Rule Modeling Exercise: Roll Initiative!

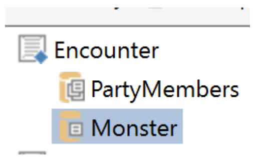
Getting started with rule authoring can feel overwhelming at first. To help with this, one of the things I suggest to our users after they finish Rule Authoring training is to take a process that they know very well and model it in InRule. So for this blog post, I thought it would be a nice exercise to model a fun process…Dungeons & Dragons!

Everybody roll initiative!

The Greatest Adventure Is What Lies Ahead

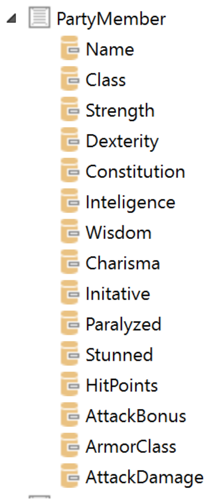
For this exercise, we're going to model some simple combat rules for D & D. For rule authoring, we usually start by building our entity structure or schema. For combat in D & D, you will have your party of adventurers and their opponent, usually a monster.

So we will have a top level entity representing the Encounter with a collection of Party Members and a Monster to fight.

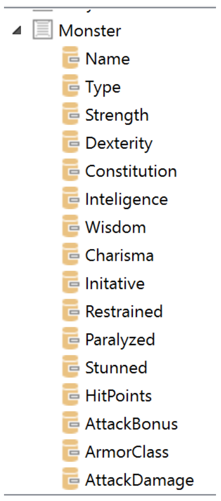


Encounter
PartyMembers
Monster

Next we'll need to define the attributes and statistics for both the party members and the monster. We'll add fields for the name, class, and statistics for their major attributes, Strength, Intelligence, Dexterity, Wisdom, Constitution, and Charisma for the adventure party as well as the monster. Plus since this is a combat simulator, we will need to also include some fields for their Hit Points (how much damage they can take), attack type (whether it is a Dexterity or Strength based weapon attack or a spell), attack bonuses, damage bonuses, their initiative (which determines in what order the party and the monster will attack), plus a few status indicators for conditions like "Paralyzed" and "Stunned". In the end we will have an entity structure like so for our Party, which we can also use for our monster.



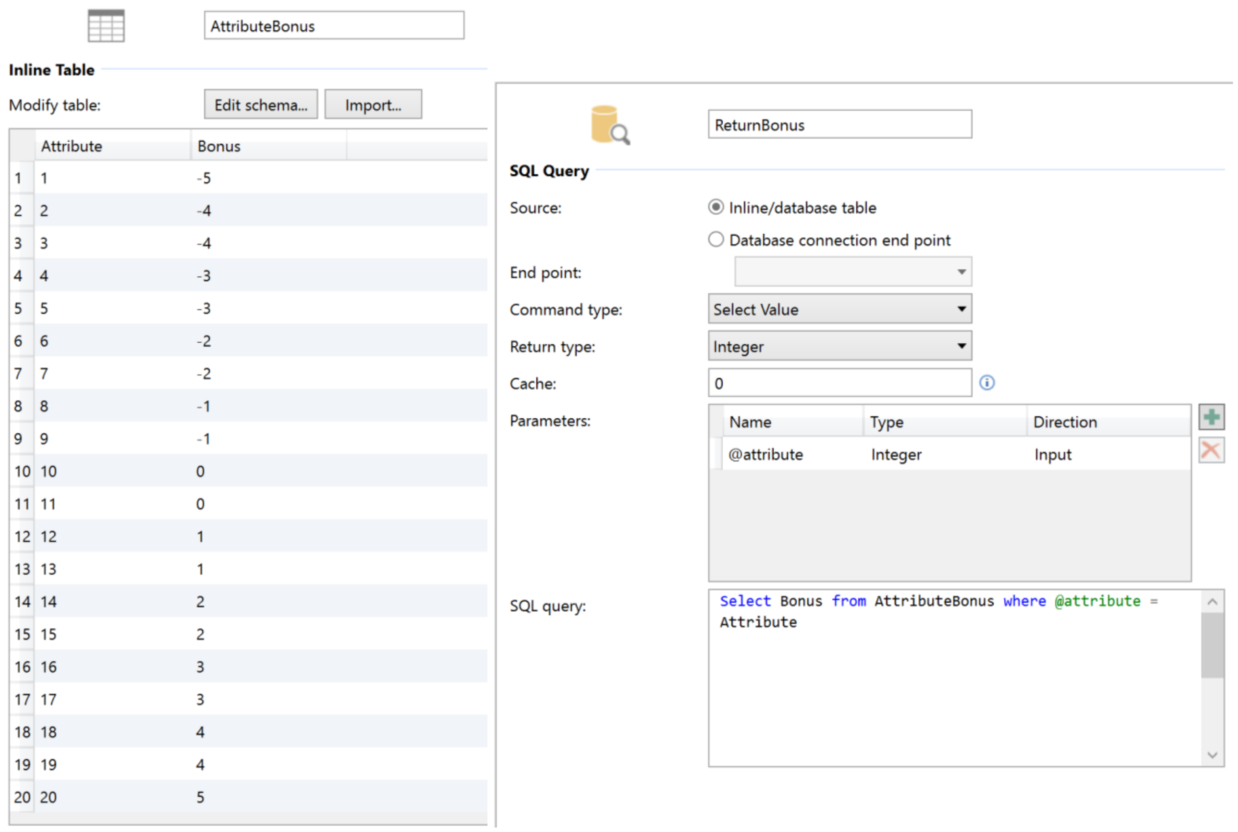
PartyMember
Name
Class
Strength
Dexterity
Constitution
Inteligence
Wisdom
Charisma
Initative
Paralyzed
Stunned
HitPoints
AttackBonus
ArmorClass
AttackDamage



