>
</tr>
<tr>
<td>Guard2 looks at MallEntry2 (15.40655, -1)</td>
</tr>
<tr>
<td>Rob meets Dan (26.09479, -1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Estimated End Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>Guard1 looks at MallEntry1 (7.69292, 7.03658)</td>
</tr>
<tr>
<td>Guard2 looks at MallEntry2 (7.69304, 8.634283)</td>
</tr>
<tr>
<td>Rob meets Guard2 (34.95145, 46.2912)</td>
</tr>
</tbody>
</table>

Figure 5: The estimated times for the events in Figure (a) are calculated and updated as shown in Figure (b).

The system must first determine the start state of the objects before the narrative execution. The overall start state is given by the collection of all states from participating objects. We only consider the states at their first occurrence \((FirstOcc[Obj])\) in the narrative. Algorithm 4 describes the implementation details for calculating the start state of the narrative. First, the initial occurrence of each object in start memories is stored in a \textit{FirstOccurrences} map \((FirstOcc)\) [see lines 2-13]. If an end event happened before a start event, it means that the start state we assumed for that object is not actually the start state of that object as an event happened before that. So, we remove that object from the \textit{FirstOccurrences} map [see lines 14-21]. Finally, in lines 22 & 23, we merge the object states to determine the start state of the narrative \((\Phi_{\text{start}}))

Once the start state has been identified, we proceed to check the consistency of the narrative and form the causal links between memory events. A narrative is considered to be inconsistent or incomplete if one or more events are missing in the discourse, or if one or more causal links are violated. In other words, the story is incomplete if the object state changes in between two consecutive events. For example, given two consecutive events \(E_1\) and \(E_2\), at the end of event \(E_1\) if character \textit{AgentA} is at location \(L_1\) and, before the start of \(E_2\), he is at \(L_2\), then an inconsistency in the narrative arises between \(E_1\) and \(E_2\). The proposed algorithm...
Algorithm 4: Method to calculate the possible start state of the narrative

```plaintext
CalculateStartState()

foreach m\textsubscript{start} ∈ M\textsubscript{start} do
  if \(!m\textsubscript{start} \rightarrow IsEstimated()\) then
    Obj\textsubscript{1} = m\textsubscript{start} → actor\textsubscript{1}
  if Obj\textsubscript{1} \notin FirstOcc then
    FirstOcc[Obj\textsubscript{1}] = m\textsubscript{start}
  else if m\textsubscript{start} \rightarrow startTime < FirstOcc[Obj\textsubscript{1}] → startTime then
    Obj\textsubscript{2} = m\textsubscript{start} → actor\textsubscript{2}
    if Obj\textsubscript{2} \notin FirstOcc then
      FirstOcc[Obj\textsubscript{2}] = m\textsubscript{start}
    else if m\textsubscript{start} \rightarrow startTime < FirstOcc[Obj\textsubscript{2}] → startTime then
      FirstOcc[Obj\textsubscript{2}] = m\textsubscript{start}

/* Remove the object if it appeared in an end event before it appeared in a start event */

foreach m\textsubscript{end} ∈ M\textsubscript{end} do
  if \(!m\textsubscript{end} \rightarrow IsEstimated()\) then
    Obj\textsubscript{1} = m\textsubscript{end} → actor\textsubscript{1}
    if (Obj\textsubscript{1} ∈ FirstOcc) && (m\textsubscript{end} → endTime < FirstOcc[Obj\textsubscript{1}] → startTime) then
      Remove FirstOcc[Obj\textsubscript{1}] from FirstOcc
      Obj\textsubscript{2} = m\textsubscript{end} → actor\textsubscript{2}
      if (Obj\textsubscript{2} ∈ FirstOcc) && (m\textsubscript{end} → endTime < FirstOcc[Obj\textsubscript{2}] → startTime) then
        Remove FirstOcc[Obj\textsubscript{2}] from FirstOcc

forall Obj ∈ FirstOcc → Keys do
  \(\Phi_{\text{start}} = \Phi_{\text{start}} \cup \{\text{FirstOcc[Obj]} \rightarrow \text{ObjectState}\}\)
```

verifies the consistency by updating the narratives state (\(\Omega_{\text{open}}\)) by adding effects (\(\Omega\)) and preconditions (\(\Phi\)) of affordances. These are derived from the end and start events respectively in chronological fashion - see Algorithm 7.

