



WHAT'S IT ALL ABOUT? SERIOUSLY?
WHEN YOU GET RIGHT DOWN TO IT?

Telling Reactive Stories in an Agent-Driven World

Spirit

Telling

Stories in
World



#Goals

- Quirky, emergent occurrences amongst the socially-simulated citizens.
- Occurrences that the system deems the most 'interesting' to be displayed (and narrated through voiceover).
- Lego-like remixing of displayed script lines.
- Select “most interesting” occurrences for an end-of-game summary for the player, complete with screenshots, to form a 'comic book' retrospective.

i.e. a system which can tell interesting stories, in real-time, in response to agent-driven behaviors.





WHAT'S IT ALL ABOUT? SERIOUSLY?
WHEN YOU GET RIGHT DOWN TO IT?

Making implicit narrative explicit.

Two-part, discrete-but-cooperative solutions

- A modified **Goal Oriented Action Planner** for citizen behaviors and decision-making.
- A **rule-based query system** for context-aware dialog and response.



1. Goal Oriented Action Planner (GOAP)



Goal Oriented Action Planner



- Discrete
- Modular
- Easy to iterate - add/remove actions & test.
- Satisfying variation of actions chosen depending on world/agent state (in theory!)

The most traditional GOAPs...

1. Planner has goals and actions.
2. Goal chosen based on dynamic weighting.
3. Regressive search to build sequence of actions which would lead to that goal.
4. Goal is solvable once it has found a final action where all its pre-requisites are already resolved.
5. Plan then reversed; actions carried out one at a time until the goal is reached.



There are always conditions...

- IS_IN_CONVERSATION
 - HAS_JOB_AT_RESTAURANT
 - IS_MANNING_WEAPON
 - IS_RUNNING_AWAY
- Each action has a list of pre-requisites and effects based on registered list of conditions.



How we deviated from traditional GOAPs

- **Dynamic Weighting**
- **Execution Contexts**
- **Futures/Promises**



Dynamic Weighting

- Goal weight as a function of **utility**.
- Once the goals are weighted, each goal is queried to see if it is currently resolvable.
- Actions in chain can dynamically self-evaluate weight.
- A weight of zero or less removes the goal or action from consideration.



Execution Contexts

- Goals and actions exist in certain **contexts**.
- Execution Contexts provide scope for goals/actions.
- Prevents evaluating rules which can't be resolved at current time.
- Easily shape expected behaviour at any particular time.
- ECs work as a stack - unrolls as contexts are popped off the stack.





Future Conditions and Broken Promises

- Futures system allows actions to be considered *without* resolving all the prerequisites themselves.
- i.e. a *promise* to get the pre-reqs resolved by an action in future.
- Actions required to take on unresolved pre-reqs.
- Action only considered for *futures* if it resolves *at least one* of the pre-reqs of current action.
- (Stops the actions from flip-flopping without furthering the goal.)



Challenges



But...

- There are actions such as **find out where nearest pub is** – which an NPC can do by either exploring, or asking someone nearby.
- How do we keep track of what an NPC knows? How do they share knowledge about the world?

2. Rule-based Query System



... for **context-aware** dialog and
response
& story generation system.



Overview

- Defining rules and responses, picked based on some criteria which needs to be met for the rule to execute.
- Inspired by system in Left 4 Dead (2012 AI Summit) ...**but** we can execute *tasks* as well as dialogue.



Overview

- Managing citizen **conversations**, sharing information, and their associated animations.
- **Narrating story events** to do with particular game states.
- **Picking most interesting** game events to display at the end of a play through.

Overview

- Rule-based XML execution environment.
- These rules can be **queried**.
- Each rule has a number of *criteria*, which need to be satisfied.
- Rule criteria tested against query from **most to least specific**.
- When rule *passes*, it triggers Rule Payload
 - **response**, and a **write-back** section.

Parts of the Query System

- Rules have:
 - Concepts, Who, **Criteria**, Rule Payload
- Rule payload has:
 - Writeback, Response Reference
- Response has:
 - Execution Item(s):
 - Say, Dialog, Audio, Emote, Story, HandleData, **Trigger/CancelExecutionContext**, NewQuery

Interconnected with GOAP

```
var response = GetResponseQuery("Ask  
                                ForDirections");  
response.Add("IsAQuestion", true);  
response.Add("AskingAbout" + knowledgeInfo.BuildingType.ToString(), true);  
response.Add("TargetBuildingType", knowledgeInfo.BuildingType.ToString());  
StartQuery(response);
```

Kicked off from a generic
“AskForBuildingLocation”
action which passes along what
type of building the NPC is
interested in.

Easy Authoring of Rules/Responses for Quick Iteration and Accessibility

Example Rules and Responses:

```
<Rule Concept="AskForDirections"
  Name="AskForDirections">
  <Criteria>
    <IsTrue>AskingAboutFood</IsTrue>
  </Criteria>
  <WriteBack>
    <Once Expires="10" />
  </WriteBack>
  <Response>
    AskForDirectionsFood
  </Response>
</Rule>
```

```
<Response Key="AskForDirectionsFood">
  <Phrase>Does {CityName}
    have anywhere I can eat?</Phrase>
  <Phrase>Hey {TheirName},
    what's good to eat here?</Phrase>
  <Phrase>Hi! I'm {MyName}. Where can
    I find some food?</Phrase>
</Say>
<NewQuery>
  <Concept>RespondToDirections</Concept>
  <Target>Nearby50</Target>
</NewQuery>
</Response>
```

Example Rules and Responses:

```
<Rule Concept="RespondToDirections" Name="RespondToDirectionsFoodLocation01">
  <Criteria>
    <IsTrue>AskingAboutFood</IsTrue>
    <HasData Type="Knowledge">
      Food
    </HasData>
  </Criteria>
  <Response>
    RespondToDirectionsFoodLocation
  </Response>
</Rule>
```

```
<Response Key="RespondToDirectionsFoodLocation">
  <Say WaitForItem="true">
    <Phrase>Check out this place!</Phrase>
    <Phrase>Food? Over there!</Phrase>
    <Phrase>Yes! Best food in {CityName}!</Phrase>
  </Say>
  <NewQuery>
    <Concept>GetDirections</Concept>
    <Target>Sender</Target>
    <IncludeData Type="Knowledge">Food
  </IncludeData>
  </NewQuery>
</Response>
```

Example Rules and Responses:

```
<Response Key="GetDirections">  
  <Say WaitForItem="true">  
    <Phrase>Hey, Thanks!</Phrase>  
    <Phrase>Great! I'll check it out!</Phrase>  
    <Phrase>Thanks for the directions!</Phrase>  
    <Phrase>Thanks {TheirName}!</Phrase>  
    <Phrase>Thanks for the directions {TheirName}!</Phrase>  
  </Say>  
  <HandleData Process="Learn">Knowledge</HandleData>  
</Response>
```





THANK YOU!

littleinvasiontales.com

Mitu: @MituK

Alan: @alanhinchcliffe