Monster
Name
Type
Strength
Dexterity
Constitution
Inteligence
Wisdom
Charisma
Initative
Restrained
Paralyzed
Stunned
HitPoints
AttackBonus
ArmorClass
AttackDamage

Knights of the Lookup Tables

One common feature of table top role-playing games is the use of lookup tables. In this case, attack bonuses are determined by their bonus score attached to the primary attribute for their class. For example, a fighter would have an attack bonus based on their Strength score while a Wizard would have a spell attack bonus based on their Intelligence score. So we can create a generic stat bonus table with a quick SQL query to return the bonus based on the attribute value.



AttributeBonus

Inline Table

Modify table: Edit schema... Import...

	Attribute	Bonus
1	1	-5
2	2	-4
3	3	-4
4	4	-3
5	5	-3
6	6	-2
7	7	-2
8	8	-1
9	9	-1
10	10	0
11	11	0
12	12	1
13	13	1
14	14	2
15	15	2
16	16	3
17	17	3
18	18	4
19	19	4
20	20	5

ReturnBonus

SQL Query

Source: [✓ Inline/database table] Database connection end point

End point:

Command type: Select Value

Return type: Integer

Cache: 0

Parameters:

Name	Type	Direction
@attribute	Integer	Input

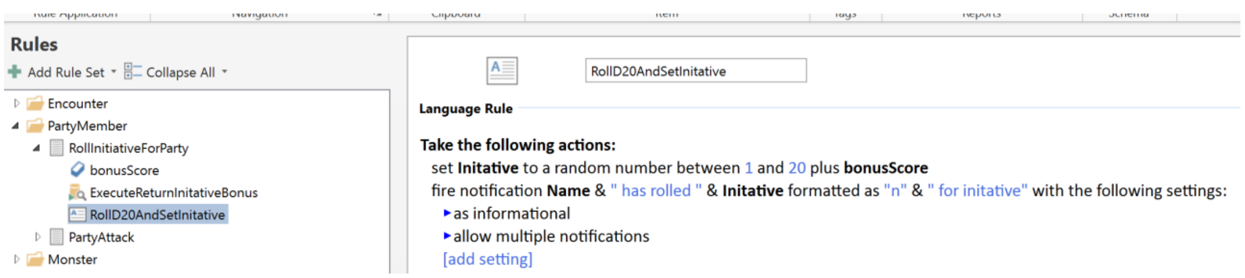
SQL query:

```
Select Bonus from AttributeBonus where @attribute =
Attribute
```

Roll Initiative!

So next, we need to determine the order, or initiative, in which our party members or monster will engage in combat. In the tabletop world, this would be determined rolling a 20- sided dice and adding the bonus to the Dexterity bonus. For our rule, I used the “Declare Variable” action, and then set that variable using the “Execute SQL Query” action.

I then used a Language rule to set the Initiative field using a generated random number between 1 and 20 and added the Dexterity bonus. Finally, I used the “Fire Notification” action to provide feedback on the results of the Initiative roll.



