GDC®

Dark Secrets of the RNG

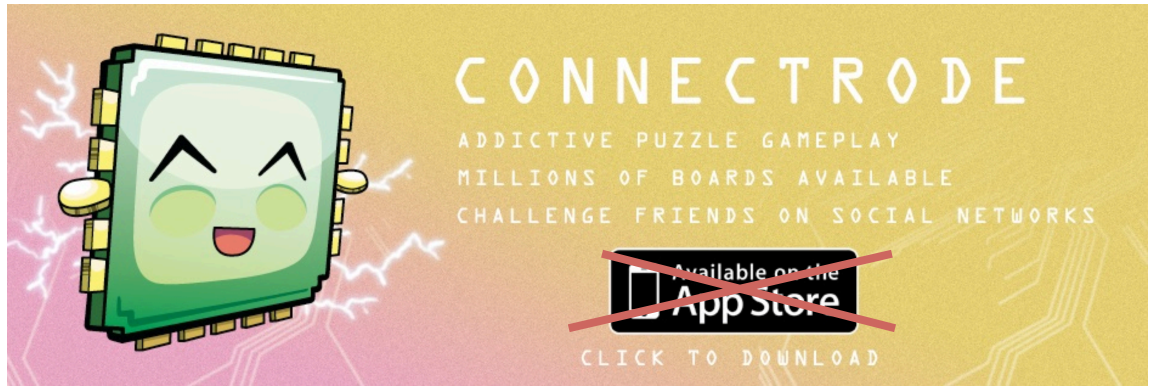
Shay Pierce
Senior Gameplay Engineer,
Dire Wolf Digital, LLC

At big studios:

MIDWAY

As an "indie":