Any affordance can be identified from a memory using its memoryName id. A (start → end) memory pair must have the same affordance. During every behavior execution, we must verify the start event \(\Omega_{\text{open}}\) with the required preconditions (line 8 of Algorithm 7). Finally, the \(\Omega_{\text{open}}\) of the overall event will be updated with the effects of the identified affordance - line 11 of Algorithm 7. In the algorithm, \(\Omega_{\text{open}}\) is defined as the set of (action → effect) pairs (\(\{< a, \phi >\}\)) where \(a\) is the last executed action with \(\phi\) as an effect. \(\Omega_{\text{open}}\) is initialized with the effects of the start affordance (\(a_{\text{start}}\)).

**Updating effects** (Algorithm 5): While updating the \(\Omega_{\text{open}}\) with effects of an affordance:

- If the condition already exists in \(\Omega_{\text{open}}\), replace the action with the new action - lines 11-14.
- For every contradicting \(< a, \phi >\) in \(\Omega_{\text{open}}\), remove it and replace it with the new action, effect entry - lines 6-10.
- If the condition is not present in \(\Omega_{\text{open}}\), create a new entry - line 16.

After verifying the affordance preconditions, an updated narrative state (\(\Omega_{\text{open}}\)) is computed and causal links can be generated in order to identify potential narrative inconsistencies.

**Verifying preconditions** (Algorithm 6): The preconditions of affordances are verified with the current narrative state when a start memory is being processed.
• If a precondition $\Phi$ is present in the $\Omega_{open}$, then a causal link is formed with that affordance over $\Phi$ - line 7.

• If $a < a, \phi >$ contradicts with the $\Phi$, then it means there is an inconsistency in the story. Hence an inconsistent link is added - line 11.

• If a $\Phi$ is not present in the $\Omega_{open}$, it means that a part of the narrative that gives rise to that condition is missing. So, an inconsistency is added (line 15).

Finally if any inconsistencies ($#IL$) are found, then the narrative is considered to be inconsistent. We generated agent memories for a sample narrative show in the supplementary video. As shown in Figure 6a, start state is derived and examples of consistent and inconsistent narratives are shown in Figures 6b & 6c respectively.

**Figure 6:** An example showing the process for resolving inconsistencies

---

**Algorithm 5:** Method to add effects to Narrative state

1. \textbf{AddEffectsToNarrativeState($m_{end}$)}
2. conditionExists = false
3. $a_e = m_{end} \rightarrow \text{GetAffordance()}$
4. \textbf{foreach $\Omega \in a_e \rightarrow \{\Omega\}$ do}
5. \textbf{foreach $<a, \phi> \in \Omega_{open}$ do}
6. \textbf{if $\Omega$ contradicts $\phi$ then}
7. $\phi = \Omega$
8. $a = a_e$
9. conditionExists = true
10. \textbf{break}
11. \textbf{else if $\Omega == \phi$ then}
12. $a = a_e$
13. conditionExists = true
14. \textbf{break}
15. \textbf{if !conditionExists then}
16. $\Omega_{open} = \Omega_{open} \cup <a_e, \Omega>$
17. \textbf{else}
18. conditionExists = false

---

**Algorithm 6:** Method to check the consistency current narrative state($\Omega_{open}$) for a start memory and add causal links

