# The General Mediation Engine

**Justus Robertson** and **R. Michael Young**
Liquid Narrative Group
Department of Computer Science
North Carolina State University
Raleigh, NC 27695
jjrobert@ncsu.edu, young@csc.ncsu.edu

## Introduction

Mediation is a plan-based interactive narrative generation process first used with the Mimesis system (Young et al. 2004). Mediation creates a cascading policy of contingency plans meant to direct NPC actions in an interactive game world. The system allows users to interact as a character in the game and conforms the series of events that play out to the choices the player makes. No matter what path the player takes, mediation guides events towards a set of desired goal states specified by an author.

Mediation takes as input a domain and problem PDDL (McDermott et al. 1998) file and outputs a policy of plans that corresponds to a branching story graph (Riedl and Young 2006). The General Mediation Engine (GME) is an implementation of mediation meant to be planner-independent. Instead of being coupled with a single planner, the system takes as input a domain and problem PDDL file along with a linear narrative generator. This generator can be any process that takes as input a domain and problem PDDL file and outputs a series of plot events as valid instantiations of operators from the domain with objects from the problem.

One shift that results from decoupling GME from a specific planner is that it now fully models the world representation, player action updates, and system updates in its search space. This representation allows GME's search to act as a unified game engine and experience manager that creates experiences based on a PDDL domain and problem file. GME simply needs a visualization layer to expose its underlying representation to the user. The power of this system is that the experience manager has a direct hook into the game world that it can use to manipulate world states and simulate possible worlds independently of a visualization.

As proof of concept, GME is equipped with a text-based interface similar to systems like MUDs (Curtis and others 1993) and text-based adventure games (e.g. Infocom games like *Zork*). This interface formats state predicates from mediation's current graph node and presents them to the user. It listens for feedback that corresponds to one of the actions available to the player character. Finally, it updates its state based on user feedback along with system and NPC responses computed for each action.

```
(:action take-cat
 :parameters (?taker ?cat ?location)
 :precondition
        (and (character ?taker)
        (alive ?taker)
        (at ?taker ?location)
        (cat ?cat) (at ?cat ?location)
        (location ?location))
 :effect
        (and (not (at ?cat ?location))
        (has ?taker ?cat)))
```

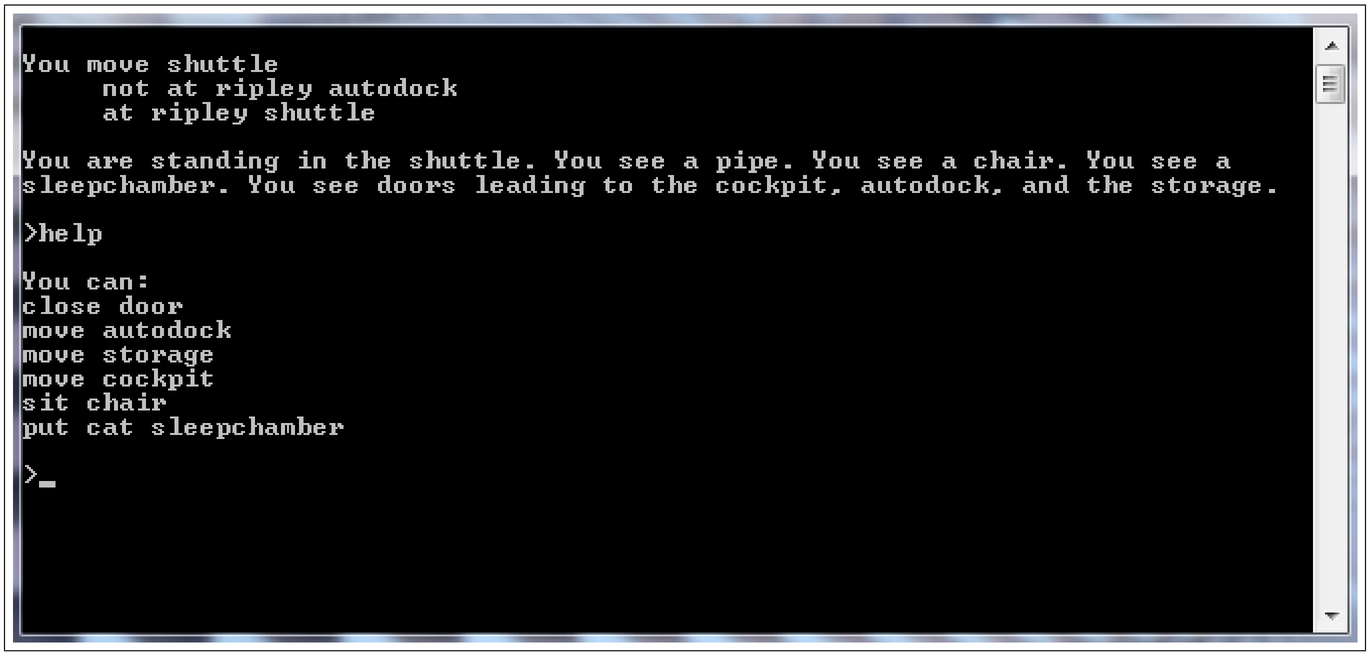
PDDL Excerpt 1: Take-Cat action in the *Alien* domain file.

