Technical Report:

Multi-Agent Narrative System Implementation

Hackfest x Datathon Generative AI Module

Generated: February 15, 2026

Executive Summary

This document describes the implementation of a Multi-Agent Narrative System for the Hackfest x Datathon Generative AI module. The system successfully implements all three mandatory components:

- **Character Memory:** Individual memory buffers for each agent
- **Action System:** Non-verbal behaviors that modify story state
- **Reasoning Layer:** Decision-making logic for Talk vs. Act

The system ensures compliance with all constraints including 25-turn limit and minimum 5 actions.

1. System Architecture

1.1 Overview

The system is built using LangGraph as the orchestration framework, with Google Gemini ("gemma-3-27b-it") as the language model. The architecture follows a modular design with clear separation of concerns.

1.2 Core Components

DirectorAgent

- Selects next speaker based on narrative flow
- Monitors action count (minimum 5 required)
- Injects narrations to advance plot
- Decides when story should conclude
- Prevents premature ending if action quota not met

CharacterAgent

- Maintains individual memory buffer
- Implements reasoning layer
- Decides between talking and acting
- Generates JSON-formatted responses
- Updates own and others' memories

StoryStateManager

- Centralized state management
- Tracks dialogue and action history
- Maintains world state
- Provides context to agents
- Enforces turn limits

2. Character Memory Implementation

2.1 Memory Structure

Each character has a CharacterMemory object containing:

- **observations:** What they've seen/heard
- **inventory:** Items they possess
- **goals:** Current objectives
- **perceptions:** Views of other characters
- **important_facts:** Key information

2.2 Memory Updates

Memory is automatically updated in three scenarios:

Self-dialogue: When a character speaks, they remember what they said

Self-action: When a character acts, they remember the action

Observation: When another character acts, all present characters observe it

2.3 Memory-Aware Context

Characters receive context including their inventory and goals, recent observations (last 5), important facts they've learned, recent dialogue (last 10 turns), and recent actions (last 5). This ensures consistent behavior based on what they know.

3. Action System Implementation

3.1 Action Types

Nine action types are implemented:

Action	Purpose	Example
LEAVE	Exit location	Leave the accident scene
SEARCH	Find objects/information	Search car for documents
GIVE	Transfer items	Give money to driver
TAKE	Acquire items	Take officer's business card
CALL	Phone communication	Call family

SHOW	Display objects	Show driver's license
THREATEN	Intimidation	Threaten to report
GESTURE	Non-verbal communication	Wave hand dismissively
MOVE	Position change	Move closer to argue

3.2 Action Effects

Actions modify the world state. Examples include:

- **LEAVE:** Updates characters_present list
- **CALL:** Sets {character}_on_call flag
- **GIVE:** Updates inventory and sets transfer flag
- **THREATEN:** Increases tension level

3.3 Action Tracking

- Minimum 5 actions required per story
- Director monitors action count
- Warning issued if story tries to end with <5 actions
- All actions logged to action_history

4. Reasoning Layer Implementation

4.1 Decision Process

Characters follow a structured reasoning process:

Analyze situation: Review goals, context, recent events

Evaluate options: Consider talk vs. act

Make decision: Choose action type

Execute: Generate response

4.2 Response Format

All character responses use JSON format. Example:

```
{ "reasoning": "I need to leave before police write ticket", "action_type": "LEAVE",
"action_target": "accident scene", "response": "Saleem quickly gets back in his
rickshaw..." }
```

This provides transparency into decision-making, structured data for processing, and a reasoning audit trail.

4.3 Talk vs. Act Logic

Characters decide to act when:

- Dialogue alone is insufficient
- Goals require physical action
- Stuck in dialogue loop
- Need to change situation dramatically

5. Narrative Flow

5.1 Turn Structure

Each turn follows this flow:

```
1. Director Selects Next Speaker
→ Considers recent dialogue/actions
→ Generates narration
→ Chooses character
2. Character Responds
→ Receives context (memory + recent events)
→ Reasons about goals
→ Decides: Talk or Act
→ Generates response
→ Updates memory
3. Check Conclusion
→ Verify turn count
→ Verify action count (≥5)
→ Check if conflict resolved
→ Continue or end
```

5.2 Constraints Enforcement

Constraint	Implementation
Max 25 turns	Hard limit in graph
Min 10 turns	Director checks before conclusion
Min 5 actions	Director override prevents early end

Max context (4000 tok)	Take last N items from history
Max output (2000 tok)	Set in LLM configuration

6. Output Files

6.1 story_output.json

Contains complete narrative trace with events array including narrations, dialogues, and actions. Each event includes type, speaker/actor, content, and turn number.

6.2 prompts_log.json

Complete LLM interaction audit trail with timestamp, agent role, full prompt sent to LLM, and raw LLM response for every interaction.

7. Technical Decisions

7.1 Why LangGraph?

- State machine abstraction for narrative flow
- Built-in state management
- Easy to add conditional logic
- Good for multi-agent orchestration

7.2 Why JSON Responses?

- Structured data easier to parse
- Forces explicit reasoning
- Enables validation
- Provides flexibility for talk/act decision

7.3 Memory Design Choices

- Recent observations (last 5): Prevents context overflow
- Separate inventories: Character autonomy
- Shared observations: Realistic information flow
- Goal tracking: Ensures consistent motivation

7.4 Action Effect System

- Simple key-value effects dictionary

- Extensible for new action types
- Immediate world state update
- Visible to all agents next turn

8. Compliance with Requirements

Requirement	Implementation	Status
Character Memory	CharacterMemory class with 5 tracked fields	✓
Action System	9 action types with state effects	✓
Reasoning Layer	JSON response with reasoning field	✓
25 turn limit	Hard limit in config + graph enforcement	✓
Min 5 actions	Director checks + override on early end	✓
story_output.json	All events with type/speaker/content	✓
prompts_log.json	Complete timestamp/agent/prompt/response	✓
Use seed story	Loads from examples/rickshaw_accident/	✓
Character consistency	Memory + persona in every prompt	✓
Free/Open model	Google Gemini free tier	✓

9. Code Quality

9.1 Modularity

- Separate files for agents, schemas, prompts
- Clear separation of concerns
- Easy to extend with new action types
- Configuration externalized

9.2 Type Safety

- Pydantic models for all data structures
- Type hints throughout
- Validation on model creation
- Clear interfaces

9.3 Error Handling

- Try-except blocks for LLM calls
- Fallback for JSON parsing failures
- Default values for missing data
- Informative error messages

10. Future Enhancements

While the current implementation meets all requirements, potential enhancements include:

- **Advanced Memory:** Semantic search over observations
- **Complex Actions:** Multi-step action sequences
- **Dynamic Goals:** Goals that change based on events
- **Emotion Modeling:** Emotional state influencing decisions
- **Action Preconditions:** Validate actions are possible
- **Multi-location:** Track character movement between locations

11. Conclusion

This implementation successfully delivers a complete Multi-Agent Narrative System with:

✓ Individual character memory
✓ Reasoning-driven decision making
✓ Rich action system (9 types)
✓ Minimum 5 actions enforced
✓ 25 turn limit
✓ Complete audit logs
✓ Modular, extensible architecture

The system demonstrates sophisticated agentic behavior while maintaining narrative coherence and meeting all technical constraints.