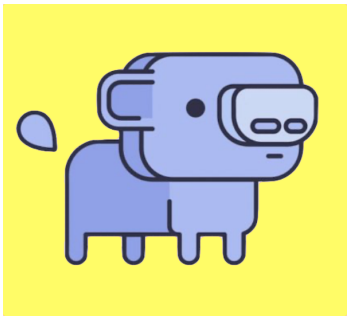


Wumpus

Here it is: the alpha package containing Wumpus and a demo storyworld, RomCom. This is an alpha package—that means that it's rough. It has bugs and lacks many useful features.

This version is like a 1965 Volkswagen bug: the rear window is broken, the windshield wipers don't work, the passenger side door is wired shut, there's a hole where the radio was, shifting from first gear into second gear takes a couple of tries, and the heater vents carbon monoxide into the car. But it works: it reliably gets you where you want to go. That's good enough for alpha release; you folks can offer bug reports and suggestions for feature improvements.

My fear is that some of the feature suggestions will be a distraction. I'm sure that, if we implement all the suggestions, the VW will be transformed into a stretch Tesla limousine with a top speed of 185 mph, a wet bar, lounge seating for twelve, and a crew of dancing girls. For now, we won't be doing that. For now, I just want to get the windshield wipers working and the rear window fixed.

I'll not be releasing the source code for a while; I expect that I'll be making so many code changes in response to bug reports that there's no point in releasing source code yet. I'll release the source code once the program has gotten into beta.

Bug Reports

For now, I need bug reports from you. Not that many people seem to know how to write a useful bug report. It really doesn't help to get a bug report saying "When I push the round thing, it doesn't work right." Here's how to write a decent bug report.

First, write a short one-line summary of the bug, something that would serve as the subject line for an email.

Next, write a short explanation of the bug, fleshing out the one-line summary in greater detail.
Write a procedure for re-creating the bug. A great many bug reports are thrown into the trash can with the letters "CNR" scribbled across them. "CNR" means "Can Not Reproduce". The programmer was unable to make the bug appear using the reporter's instructions. Your procedure should state EXACTLY what I should do to reproduce the bug. You should test your procedure to make certain that it produces the results that you think wrong.

At the end of the procedure, you must state

1. What you expected to happen after the last step of the procedure.
2. What actually happened after the last step of the procedure.

Here's an example based on a real bug I just fixed:

Summary: A newly-created encounter shows the contents of the previously edited encounter.

Explanation: when you create a new encounter, it shows the contents of the encounter that you were previously editing as if they were the contents of the newly-created encounter.

How to Reproduce: Open file RomCom35.xml. Select the encounter "PrepareForDate". Click on the add button for encounters (the + button to the left of the "Encounters" title). Enter the title of the new encounter in the dialog box. Click on the "OK" button at the bottom of the dialog box. Observe that the new encounter shows the introductory text, options, reactions, and other properties of the "PrepareForDate" encounter.

I expected that the screen would display null entries for all the properties of the encounter.
Instead, the screen displayed the entries from the PrepareForDate encounter.

I'll send you an email response to your bug report presenting one of three outcomes:

A. That's not a bug, that's a missing feature.
B. I was able to reproduce the bug and correct it.
C. CNR.

Feature requests

I'm sure that you'll have plenty of ideas for feature additions. I will maintain a list of all proposed feature additions. I will prioritize these proposals based on my judgement of the ease of implementation and the relative value of the feature. As time permits, I will implement features in order of priority. This means that I will not be able to implement some features.

I already know three features that are truly needed but I can't do. The first is the ability to rename encounters. The second is the ability to reorder encounters in the encounter list. Neither of these is easily done in Java. For now, you can accomplish these goals by simply editing the XML file. You DO have a text editor, don't you?

The third necessary feature is a cut, copy, and paste capability for the text. This is not that difficult to do, but it does involve a few score lines of code, and I just haven't had the time to implement it.

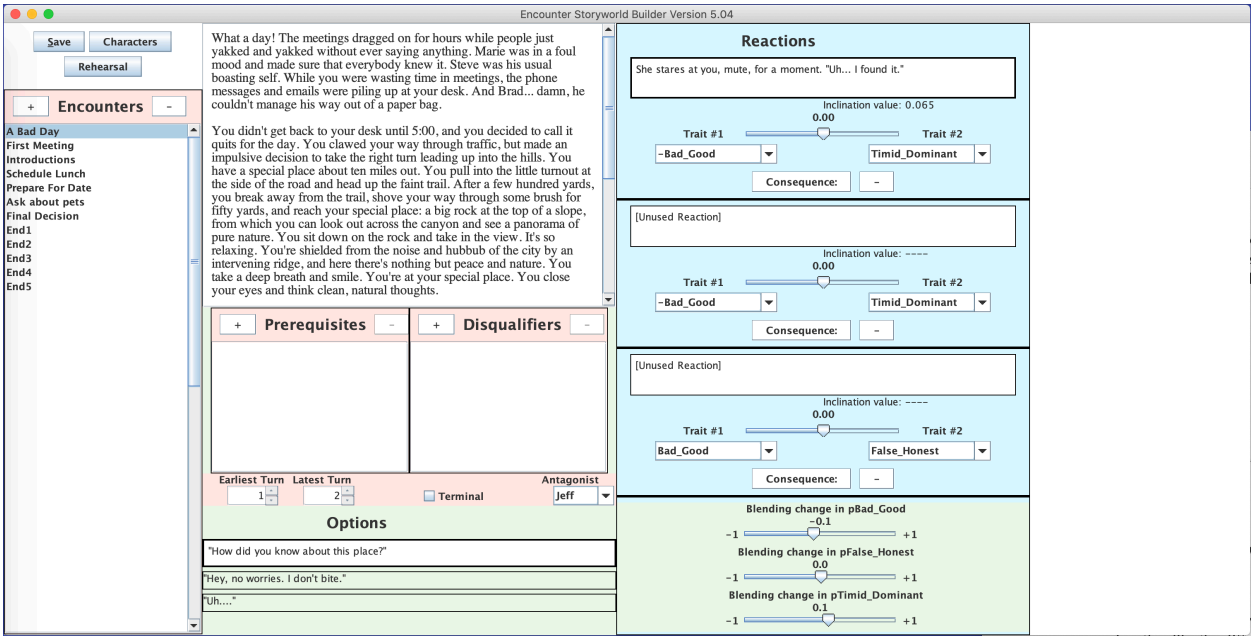
We'll spend the month of October messing around with this. I'll be issuing new versions frequently. Sometime after that, we'll organize a team to take over the project, and they can hire as many dancing girls as they want.

