Utilizing Graph Rewriting for Offline Narrative Generation

Ben Kybartas

260477933, McGill University, Montreal, Quebec

Abstract

In this paper we present a system which uses graph rewriting techniques to create interesting and unique narratives for use within a video game environment. The system uses a graph containing an abstract representation of the game world that consists of entities and relations between entities. By discovering specific patterns within this graph, a narrative skeleton is created. From this skeleton, potential narrative "twists" are found by again searching for specific patterns within the game world. The narrative, in turn, causes changes within the game world. The final system creates a dynamic game environment which inspires narratives and in turn reacts to these narratives accordingly resulting in a unique and ever changing game experience.

Keywords: Narrative Generation, Procedural Content Generation, Graph Rewriting

1. Introduction

Procedurally-generated content (PCG) has long since been a valuable area of research for the video games industry as it allows for the creation of game content without the intervention of game designers. While the realm of potential research areas for PCG in computer games is vast, one area of particular interest is that of narrative generation.

Narratives, being particularly popular in role-playing games, immerse the player within worlds where they need to feel that their actions have an effect on, and are motivated by, the game world around them. In this sense,

Email address: `ben.kybartas@mail.mcgill.ca` (Ben Kybartas)

narrative is used to give the player a sense of worth in the world in which they inhabit. For proper generation to occur, the narrative must be closely tied to events and relations within the virtual world in which the game takes place.

We will first present related work in 2. Following this, in 3, we will present the formal model which we are using for our system. In 4 we will discuss the implementation of the system and show a simple example using the system. Next, in 5 we comment on several features of the system. Lastly we present conclusions and future work in 6 and 7 respectively.

2. Related Work

The fundamental design of the system revolves around having two graphs, one of which represents the *Game World* and the second represents the *Narrative* itself. The game world influences the creation and development of the narrative, and the actions taken in the narrative modify the entities and relations in the game world. This concept is similar to several approaches of *cognitive modeling*, such as those used in the ACT-R system present in Stewart and West (2006). In ACT-R, the system uses an abstract representation of the *brain*, which is composed of objects and relations, and *actions* which occur as reactions to the current state of the *brain*. For our system, the *brain* will be our game world graph, and the *actions* will be the events in the narrative we are generating.

Focusing instead on the existing work in the field of narrative generation, there have been some notable examples of narrative generation techniques based on the current state of the game world. The game "Prom Week" from McCoy et al. (2010), utilizes social interactions between characters in order to create *emergent* and *online* narrative situations. The game is essentially oriented around using social interations to achieve goals within the game, such as becoming the king/queen of prom, gaining friends or enemies, etc.

Chang and Soo (2009) presented a method of narrative generation which uses the idea of planning within a game environment in order to generate narratives. In the paper *beliefs* and *motivations* are used to guide characters towards goals, where the actions taken by a specific character may influence the beliefs and motivations of other characters.

From the commercial perspective, the recently released game "The Elder Scrolls: Skyrim", from Bethesda Softworks, uses a basic form of narrative generation as a means of generating infinite quests for players, Lenhardt

(2012). The system, dubbed "Radiant Story", attempts to generate quests which guide players to locations they haven't previously visited, encouraging exploration as well as narrative involvement.

3. Modelling

3.1. Concept

For this system, we will be using two graphs. The first graph will contain the state of the game world, which will be modeled as a series of *entities* and the *relations* between these entities. The second will be our narrative, which is modeled as a series of *events* and the *links* between these events.

The system will utilize both these graphs, using the game world as a means for locating potential stories using a set of *narrative generation rules*, which will then update our narrative. The events which occur within the narrative will, in turn, cause changes to the entities within our game world and to the relations between our entities.

3.2. Game World Graph

3.2.1. Entity

Our game world graph is composed of entities and relations. An entity is defined as follows:

$$e =< n, A, R_{in}, R_{out} >$$

Where n represents a unique identifier for our entity. For example, for an non-playable character (npc), the unique identifier would most likely be the name of the npc.

A represents a set of attributes. An attribute consists of both an *identifier* and a *value*. Taking the example of an npc, If we want to denote the jobs of npcs in our game environment then a sample attribute would be rank, knight for a knight and rank, king for a king. Therefore a sample entity would be the following:

$$< "Arthur", \{"rank" : "king", "alive" : True\}, [], [] >$$

Note that we choose to denote our attribute set in this fashion since in our implementation we use dictionaries for our attributes, and above is how a dictionary is declared in python.

