

Kapil Somani

Action Generator

Project Report

CSC 582 - Computer Models of Interactive Narrative

North Carolina State University, Raleigh

`kmsomani@ncsu.edu`

1 Project Description

The project aims to generate stories based on given set of rules. We provide the a story with a set of entities in the story. We also state a set of true/false statements in the world at the beginning. Along with it, we present a set of valid actions with pre conditions and post conditions. The program iteratively checks for each combination of values for each variable and check for consitions which are supposed to be true or false in the world. The values satisfying the pre conditions are then presented to user, from which he can choose a desired action. The story keeps on building up until the user wishes to finish the story.

The primary purpose of this implementation is to enable user to generate different stories on same set of rules/conditions. The program essentially checks the pre-conditions specified to perform given rule. If the all the preconditions are valid, the user can take that action and appropriate changes are made in the world state. Successive actions depend on the latest world state present in the system.

The author has to feed the building blocks for his story. Entities involved in the story, starting state and list of possible actions in proper format. The implementation will provide user with all the actions he can take, given the pre-requisites are satisified. Changes are made to world state based on post condition stated along with action description.

The program also keeps track of states it has seen so far. So if the story moves into world state which was previously seen by the system. If encounter a **“repeated state”**, the story will notify the narrator that he has entered into one of the previously encountered state. This in a way, may help narrator to repeat any possible loops in the story.

2 Design Documents

The key lies in representation. The world state is represented in the form of semantic network. Semantic network is one the common knowledge representation technique used in prolog like systems. Semantic networks are directed or undirected graph consisting of vertices representing entities and edges representing relation between entities.

Breaking down actions into simple set of pre conditions and post conditions makes it easy to feed into the system and make changes in the system. Semantic representation involves use of small tuples representing the relations between entities.

Figure[1] shows the structure used to represent story.

Story	entities	<entity1> <entity2>	[list of values] [list of values]
	actions	do	[action name]
		parameters	[list of variables]
		pre_conditions	[set of pre-conditions]
		post_conditions	[set of post-conditions]
		.	
		.	
		.	

Figure 1: Structure to represent story

Given implementation describes the world, actions from a very famous puzzle involving saints and demons. We have saints and demons standing on one bank of river bank, our aim is to safely send all the saints to other side of river bank. However, the boat can carry almost two passengers at one time. Also, if the number of demons exceed the number of saint on given bank, the demons will eat the saints and you lose the game.

With these story, we build our game with entities as saints: S1, S2 and S3; and demons D1, D2 and D3. We have actions ROW-LEFT-ALONE, ROW-RIGHT-ALONE, ROW-LEFT, ROW-RIGHT, which are self explanatory.

3 Demo

To execute the attached code, please make sure that Python 2.7.3 or higher installed in the system. The code has 3 major classes, all which have appropriate comments at required places. A brief description about each class is as follows:

1. Application.py

This class contains the main method for the application. It creates an instance of Planner object and makes call to various functions which will list possible actions based on current world state. The user will be prompted to select one among multiple options presented to him and decide the course of the story.

2. Story.py

This class stores the layout for the story. It stores multiple story boards and user can select one among them to design his own story. Each Story object has following attributes.

- (a) *title*: stores the title of the story.

- (b) *edge*: Stores set of valid consitions in the world. Edge data consists of tuple with 3 items, describing the relation between two entities. Edge data is reprinted in following format `<entity1, relationship, entity2>`
- (c) *entities*: dictionary of participating entities in the story. The variables formulated in the pre-conditions and post-conditions take values from items listed in entities.
- (d) *actions*: Stores list of action, which can be performed in the given story. Each action must has following attributes:
 - i. *do* : descriptive name for the action.
 - ii. *parameters* : list of variables in action signature. the values of parameters are decided by comparing them with world states
 - iii. *pre-conditions* : set of conditions which should be valid in world to perform given action.
 - iv. *post-conditions* : set of conditions which hold true in world after the given action is performed.

3. Planner.py

This class is the logical crux of the application. It checks the all the actions with all possible values for the variables, and decide which pre-conditions hold true for current state. Actions will all pre-conditions satisfied are presented to user and user selects one of the action. The action is performed and appropriate changes are made in the world. The class also checks if the current state is same as one of previously observed states, if found, it notifies author, which will help him to eliminate loops in stories if possible.

Besides we also have 2 helper classes

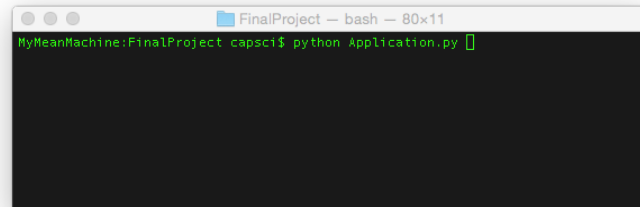
1. HashTable.py

Implements hash table data structure to check for duplicate actions generated during execution. Any duplicate actions are removed and not presented to user, to remove redundancy of actions presented to user.

