Beep! Beep! Boom!: Towards a Planning Model of Coyote and Road Runner Cartoons

ABSTRACT

In currently available games, there is a lack of physical humor based games. The humor in current games is largely linguistic, and hand authored. In this paper we propose a A Cartoon Making Experience (ACME), a system that includes physical humor, based on The Road Runner and Coyote cartoons in which the player can add and remove objects from the world. Then using an HTN planner, the game plans out a gag based on the item attributes as well as the state of the world.

Keywords

Computational humor, cartoons, artificial intelligence, planning.

1. INTRODUCTION

Humor is an important part of entertainment: it is everywhere from comedic movies, to sitcoms on television, to standup comics live at comedy clubs. In spite of this prevalence of comedy in other forms of entertainment, it plays a relatively small role in interactive digital media, especially games.

There are a few instances of games that use comedy as a main gameplay element. Games such as the *Sam and Max* series use a team of writers to write and pre-script the actions and dialogue that make the game funny. This is a highly labor-intensive task in which writers have to spend time creating the content for the game, limiting the re-playability of the game. In addition, the humor is often language based, which gives it strong ties to a particular cultural, relying on puns, cultural references, etc. This makes the translation of the jokes very difficult and may seem unfunny to someone without the requisite cultural background.

The Sims, in contrast, uses visually based humor. The game uses a humorous model of human nature and intersperses visual gags during gameplay. Tim Schafer counts *The Sims* as a funny game and adds, "How can someone peeing on the floor not be funny?" [4] Schafer says the humor in *The Sims* "comes from everyday situations, not the fantasy worlds that games are usually drawn from, so it's very different and a wider variety of people can relate to it." Furthermore, the comedy in *The Sims* takes advantage of its action-based interaction to create a game that is funny through natural interaction that the player is already having with the game, rather than something tacked on.

Our work here starts from these two observations about why the humor in *The Sims* is successful: humorous actions are key gameplay elements, and the primary type of humor is physical comedy. However, unlike The Sims, we're interested in exploring AI models that explicitly reason about and integrate humor into gameplay.

Our game, A Cartoon Making Experience (ACME), is based on the Warner Brothers' Wile E Coyote (the Coyote) and Road Runner (the Road Runner) cartoons. These cartoons are ideal because their main focus is physical humor. Chuck Jones, in his book *Chuck Amuck* [7], says:

> "The Road Runner and Coyote cartoons are known and accepted throughout the world—perhaps the lack of dialogue is one reason. If you want to laugh, you can do so at any time, whether in Danish, French, Japanese, Urdu, Navajo, Eskimo, Portuguese, or Hindi, "Beep-Beep!"

The Coyote and Road Runner cartoons have the additional advantage that the explicit rules used in the creation of the cartoons help ensure internal consistency [7, 10]; this rule-based consistency makes it easier to capture the humor of these cartoons in a computational system. Since games are procedural rule-based systems, basing a humorous game on a cartoon with internal rule structure of its own makes for greater potential to integrate the humor into the actual gameplay than might otherwise be possible.

We make several contributions in this paper. First, we formally analyze the Coyote and Road Runner Cartoons (CRRC) and identify the gag structures they use, as well as some salient information from the cartoons. We combine this analysis with a survey of some theories of humor and how they related to cartoons. We then use these insights to build up a hierarchical task network (HTN) planning domain that can represent the CRRC world and generate action sequences within it that follow its internal rule structure. Finally, we integrate the planning domain with an interactive player experience that allows the player to take on the role of the cartoonist in the world.

2. RELATED WORK

Research in computational humor has traditionally focused on linguistic forms of humor. Early work began with the creation of puns called Tom Swifties [8] and continued with work on creating punning riddles using the system JAPE [1]. A more recent system, STANDUP, is also based on punning riddles [2, 12]. There is also a system, HAHAcronym, that creates humorous acronyms [13]

Another area of linguistic computational humor is joke detection and understanding. This includes recognition of puns and jokes and detection of knock-knock jokes [15] and a program to detect puns in Japanese [14].

Cavazza et. al. work on physical humor in interactive narrative settings [3]. They first started using Heurstic Search Planning (HSP) to create a humorous digital storytelling experience. The system used another cartoon, the Pink Panther, to show how plan failure could be used for comedic effect in the world. Plan nodes that fail are considered to be funny because they rely on the expectation of something happening; humor occurs when the expectation is violated. For our project we take the idea of plan failure being funny and add conditions to pick when the plan should fail.