Lastly, R_{in} and R_{out} are sets which define the incoming and outgoing relations for this entity.

3.2.2. Relation

A relation in our implementation is defined as follows:

$$r =< a, e_{from}, e_{to} >$$

Here, a represents a single attribute which defines our relation. The attribute once again consists of an *identifier* and a *value*. In this case, these roughly translate to *relation* and *reason*. For example. If the knight is friends with the king because knight's are loyal to the king then the relation connecting the knight to the king would be given the attribute {"friends":"rank"} . This translates to: the knight is friends with the king because of the rank of the king. Note that rank is the same identifier we used to describe one attribute of the king in the above example. This means we can further state that the knight is friends with the king because he is the king.

e_{from} and e_{to} denote the outgoing entity and incoming entity respectively. For our above example, the e_{from} would be the knight and the e_{to} would be the king, since we are describing the relation of the knight with respect to the king. The relations are not two way, the king may not be friends with the knight, or may be friends for a different reason, hence why our relations are one way only.

3.2.3. Game World Graph

Using the above two definitions our game world graph can simply be defined as follows:

$$G =< E, R >$$

Where E represents the set of all our entities and R represents the set of all of our relations.

3.3. Narrative Graph

3.3.1. Event

Our narrative graph is very similar in definition to the Game World graph as it involves entities and relations. However for our narrative graph, our entities are technically *narrative events*. An event is defined as follows:

$$x =< n, A, Y_{in}, Y_{out}, e >$$

n, and A are the same as above, except in this case they are used to describe the *name* and *attributes* of an event in particular. Y_{in} and Y_{out} once again refer to the incoming and outgoing relations for our narrative graph. To differentiate, here we will call relations *links*, in order to differentiate them from the relations described above.

e here refers to a target entity, since each event in the narrative is related to a specific entity within our game world. For instance, if our narrative event is to murder an npc, the npc to be murdered would be our target entity.

3.3.2. Link

Links in our narrative graph are defined as follows:

$$y =< x_{from}, x_{to} >$$

For a link, we only need to know from which event we have come and to which event we are headed. These are labeled x_{from} and x_{to} respectively.

3.3.3. Narrative Graph

One again, we can simply define our narrative graph using the above two rules as follows:

$$N =< X, Y >$$

Where X represents the set of all our events and Y represents the set of all our links.

3.4. Graph Rewrite Rules

3.4.1. Narrative Generation Rules

Our system is further defined by a set of rewrite rules, used both for narrative generation and for narrative rewriting. For a narrative generation rule, we use the following definition:

$$u =< G_{cond}, G_{outcome}, N_{outcome} >$$

Where G_{cond} represents our game world *condition*, ie. we need G_{cond} to be a subset of our main game world G. We use the subgraph isomorphism algorithm presented by Ullman (1976) for testing this condition. If the condition is met, then we generate the narrative given in $N_{outcome}$. Additionally, we link social outcomes, defined as $G_{outcome}$ to certain events in the narrative structure. For instance, if a character is murdered at one event in the narrative, then we will need an update rule which sets the character to be dead and update their relations accordingly (such as making all their friends and lovers hate the murderer).

3.4.2. Narrative Rewriting Rules

A narrative rewriting rule is declared as follows:

$$v =< G_{cond}, N_{cond}, G_{outcome}, N_{outcome} >$$

Similarly to the narrative generation rule, we first have a game world condition, G_{cond}, that must be met in order for our rewriting to occur. However for a rewrite rule, we also have a N_{cond}, which is the condition that the narrative must meet in order for the rewrite rule to occur. Here, we again have two outcomes, the first being the changes to the game world graph, $G_{outcome}$, and the second being the changes to the narrative graph, $N_{outcome}$.

3.5. System

Putting together all that we have above, we can define our overall system as follows:

$$sys =< G, N, U, V >$$

Where G is our game world graph, N is our narrative graph, U is the set of all narrative generation rules, and V is the set of all narrative rewrite rules.

