CARPS - Customizable Abstract Roleplaying System Version 4.0

Download (109Kb .pdf)

By Nick Thomas

Well, here it is; version 4.0 of CARPS. In the making of this, I trashed every single rule. This is a completely different game system than 3.5. I think you will find it quite drool-worthy. A lot of the rules seem kind of crazy and pointless, but they are all there for a damn good reason; don't question the RPG designer! At the bottom of the page I have added a justification for every rule. Finally, this is not for beginners. This is pretty advanced and makes no attempt to explain the basics of roleplaying. Newcomers would be better off with Version 3.5 or CARPS lite.

Object Description

Objects are anything in the universe. This means living creatures, inanimate objects, and even forces of nature, such as gravity. The following sections will explain how objects are described in CARPS. Other RPGs use skills and ability scores, but CARPS is a little different, as you will discover.

Layers Basics

Objects are described in terms of layers. There is a flowchart displaying all of the objects, categorized into layers. There are any number of layers, each becoming more precise, until the final layer represents a specific object. If this makes no sense, let me explain using an example. Okay, we have Bob the Human. He is our fifth layer. The fourth layer is the generic human, from which he inherits humanlike traits. The third layer is the generic mammal, from which he inherits mammal-like traits. The second layer is the generic animal, from which he inherits animal-like traits. The first layer is the generic living creature, from which he inherits the traits of living creatures. Of course, bob isn't the only thing on the flowchart. There would also be other humans stemming off from the generic human, other species of mammals stemming off from the generic mammal, and so on. Okay, so we know Bob is a Human, a Human is a Mammal, a Mammal is an animal, and an animal is a living creature. In CARPS, you would shorthand that sentence like this: bob.human.mammal.animal.living creature. This makes traveling up the flowchart much less painful. By the way, it is called "dot notation".

Note: I would like to note that about a week after making up this rule, I came across an RPS using the same mechanics, namely Alternate Realities. This is purely a coincidental great-minds-think-alike type thing, and I am leaving this note here to avoid getting emails accusing me of copying off of AR. I got this idea looking through the DOM in my JavaScript manual. If you would like to email me for any other reason, you may do so here.

Properties

Okay, so we know that Bob is a human.mammal.animal.living creature, but that doesn't tell us much about Bob as an individual. Heck, we don't even know anything specific about humans, mammals, animals or living creatures! This is where Properties come in. All objects have properties. A property is an aspect of an object which describes them; if objects are nouns, properties are adjectives. For instance, one of Bob's properties might be his ability to swing a sword real well. Note that the bottom layer isn't the only one with properties. All of the other layers have properties, and the bottom layer "inherits" these properties. Thus, if one of a human's properties is that they have arms and legs, Bob has arms and legs.

Values

At this point we know that Bob can swing a sword pretty well, but we don't know how well. At risk of overusing the phrase, this is where values come in. Each property has a value assigned to it; this value represents its effects. A value is described using either quantative or qualitative data. Quantative data is a number. It is used when the value represents a quantity of something or another. Let's go back to Bob. His ability to swing a sword pretty well would be quantative data (his amount of skill using a sword). Now, qualitative data is more complex. Qualitative data encompasses anything that is not measurable (i.e. is not a quantity of anything). Bob's posession of limbs (inherited from the human layer) would be a qualitative value. I guess that's about it for values.

More on Layers

This is simply a list of bits and pieces about layers that I omitted earlier for simplicity.

- The flowchart of objects in reffered to as the UOM (Universal Object Model).
- Each object should have its properties listed on the UOM, but specific objects shouldn't be listed at all.
- No object on any layer should ever connect to more than one higher-level object. If you have plant-being as an object, it must have its own section on the UOM.
- When making a non-bottom-level object, you must be sure that all properties you give it will apply to every object that inherits them. For instance, the generic human could not say that it was 5 feet tall, otherwise all humans would be 5 feet tall. The exception is for mutations, such as midgets and siamese twins, who are exceptions to the rule.
- Sometimes a non-bottom-layer object will have a property (that all objects branching off of it will inherit) but not a specific number. In this case, the property is listed, but not given a value. All properties of all bottom-layer objects must have values.

