Kill the Dragon and Rescue the Princess: Designing a Plan-based Multi-agent Story Generator

Iván M. Laclaustra, José L. Ledesma, Gonzalo Méndez, Pablo Gervás
Facultad de Informática
Universidad Complutense de Madrid
Madrid, Spain
{ilaclaus, josledes, gmendez, pgervas}@ucm.es

Abstract

We describe a prototype of a story generator that uses a multi-agent system and a planner to simulate and generate stories. The objective is to develop a system that is able to produce a wide range of stories by changing its configuration options and the domain knowledge. The resulting prototype is a proof of concept that integrates the simplest pieces that are necessary to generate the stories.

Introduction

When trying to generate stories automatically, it is mandatory to research how actual stories work. That, inevitably, makes you think: "What makes a story interesting?".

While researching for this project, we realized that in a story, most of the times, the most important thing is not WHAT, but HOW things happened. This represents a huge challenge, since it is difficult to simulate things such as time (we must be able to simulate time, so that things are not done immediately), conversations (they have to be fluid, spontaneous), and many more. Similarly, there are some actions that lack interest in themselves, but may have some if combined with others. For example, eating or sleeping, are actions that may not appear in the final story, but may be worthy of attention if the character meets someone while eating. Of course, some of the stories generated will just be sets of facts without any relation or interest, but that is part of the process.

One of the ways we have for generating stories is by simulating them. Then, you just have to run the simulation and see what happens. We achieve this by simulating the stories using autonomous intelligent agents. Each of the agents of the story is going to act as a character, which will act independently from the others, but depending on the story world's state. Then stories are generated by "filming" what these actors do and say. Our main goal for now, is to make a small Dungeons & Dragons story, which has more than one ending.

Related Work

The first story telling system for which there is a record is the Novel Writer system developed by Sheldon Klein (Klein et al. 1973), which created murder stories within the context of a weekend party. It relied on a microsimulation model where the behaviour of individual characters and events were governed by probabilistic rules that progressively changed the state of the simulated world (represented as a semantic network). The flow of the narrative arises from reports on the changing state of the world model. A description of the world in which the story was to take place was provided as input. The particular murderer and victim depended on the character traits specified as input (with an additional random ingredient). The motives arise as a function of the events during the course of the story. The set of rules is highly constraining, and allows for the construction of only one very specific type of story. The world representation allows for reasonably wide modeling of relations between characters. Causality is used by the system to drive the creation of the story but it is not represented explicitly.

TALESPIN (Meehan 1977) is a system which tells stories about the lives of simple woodland creatures. TALESPIN was based on planning: to create a story, a character is given a goal, and then the plan is developed to solve the goal. TALESPIN introduces character goals as triggers for action. Actions are no longer set off directly by satisfaction of their conditions; an initial goal is set, which is decomposed into subgoals and events. TALESPIN introduced the possibility of having more than one problem-solving character in the story (and it introduced separate goal lists for each of them). The validity of a story is established in terms of: existence of a problem, degree of difficulty in solving the problem, and nature or level of problem solved.

Lebowitz's UNIVERSE (Lebowitz 1985) modelled the generation of scripts for a succession of TV soap opera episodes. It aimed at exploring extended story generation, a continuing serial rather than a story with a beginning and an end. It is in a first instance intended as a writer's aid, with additional hopes to later develop it into an autonomous storyteller. The actual story generation process of UNIVERSE uses plan-like units (plot fragments) to generate plot outlines. Plot fragments provide narrative methods that achieve goals, but the goals considered here are not character goals, but author goals. This is intended to allow the system to lead characters into undertaking actions that they would not have chosen to do as independent agents. The system keeps a precedence graph that records how the various pending author goals and plot fragments relate to each other and to events that have been told already. To plan the next stage

of the plot, a goal with no missing preconditions is selected and expanded.

The line of work initiated by TALESPIN, based on modeling the behaviour of characters, has led to a specific branch of storytellers. Characters are implemented as autonomous intelligent agents that can choose their own actions informed by their internal states (including goals and emotions) and their perception of the environment. Narrative is understood to emerge from the interaction of these characters with one another. This guarantees coherent plots, but, as Dehn pointed out, lack of author goals implies they are not necessarily very interesting ones. However, it has been found very useful in the context of virtual environments, where the introduction of such agents injects a measure of narrative to an interactive setting.