The paper written by Thawonmas et al also used HSP, in this case to plan the actions of character that mimics the actions of a player-controlled character in a scene based on the Mr. Bean world [16]. The player-controlled character is a semi-autonomous agent who responds to controls by the player by doing things, but still contains its internal game state that is important for narrative goals. There also exists a character in the world that is a fully autonomous plan-based character whose goal is to mimic the actions of the player's character, in a way that seems funny to the player. The state of the world and the state of the characters in the world are important mechanisms for determining how the plans should succeed. In contrast our characters are fully autonomous, but like Thawonmas, we do that have plans that are affected by world state, and the actions of the players have an effect on the gameworld.

In contrast, the second system by Thawonmas et al uses an extended HTN system, also set in the Mr. Bean world [17]. They use "executability conditions" as additional constraints on the HTN operators. When the preconditions are met, but an executability condition fails, this represents a failure of the action to execute. For instance if we take an operator `pickup`, it has the precondition `available`, and it results with an `object-in-hand`, in contrast `pickup-silently` has the same precondition and result, but has an executability condition `not-someone-aware-of`. Therefore if the `pickup-silently` was chosen rather than just `pickup`, and the `not-someone-aware-of` failed, the action would fail but would be deemed humorous and represent the character trying to pickup an object silently but someone seeing him. When executability conditions fail an action, a heuristic is added, which helps determine what action is taken next, or, if high enough, represents the trying and failure of multiple actions, which kicks it up to a higher task. We use a similar sort of heuristic for determining when the failure of the Coyote's plan takes place and the gag planning occurs.

Finally, our work takes inspiration from a formal analysis of the consistency of the Coyote and Road Runner world [10]. In that work, a Road Runner micro-world was encoded using a modal-temporal formal logic. This logic was used to show that internal consistency was maintained in the gags even though the gags were not realistic. This was due to internal rules that were always followed, but that took into consideration attributes of the Coyote such as belief state in determining the outcome of a rule.

3. ANALYZING THE CARTOONS

In his book *Chuck Amuck* [7], Chuck Jones describes the world background and rules he used to create CRRC. The Coyote is meant to be his own worst enemy in the cartoons; it is his persistent chasing of the Road Runner, even though he has never caught the Road Runner, which is the root cause of all his problems. This is important to note in that the main cause of the problems for the Coyote must be his own actions and not bad things that happen to him due to the intentions of others or randomly (acts of God).

The book references 10 rules that were generally followed while creating the cartoons. They are:

1. Road Runner cannot harm the Coyote except by going "beep, beep".
2. No outside force can harm the Coyote -- only his own ineptitude or the failure of Acme products.
3. The Coyote could stop anytime -- IF he was not a fanatic. (Repeat: "A fanatic is one who redoubles his effort when he has forgotten his aim." —George Santayana).
4. No dialogue ever, except "beep, beep".
5. Road Runner must stay on the road -- for no other reason than that he's a Road Runner.
6. All action must be confined to the natural environment of the two characters -- the southwest American desert.
7. All tools, weapons, or mechanical conveniences must be obtained from the Acme Corporation.
8. Whenever possible, make gravity the Coyote's greatest enemy.
9. The Coyote is always more humiliated than harmed by his failures.
10. The audience's sympathy must remain with the Coyote.

While these rules are not a perfect indicator of what actually happens in the cartoons, they give an idea of what rules the cartoonist used to create the internal consistency that helped the cartoons to be so popular.

While watching the cartoons, we extracted a model of how most of the Road Runner cartoons are laid out. A typical cartoon starts with the coyote chasing after the Road Runner, often carrying a knife and a fork. The Road Runner gets away from the Coyote because the Road Runner is the faster of the two. This scene introduces the two characters; but most importantly it shows the initial motivation that the Coyote has for catching the Road Runner.

The cartoon then follows with a series of scenarios of varying length. A typical scenario contains multiple sections that broadly relate to the item that the Coyote is trying to use to catch the Road Runner. The first section is where the Coyote receives the item he needs from the Acme Corporation. He then prepares the item for use, taking the necessary steps needed to set it up for use. Finally he uses the item. The gags predominately take place when trying to use the item, but they can take place in any part of the process. The gags are related to the Coyote's own setup and use of the item against the Road Runner, and his inconsistent understanding of how the world works. The world of CRRC does not realistically model the physics of the real world. Rather, gags establish the Coyote's expectation and then deny that expectation to achieve a humorous effect.

