The Mechanicomicon

Joe Foley

Version 72
2007-08-16
http://web.mit.edu/foley/Public/Mech/mechanicomicon.pdf

Contents

1 Introduction

After being suckered into helping with assorted mechanics in assorted games, I realized that I ended up doing the same thing over and over. This realization led me to believe that a common toolset should be created for GMs trying to add mechanics in their games. I've tried to make a number of tools for making such mechanics easier to build, faster to build, and most importantly, less error prone.

You also probably will have wanted to read Jake Beal's "Definitive Guide to Writing Guild Games for the Rest of Eternity" as it has useful suggestions also, and I make some assumptions based upon a reader's knowledge.

2 Thinking about Mechanics

First off, before designing a mechanic, you need to figure out if you need a mechanic at all. In the *Good Old Days* there were almost no mechanics in games. Everything was done by getting information out of people and piecing it together to figure out who to kill. Well, it's not quite true that there were no mechanics – there still had to be a way for people to die. However, before getting into the nitty gritty of building the mechanic we should ask ourselves a few questions.

2.1 Do We Need a Mechanic?

As mentioned before, the guild got by for the longest time without the need of strange and complex mechanics. Most games don't actually need all the mechanics they think they do, and if they choose the wrong subset are missing important things from the game because they spent too much time on extraneous stuff.

A Mechanic should be thought of as a means to an end, in most cases the end being the completion of a plot. For example, say Jim and Ken are trying to build the Ultimate Veapon using a Toaster, Laser Pistol, and Salad Colander. Is it out of the question to say if they get all the items together they can magically create the Ultimate Veapon? Or is that too easy? Think about the following questions to determine if you need a mechanic.

1. How many people should it take?
2. How many man-hours (on average) should it take?
3. What are the win conditions, anyway?
4. What in-game things will it require to do this Mechanic?
5. Is this an important plot for a group/person?

6. How will finishing this Mechanic affect the rest of game?

2.2 What Kind of Mechanic Do We Need?

In order to make the game's mechanics cohesive, a given mechanic should emulate some part of the real world effectively. For instance, Jim and Ken (as mentioned above) might have a research notebook that tells them to get the Toaster, Laser Pistol, and Salad Colander. They then need to plug the Toaster in. This seems reasonable. A somewhat strange suggestion would be for them to go on a riddle-trail to figure out what they need to do next. This is because it's hard to explain why a dot-trail helps the players figure out how the Toaster and other items combine to make a weapon of mass destruction.

To this end, I've tried to group together most of the mechanics I've seen into categories:

- Tunnel/Packet: Packets on the walls with things you draw out of them
- Research Notebooks: ROT-N'd books with instructions and results
- Trails – Riddle, Picture: Series of clues to places on campus with dots hidden there.
- Puzzles – Trails, TimeWasters: Usually tailored to an individual mechanic
- Geographical: Clues for locations are given, then the points on the map draw something significant
- Rituals – Getting the right people/things together, and saying the right things
- Hunts – RTIs, Information, People: Finding the right things to use
- Games – Wargames, Sitdown games: For opposed mechanics, having a game for them to play is great. Usually requires a little bit of mapping though.
- Custom – Mechanics made for a given game/mechanic from scratch

Whenever possible, try to stay away from Custom unless you have lots of spare time. It allows you to make interesting mechanics, but just takes way too long.

2.3 Desirable Traits in a Mechanic

There are a number of things that you should make sure you do when using/designing a mechanic:

1. **Make sure it is not GM intensive:** You won't have enough time to deal with all of the mechanics and keep your game running if you don't follow this trait
2. **Make it easy to recreate:** PhysPlant regularly grunges things. This means packets, paper, etc. will randomly disappear. To prevent this from screwing up your game, make sure you can replace parts of game if they vanish and do so quickly. Also, make sure you know where everything is!

3. **Document and make the Instructions Simple:** Make the instructions simple enough that you can hand them off to a helper or another GM when you don't have the time to take care of it. Remember, GM job security is a big no-no. You never want to be the only person who knows how to run something[1].

4. **Simple to Create, a Lifetime to Master:** You want your mechanic to be interesting but don't want to spend forever building it. This means you want them to be one-way problems in general – easy to create but hard to solve. If it's too easy, then whats the point?

5. **Randomness is your Friend:** You can confuse people into thinking you have put a lot of effort into placing a number by having them mostly be different. You can get the exact same effect by making them random. Another way to think of it is that *when the number doesn't matter, pick randomly!* This also means you can write a script to autogenerate the numbers... one less thing to think about!

6. **Categorize by Threes:** This is mostly relavent to when you need to assign stats that relate to a mechanic. Usually you just want to create three classes of stats: good, bad, and average. There of course can be exceptions to these numbers, but they will be the special cases.