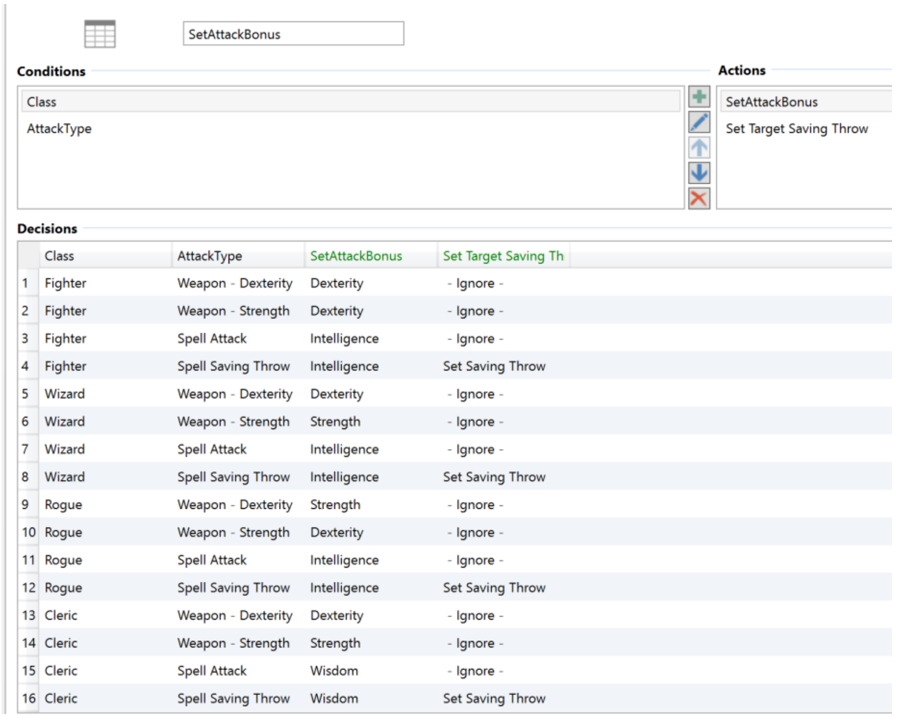
Take the following actions:
set **Initative** to a random number between 1 and 20 plus **bonusScore**
fire notification **Name** & " has rolled " & **Initative** formatted as "n" & " for initative" with the following settings:
▸as informational
▸allow multiple notifications
[add setting]

Let's Get Ready to Rumble!

Now that we have our order determined, we started building the attack rules. For our party, we first need to determine the attack bonus. Different classes can have different attack bonuses depending how they attack, for example our fighter swinging a heavy axe would use the bonus score from his Strength score while a nimble rogue with a bow would use a Dexterity based attack.

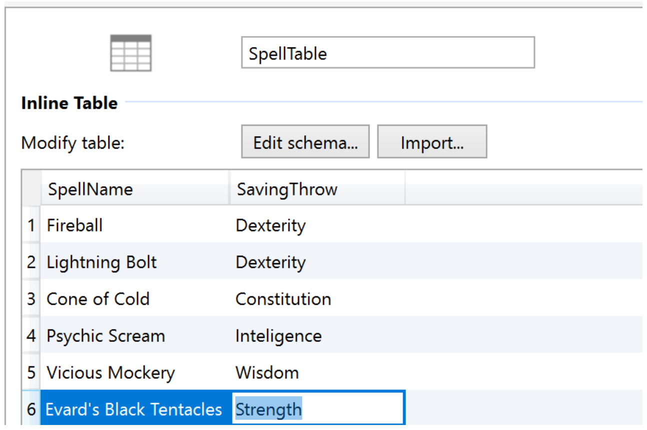
Spell casters add an additional point of complexity in that some spells use an ability based attack roll. For example Wizards would use their Intelligence for a spell whereas a Cleric would use Wisdom. Other spells require a saving throw to be made by the target to avoid the worst of the spell's effects.

Because of this complexity, I used a combination of a decision table and the Execute SQL Query action to define the bonus based on their class and how they are attacking.



	Class	AttackType	SetAttackBonus	Set Target Saving Th
1	Fighter	Weapon - Dexterity	Dexterity	- Ignore -
2	Fighter	Weapon - Strength	Dexterity	- Ignore -
3	Fighter	Spell Attack	Intelligence	- Ignore -
4	Fighter	Spell Saving Throw	Intelligence	Set Saving Throw
5	Wizard	Weapon - Dexterity	Dexterity	- Ignore -
6	Wizard	Weapon - Strength	Strength	- Ignore -
7	Wizard	Spell Attack	Intelligence	- Ignore -
8	Wizard	Spell Saving Throw	Intelligence	Set Saving Throw
9	Rogue	Weapon - Dexterity	Strength	- Ignore -
10	Rogue	Weapon - Strength	Dexterity	- Ignore -
11	Rogue	Spell Attack	Intelligence	- Ignore -
12	Rogue	Spell Saving Throw	Intelligence	Set Saving Throw
13	Cleric	Weapon - Dexterity	Dexterity	- Ignore -
14	Cleric	Weapon - Strength	Strength	- Ignore -
15	Cleric	Spell Attack	Wisdom	- Ignore -
16	Cleric	Spell Saving Throw	Wisdom	Set Saving Throw

Next we need to determine if the attack will require a saving throw or would be a regular attack. So for this point I will branch the logic using an "if then else" statement. If it is a Spell requiring a saving throw, we need to roll a D20 and then the target would add the bonus to their attribute rated saving throw. I created short spell table and what saving throw is required.