How to use Wumpus

Wumpus is simple to use but lacks many of the conveniences of modern programs. The difference between using Wumpus and using a conventional program is rather like the difference between camping out and staying in a hotel room.

When the program launches, it will demand a file to load. It should give you just one choice. You'll have to make the agonizing decision about this all by yourself.

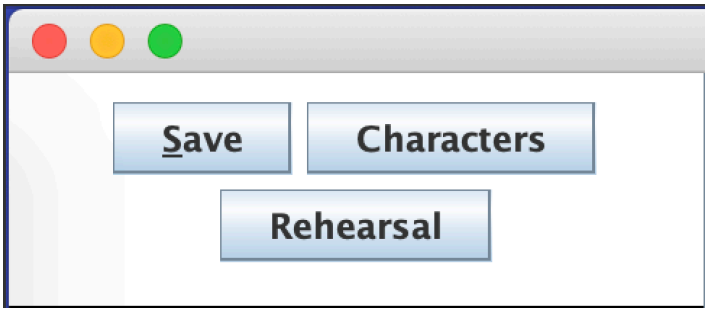
With the RomCom storyworld loaded, you'll see a window like this:



I'll move from top left to bottom right in explaining the contents of this window.

Buttons

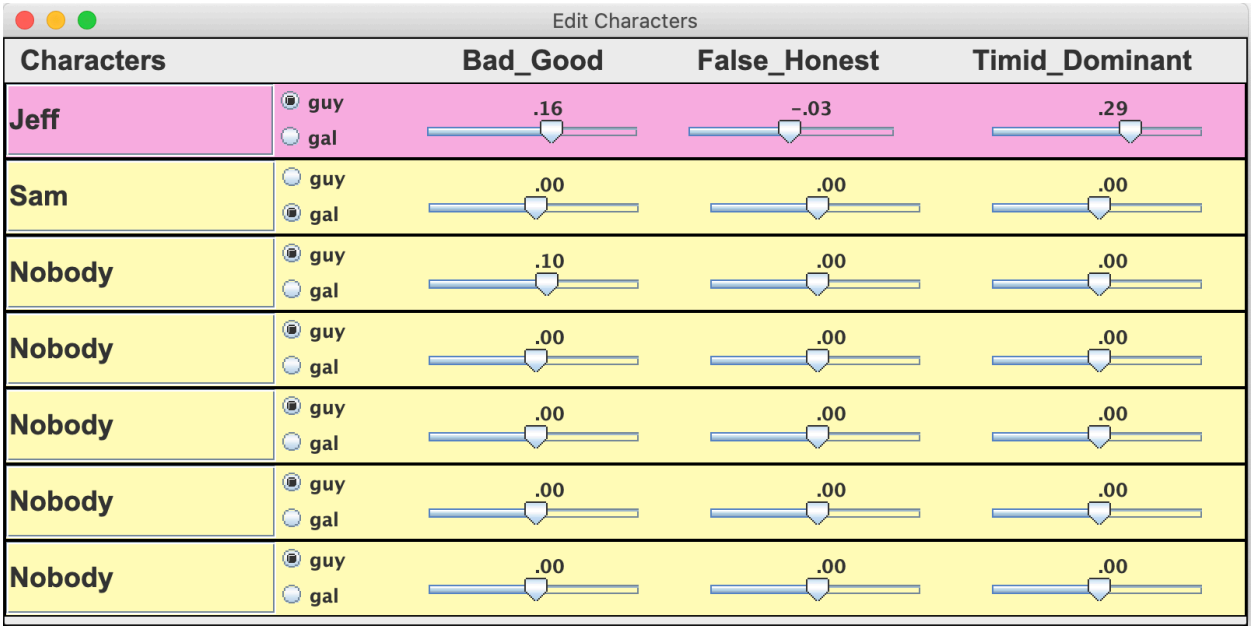
First we have three buttons for you to play with:



The first button saves the storyworld in its current state. This is the only way to save the file; there is no keyboard function for saving the file.

The Character Editor

The second button opens the character editor. When you press this button, you get a new window opening up. It looks like this:



The top row shows the Protagonist (the player). The sliders in that row show the Protagonist's personality trait values. I have set Jeff to be a slightly good fellow with a tiny trace of dishonesty and a typical male desire to dominate.