The Coyote's actions are directly a consequence of his fanaticism: a smarter creature would stop chasing the Road Runner after so many failures, or at least would be more careful in what methods he used to enact his goal. The Coyote always has tries to catch the

Road Runner, and due to his own incompetence fails to do so. The gags are based on the Coyote trying and failing to catch the Road Runner.

When the gags happen, and what the gags are, is an important part of the cartoons. The viewer has an expectation that something bad is going to happen to the Coyote. The key factor in a particular cartoon is determining what will happen and when will it happen. Many of the gags happen quickly and this expectation is kept. The dynamite blows up as soon as the Coyote lights it; the gun he is pointing at the Road Runner shoots him immediately.

There are plenty of other instances where gags play with these expectations, usually to enhance expectation or to leave the viewer unsure of when the shoe will actually fall. For example, we take a scene where the Coyote has set up a trap to catch the Road Runner; this trap is a trip wire that is attached to a gun. The Road Runner passes by it without harm. The Coyote wonders what went wrong, so he tests the trap out himself. In a standard version of this gag, the trap would immediately go off and the Coyote would be harmed. However when the cartoon is playing with expectation, the coyote might go through the trap several times before it fires at him or it might not go off until some further gag triggers it. This increases the expectation of the viewer because the Coyote's actions always end up causing him harm, but in this case have not done so yet, leaving the viewer waiting to see how they will end up causing harm eventually.

4. WHAT IS HUMOR

In his book *Semantic Mechanisms of Humor* [11], Raskin summarizes humor theories as belonging to one of three categories:

1. Incongruity – These theories deal with the cognitive response to humor.
2. Disparagement or superiority – These theories deal with behavioral response to humor.
3. Psychoanalytical - These theories are concerned with the release from repression that the humor allows.

There have been many theories created in these categories, and some of the theories have been around for thousands of years; yet none are sufficiently precise and detailed enough to be directly implementable [12]. Consequently, what typically happens in the computational humor literature is that systems take inspiration from one of the grand unified theories, but then use a theory more specifically related to their type of system that or more specifically fits the situation.

The theory we use was taken from Charles Gruner's book *The Game of Humor* [5]. Its premise is that laughing equals winning. He defines winning in its broadest sense of "getting what you want." He goes into further detail with a three-part sub-thesis that states: There will always be a winner and loser, a humorous situation will always be incongruous, and there is an element of surprise in humor. This theory is a form of superiority theory.

In this theory, the winner of the situation is someone who gets what they want. Often, when telling a joke or pun, the winner is the teller of the joke, who is able to assert his superiority over the situation by getting a laugh. Punch-lines for jokes and the wordplay for puns form the basis of the incongruity and surprise. The listener of a joke is presented with a view of the world that is consistent, and then at the last second a twist or surprise ending changes the consistent world as told in the joke. Puns achieve the same effect, but instead bring expectations for words based on the listener's knowledge of their normal meaning, then take it and twist it around to a surprise meaning.

Physical humor can even easily be shown to fit in this theory: the winner is the viewer rather than the one who is having a possibly painful experience inflicted on them. The gags themselves are incongruous to expectation of what should happen in a normal situation and this is often exacerbated by the fact that the one who has the gag performed on them gets up unharmed afterwards.

The CRRC falls very neatly in to the thesis; the winner is the Road Runner, the situations that the coyote gets into are incongruous both to what the Coyote expects should happen and also to the viewers' expectations of physical reality. Finally there is always an element of surprise; this usually manifests in the timing of the gag.

We can therefore use this thesis about winning, and more specifically the sub-theses, to map some of the cartoon concepts and encode them.

5. GAME ARCHITECTURE

Our system, ACME, revolves around the Coyote's eternal quest to catch the Road Runner using whatever means he possible. ACME consists of two parts: a virtual environment in which the Coyote and Road Runner exist as autonomous agents, and an HTN planning domain.