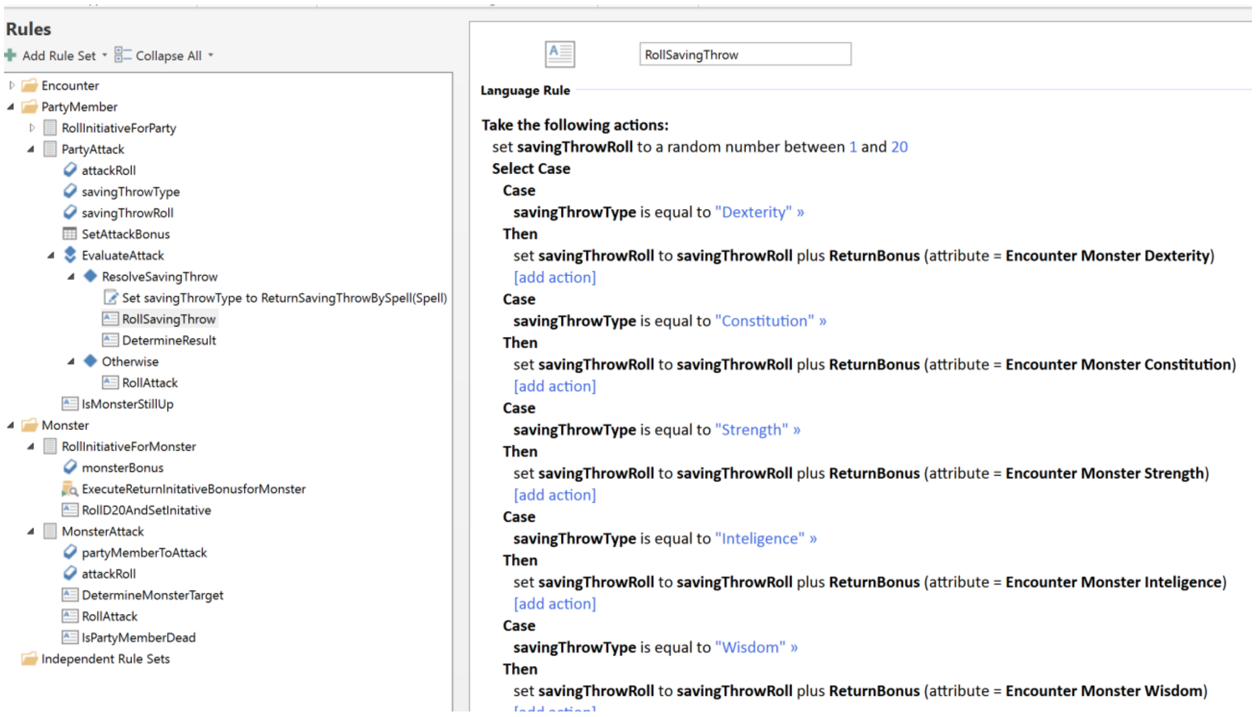
SpellTable

Inline Table

Modify table: Edit schema... Import...

	SpellName	SavingThrow
1	Fireball	Dexterity
2	Lightning Bolt	Dexterity
3	Cone of Cold	Constitution
4	Psychic Scream	Inteligence
5	Vicious Mockery	Wisdom
6	Evard's Black Tentacles	Strength

This way we can use a query to look up the results within a select case statement as illustrated in the rule below.



Rules

Add Rule Set · Collapse All

```
Encounter
PartyMember
  RollInitiativeForParty
  PartyAttack
    attackRoll
    savingThrowType
    savingThrowRoll
    SetAttackBonus
    EvaluateAttack
      ResolveSavingThrow
        Set savingThrowType to ReturnSavingThrowBySpell(Spell)
        RollSavingThrow
        DetermineResult
      Otherwise
        RollAttack
    IsMonsterStillUp
Monster
  RollInitiativeForMonster
    monsterBonus
    ExecuteReturnInitativeBonusforMonster
    RollD20AndSetInitative
  MonsterAttack
    partyMemberToAttack
    attackRoll
    DetermineMonsterTarget
    RollAttack
    IsPartyMemberDead
Independent Rule Sets
```