The yellow rows beneath present the other characters. However, the values in the sliders don't represent that character's personality trait values; instead, they present the yellow character's pValues towards the magenta character. In this example, everybody else is completely neutral towards other people because they've never met.

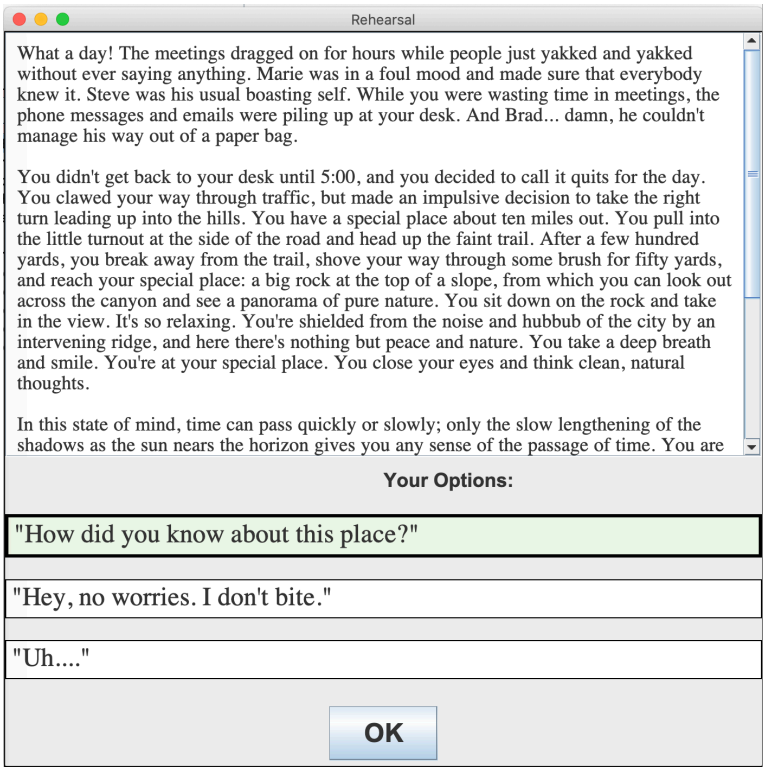
Characters named "Nobody" are never accessed by the engine. If you wish to establish a new character, just click inside the name box and type in a new name. If you wish to delete an existing character, nullify them by changing their name to "Nobody".

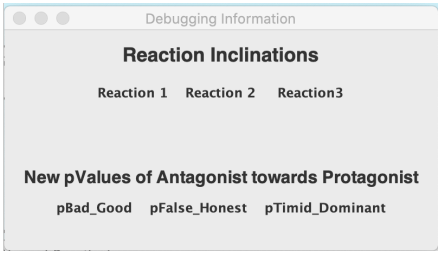
Note also that you can select the gender of each character with the little radio buttons. We have no males or females, no men or women, boys or girls in Wumpus — just guys and gals, because they have shorter labels.

You can change relationships simply by using the sliders. Unfortunately, there's a bug in Wumpus that makes the sliders jump around stupidly. I'll be complaining bitterly about this bug until I find somebody else to fix it for me.

Rehearsal

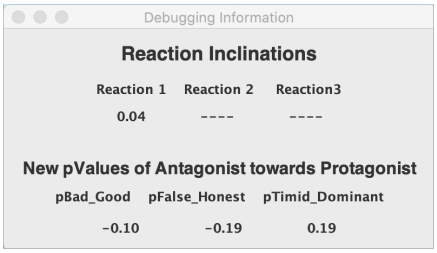
The last button raises the rehearsal. This is an even more complicated doodad:



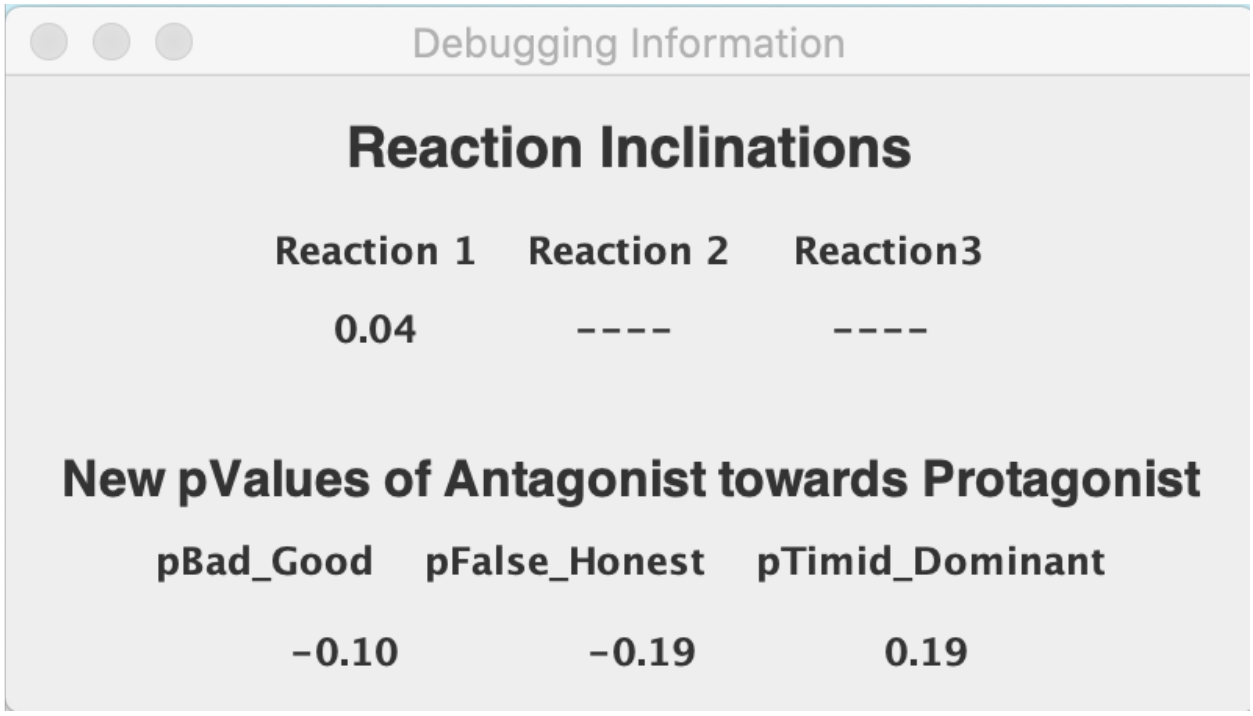


It is broken into two windows because the left window will eventually become the window that players will use, while the right window is only for developers to see. You cannot edit anything in either window; these exist only to give you information.

In the left window, you see the “introductory text” that introduces the encounter. Below it you see the options available to the player. You can execute the same commands that a player would be able to execute. You can click on one of the options to select it, and you click on the “OK” button to proceed to the next step in the story. When you do click on the OK button, you’ll see the antagonist’s reaction to your option. The debugger window will also show useful information:



Let's zoom in on the debugger window:

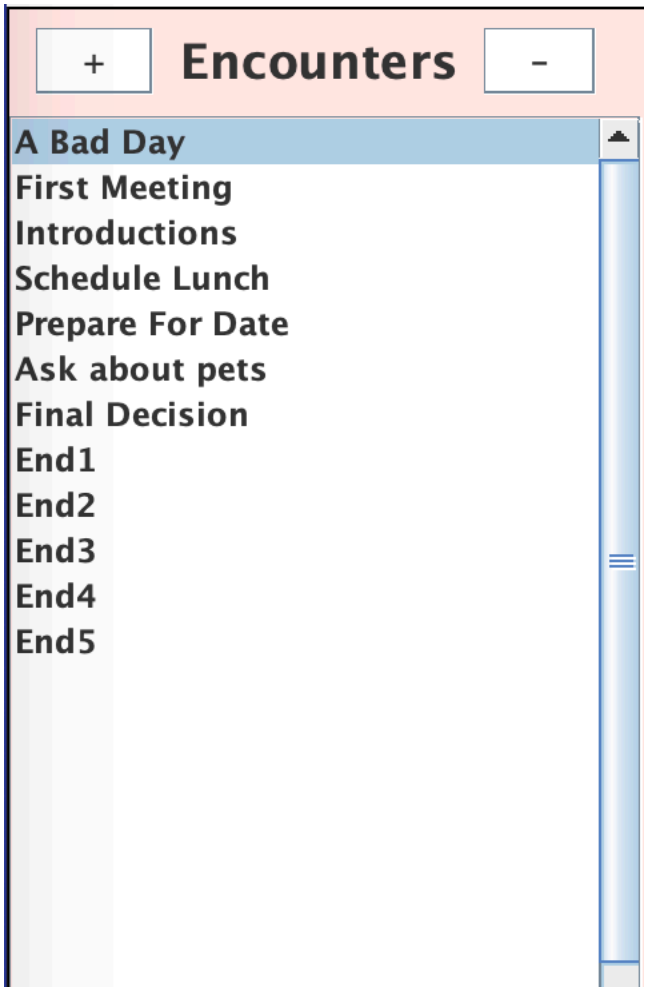


The upper row tells you how the antagonist made her decision by presenting the inclination values of each of the three reactions. In this example, she had only one possible reaction, so the other two reactions are blanked out.

The second row shows how the antagonist's pValues towards the player have changed. It shows the final results of the changes, not the magnitudes of the changes themselves. This, you will find, is very useful information when you're tuning the storyworld.