DEEP PLAID GAMES

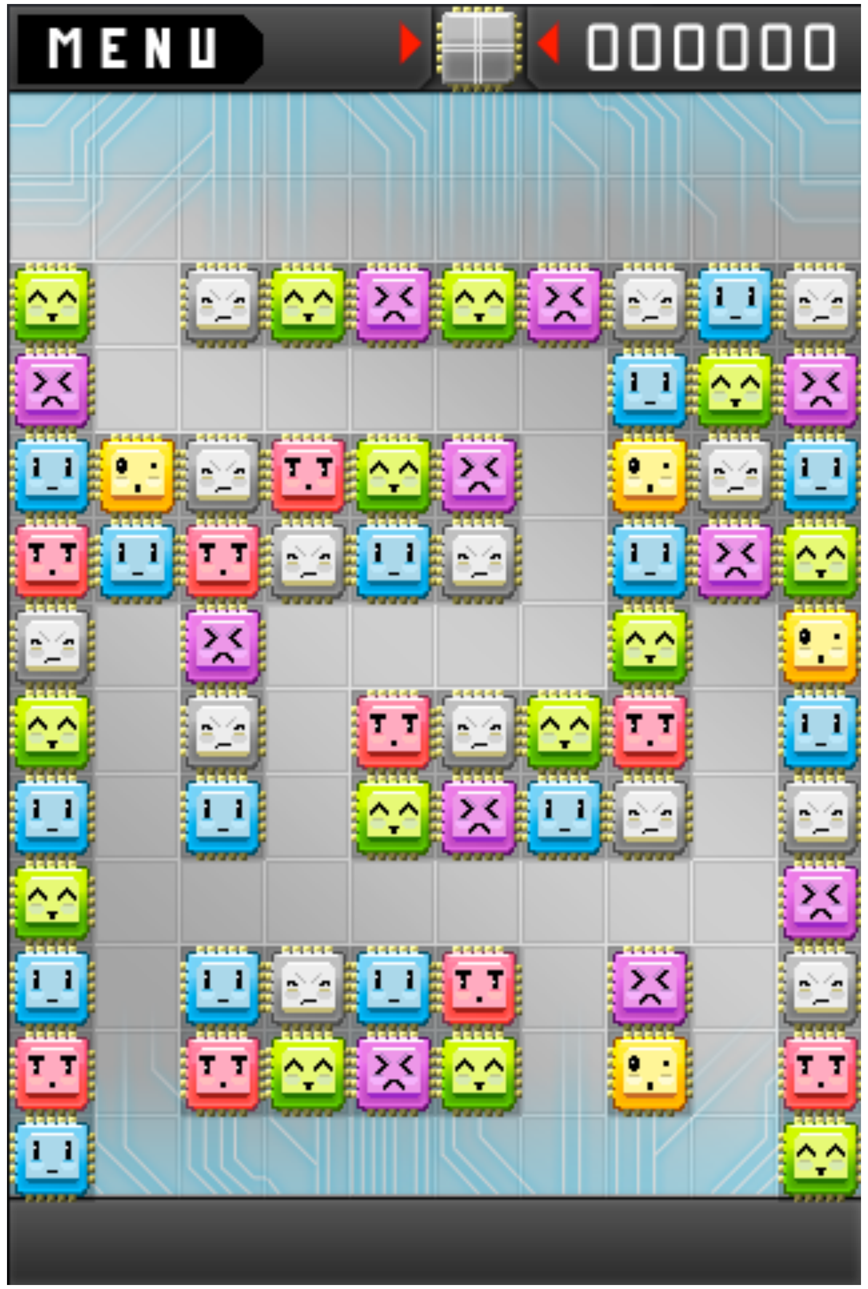

CONNECTRODE
ADDICTIVE PUZZLE GAMEPLAY
MILLIONS OF BOARDS AVAILABLE
CHALLENGE FRIENDS ON SOCIAL NETWORKS
~~Available on the App Store~~
CLICK TO DOWNLOAD

Nowadays:

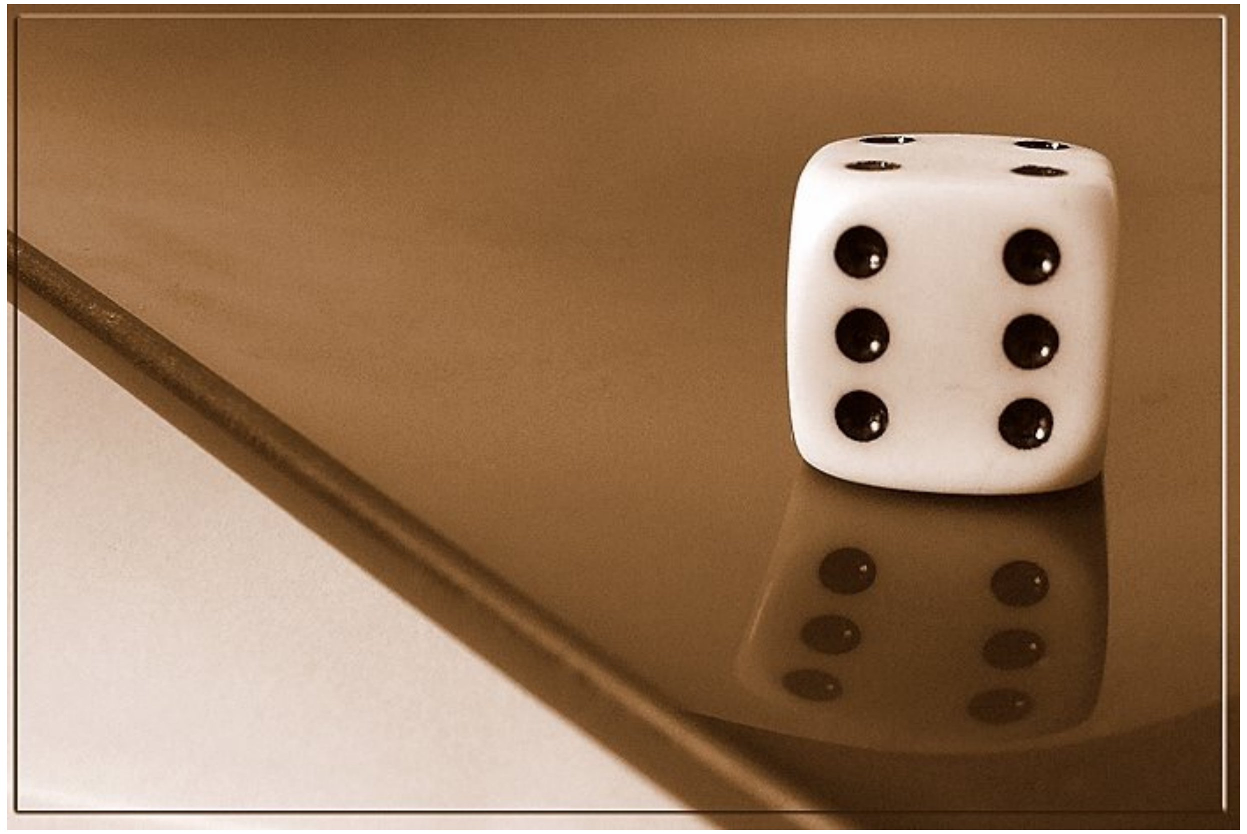
DIRE WOLF®

RNG Case Study: Connectrode

- Spawns a piece each turn: one of six colors, at random.
- Bug: same color 9 times in a row!
- Simple Solution: "Bagging!"



A Subtle But Important Distinction



"Sampling WITH Replacement"

"Sampling WITHOUT Replacement"

Bagging & Edge Cases

- Evens out distribution
- Limits/eliminates “streaks”
- Must handle edge cases (literally)!
- If you shuffle three decks together (and reshuffle when empty)… what’s the longest possible streak of repeats you can draw?

Dark Secret of RNG #1:

Every random edge case that *can* happen, *will* happen to a live player (eventually!)

As the gameplay programmer, you are the last line of defense for sniffing out and accounting for those edge cases

INTRODUCING: THE PITY TIMER

(Because "random" means: "yes, Guildenstern, it *is* possible to lose 100 coin flips in a row"!)

"We added a system in the expansion that tracks the amount of time you spend fighting creatures without finding a legendary and after a certain period of time will slowly start increasing the legendary drop rate. Once a legendary drops for you … we reset that timer."

…

"If we want you to get a legendary every 2 hours, the system basically says 'ok it's been like double/triple that period of time, just help the guy out!'"

Diablo 3 developer Travis Day, 2014

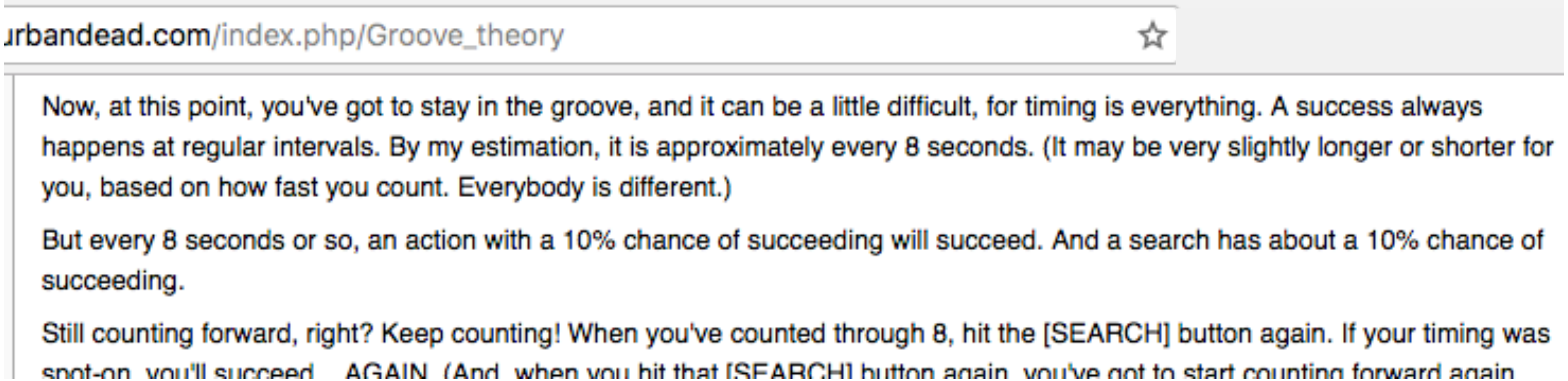
More Dark Secrets of RNG:

Secret #2: Human brains are *terrible* at randomness!

…So many games use tricks to emulate “fair” randomness (i.e. the behavior players expect).

Secret #3: Players will *still* always complain that your RNG is broken and unfair.