RollSavingThrow

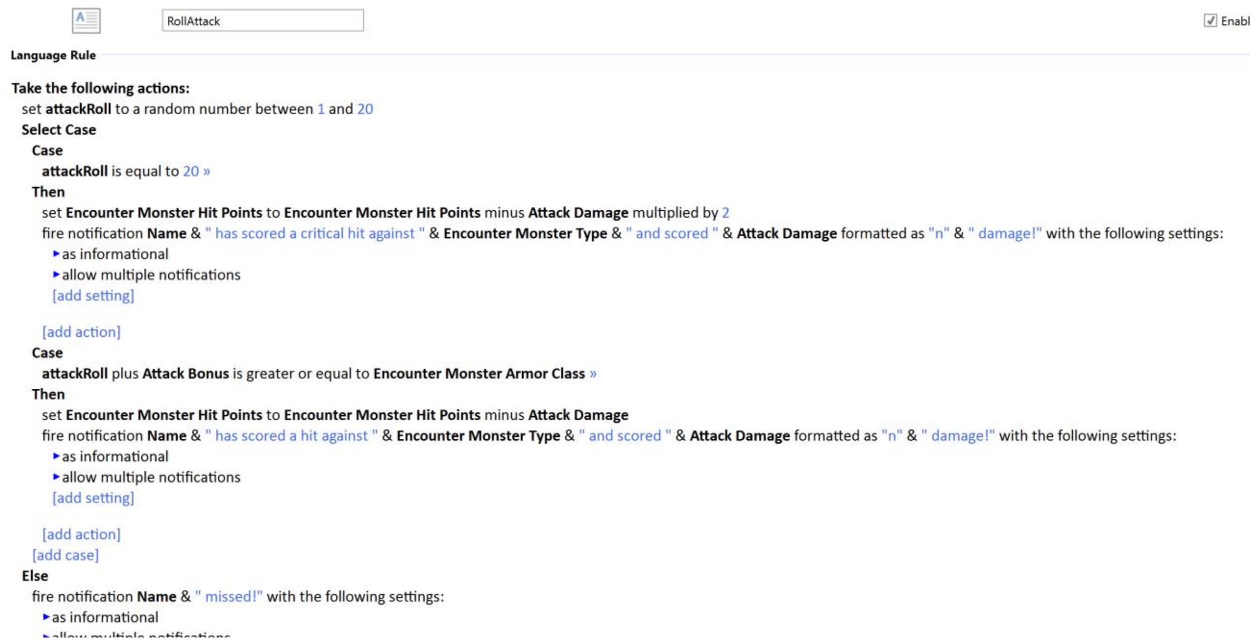
Language Rule

```
Take the following actions:
 set savingThrowRoll to a random number between 1 and 20
 Select Case
  Case
   savingThrowType is equal to "Dexterity" »
  Then
   set savingThrowRoll to savingThrowRoll plus ReturnBonus (attribute = Encounter Monster Dexterity)
   [add action]
  Case
   savingThrowType is equal to "Constitution" »
  Then
   set savingThrowRoll to savingThrowRoll plus ReturnBonus (attribute = Encounter Monster Constitution)
   [add action]
  Case
   savingThrowType is equal to "Strength" »
  Then
   set savingThrowRoll to savingThrowRoll plus ReturnBonus (attribute = Encounter Monster Strength)
   [add action]
  Case
   savingThrowType is equal to "Inteligence" »
  Then
   set savingThrowRoll to savingThrowRoll plus ReturnBonus (attribute = Encounter Monster Inteligence)
   [add action]
  Case
   savingThrowType is equal to "Wisdom" »
  Then
   set savingThrowRoll to savingThrowRoll plus ReturnBonus (attribute = Encounter Monster Wisdom)
```

Then we can roll out saving throw and if it is under the amount the target takes the damage and if it is equal to or over then they can avoid it.

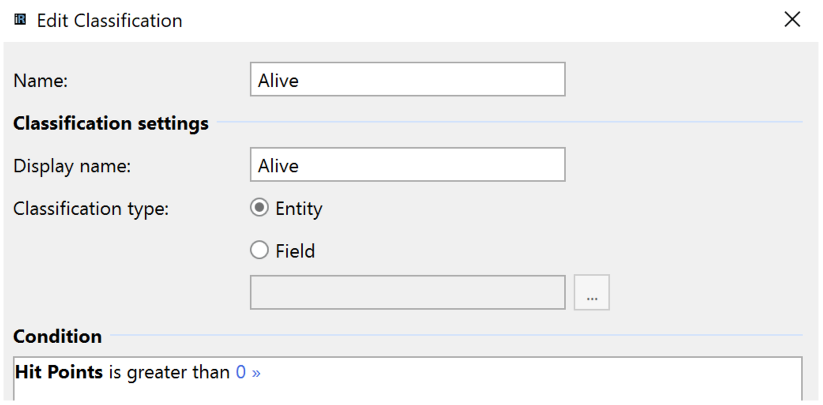
Then we need to determine if our regular attacks will succeed. In the tabletop world, we would roll a 20 sided die, add the bonus score, and compare the result against the opponent's armor class. If the modified roll is higher, then the attack succeeds and we can determine damage. If it is lower, then it will fail. But if the roll before the modifier was a 20, then that would be a critical hit and then the attack does twice the damage!