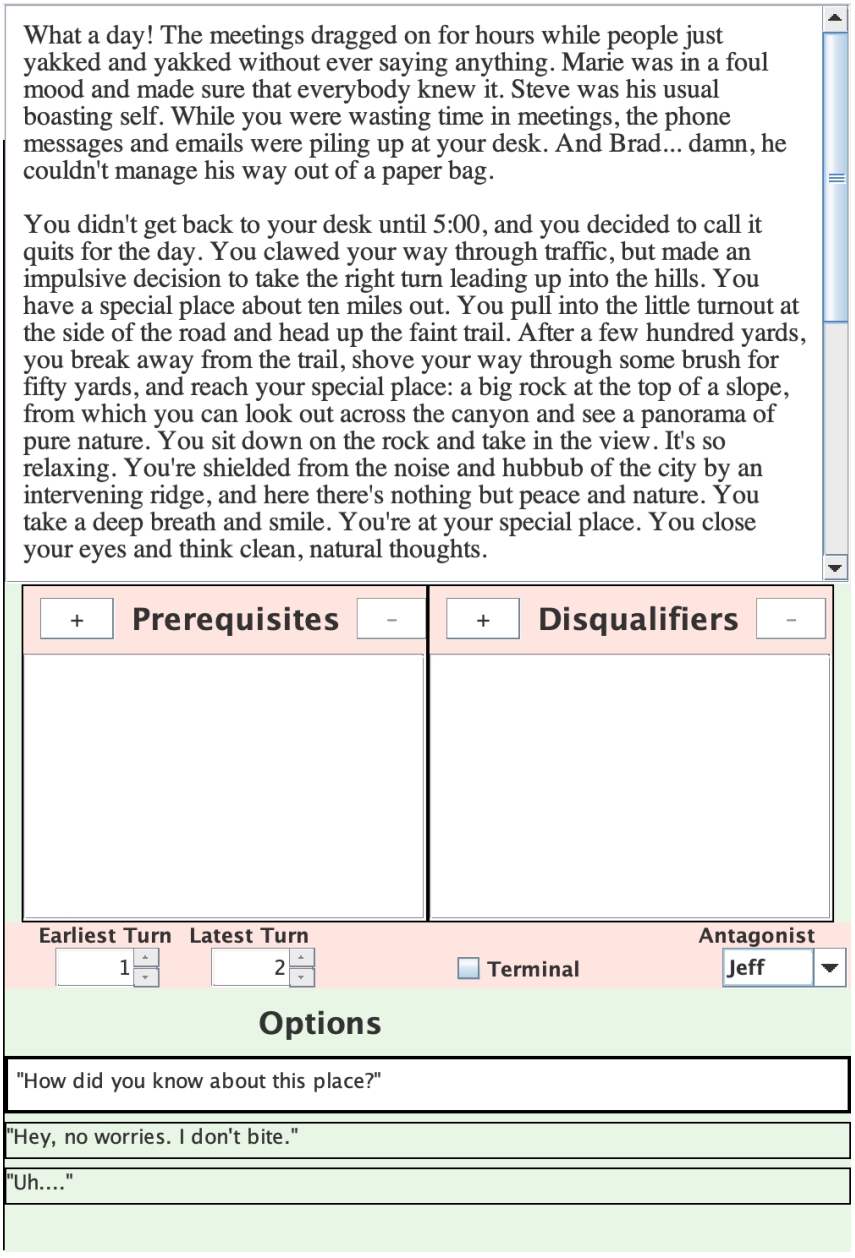
The Encounter List

Just below the button panel is the Encounter List:



This lists all of the encounters in your storyworld. The plus button at the top left adds a new encounter, which you must name. The minus button at the top right deletes whatever encounter you have selected. To edit an encounter's properties, you click on its name in this list, and all the relevant information appears in the other two columns.

The Center Column



At the top, you have the introductory text. You can edit it here, but you cannot use cut, copy, or paste. Find somebody else to yell at over that shortcoming.

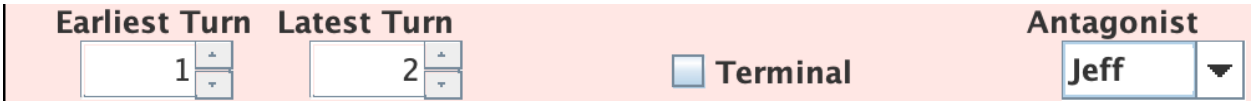
Prerequisites and Disqualifiers

These are encounters that are either required for this encounter to take place, or that prevent this encounter from taking place if they have themselves taken place. For example, suppose that the encounter you are editing is about you riding your horse. However, there is a previous encounter in which somebody gives you the horse. That previous encounter is a prerequisite to this one; you can't ride a horse you don't have. So you would enter the previous encounter as a prerequisite. You do this by clicking on the plus button to the left of the word "Prerequisites" and selecting the appropriate encounter from the pop-up list. You can delete a prerequisite by selecting it and pressing the minus button to the right of the word "Prerequisites".

Disqualifiers are encounters that would prevent the edited encounter from taking place. For example, if there's a previous encounter in which the horse is killed and eaten by a monster, then you can't go riding on it, so the encounter "Monster eats horse" would be entered in the Disqualifiers box.

Odds and Ends

Next comes a little pink bar:



The first doodad specifies the earliest turn number on which this encounter can appear. The second doodad specifies the latest turn number on which this encounter can appear. These two numbers establish a time window that the engine uses to decide which encounter to present.

Next comes the “Terminal” checkbox. If checked, then this encounter is the last in the storyworld and the engine quits after this encounter is done.

Lastly comes the “Antagonist” slot. This allows you to specify whom this encounter is addressing. This is important! Make sure that it shows the person whom the player is interacting with in this encounter.

Options

The bottom slot is where you enter the text for the three options — or fewer — that you offer your player. Usually that will be a statement of some sort, in which case you put the words inside quotation marks. However, you can also present physical actions, in which case the protocol is to present it in first person, as in:

I punch and kick the innocent young orphan.