2. Printer.py

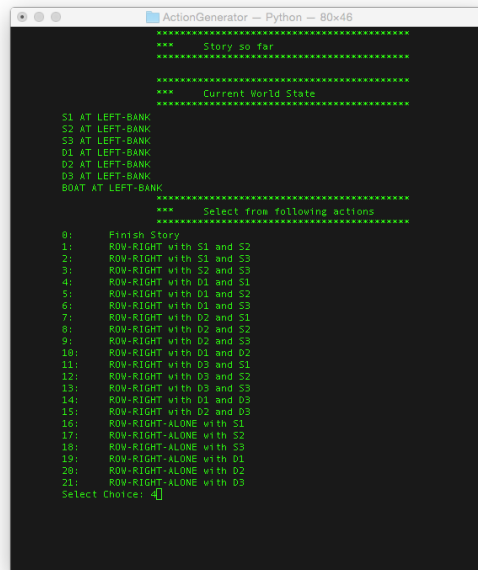
Handles print formatting for printing actions and world states and actions. NLP can be incorporated to make more sophisticated statements.

Following are some snapshots from the code in execution. Please make sure the terminal window has enough height to accomodate all the printed text:



```
FinalProject — bash — 80x11
MyMeanMachine:FinalProject capsci$ python Application.py
```

Figure 2: Invoking executable



```
ActionGenerator — Python — 80x46
*** Story so far
*** Current World State
S1 AT LEFT-BANK
S2 AT LEFT-BANK
S3 AT LEFT-BANK
D1 AT LEFT-BANK
D2 AT LEFT-BANK
D3 AT LEFT-BANK
BOAT AT LEFT-BANK
*** Select from following actions
0: Finish Story
1: ROW-RIGHT with S1 and S2
2: ROW-RIGHT with S1 and S3
3: ROW-RIGHT with S2 and S3
4: ROW-RIGHT with D1 and S1
5: ROW-RIGHT with D1 and S2
6: ROW-RIGHT with D1 and S3
7: ROW-RIGHT with D2 and S1
8: ROW-RIGHT with D2 and S2
9: ROW-RIGHT with D2 and S3
10: ROW-RIGHT with D1 and D2
11: ROW-RIGHT with D3 and S1
12: ROW-RIGHT with D3 and S2
13: ROW-RIGHT with D3 and S3
14: ROW-RIGHT with D1 and D3
15: ROW-RIGHT with D2 and D3
16: ROW-RIGHT-ALONE with S1
17: ROW-RIGHT-ALONE with S2
18: ROW-RIGHT-ALONE with S3
19: ROW-RIGHT-ALONE with D1
20: ROW-RIGHT-ALONE with D2
21: ROW-RIGHT-ALONE with D3
Select Choice: 4
```

Figure 3: Choose action for listed actions

```
ActionGenerator - Python - 80x46
*** Story so far ***
D1 and S1 sat in the boat and roved to right bank
*** Current World State ***
S2 AT LEFT-BANK
S3 AT LEFT-BANK
D2 AT LEFT-BANK
D3 AT LEFT-BANK
D1 AT RIGHT-BANK
S1 AT RIGHT-BANK
BOAT AT RIGHT-BANK
*** Select from following actions ***
0: Finish Story
1: ROW-LEFT with D1 and S1
2: ROW-LEFT-ALONE with S1
3: ROW-LEFT-ALONE with D1
Select Choice: 1
```

Created by Paint X

Figure 4: Choose next action for listed actions, the story is in formation

```
ActionGenerator - Python - 80x46
*** Story so far ***
D1 and S1 sat in the boat and roved to right bank. D1 and S1 sat in the
boat and roved to left bank
*** Current World State REPEATED ***
S2 AT LEFT-BANK
S3 AT LEFT-BANK
D2 AT LEFT-BANK
D3 AT LEFT-BANK
D1 AT LEFT-BANK
S1 AT LEFT-BANK
BOAT AT LEFT-BANK
*** Select from following actions ***
0: Finish Story
1: ROW-RIGHT with S1 and S2
2: ROW-RIGHT with S1 and S3
3: ROW-RIGHT with S2 and S3
4: ROW-RIGHT with D1 and S1
5: ROW-RIGHT with D1 and S2
6: ROW-RIGHT with D1 and S3
7: ROW-RIGHT with D2 and S1
8: ROW-RIGHT with D2 and S2
9: ROW-RIGHT with D2 and S3
10: ROW-RIGHT with D1 and D2
11: ROW-RIGHT with D3 and S1
12: ROW-RIGHT with D3 and S2
13: ROW-RIGHT with D3 and S3
14: ROW-RIGHT with D1 and D3
15: ROW-RIGHT with D2 and D3
16: ROW-RIGHT-ALONE with S1
17: ROW-RIGHT-ALONE with S2
18: ROW-RIGHT-ALONE with S3
19: ROW-RIGHT-ALONE with D1
20: ROW-RIGHT-ALONE with D2
21: ROW-RIGHT-ALONE with D3
Select Choice: 
```

Created by Paint X

Figure 5: Notified when you enter one of previously encountered state

```

ActionGenerator -- bash -- 80x48
=====
*** Story so far
=====
S1 and S2 sat in the boat and rowed to right bank.
=====
*** Current World State
=====
S3 AT LEFT-BANK
D1 AT LEFT-BANK
D2 AT LEFT-BANK
D3 AT LEFT-BANK
S1 AT RIGHT-BANK
S2 AT RIGHT-BANK
BOAT AT RIGHT-BANK
=====
*** Select from following actions
=====
0: Finish Story
1: ROW-LEFT with S1 and S2
2: ROW-LEFT-ALONE with S1
3: ROW-LEFT-ALONE with S2
=====
Demons ate the saints
you lost!
MyMeanMachine:ActionGenerator capscl$ 
```

Created by Paint X

Figure 6: If you make a wrong move, the story ends