For this part of the rule, I used the select case action to create the condition statements and used the Fire Notification action to provide feedback on the success of the attack. Any successful attack will deduct the damage dealt from the opponents Hit Point score. Note: this is a simplified version of combat and if we wished we could expand this to include damage rolls, saving throws, and other rules.



```
RollAttack                                                        ✓ Enabl
Language Rule

Take the following actions:
  set attackRoll to a random number between 1 and 20
  Select Case
    Case
      attackRoll is equal to 20 »
    Then
      set Encounter Monster Hit Points to Encounter Monster Hit Points minus Attack Damage multiplied by 2
      fire notification Name & " has scored a critical hit against " & Encounter Monster Type & " and scored " & Attack Damage formatted as "n" & " damage!" with the following settings:
        ▸as informational
        ▸allow multiple notifications
        [add setting]

      [add action]
    Case
      attackRoll plus Attack Bonus is greater or equal to Encounter Monster Armor Class »
    Then
      set Encounter Monster Hit Points to Encounter Monster Hit Points minus Attack Damage
      fire notification Name & " has scored a hit against " & Encounter Monster Type & " and scored " & Attack Damage formatted as "n" & " damage!" with the following settings:
        ▸as informational
        ▸allow multiple notifications
        [add setting]

      [add action]
    [add case]
  Else
    fire notification Name & " missed!" with the following settings:
      ▸as informational
```

After our attack, we then need to check to see if the opponent is still alive or if the Hit Point score is greater than zero. This definition was a great opportunity to create a simple classification. Classification is a simple vocabulary template that will evaluate for true or false for a given entity or field. By using this, we can define in our own language that our monster or party member is Alive if their Hit Point score is above zero, as seen below.



Edit Classification

Name: Alive

Classification settings

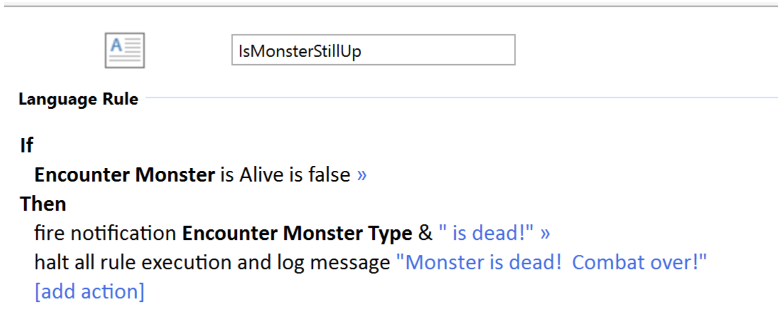
Display name: Alive

Classification type: [● Entity] ○ Field

Condition

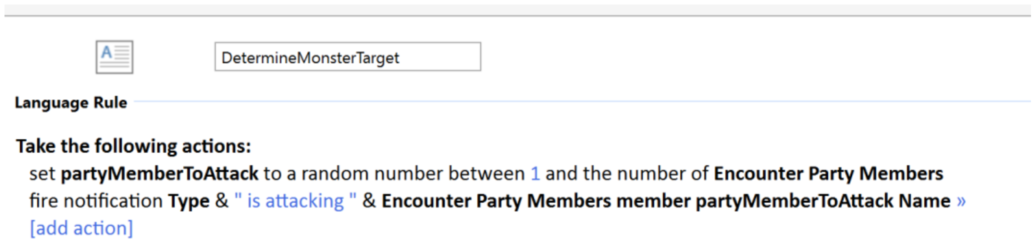
Hit Points is greater than 0 »

By doing this, our rule becomes more transparent as we check the condition of the target after the attack.