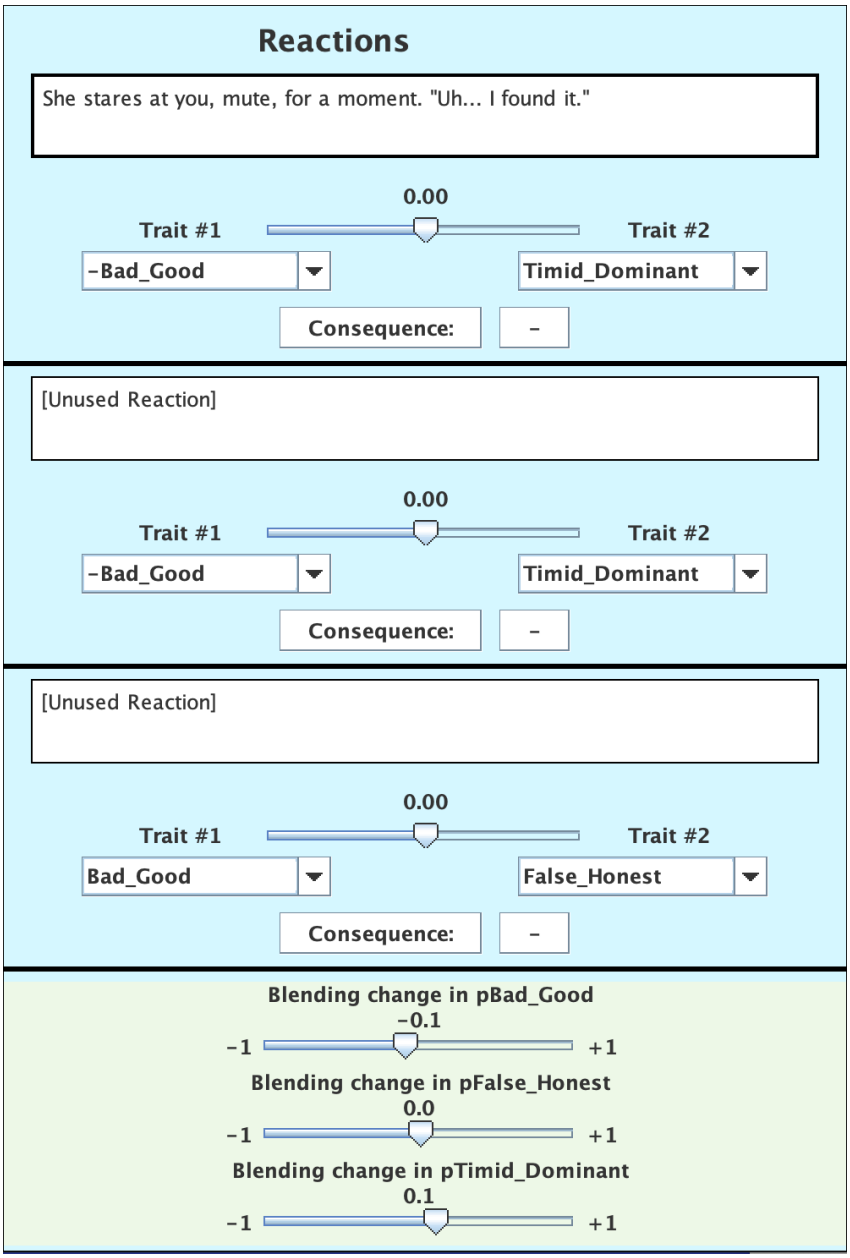
This won’t affect the behavior of the engine, it’s just the protocol we use for the benefit of the player.

If the option text is <u>exactly</u> this:

[Unused Option]

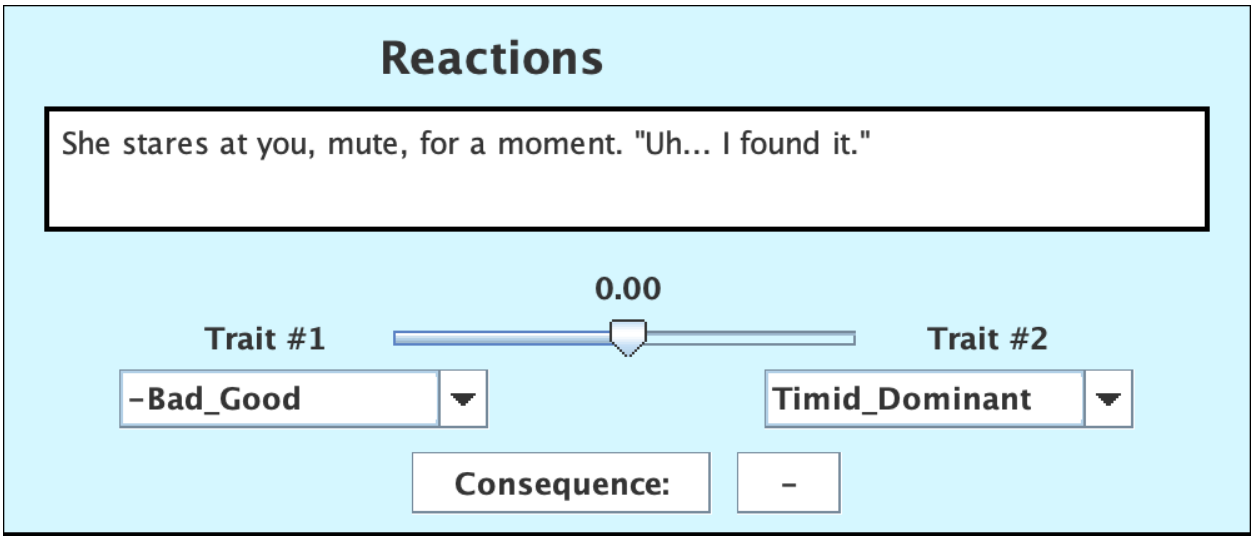
then the engine will ignore this option. But if you make the slightest, tiniest, most insignificant change in this text, then the engine WILL activate this option, and your player will end up being able to ‘Unused option’.

The Right Column



This has just two sections, but they're complicated. We begin with the upper section that specifies the three possible reactions. As with options, a reaction that says "[Unused Reaction]" will not be considered by the engine, but if it deviates from this exact spelling, the engine will use it. Do

you think that's bad user interface design? If so, go design your own feculent system!



There are seven separate doodads in this panel. The first one is easy: it's where you type in the text that the player sees when this reaction is selected by the engine.