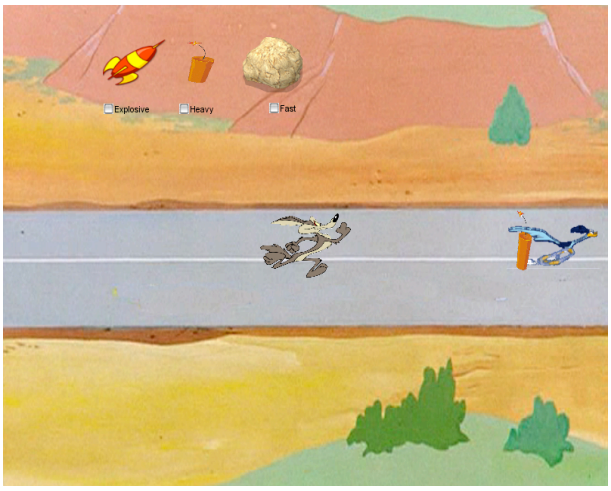

Figure 1: The dynamite was placed by the player ready to pick up.

The game environment is made up of a scene set in the American southwest with the Coyote chasing after the Road Runner. This environment is the player's means of playing, where they take on the role a cartoonist directing a cartoon. The player views the action from a third person perspective, in much the same view they would have if they were watching the cartoon.

The player can add items by clicking on the icon representing the icon of their choice. The player's interaction with the world involves the physical manipulation of objects in the world, specifically the addition, placement and removal of items from the world. Additionally, these items can have player-modifiable attributes (e.g. explosive, fast or heavy), which are set before the player places the item. A secondary interaction allows the player to drag and drop the Coyote around in the environment, allowing to player to have further say in what gags are created.

Figure 1 shows the placement of a piece of dynamite by the player in front of the Coyote. The attribute of explodable was checked setting it for later.

The Road Runner and the Coyote are autonomous agents that exist in this environment. They each contain simple instructions for interacting between themselves. The Road Runner is the simpler of the two, running as a simple finite state machine (FSM) with two states: The Road Runner is either running from the Coyote or is at standing still, and which state he is in is based purely on the distance to the Coyote.

The Coyote also uses a FSM to control the behavior in the game world. The Coyote has a base state of chasing after the Road Runner; however whenever there is an item placed in the world near the Coyote, he will switch states from chasing the Road Runner to going towards the item. When the Coyote is sufficiently near the item, the Coyote switches into the planning state.

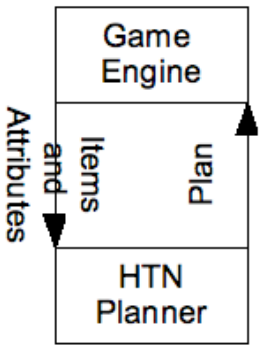

Figure 2:The connection between the game engine and the planner.

The game engine takes the current world state, including all the items and their attributes, and creates the initial predicates and defines the problem to run. The game engine then executes the planner and retrieves a plan. The game engine takes this plan and uses it as input to send commands to the Coyote and items in the system. This flow is shown in Figure 2. Once the plan is executed, the Coyote returns to his default chasing state (he never gives up!).

To create the plans, we use the JSHOP2 [6] HTN planning system. This system was chosen for the initial iteration of this project due to the ease of knowledge engineering within an HTN framework, supporting rapid development and iteration of our ideas.

6. PLANNING

To create a plan, we need to take two competing goals and merge them into one sequence: the desire of the Coyote to catch the Road Runner, and the system's goal to prevent the Coyote from catching the Road Runner in the most humorous way possible.

The Coyote relies on items to accomplish his goal; therefore the best way for the system to prevent the Coyote from reaching his goal is to subvert the nature of the item towards the prevention rather than the accomplishment of the Coyote's goal. To this end, we created items with types and attributes that the system can use in deciding how to plan out its grand goal.

Table 1: This table gives examples of type and attribute create gags

Type	Attribute	Gag
is-projectile	is-explosive	explode
	is-fast	hit-coyote
	is-heavy	drop

The system needs to decide when and how this subversion (gag) plan interacts with the Coyote's plan to catch the Road Runner. Information is shared by having the Coyote state and other portions of world state passed on from the Coyote's plan to the gag plan. An anticipation metric, along with a heuristic, decides when to move from one plan to the next.

Below we discuss the formalization of CRRC as an HTN domain, and the creation of the Coyote and gag plans, in more detail.

6.1 Domain Constructs

We initially need an HTN domain to represent the CRRC world. A HTN domain is made up of two types of tasks: primitive tasks and compound tasks. In JSHOP2, operators represent primitive tasks and methods represent compound tasks. Operators are the only construct that is allowed to change the state of the world. We therefore need to logically split the operators in to two types, internal and external operators, to match plan operations that are meant for internal plan bookkeeping, or external actions done by the game.