Seed Wisely, Or Thou May Fall Prey to ***Groove Theory!***

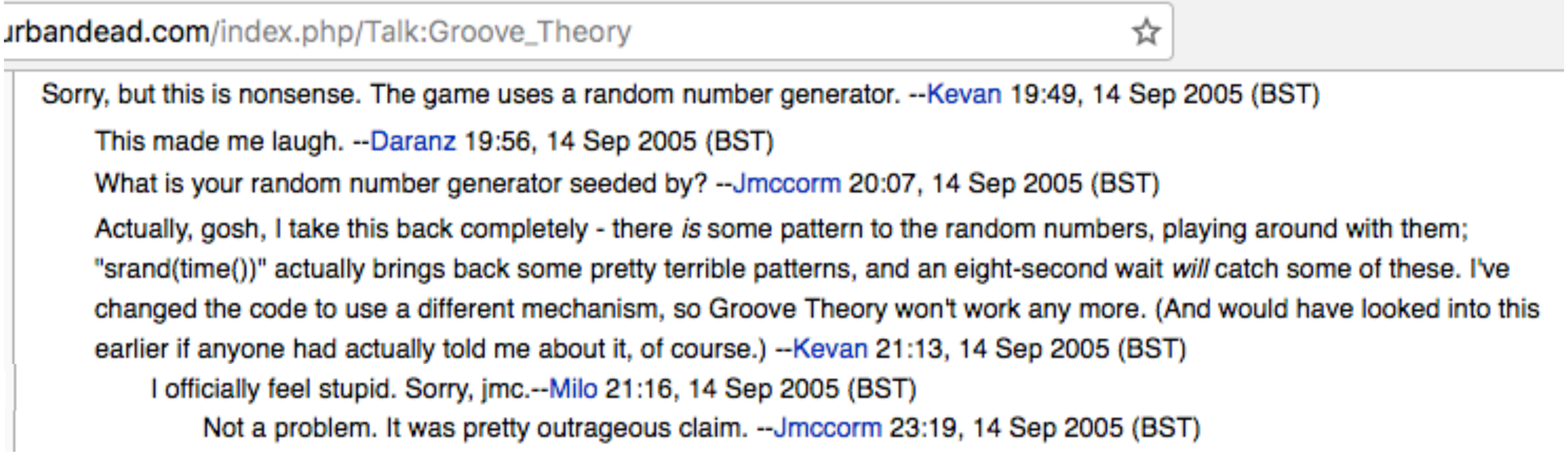


urbandead.com/index.php/Groove_theory ☆

Now, at this point, you've got to stay in the groove, and it can be a little difficult, for timing is everything. A success always happens at regular intervals. By my estimation, it is approximately every 8 seconds. (It may be very slightly longer or shorter for you, based on how fast you count. Everybody is different.)

But every 8 seconds or so, an action with a 10% chance of succeeding will succeed. And a search has about a 10% chance of succeeding.

Still counting forward, right? Keep counting! When you've counted through 8, hit the [SEARCH] button again. If your timing was spot-on, you'll succeed... AGAIN. (And, when you hit that [SEARCH] button again, you've got to start counting forward again



urbandead.com/index.php/Talk:Groove_Theory ☆

Sorry, but this is nonsense. The game uses a random number generator. --Kevan 19:49, 14 Sep 2005 (BST)

> This made me laugh. --Daranz 19:56, 14 Sep 2005 (BST)
>
> What is your random number generator seeded by? --Jmccorm 20:07, 14 Sep 2005 (BST)
>
> Actually, gosh, I take this back completely - there *is* some pattern to the random numbers, playing around with them; "srand(time())" actually brings back some pretty terrible patterns, and an eight-second wait *will* catch some of these. I've changed the code to use a different mechanism, so Groove Theory won't work any more. (And would have looked into this earlier if anyone had actually told me about it, of course.) --Kevan 21:13, 14 Sep 2005 (BST)
>
>> I officially feel stupid. Sorry, jmc.--Milo 21:16, 14 Sep 2005 (BST)
>>
>>> Not a problem. It was pretty outrageous claim. --Jmccorm 23:19, 14 Sep 2005 (BST)

The Dark Secret of Gameplay Programming:

The designer(s) will ALWAYS change their minds about everything possible!