Underneath that is a slider. This controls the weighting of the two traits specified just underneath the slider. Those traits are specified with pop-up menus. There are three traits; you can specify either the trait or the pValue of the Antagonist towards the Protagonist. Remember that you specified the Antagonist in the middle of the center column? This is why it's so important to specify the Antagonist!

You can also specify the negative value of each of the Traits and their pValues. You would, for example, use pBad_Good in situations where the Antagonist liking the Protagonist inclines the Antagonist TOWARDS this reaction, and you would use the negative value of pBad_Good in situations

where the Antagonist liking the Protagonist inclines the Antagonist AGAINST this reaction.

Lastly, at the very bottom of the panel is a little pop-up menu that allows you to specify a direct consequence for this reaction. If this reaction is selected by the engine, then the consequence specified here will be selected as the next encounter executed. The title of the consequence will be inserted between the Consequence pop-up menu and the deletion button to its right. Yes, that deletion button deletes any existing consequence.

Notes on Storyworld Construction

I urge you to begin by messing around with the demo storyworld, RomCom. I tried to include most of the basic elements of storyworld design into this demo, but its small size makes it impossible to truly demonstrate the principles of good storyworld design. I have not yet built a big storyworld, although I've been working on the Le Morte D'Arthur storyworld for a year now, and have spent a lot of time thinking about its structure as I've built it.

Small steps, not big leaps

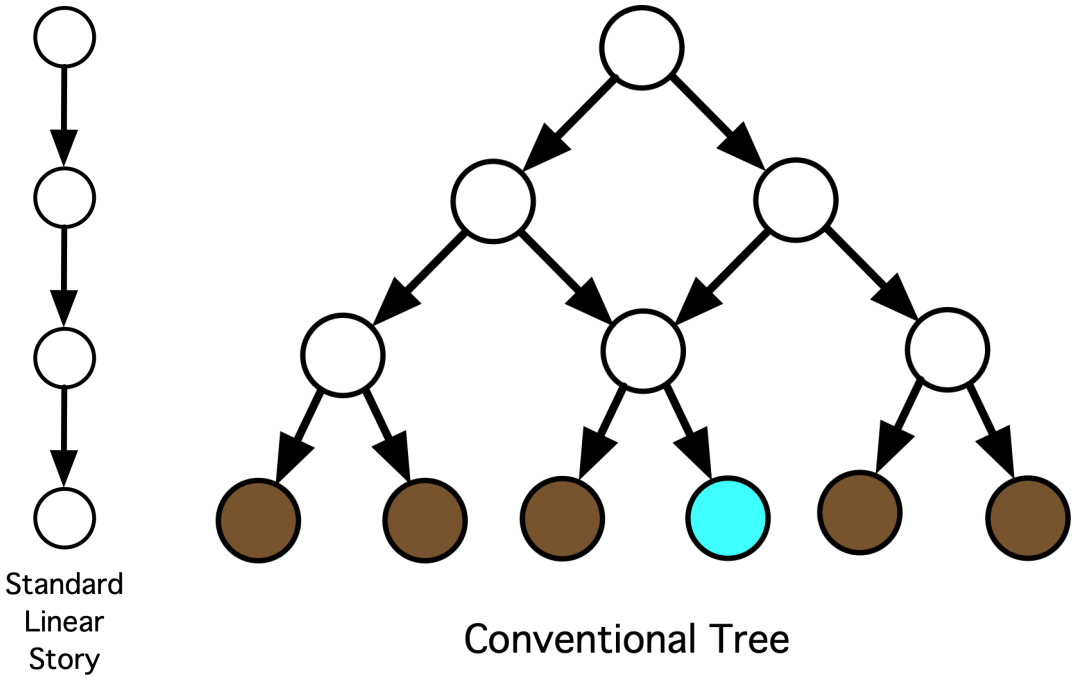
A good storyworld plays like bringing in a big fish with a weak fishing line. Unlike a game, storyworlds don't have big dramatic moments where the player decides the fate of the universe — not until the very end. Instead, the dramatic action proceeds in lots of small steps, some forward, some backward. Sometimes, the player must accept reversals, just as a fisherman must let the fish have the line when he pulls hard. At other times, the player can reel in the drama, just as the fisherman reels in the fish when it pauses.

The underlying strategy is for the player to improve the pValues of the other characters towards the player. This process takes place through a multitude of small steps. People don't fall in love after one clever comment. People don't commit to follow a leader after he does one heroic thing. Relationships develop over hundreds of small steps. Inevitably, there are setbacks, and recovering from the setbacks is a crucial part of the process.

Sometimes improving your relationship with one character hurts your relationship with another character. Balancing one gain against another loss is also part of the process of building relationships.