4. Implementation

The system was implemented using Python, following the above definitions for each individual component. At the start, we only need to define our main social graph. A sample social graph is shown below:

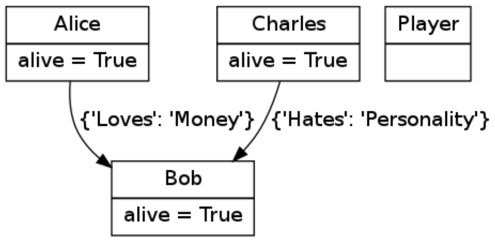

Figure 1: Our Starting Game World

In this example we have four entities, and two relations. Note that at the start we have left the player as *neutral* meaning they have no relation to any of the npcs in our game world at the moment. This allows us to simulate a *fresh* world, in which the player has not performed any actions and made no changes to the game world. In order to develop our narrative skeleton, we search through our rule set and attempt to find a matching rule. For this example, we are only assuming one rule, in which the player is asked by a character to kill their enemy. The rule is represented as follows:

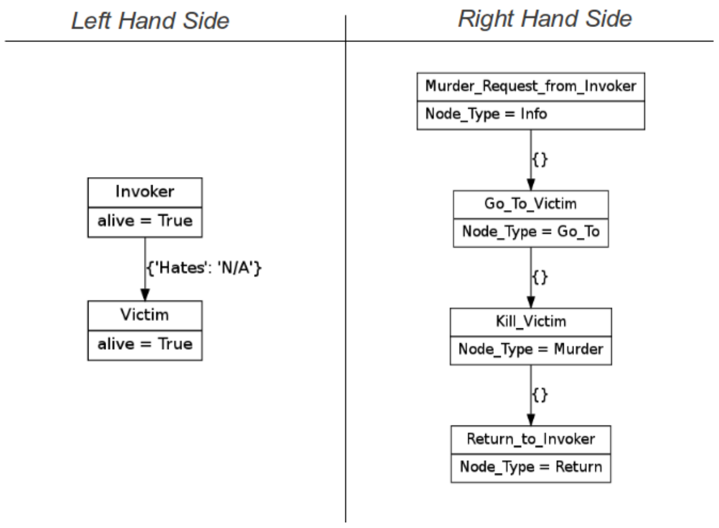

Figure 2: A Sample Narrative Generation Rule

On the left-hand side, we have the *condition* for our rule, and on the right-hand side we have the *result*. In this case, the *result* will be our narrative skeleton. In checking this rule, the system will notice that Charles has a hate relation to Bob. Note that using *N/A* as a reason means that we do not care *why* one character hates another, the rule is satisfied simply if there exists *any* hate relation. The scheduler then takes our two target npcs, Charles and Bob, and creates our narrative skeleton, shown below:

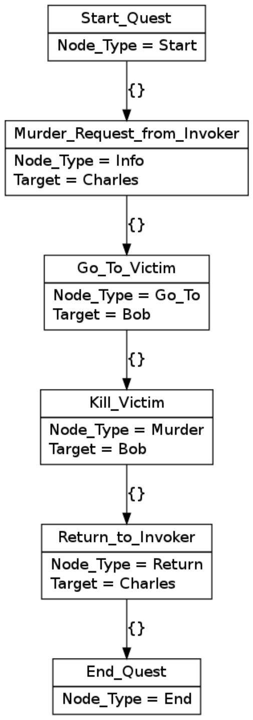

Figure 3: Our starting narrative skeleton, notice how the *target* gives the name of the target entity for each event (ex. for Kill Victim, the target you are killing is Bob)

The next step of the generation stage is to simulate execution of the narrative, using each event as a discrete unit of time, by moving from event to event and checking if there are any game world modifications at any event. In this case, at the "Kill Victim" stage, we will need to set the *alive* attribute of our victim to be *False*. However, on top of this, the scheduler is also checking if there are any cases where we can apply a narrative rewriting rule. Again, we assume only one rewrite rule, which is defined as follows:

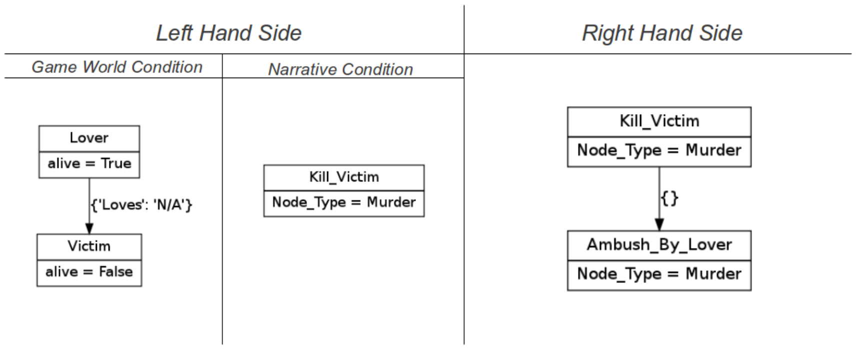

Figure 4: An example narrative rewrite rule

Here we have both a game world condition and a narrative condition. When the simulator reaches the "Kill Victim" stage, it will check if there are any relations in the graph where there is a character who has a love relation to the victim of the murder. For our game world, it will discover that Alice loves Bob. The system will then update the narrative to contain this new twist, where the player must additionally kill the lover of their victim as well as the victim themselves. Since this is our only rewrite rule, the system will finish simulating the narrative. The resulting narrative with the new twist appears as follows:

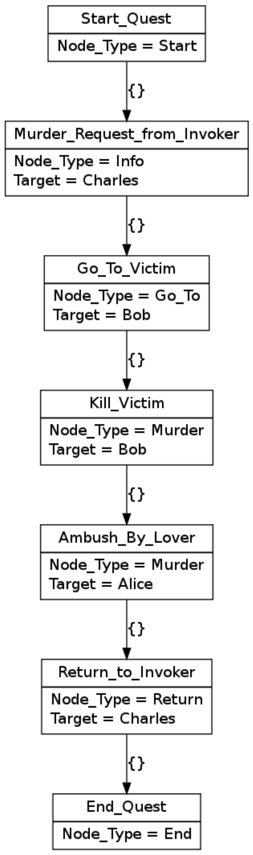

Figure 5: The final narrative after applying the narrative rewrite rule

As mentioned before, while the narrative is simulating, it is also making updates to the game world graph. For this story we are murdering two characters. The updates to the graph are also rules. For a murder, we update the graph based off the relations of the murdered character. If the character was friends with or loved by any other characters, then they develop a hate relation towards the murderer. If the character was hated by any other characters, then they develop a friends relation towards the murderer. Additionally, the murdered character loses all outgoing relations, since the dead character's relations to other entities no longer matters. The final graph graph is shown here:

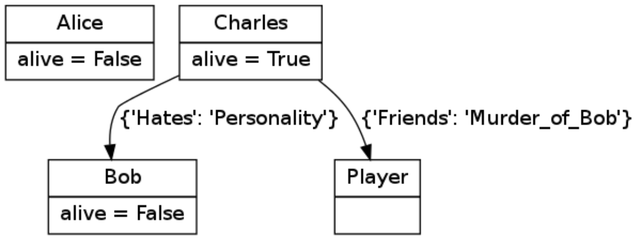

Figure 6: The Final Game World given our narrative. Note that the character is no longer neutral (ie. Charles now has a relation to the player)

We can see that both Alice and Bob were killed throughout the duration of our narrative. We can also see that Charles now has a friendship relation to the player, and the reason for this is that the player has killed Bob, whom Charles hated. For comparison, if simulate the narrative except remove the Ambush scenario, where the player must also kill Alice, we get instead the following game world graph:

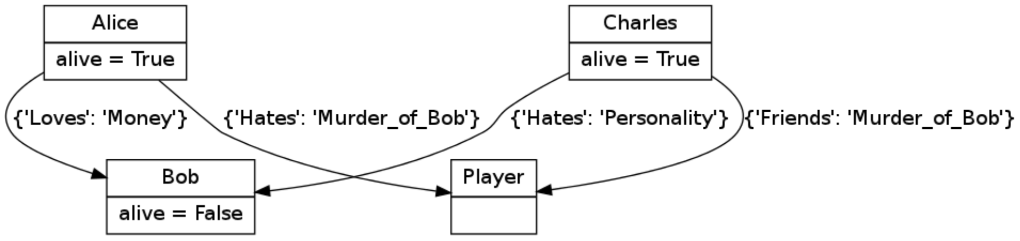

Figure 7: The Final Game World if Alice remains alive. Note that both characters now have conflicting views of the player due to their previous relation to Bob

In this instance, We see that Alice now has a hates relation to the player, since she has a love relation to Bob, whom we have previously murdered.