Action Resolution

Action resolution implies the determining of the winners of contests between objects. Contests can be as simple or complex as objects themselves. If Bob were trying to swing a sword at Jim the Orc, it would be a contest between Jim and Bob. If Bob were trying to climb a steep slope, it would be a contest between Bob and the force of gravity. These contests are referred to as checks. Note that not everything Bob does would require a check; when the objects are coexisting peacefully no check is needed. He wouldn't need a check to start his car, but would need one to break into someone else's car.

Check Methods

Every check will have a method with which it is resolved. The default method is that each object will take one of their quantative properties (appropriate to the task), weigh them against each other, and whichever has the greater property prevails. Here's an example: *Bob swings his sword at Jim. Bob has a sword-swinging property of level 7, and Jim has a dodging property of 4. Bob cleaves through him.* Anyone who has played an RPG before will know that this is a really stupid check resolution system. Well, I'm not done! Read on.

Variables

Almost all of the time, there is more going on than Bob and Jim. There will no doubt be outside forces interfering, no matter how subtle. Maybe Jim is wearing heavy armor that inhibits his dodging ability, for instance. Maybe Bob just got lucky. This type of thing is described using variables. No connection to the type of variable found in Algebra; don't worry. Okay, so here is how variables work (finally). Whenever there is a condition in effect which would hinder or help a participant, it is translated into a numeric value with size in proportion to its significance and either added or subtracted from the relevant property of the one affected, though only until the check is resolved. In english please! Okay, if Jim is having trouble dodging Bob's blade due to his armor, he gets 2 subtracted from his dodging property value of 5. This leaves him with 3. Please note that Jim has not lost two points off of his dodging value permanently; it is only decreased for purposes of this one task. Of course, there are any number of variables you could come up with for a given task, but that would eat up an awful lot of time. Instead of factoring in every little variable, you could just examine the general situation and decide whether the odds are in or against Jim's favor, then do the same for Bob, and give each of them general variables based on the analysis. Oh yes, for clarification, qualitative values are applied as variables whenever they are relevant.

More on Methods

The smarter readers will have noticed that I failed to explain non-default methods in the Methods column. Well, methods are pretty open-ended. Actually, totally open-ended. Whenever the default method is unsatisfactory for a certain check or something like a check, you can add on to or rework it to create new methods to be used for certain tasks. Now, some other important points about checks:

Multi-Participant Checks:

Many checks will have more than one participant. For these checks, simply modify the method in whatever way makes sense.

Multi-Property Checks:

Some (if not most) default-method Checks will use more than one relevant property. For these Checks, simply add together the relevant properties' values for each object. Let's get back to Bob and Jim. Bob has a sword skill property of 5, and a strength property of 6. Jim has a dodging skill property of 3, a reflexes property of 4 and an agility property of 5. Bob adds together his two properties; 5 + 6 is 11. Jim adds together his two properties; 3 + 5 + 4 is 12. We'll ignore variables for now for simplicity. Since the sum of Jim's relevant properties exceeds the sum of Bob's relevant properties, Jim manages to dodge the sword blow.

Conditions

Things happen to objects that affect their properties. If Jim gets wounded, how does this affect his performance in combat? This type of thing is described using conditions. Whenever a significant but temporay change is made on an object, it is recorded. This is a condition. Whenever the condition would have a significant impact on the object, it is applied as a variable. Sometimes an object will have to undergo a check due to a condition. For isntance, Jim might have to undergo a task to avoid keeling over upon being slashed with Bob's sword. But what would he make the task against? This brings us into our next topic, unopposed checks.

Unopposed Checks

This is a type of check in which one object needs to make a task but there is clearly no object to oppose it. In this case, a Target Number (TN) for the check is set by the GM to use in place of a property of another object. You can also use unopposed checks whenever you don't feel like figuring out a bunch of objects and properties. It's a good shortcut.

Permanent Changes

Some conditions don't ever wear off. Conditions like this don't have to be recorded; just make the

appropriate changes to properties. This type of change can also include switching to a different higher-level object (for instance if Bob got turned into a frog by a witch, his new higher-level object would be a frog, not a human). Technically this can also happen with conditions.