The internal operators exist as a bookkeeping construct that allows the plan to update the state of the world; these include such operators as `time-for-gag` that updates the time when a gag plan should start. This internal state is used by methods and operators during planning, but does not directly effect the execution of a gag in the game engine. External operators. such as `explode-item`, are executed by the game engine when the plan is passed to the game engine.

We use types and attributes for each item, which allows the planner to know what sort of function the item has in the world, and what kind of gags are available to use. Each place-able object has a type associated with it, and may have attributes associated with it when placed in the virtual world by the cartoonist.

The type of the object relates the item to its function in the gameworld. A type is pre-set and static in an item. There are three

distinct types of item; `is-moveable`, `is-projectile`, or `is-trap`. The type `is-moveable` references objects that allow the Coyote to move quickly around the world, including such items as rockets, skates or motorcycles. The type `is-projectile` are items that the Coyote can use to shoot or throw, including items such as dynamite, guns or bullets. Finally `is-trap` references trap items that are supposed to catch the Road Runner these would include items such as rope or a tripwire.

An attribute shows what sort of gag can occur on that item. It is a way for the player to specify what can happen to the items in the gag process, and at a higher level it gives the system an idea of how it can use the item to prevent the Coyote from reaching his goal. Attributes included in our system are `is-heavy`, which tells the system that the object is heavy and therefore can be used to fall or crush the coyote, `is-explosive`, which means the item is explosive and therefore can explode, and `is-fast`, which means the item is quick and therefore susceptible to crashing into things.

The types and the attributes together are a primary way that the system uses to generate gags. For example, Table 1 shows all the gags that can occur given the Type `is-projectile` and the attributes `is-explosive`, `is-fast`, and `is-heavy`. Each of these produce a different output relating to the item, but new items can also be created easily and can be used to create gags immediately without any additional need to write code. While attributes link items to plans, state of the world also plays an important role in plans.

The state of items and the Coyote are important factors when deciding the flow of the planning process and the interaction of the two plans with each other. State associated with the Coyote can determine how a gag will be performed, and what consequences there might be to the gag. For instance an `on-item` is set when the Coyote sits on a rocket. The links the Coyote to the rocket and when the rocket moves he also moves. It also gives the system the power to make the Coyote fall off the rocket or be stuck to the rocket.

The two plans also interact when they switch from one plan to the next. This is decided based on anticipation and timing in the system. During the Coyote's planning phase, the system keeps record of how many steps the Coyote has gone through. These steps model the player's anticipation of a gag happening at a certain point, and this anticipation increases linearly with time. A function that relates the current state of the Coyote's plan and the current item then returns a weighted probability based on the likelihood of a gag happening given that item and the current state. The anticipation metric and the weighted probability are checked against a random number. If the metric is high enough the Coyote's plan will end and the gag planning will start to take place.

These constructs in the HTN domain allow us to create two separate plans that are able to interact and come with comedic events.

6.2 Coyote and Gag Plans

The flow of the plans is shown in Figure 3. A box designates each method. The methods attached to the main method can be called at any time when the preconditions are met. The Coyote's portion of the plan is designated by the get item, prepare item, and use item. The gag portion is when gag, what gag, how gag and consequences.

The planning starts in a state near an item that the Coyote believes will help him to catch the Road Runner. To use the item the Coyote must first obtain the item, then he must prepare it for use, and finally he must properly use the item. Obtaining an item happens the same for any item in the game; however the preparation and use of items are specific to the item that is being used.

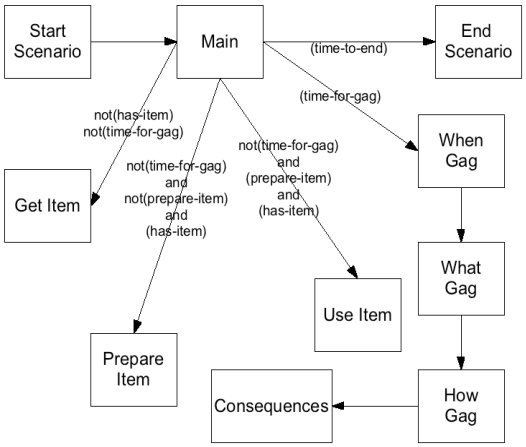

Figure 3: The global planning layout.