5. Comments

The system we presented uses a very simplified view of relations between entities within the game world, such as the relations between npcs being modeled as simple *relation/reason* pairs. The advantage of this is that we want to use a minimal vocabulary to define our game world. If relations are too specific, then we reduce the number of matching subgraphs (as well as the likelihood of finding *any* matching subgraph). For example, the relationship set $friend, neutral, enemy$ means that any npc relation falls into one of these three categories. Thus, if we have a rewrite rule which requires a $friend$ relation, then we are likely to get a result. However, using a set that small may be inadequate for describing the world which we want to create. Therefore, there is a trade-off between the size of the relationship set and the number of subgraphs found for a given rewrite rule.

This system differs from the previously mentioned work in that it aims to generalize the process of defining a game world and defining a narrative and to make it simple enough that a game designer would easily be able to work with it if they wished to use such a system for their game. Likewise, the tool can be extremely specific, as mentioned above if the designer wished for very specific relations to exist, or for very specific conditions to be used when creating a narrative, then these could be added easily.

6. Conclusions

In this paper we presented a means of generating narratives for games by applying graph rewriting techniques. We used a graph representation of our game world based off entities within the world and their relations between each other. We searched for specific entity/relation subgraphs within this world and used those to create a narrative skeleton. We then simulated this narrative, rewriting parts of the narrative to create a more dynamic story. The result gives us our game narrative as well as an updated game world graph. The resulting system is very easy to understand visually and could easily be understood by a designer aiming to use this system to create their own narrative generation system.

7. Future Work

7.1. Online Narrative Generation

The system implemented first created a narrative skeleton and then simulated it in order to search for narrative twists. The result of this was our narrative. This entire process was run *offline*, meaning we are expected to generate the entire narrative before presenting it to the character. An alternative to this system is to use an *online* generation system. For this system, we would immediately present the narrative to the player. As the player worked their way through the narrative, the system would dynamically check for any potential narrative rewrites, and then present them when necessary. Using this system would reduce the generation time as we would not need to perform a narrative simulation and would also present a more dynamic experience to the user, where changes in the game world would produce immediate reactions in the narrative, as opposed to having all events predetermined.

7.2. Branching Narrative

Another interesting area for narrative generation is the idea of creating *branching* narratives, where there are different paths throughout the narrative. Some of these paths may lead to completely different results and other paths may simply be alternative methods of achieving the same goal. This system would work both as an offline or online system. For an offline system, we would need to simulate all possible paths through the narrative in order to create a large narrative containing all possible paths and outcomes. For an online system, we would simply need to continue to generate new narrative segments depending on whatever choices the player makes. Having a branching narrative again gives more control of how the story unfolds to the player, resulting in a more engaging and personal experience for each player.

8. References

Chang, H.-M., Soo, V.-W., September 2009. Planning-Based Narrative Generation in Simulated Game Universes. IEEE Transactions on Computational Intellignence and AI in Games 1 (3), 200–213.

Lenhardt, H., January 2012. Bethesdas Nesmith reflects on the difficult birth of Skyrims Radiant Story system. Accessed on April 2012.
URL `http://venturebeat.com/2012/01/27/bethesdas-nesmith-reflects-on-the-difficult-birth-of-skyrims-radiant-story-system/`

McCoy, J., Treanor, M., Samuel, B., Tearse, B., Mateas, M., Wardrip-Fruin, N., June 2010. Authoring Game-based Interactive Narrative using Social

Games and Comme il Faut. In: Proceedings of the 4th International Conference and Festival of the Electronic Literature Organization: Archive and Innovate. Providence, Rhode Island, USA.

Stewart, T. C., West, R. L., 2006. Deconstructing ACT-R. In: Proceedings of the Seventh International Conference on Cognitive Modeling. Trieste, Italy, pp. 298–303.

Ullman, J. R., January 1976. An Algorithm for Subgraph Isomorphism. Journal of the Association for Computing Machinery 23 (1), 31–42.