There are also some general tests for a mechanic[2]:

1. **Brute Force:** *Can I sit down with a computer or pencil and paper and just optimize the mechanic to death?* Make sure that there is either too much complexity/hardness or enough randomness that someone can't just sit down and figure out the guarunteed way to win. Deterministic systems aren't nearly as much fun as ones with a little chaos.

2. **Rationalization:** *Well, I thought you meant this so I did it, but I didn't bother asking about it later because it made sense to me.* Make sure that someone cannot delude themselves into doing things that you don't want to do. Many times this is from having too many grey areas or strange interactions. It also stems from not having crystal-clear goals listed at the bottom of their sheet. Sometimes its just the person, and there is not much you can do about it other than to cast them carefully.

3. **Adrenaline:** *Can I use my real-life boosted reflexes and martial arts training in the SIK game?* Make sure that people who have significantly stronger muscles don't accidentally kill/injure other players through your mechanic. This usually requires putting

[1] Nanopunk GM One:"Oh no! Ken's passed out, and the tunnels aren't up!"
Nanopunk GM Two:"Put up a time-waster, we'll deal tomorrow"

[2] You may know other names for these, but I chose ones that wouldn't be hard to understand after a couple years.

in annoying rules that are neccessary to keep them from having to check their swings[3]. [4]

4. **Stupidity:** *Can I completely misunderstand the mechanic and make it no fun for other players when I screw up?* Make sure that your mechanic is simple to use and that the directions are clear. Also, put in checks in the mechanic so that other people(GMs or players) check the person's use of the mechanic from time to time. The mechanic should also be designed such that a clueless player does not eit interactions with a clueful player. Their cluelessness should only affect their own play if you've done partitioning correctly. Lastly, put in some fake entries/metagame traps, so that the person using the mechanic knows when they've messed up.

3 Tunnel/Packet

4 Research Notebooks

I've created a tool to help in the construction of these little devils. It's in `Notebooks` as `maketrees.pl`. Use of it is mostly intuitive, except for the RANDSEED option. The perl script cheats on its pseudo-randomization algorithm to make sure that it creates the same trail order if you run it again after fixing a typo.

5 Trails

Since trails are something that are easy to screw up and need to be maintained often (dots fall down, people complain that something isn't there when it is), I wrote a couple of scripts for generating/maintaining the trails. They live in `Trails`.

5.1 Riddle Trails

These tend to be really easy to make too hard. The best way to generate them is to wander around campus with a tape recorder in hand describing the interesting things you see.

[3] Another good way to avoid this is to make sure that props are very safe, and that there isn't boffer combat.

[4] Notice that I didn't say they couldn't injure themselves...

5.2 Picture Trails

The digital age has made these much easier to deal with. You wander around and take pictures of interesting but obscure things on campus. I've created a tool to help catalog these and actually turn them into a series of dot numbers.

6 Puzzles

I don't have much to say in the way of puzzles, other than to make sure someone actually tries doing them other than the person who picked them. This is known as "sanity-checking", something that doesn't happen enough in games.

7 Geographical

To make these, you'll need a map of MIT. You can get these from the Information Center at Lobby 7. Figure out the points that you want to be in the series and write down room numbers. Figure out if you're going to make a Riddle or Picture Trail, and follow the directions for that.

8 Rituals

9 Hunts

10 Games

Try to find mechanics that have been marketed and tested to include in your game. One of my best examples of this was a drug-running mechanic that used a modified Battleship. Battleship was also used later for a sub-hunting game. You can see how both of these mechanics required a "search and destroy" style of task that worked well with Battleship.

Some of your mechanics may require "teaching" of a skill to another player. I recommend Zendo as a good style of game to look for. Part of Zendo is the "Master" trying to teach a "Student" a rule through "Koans" which are effectively little tests of the rule.

11 Custom

I have been dumb enough to want a custom mechanic. To this end, I spent way too much time playing with perl and postscript. However, you can profit from my stupidity. I've included in the `PostScript` directory a file `satellite.ps` which has a number of the crazy hacks I did to make postscript follow my bidding. You'll need to understand a little postscript before using it, but it should help you get started.

12 Other Useful Stuff

12.1 Dist

Quite often there are public documents(rules, public mechanics, etc.) that you need to distribute to a bunch of places. In the `Dist` directory is a sample `updatepublic.sh` for updating a webserver and a AFS Public directory with the latest documents.

12.2 Crypt

Sometimes you want to make the person work a little for the answer to their problem. One simple way to do this is by doing a single-letter substitution cypher. I wrote a simple little tool for generating these randomly: `Crypt/gencypher.pl` which will create two little perl scripts that you use for encrypting text or decrypting it.