Obtaining the item is a simply a matter of moving to the item and then retrieving for use. The Coyote location is moved to be next to the item then gets item, a state predicate is set linking the Coyote to the item.

Once the Coyote obtains the item, he can start to prepare the item. Unlike getting the item, preparing the item has to be written for each item. This is done because items that have the same type and attributes may not necessarily have the same methods for preparation. For instance a pair of skates and a rocket, both have the type `is-moveable` and have the attribute `is-fast`. Both may have the same sorts of gags, but preparing a rocket requires getting on and lighting a match, where the skates just need to be put on. There are some predicates that are set in the preparation of items that can be referenced later by the gags. In the case of the skates versus the rocket both would contain the predicate `on-item`, though the rocket may contain additional predicates.

Using the item uses a similar sort of mechanic as preparing the item; each item has its own methods and operators that it calls in normal use of an item. However these operators will change the state in standard ways that the gag system can use later to determine the gag or effects of the gag.

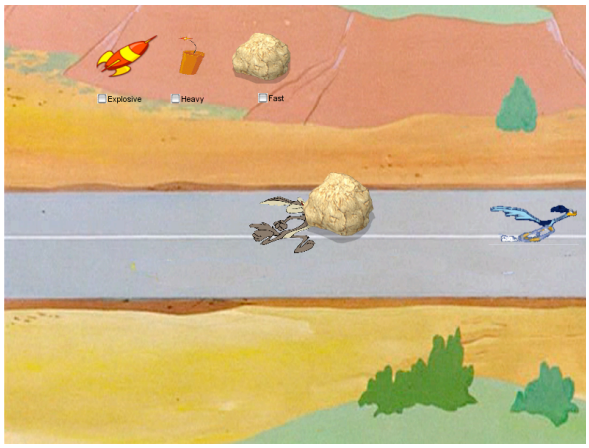

Figure 4: Boulder falling on the Coyote will never be followed by another gag.

During the previous steps, the system is calculating the anticipation metric and the heuristic to decide whether or not it is time for a gag. When calculation is satisfied based on these factors, the system will stop working on Coyote's plan and instead start to work on the gag planning.

The gag system is made of four questions that must be answered. The first is when to trigger the gag, then what gag is going to happen, then how the gag will happen, and finally what the consequences are. This first question about when the gag should occur has at this point already been answered by the anticipation metric and heuristic, but the rest of the questions must be answered.

What gag takes place must be calculated using the type of the item and its attributes, filtering out any gags that would not make sense for the time the gag takes place. As noted in Table 1 above the type of the item and the attributes of the item determine what gags can be performed. If multiple attributes are selected, then multiple gags are possible. The system takes these gags and randomly picks one. However, when the gag takes place determines if a gag should take place. For instance, if the Coyote is planning on throwing a piece of dynamite at the Road Runner, it makes no sense if the dynamite explodes before he has even gotten to it. It also violates the idea that the Coyote is the cause of his own destruction. The plan would make more sense if the Coyote were at least in the process of lighting the dynamite before it exploded.

```
move-object coyote rocket
get-item coyote rocket
get-on coyote rocket
prepare-object coyote rocket
chase-roadrunner coyote rocket
ready-for-gag
set-gag rocket is-fast
move-object rocket terrain
crash-into rocket terrain
explode rocket
drop boulder coyote
```

Figure 5: Example Gag Plan

The gag will not make sense until we figure out how the gag took place. How the gag happens is dependant on a number of factors, including when the gag happened and what gag was done. Other factors include the state of the Coyote in the world, as well as the state of the item compared to the Coyote in the world. For instance, taking the lit dynamite example above; if the Coyote had thrown the dynamite at the Road Runner, it would be in a different location, therefore the dynamite would have to make it back to the Coyote before exploding, or the gag would not work. However, if the Coyote was still holding the dynamite in his hand, it could explode right away because all the conditions for a humorous explosion are met.

Finally, the state of the world and the gag that was picked determine the consequences of the gag. The consequences are specific to the gag. For example, the explode gag looks at the state of the world and finds any nearby items with higher elevation. If there is such an item, it will call the fall gag using the elevated item. This can bring chaining of gags: a consequence can call other gags, thus in the correct circumstances a number gags could be performed from one initial action.

The current model just has one gag for each type-attribute pair. How the gag occurs and the consequences of the gag occurring provide the variety to the gags, because this is where most of the external state references come in to play.