1. \textbf{UpdateNSForStartMemory($m_{start}$)}
2. conditionExists = false
3. $a_s = m_{start} \rightarrow \text{GetAffordance()}$
4. \textbf{foreach $\phi \in a_s \rightarrow \{\phi\}$ do}
5. \textbf{foreach $<a, \phi> \in \Omega_{open}$ do}
6. \textbf{if $\phi == \phi$ then}
7. $L = L \cup <a, \phi, a_s>$
8. conditionExists = true
9. \textbf{break}
10. \textbf{else if $\Phi$ contradicts $\phi$ then}
11. $IL = IL \cup <a, \Phi, a_s>$
12. $\phi = \Phi$
13. conditionExists = true
14. \textbf{break}
15. \textbf{if !conditionExists then}
16. $IL = IL \cup <a_{start}, \Phi, a_s>$
17. \textbf{else}
18. conditionExists = false

---

6.3 Narrative Extrapolation

In this phase, each inconsistency generated by Algorithm 7 in the Narrative Validation process is fed to a planner to generate plausible predictions of the story outcome. We use a modified version of a partial planner for story completion.

The partial planner does an exhaustive search all possible plans. We populate an initial partial plan ($\pi_i$) by generating ordering constraints based on the chronological ordering of memories, i.e. if an event of $m_1$ happens before the start of $m_2$, then the $m_1 < m_2$ is created. We deduce that an action $a_1$ happens before $a_2$, by
Algorithm 7: Method to check the narrative consistency and form causal links

1 CheckNarrativeConsistency()
2   i = 0, j = 0
3   isNarrativeConsistent = true
4 CalculateStartState()
5   $a_{start} = \text{CreateAffordaceWithEffects}(\Phi_{start})$
6   do
7       if $M_{start}[i] \to start\text{Time} < M_{end}[j] \to end\text{Time}$ then
8           UpdateNSForStartMemory($M_{start}[i]$)
9           i = i + 1
10      else
11         AddEffectsToNarrativeState($M_{end}[j]$)
12         j = j + 1
13   while (i < $\#M_{start}$ && j < $\#M_{end}$)
14   if $\#IL > 0$ then
15       isNarrativeConsistent = false

simply determining if $a_2$ starts after $a_1$ finishes.

For each inconsistency in $IL$, a set of possible partial plans ($\Pi$) is generated by executing the partialPlanner in plan space $\pi$ - line 6 of Algorithm 8. Then the user selects the plans (Narrative discourse) and the $\pi$s is updated accordingly - see line 8 of Algorithm 8. This procedure is repeated until all the inconsistencies are resolved. If the algorithm can’t resolve an inconsistency, Planner in line 6 returns a null plan ($\emptyset$), thereby terminating the process (line 10 of Algorithm 8) Figure 7 shows an example where partial plans are generated for inconsistencies.

7 Conclusion

We presented the techniques to reconstruct narratives from multiple agent memories, the data structures used, and the mechanics utilized for solving potential conflicts in the process of narratives reconstruction. We demonstrated the potential of our approach in a pre-scripted virtual environment where the partial memories of different subsets of actors in the story can be merged, to generate a plausible, complete reconstruction of the entire story. These preliminary demonstrations are for small-scale stories involving few participants, but these theoretical concepts are scalable, and can be applied to larger-scale narratives in even more complex settings.

There are many avenues of future exploration. The agent model can be further developed for more complex scenarios and should be checked for its robustness to see if it can be used in scalable real world scenarios such as surveillance and crime reconstruction scenes. We would also like to explore the integration of stochastic memory representations for probabilistic recounts of memories with missing, noisy perceptual signals.

Algorithm 8: Algorithm to generate possible complete narratives

1 begin
2   GenerateOrderingConstraints()
3       $\pi_i = \langle A, L, O \rangle$
4       $\pi = \pi_i$
5       do
6           $\Pi = \text{Planner}(IL \to \text{pop}(\cdot), \pi)$
7           if $\Pi \neq \emptyset$ then
8               $\pi = \text{SelectFromUI}(\Pi)$
9           else
10              Show("No narratives found")
11       return
12   while $\#IL > 0$
References


[16] Lotzi Boloni. Xapagy cognitive architecture.