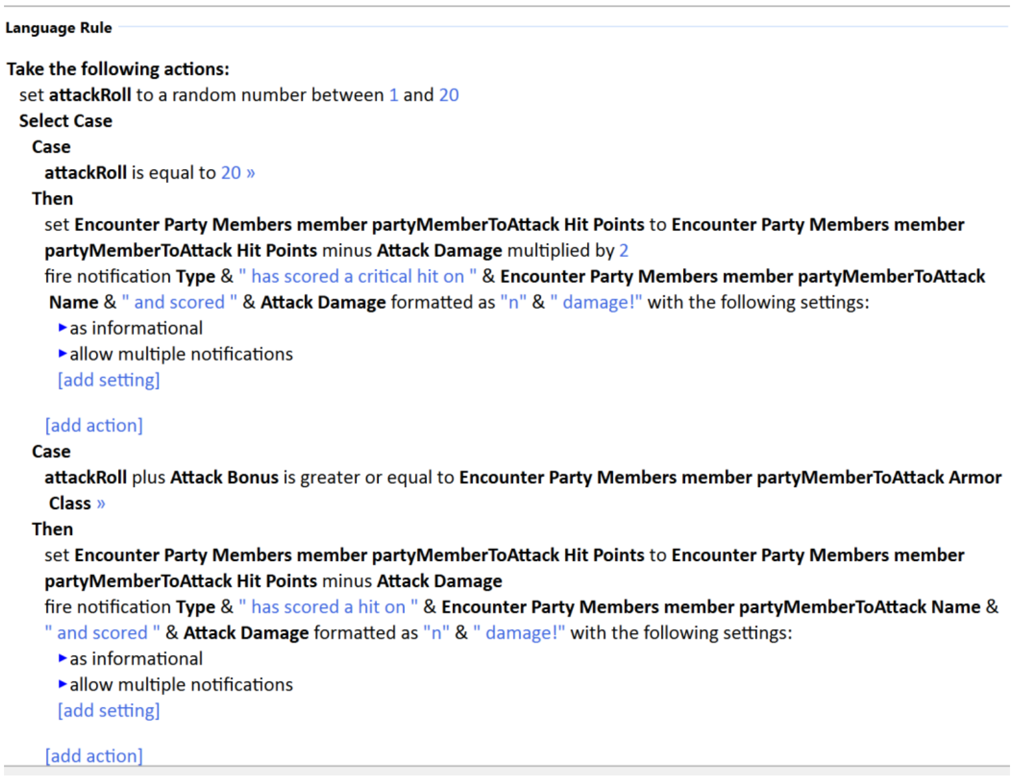
```
IsMonsterStillUp

Language Rule

If
  Encounter Monster is Alive is false »
Then
  fire notification Encounter Monster Type & " is dead!" »
  halt all rule execution and log message "Monster is dead!  Combat over!"
  [add action]
```

For the monster, we will execute a similar structure for the combat but monsters generally have set stats with attach and damage bonuses, so we will not need a decision table for that, but we do need to know which person the monster will target. I've chosen to do this randomly, using a declared variable to store the party member to attack, but other options could be created like targeting the party member with hit points.



```
DetermineMonsterTarget

Language Rule

Take the following actions:
  set partyMemberToAttack to a random number between 1 and the number of Encounter Party Members
  fire notification Type & " is attacking " & Encounter Party Members member partyMemberToAttack Name »
  [add action]
```

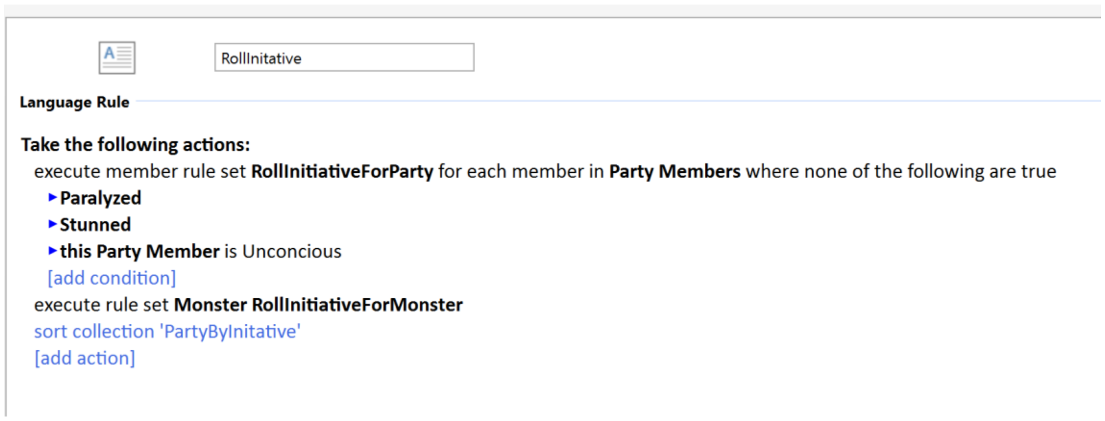
We can then use that integer to have the attack target the identified member.



```
Language Rule

Take the following actions:
  set attackRoll to a random number between 1 and 20
  Select Case
    Case
      attackRoll is equal to 20 »
    Then
      set Encounter Party Members member partyMemberToAttack Hit Points to Encounter Party Members member
      partyMemberToAttack Hit Points minus Attack Damage multiplied by 2
      fire notification Type & " has scored a critical hit on " & Encounter Party Members member partyMemberToAttack
       Name & " and scored " & Attack Damage formatted as "n" & " damage!" with the following settings:
        ▸as informational
        ▸allow multiple notifications
        [add setting]

      [add action]
    Case
      attackRoll plus Attack Bonus is greater or equal to Encounter Party Members member partyMemberToAttack Armor
       Class »
    Then
      set Encounter Party Members member partyMemberToAttack Hit Points to Encounter Party Members member
      partyMemberToAttack Hit Points minus Attack Damage
      fire notification Type & " has scored a hit on " & Encounter Party Members member partyMemberToAttack Name &
      " and scored " & Attack Damage formatted as "n" & " damage!" with the following settings:
        ▸as informational
        ▸allow multiple notifications
        [add setting]

      [add action]
```