Also note that some gags, such as in Figure 4, do not call anything else and therefore are often used for the chaining mechanisms. The final state of a plan is when then the original item is destroyed, which occurs after all the consequences for those items are finished.

7. RESULTS

In Figure 5 we show an example plan that the ACME planner would produce and then send on to the ACME engine. This plan starts with the Coyote being near a rocket that is of type is-move, and has the attributes of `is-explosive` and `is-fast`. In the world also exists a boulder, which has an attribute of being is-heavy.

The plan starts with the Coyote in `get-item` state moving to the rocket and getting the rocket. The coyote is able to successfully get the item and then moves in to the preparation of the rocket. The rocket preparation consists of merely getting on the rocket to use. He then using the rocket is successfully able to start chasing after the Road Runner. The system decided to only let him chase for a short time before deciding that a gag was to take place.

The system decided it should crash the rocket; however it could have also decided to explode based on the fact that the rocket is both explosive and fast. Once it decided that it would crash, it tried to figure out how it would crash. It looked for appropriate objects to crash into and decided that crashing into the terrain was the best option. It therefore moved the rocket to the terrain and told it to crash. It then looked at the consequences possible for an item with its attributes, and it found that it could also explode. Using a probability function, it decided that it would explode and so it called the explode function. The explode function made sure

the explosion was happening at the point where the Coyote was located, and then it exploded. The consequences were called and explode looked and found the boulder. The boulder is nearby and above the Coyote's position, so it called the gag drop to drop the boulder on the Coyote. The drop boulder is an end gag so there are no other gags called from the consequences. Figure 6 shows a screenshot of the output when the plan is executed in the game engine.

For this example we could have taken another item such as a motorcycle and easily added it to the list of items and placed it down instead, and if the motorcycle had the similar attributes the gag would have been similar. In addition adding new gags including what, how and the consequences is easily done. It just requires writing the gag and some simple integration with the existing system. The structure is modular and easily extensible.

This also means that minimal amount of work can create new sets of gags because the new structure can work with the existing structure. So if there was a consequence where the Coyote would fall off the rocket instead and then have it crash in to him. The n we could easily create a slippery attribute where the Coyote fell off the rocket and then have the fast attribute crash in to the Coyote. The other way could be written as well where the Coyote crashed and then slipped off the rocket humorously.

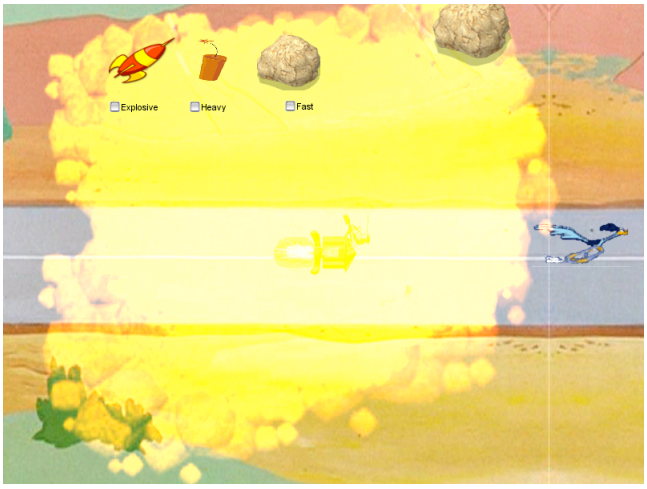

Figure 6: A gag which explodes the rocket on the Coyote and is about to drop the boulder on him.

The gags that occur in our system could occur in the CRRC. Crashes and exploding rockets do happen CRRC however ACME misses out on two important aspects of the CRRC: the fine grained movement and animation that partially makes the cartoons so funny.

8. CONCLUSION AND FUTURE WORK

In this paper we formally analyzed what goes in to a Coyote and Road Runner Cartoon. The analysis showed how the gags take place and also how the gags themselves are modeled. We then took this analysis and built a HTN domain model that incorporated the ideas of a Coyote plan and a gag plan that interact with each other. We used the idea of anticipation and heuristics to decide the best time to call the gag plan. Then, using item types and attributes, we built a generic gag planning system. The Coyote plan is split into sections about the retrieving, preparing and using an item. The gag planning is split, answering the questions: when does the gag happen, what gag happens, how does the gag happen, and finally what are the consequences to the gag happening. We finally showed an example of a plan and gave the some of the advantages and disadvantages of the planning mechanism.