The Virtual Storyteller (Theune et al. 2003) introduces a multi-agent approach to story creation where a specific director agent is introduced to look after a plot. Each agent has its own knowledge base (representing what it knows about the world) and rules to govern its behaviour. In particular, the director agent has basic knowledge about plot structure (that it must have a beginning, a middle, and a happy end) and exercises control over agent's actions in one of three ways: environmental (introduce new characters and object), motivational (giving characters specific goals), and proscriptive (disallowing a character's intended action). The director has no prescriptive control (it cannot force characters to perform specific actions). Theune et al. report non-structural rules are contemplated, to measure issues such as surprise and "impressiveness". The Virtual Storyteller includes a specific narrator agent, in charge of translating the system representation of states and events into natural language sentences. The development effort on the narrator seems to have focused on correct generation of pronouns to make the resulting text appear natural.

The story generator

The objective of this work is to develop a story generator that can generate different stories using the same initial information and that, in addition, can be easily modified to generate a wider range of stories.

With these objectives in mind, we have developed a first prototype that works as a proof of concept to test our approach. This prototype has been developed using very simple, unsophisticated components with the aim of substituting them with more complex ones once the feasibility of the solution has been tested.

The generator is structured in four modules, each of them with their corresponding configuration files: a multi-agent system, which contains an agent for each character and a set of managing agents (currently the world agent, the simulation agent and the director agent), a logger (in charge of collecting the events of the story), a planner (what the characters use to know what to do), and the world (contains the map where the characters interact).

The world

The world is basically a map with different locations, connected by paths between them, in order to make the characters move around it. The prototype we have built has a map formed by three locations:

- Castle: Where the king and the princess are.
- Village: Where the knight starts at.
- Cave: Dragon's home.

Since one of our main goals is to make this storyteller easy to configure, we decided to use text files to load the map and the objects present in each location. The map is structured as an XML file that contains a list of locations with pointers to the locations they are connected to, and the objects and characters situated there, so it works as a graph.

The multi-agent system

The multi-agent system is implemented using the JADE (Bellifemine, Caire, and Greenwood 2007) agent platform. This first prototype generates stories with four types of characters:

- Princess: the character around which the story is built up.
- Dragon: its goal is to kidnap the princess and hold her prisoner in his cave.
- King: the father of the princess. When his daughter is kidnapped, his goal is to find a suitable knight and hire him to kill the dragon. If the knight fails, the king looks for another one, until the princess is safe and sound back in her father's castle.
- Knight: He has no goals until the princess is kidnapped. From then on, his goal is to kill the dragon and take her back to her father.

New stories can be created by simply adding more characters of a type, which are specified at the beginning in a configuration file. In addition, the director agent may create them if it fits the objectives of the story. For example, creating more than one knight, when the princess is kidnapped, the king will look for all the knights available, and will hire the one with lowest fees.

Each character works as a finite state machine consisting of one state per behavior type and a "waiting" state where they are when they don't have active goals.

The world agent is in charge of managing the map, so that all the other agents have a consistent view of the world. Every time a character moves to a new location, he has to send a message to the world agent, so the map gets updated.

The simulation agent is in charge of managing the result of the actions that cannot be directly obtained by the planner, such as the result of the battle between the dragon and the knight.

Finally, the director agent is the one in charge of creating all the necessary agents of the story, these being: the world agent, the simulation agent and the characters. It also makes the necessary decisions to keep the story going, such as setting new goals for the characters. Currently, these decisions are hand written in a configuration file, but the purpose is for this agent to be able to generate them dynamically according to certain heuristics or ask the user to suggest what the new goals should be, in order to make the generator more interactive.

Planning

Each character's actions are driven by their own goals, which are used to plan the sequence of actions they have to carry out to achieve these goals. At the beginning we thought of using just one planner to generate the whole story, but soon it was clear that the planning process would be costly, that the number of possible stories would be small and that it would be difficult to obtain valid plans for agents with conflicting interests. Therefore, we decided it would be more suitable to use separate planners for each agent, so that each of them could make their own plans according to their interests and, in case of conflict, they would have to create new plans to achieve their goals.