Controlling the Combat

After I had the basic combat rules authored, I opted to build a couple of controlling rulesets to coordinate the execution of the rules. For the Initiative rules, I created a Language rule with the Execute Member Ruleset action. I used the filter option combined with the "none of the following are true" template, so we don't roll initiative for party members that are unable to act. I also used the sort collection action to place the party in order of their Initiative score.

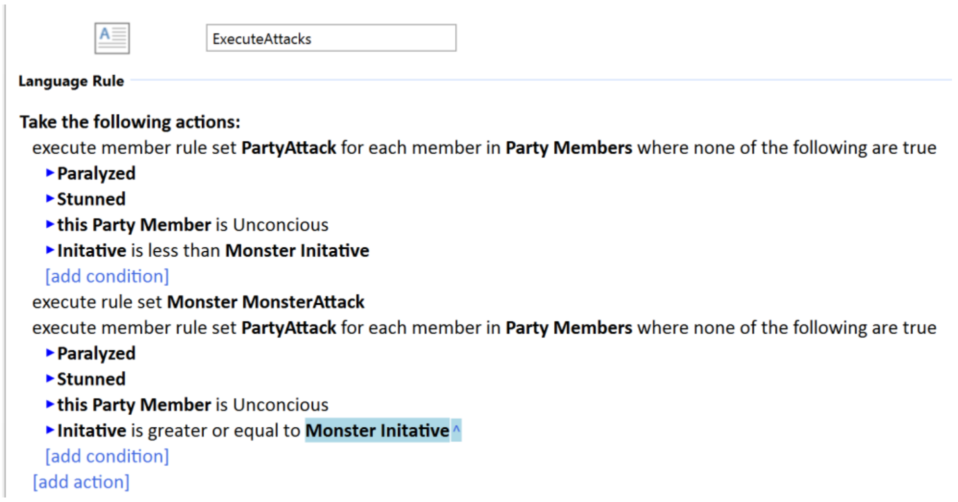


RollInitative

Language Rule

Take the following actions:
execute member rule set **RollInitiativeForParty** for each member in **Party Members** where none of the following are true
▸**Paralyzed**
▸**Stunned**
▸**this Party Member** is Unconcious
[add condition]
execute rule set **Monster RollInitiativeForMonster**
sort collection 'PartyByInitative'
[add action]

For controlling the attack rolls, I used a similar technique using the Execute Member Ruleset action. Since we sorted the collection of party members in the step above we can be sure the party will go in order of their Initiative Score.

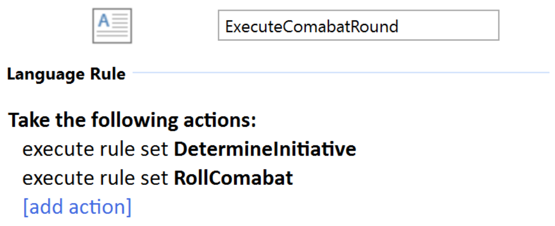


ExecuteAttacks

Language Rule

Take the following actions:
execute member rule set **PartyAttack** for each member in **Party Members** where none of the following are true
▸**Paralyzed**
▸**Stunned**
▸**this Party Member** is Unconcious
▸**Initative** is less than **Monster Initative**
[add condition]
execute rule set **Monster MonsterAttack**
execute member rule set **PartyAttack** for each member in **Party Members** where none of the following are true
▸**Paralyzed**
▸**Stunned**
▸**this Party Member** is Unconcious
▸**Initative** is greater or equal to **Monster Initative**
[add condition]
[add action]

Finally, we can coordinate these two rules with one more controlling rule, shown below.



ExecuteComabatRound

Language Rule

Take the following actions:
execute rule set **DetermineInitiative**
execute rule set **RollComabat**
[add action]

Feel free to download this rule modeling exercise from our GitHub site.

Hopefully, this exercise will help illustrate how you can use irAuthor to model the processes that are important you and your organization!

Happy Authoring!