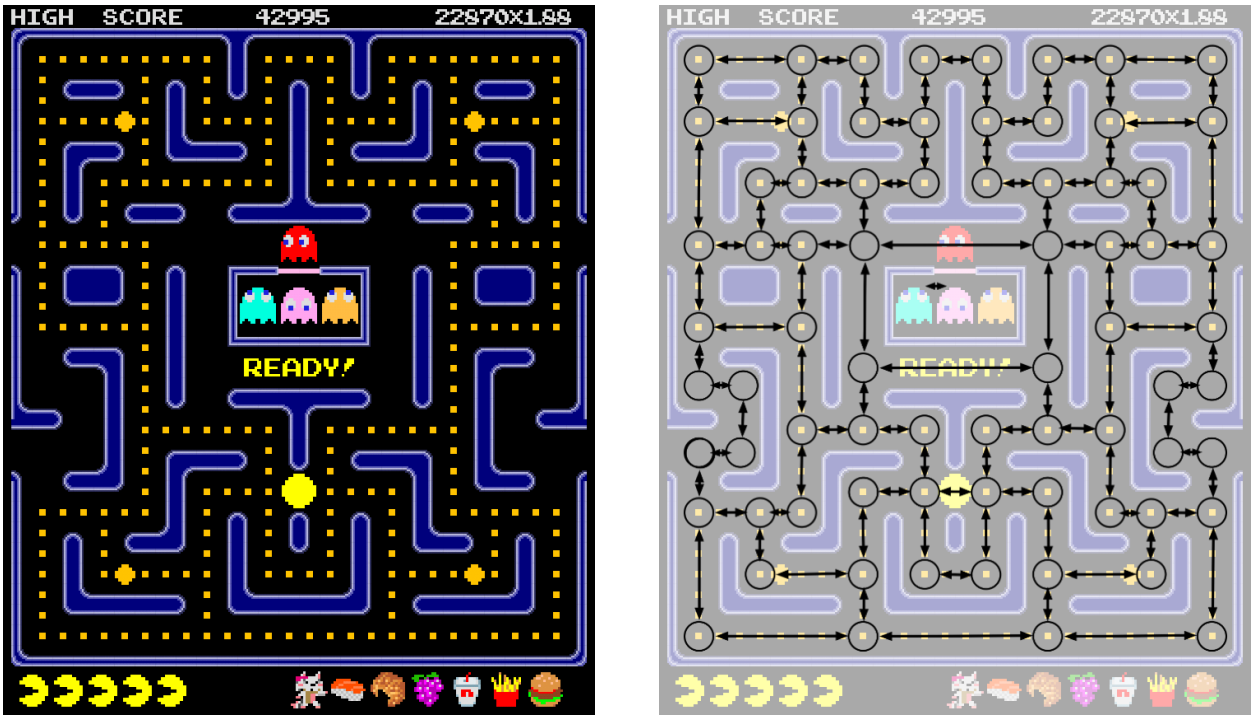
It's not the path

People who labor under the misconception that a storyworld is like a text adventure tend to think in terms of a network of nodes to traverse, with one node representing success, and that node can be reached only by tracing the correct path. This is double-plus ungood wrongthink. The player's success or failure is established by WHAT he does while traversing the network, not WHERE he goes. Theoretically, it should be possible to create a simple linear path that the player traverses every time he plays, but the player's choice of options in each case determine the outcome. Here's are some diagrams showing various networks, paths, and trees:



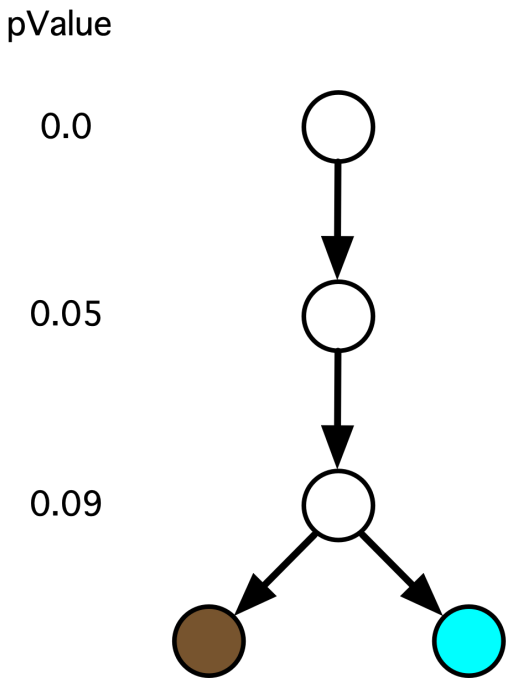
A story is simply a linear sequence of events, with no choices available to the reader. The simplest and most obvious way to make it interactive is with a conventional tree structure where the player's choices lead to different outcomes. The flaw in tree structures is the exponential explosion of nodes; nobody can build a tree big enough to provide an interesting experience.

The big step forward is a directed graph with global variables whose values change incrementally as the graph is traversed. An early and very simple version of this was Pac-Man:



The directed graph is shown on the right; the dots are the global variables that are changed by traversal through the graph. However, this is a simple application of the idea of using a directed graph with global variables; the Storytron technology took this much further with a much richer set of global variables that could be altered by the player's behavior as he traversed the graph.

Here is a very, very simple version of the graph for an encounter-based storyworld:



Here it is the pValues that are altered by the player's behavior as he traverses the simple story-like graph.

RomCom

The demo storyworld is RomCom. It presents a simple story of a guy and a gal meeting. It's nowhere near polished and perfect; there are plenty of tuning problems, but my intention is to give users the opportunity to mess around with a storyworld.

The first four encounters (A Bad Day, First Meeting, Introductions, and Schedule Lunch) are prefatory. They are meant to provide an introduction to the general situation and the characters. The player does not have any leeway for meaningful interaction. This is necessary to encounter storyworlds; the player should not have to make decisions before the basic situation has been laid down. Only after the player has gone through the basics should the player be presented with dramatically meaningful choices. This takes place in the next five encounters (Prepare for Date, Opening Conversation, What's Your Job, Are You From Here?, and Ask About Pets). These lead to the concluding encounter, Final Decision, in which the results of the player's actions are brought to bear.

There are five endpoints to the game, each reached in a different manner.