We decided to use a STRIPS-based planner (Fikes and Nilsson 1971), since it is quite simple and it is a straightforward option to generate simple stories. In addition, we wanted it to work with PDDL (McDermott 1998) so it would be easy to substitute it with a more sophisticated one in the future.

With this choice, adding a new character to the story involves the creation of another class with the character and two PDDL files, one for its actions, and one for its initial state and goals.

We decided to use the JavaFF planner (Coles et al. 2008) because it works with PDDL and it is open source. The planner takes the domain and the problem in PDDL as inputs, and writes the plan (as a list of actions) into an output file. Since it is open source, we were able to modify it, in order to make the planner return a list of actions (the data structure managed by the planner) instead of writing it to a file. By just adding new actions to the character's PDDL file, new stories are generated, as plans may change including these new actions.

At the time of writing this paper, agents make their plans sequentially (one makes its plan and executes it, then the next one), so that they don't interfere with each other's goals while executing their plans. This reduces the richness of the generated stories, but it is still a good solution to test the validity of the proposed solution.

Capturing the events of the story

As we already said, the only important things are not only the events themselves, so we need a way to gather what happens in the story, but also what is "said" and in what context. Namely, we need a log of everything that happens in the simulation, including the actions that are carried out and the messages exchanged between the agents. We have used the log4j library (Gulcu 2003), which allows the user to enable logging at runtime without modifying the application binary. It also allows us to decide what to enter the log (in our case, it would be everything), the layout, what to save in the log (date, action, agent) and more. Everything is configurable via a parameters file, and will be saved as a log file.

This log is what enables us to actually know what has happened in a certain story, what actions were executed and what was said (scilicet, what messages were interchanged between the agents). However, we must keep in mind that not all the exchanged messages are likely to appear in the final story. For example, all characters have to send a message to the world agent when moving, in order to keep the map updated. These messages should not appear in the story, as their goal is to guarantee internal consistency.

Results

We have implemented a simple prototype where all the described components work together to generate simple, short variations of a story (in Spanish) where a dragon kidnaps a princess and her father the king manages to hire a knight who rescues her and takes her back to her father:

El rey Felipe está preparado.
La princesa Laura despierta.
La princesa sale del castillo.
El dragón Draco emprende el vuelo en busca de alguna princesa desprotegida.
La princesa Laura ha sido secuestrada.
El rey intenta pedir rescate para la princesa Laura.
El caballero Rafael entra en escena.
El rey intenta pedir rescate para la princesa Laura.
El caballero Rafael busca al dragón Draco.
El dragón Draco ha muerto en batalla.
La princesa Laura fue liberada.
El rey entrega 50 monedas al caballero Rafael.
La princesa llega al castillo con el caballero Rafael.
La princesa Laura pone fin a su aventura.

As far as we have been able to test, it is easy to modify the world map to add new locations and situate the characters in them, so they have to make longer journeys to achieve their goals. It is also easy to add new characters of existing kinds so, for example, we can add a second dragon that tries to kidnap the princess from the first one's den.

To make further changes, such as adding new types of characters or actions, it is already necessary to modify the source code of the generator, as well as the domain knowledge, but the code is sufficiently well crafted so that these changes can be easily made. We still have not tested how easy it is to generate a story in a different domain, such as a superheroes story, a western or a love story, but as far as we can see now it may be more painstaking than difficult.

As of now, the stories we generate consist of all the events that take place in the simulation, so our current work is focused on the content extraction, so that we can tell just the relevant events in a relevant order.

To transform the generated logs into text we are using the TAP text generator (Gervás 2011) that receives a crafted set of information and transforms it into an ordered set of sentences that replicates the events that took place in the simulation in the form of a story.

Therefore, in a still simple way, we have developed a story generator that, by means of simple modifications, is able to generate a fair amount of different, although related, stories.

Future Work

Some of the goals we had in mind at the beginning of this project could not be achieved, mostly because of time constraints. We describe some of them here, so they can be used as a starting point for future contributions.