Miscellaneous

The Nitty-Gritty

These are assorted little details which don't belong anywhere else or have to do with metagame (outside the game) stuff.

Who makes up what?

Who makes up the properties for objects, and who determines their values? For that matter, who makes up the objects themselves? Well, players make up their characters and all of their properties (including their values; players who abuse this rule should recieve their just desserts), though the GM can veto properties which he deems to be too powerful, inappropriate, or not allowable in the setting of the campaign. With the GM's permission, players can also add other objects to the UOM. The GM makes up the rest of the objects in his world and determines their properties.

What numbers do I use?

You may be asking yourself what numbers to assign to your properties. Well, this is where the Relativity Principle comes in. Don't worry; this isn't rocket science. All this means is that everything is relative. You can have your numbers as large or as small as you want, as long as they are in proportion to each other. You can even go into negatives if you want. For instance, you could say that Bob's sword skill was 5 and Jim's dodging skill was 3, or you could say that Bob's sword skill was 500 and Jim's dodging skill was 300, and there would be no difference. It all depends on how precisely you want to measure your values. The only goal is to create an internally consistent environment.

Concerning layout of the UOM

- Don't display bottom-level objects on the UOM.
- Don't display insignificant objects like pencils on the UOM. As a rule of thumb, only display objects whose lower layers are involved in tasks.
- Write down objects' properties next to their names on the flowchart.
- Don't try to write down all significant objects beforehand. Just write down a few bare-bones objects and add more as you need them.
- Don't worry about writing down all of the properties of an object beforehand. Just write down a few basic ones and add more as you need to. You could even lump together the basic ones. For instance, all physical traits that define humans could be lumped together under the qualitative property of 'basic human traits'.

Advancement

Most other game systems have at least a basic guide for tracking the advancement of characters. CARPS has no such thing. Actually, it does, in the form of permanent changes, but no specific method for determining how much an object's property's value increases when. This is impossible anyway, due to the relativity principle. This gives the GM the opportunity to increase characters' values by however much he sees fit whenever he sees fit.

Rules Justifications

Why Objects? By using objects instead of PCs, NPCs, and inananimate objects, CARPS allows for very weird settings and complex situations by taking away the limitations of only describing animate objects.

What's With the UOM? This thing has an array of benefits. The most important one of these is that, by predefining attributes of an object of a certain type, you shave time off of the creation process, make sure that all objects of a certain type have certain properties without using attributes and thereby giving bricks dexterity properties. The UOM also offers quick reference (instead of slow reference looking up stuff in books). Need I say more?

Properties: Less rules are better than more rules, right? Well, with properties (instead of the skills, traits and attributes as well as unpsoken physical traits of CARPS 3.5), everything is rolled up in one little rule! In tandem with the UOM, this allows for Attributes to be used on multiple levels (including multiple sets of attributes!). This results in a much more flexible and overall awesome system.

Methods: While most checks can be resolved using the default method, not all of them can. Nor is it possible to create a method for every check imaginable. Therefore, the method system is open-ended to allow for users to tailor it to their needs.

Variables: This is an extension of a rule which exists already in most RPGs, and seems to work, therefore it should be kept. If it ain't broke, don't fix it.

Why no Dice? It seems to me that there is no true randomness in the world. If Bob swings a sword at Jim in exactly the same way as the last time and Jim does the same thing in defense, the same thing will happen, assuming the same conditions as the last time. Duh. This means that the players will have to vary their routines to get the results they want.

Conditions: It seems that all other RPGs are lacking in this area. They have open-ended rules for everything else, and have to waste time with the exact effects of every little thing because they don't have conditions. Conditions are a real innovation in RPG design.

Unopposed Tasks: This is how almost all checks in most other RPGs work. It is needed only rarely in CARPS, but still useful. It also makes for a good shortcut. Once again, if it ain't broke, don't fix it.

Credits

Concepts: Nick Thomas
Writing: Nick Thomas
Editing: Nick Thomas
Publishing: Nick Thomas
Ideas and Inspiration, with all due respect:
GURPS, **FUDGE**, **Risus**, **Alternate Realitites**, **and of course SLUG.**