There are three main areas of improvement that we see for work in the future: first converting from HTN planning to a more hybrid approach, second improving on the links between theories of humor and finally creating a new class of items that correspond to effects in the world.

The first thing is to create a hybrid of ABL [9], a reactive planner and HSP. This integration would bring a good measure of authorial control, integration with the environment and allow for more emergent planning. ABL has similar structure as HTN planner but has sensors and actions that talk directly to the game engine and it works in real-time. We would be able to make the Coyote a fully integrated ABL agent, having all of his planning take place in ABL. HSP provides more variation because it dynamically generates a plan given a goal as opposed to the tree nature of HTN. This would provide a bigger variety of gags and therefore could be more interesting.

A more detailed anticipation metric and other ideas taken from humor theory would add to the cartoons. This would involve creating a more robust player model, taking into account and looking at ideas and areas that people may find funny. To accomplish this we would also have bring in some gag and state history, to be able to better anticipate what would be funny and when it would be funny. These ideas would also be better integrated throughout the planning process rather than just a link between the Coyote plans and the gag plans.

Finally, adding effect-items in to the world would help the player have more control over the gag plans and bring new consequences to the gags. Items would affect the physical forces in the world and could provide hints to the planner about the funniest places that they could go. The effect-items would be purely for plan modification.

9. REFERENCES

[1] Binsted, K., 1996. Machine humour: An implemented model of puns, University of Edinburgh.

[2] Binsted, K., Bergen, B., Coulson, S., Nijholt, A., Stock, O., Strapparava, C., Ritchie, G., Manurung, R., Pain, H., Waller, A. and O'Mara, D.,2006. Computational Humor, IEEE Intelligent Systems, 21 (2006), pp. 59-69.

[3] Cavazza, M., F. Charles and Mead, S. J., 2003. Generation of Humorous Situations in Cartoons through Plan-based Formalisations,CHI-2003 Workshop: Humor Modeling in the Interface.

[4] Gonzalez, L., 2005. A Brief history of video game humour, Gamespot.com.

[5] Gruner, C. R., 1997 The game of humor : a comprehensive theory of why we laugh, Transaction Publishers, New Brunswick, N.J.

[6] Ilghami, O. and Nau, D. S., 2003. A General Approach to Synthesize Problem-Specific Planners.

[7] Jones, C., 1989 Chuck amuck : the life and times of an animated cartoonist, Farrar, Straus & Giroux, New York.

[8] Lessard, G. and Levison, M., 2002. Computational modelling of linguistic humour: Tom Swifties,ALLC/ACH Joint Annual Conference, Oxford, pp. 175-178.

[9] Mateas, M. and Stern, A.,2002. A Behavior Language for Story-based Believable Agents., IEEE Intelligent Systems, 17 (2002), pp. 39-47.

[10] McCartney, R. and Anderson, M., 1996. The believability of Road Runner cartoons: logical consitency conquers unreality,AAAI Technical Report.

[11] Raskin, V., 1985 Semantic mechanisms of humor, D. Reidel Pub. Co. ;, Dordrecht ; Boston Hingham, MA.

[12] Ritchie, G., Manurung, R., Pain, H., Waller, A., Black, R. and O'Mara, D., 2007. A practical application of computational humour, in A. Cardoso and G. A. Wiggins, eds.,Proceedings of the 4th International Joint Conference on Computational Creativity, London, pp. 91-98.

[13] Stock, O. and Strapparava, C., 2002. HAHAcronym: Humorous Agents for Humorous Acronyms.,The April Fools' Day Workshop on Computational Humour., Trento.

[14] Takizawa, O., Yanagida, M., Ito, A. and Isahara, H., 1996. On Computational Processing Of Rhetorical Expressions - Puns, Ironies And Tautologies.,Proceedings of Twente Workshop on Language Technology 12., University of Twente.

[15] Taylor, J. and Mazlack, L., 2004. Computationally Recognizing Wordplay In Jokes,,CogSci 2004, Cognitive Science Conference Proceedings, pp. 1315-1320.

[16] Thawonmas, R., Hassaku, H. and Tanaka, K., 2003. Mimicry: Another Approach for Interactive Comedy,GAME-ON, pp. 47-.

[17] Thawonmas, R., Tanaka, K. and Hassaku, H., 2003. Extended Hierarchical Task Network Planning for Interactive Comedy,PRIMA, pp. 205-213.