One of the first thing that comes to mind is expanding the world. As the characters and world we are using now are very limited, stories generated are just little paragraphs and there are not many variations between different executions of the application. Just by adding new locations and new characters, we will be adding more possibilities to the story to move along, so that we get more possible stories, which become more intricate at the same time.

As we said before, at the moment, the characters in our application work sequentially, for practical reasons. This reduces the possibilities of the stories generated, since it is more difficult for conflicting interests to appear, or for characters to collaborate to achieve a common goal. A good improvement would be to make all the characters work in parallel, so they would make their plans based on the initial state. While executing their plans, the actions of some characters may interfere in the plans and goals of others. There is when re-planning comes in. Re-planning would make characters interact a lot more, making them compete for the resources to achieve their goals.

In addition, we may want to increase the richness of the stories by making the characters more complex. Adding a slight mood to the characters can make possible stories increase significantly, as the same character may have different behaviors with different moods. Another possibility would be to add feelings and even personality traits.

A lot of richness can also be added via expanding the map. Having a sub-map inside every location would make much more complex plans. Each location can contain different objects, usable and decorative, so the characters can interact with them. For example, you could have a dragon which cannot be killed without a magical sword, so the knight has to find the hidden key to get it.

A much more difficult (and interesting) goal is to make the theme of the story configurable. The idea is to create a configuration file where you can state the theme of the story. That would make everything more difficult, since you can't work with the characters directly. The agents can adopt the role of "actors", instead of characters. With that, there would be a "main character", an "antagonist", a "damsel in distress", and various "secondary actors" in each story. By doing this, you could include in the theme configuration file the names of the characters, their mood (if any), their role, how their actions work (the action "attack" for a knight would make him use his sword, while for a policeman, it would make him use his gun), and have a PDDL file of actions for each role.

Another improvement would be to make the user take the role of a character, so his decisions affect the final result of the story. At first, it could work as in conversational adventures (Montfort 2004), so the user tells the system what actions to carry out. After that, the system would work just as it usually does.

Finally, another option is to give the characters the possibility of making up the details of the story. For example, in our story, the knight could pretend he has a magical weapon to kill the dragon. This endows the stories generated with a whole new level of richness, because new facts are created on the fly.

Acknowledgments*

This paper has been partially supported by the projects WHIM 611560 and PROSECCO 600653 funded by the European Commission, Framework Program 7, the ICT theme, and the Future Emerging Technologies FET program.

References

Bellifemine, F. L.; Caire, G.; and Greenwood, D. 2007. *Developing multi-agent systems with JADE*. Wiley series in agent technology. Wiley.

Coles, A.; Fox, M.; Long, D.; and Smith, A. 2008. Teaching forward-chaining planning with javaff. In *Colloquium on AI Education, Twenty-Third AAAI Conference on Artificial Intelligence*.

Fikes, R. E., and Nilsson, N. J. 1971. Strips: A new approach to the application of theorem proving to problem solving. In *Proceedings of the 2Nd International Joint Conference on Artificial Intelligence*, IJCAI'71, 608–620. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Gervás, P. 2011. UCM submission to the surface realization challenge. In *Surface Realization Challenge. Challenges 2011 Session at 13th European Workshop on Natural Language Generation (ENLG 2011)*.

Gulcu, C. 2003. *The Complete Log4j Manual*. QOS.ch.

Klein, S.; Aeschliman, J. F.; Balsiger, D.; Converse, S. L.; Court, C.; Foster, M.; Lao, R.; Oakley, J. D.; and Smith, J. 1973. Automatic novel writing: A status report. Technical Report 186, Computer Science Department, The University of Wisconsin, Madison, Wisconsin.

Lebowitz, M. 1985. Story-telling as planning and learning. *Poetics* 14:483–502.

McDermott, D. 1998. PDDL - the planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

Meehan, J. R. 1977. TALE-SPIN, an interactive program that writes stories. In *In Proceedings of the Fifth International Joint Conference on Artificial Intelligence*, 91–98.

Montfort, N. 2004. *Twisty Little Passages: An Approach to Interactive Fiction*. Cambridge, MA, USA: MIT Press.

Theune, M.; Faas, E.; Nijholt, A.; and Heylen, D. 2003. The virtual storyteller: Story creation by intelligent agents. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment (TIDSE) Conference*, 204–215.