## System Input

The input to GME is a PDDL 3.1 domain and problem file. Domain files specify a library of action operators that can be instantiated in the game world and the problem file specifies an initial and goal state of the game world. These files are stored in a *Benchmarks* folder in the GME directory. The player can choose from among these game domains at the top level of the GME system. Each domain and problem file combination contains a unique game experience. Several domain and problem files come packaged with GME, including scenes modeled from *The Evil Dead* and *Alien*. New domain and problem file combinations can be created and used with GME by creating a new directory with a unique name inside the Benchmarks folder.

The system currently supports domains with conjunctive STRIPS operators and conditional effects. An example conjunctive STRIPS operator from the *Alien* domain is given in PDDL Excerpt 1. Conditional effects are axioms given in the effect list of an operator that specify optional circumstances that trigger additional predicates to become true.

Once a problem and domain are chosen by the player the system parses the files and initializes its game tree representation. It reads in the initial world state from the problem file and uses it as the root state node of the game tree. It reads in the possible actions from the domain file and caches all possible instantiations of user-performed actions for later use. The system then queries its planner using the PDDL files to find a series of actions it wishes to play out in the game world.



```
You move shuttle
     not at ripley autodock
     at ripley shuttle

You are standing in the shuttle. You see a pipe. You see a chair. You see a
sleepchamber. You see doors leading to the cockpit, autodock, and the storage.

>help

You can:
close door
move autodock
move storage
move cockpit
sit chair
put cat sleepchamber

>_
```

Figure 1: A possible state in the *Alien* domain.

## Game Tree Expansion

Once the current state information is prepared, the system begins expanding its game tree representation. It checks to see what cached player actions have their preconditions fulfilled by the initial state. For each valid player action the system expands the graph by computing a system response, updating the state representation, and creating a new game tree node. This graph expansion process is done on-line in conjunction with a set of knowledge axioms and a text interface that expose state information and possible player actions to the user. The system expands the graph, takes NPC action, and updates state information based on what actions the player takes in the interface.

The system currently uses the Fast Downward (Helmert 2006) planner to construct narrative trajectories, but with minor changes can support any planner that takes PDDL 3.1 input through a command line call and outputs plans as a series of fully ground operators from the input domain.

## Interface

The system exposes state information and receives player actions through a text-based interface. The system is equipped with interchangeable observation axioms that filter the full world state according to what the player can see. It also has natural language formatting rules that output observed state predicates in English sentences instead of the underlying logical formalism. The interface accepts commands from users that correspond to enabled PDDL actions from the domain. The system supports a shorthand for specifying actions by typing the action name and a number of trailing variables that correspond to '-' symbols in the action title. So, instead of typing *(move-location player outside house)* a user can type *move outside* to specify the action.

## Demo

An early demo of GME with installation and usage instructions can be found at:

http://go.ncsu.edu/gme

## Conclusion

The General Mediation Engine is a game engine and experience manager that uses PDDL domain and problem files to generate gameplay. This system is interesting because it provides an experience manager direct access to world states and actions that can be manipulated independently of the visual experience. The system currently uses a text-based interface to expose its internal representation to the user. This visual layer may later be substituted for more robust 2D or 3D graphical models.

## References

Curtis, P., et al. 1993. LambdaMOO Programmers Manual. *Xerox Parc*.

Helmert, M. 2006. The Fast Downward Planning System. *Journal of Artificial Intelligence Research* 26:191–246.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. *PDDL - The Planning Domain Definition Language*.

Riedl, M. O., and Young, R. M. 2006. From Linear Story Generation to Branching Story Graphs. *Computer Graphics and Applications* 26(3):23–31.

Young, R. M.; Riedl, M. O.; Branly, M.; Jhala, A.; Martin, R. J.; and Saretto, C. J. 2004. An Architecture for Integrating Plan-Based Behavior Generation with Interactive Game Environments. *Journal of Game Development* 1(1):51–70.