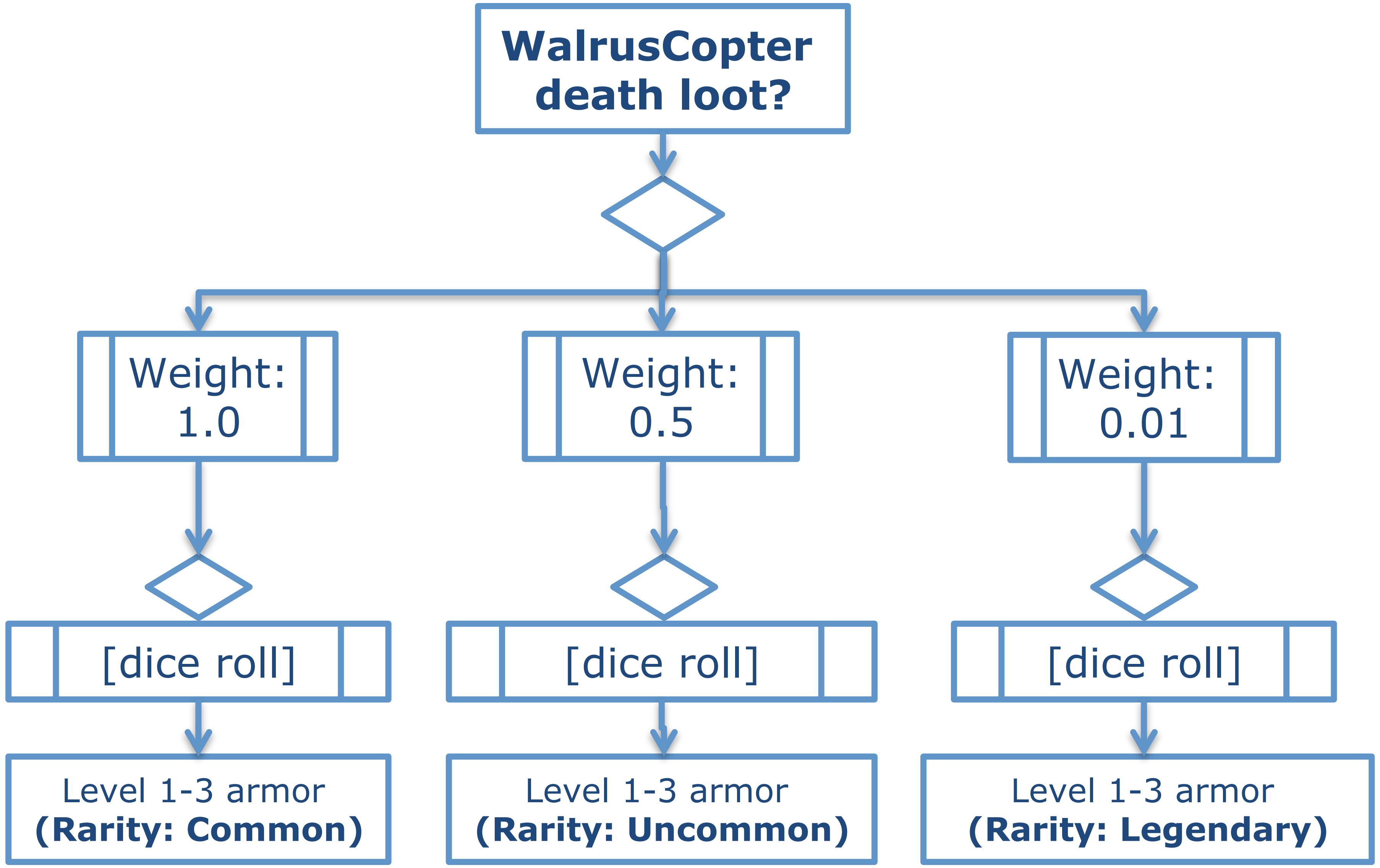
...Your job is to plan for it, and enable it, by using good, flexible coding approaches!

Example “Pity Timer” implementation

```
var rarityLootLookup = new LookupTable<LookupTable<ILootItemArchetype>>(
  new RNG(seed),

  new LookupItem<LookupTable<ILootItemArchetype>>(commonLootLookup, 1f),
  new LookupItem<LookupTable<ILootItemArchetype>>(uncommonLootLookup, .5f),
  new LookupItem<LookupTable<ILootItemArchetype>>(
    legendaryLookup,
    .01f,
    li => li.Weight = li.StartingWeight,  // Reset weight whenever it's picked
    li => li.Weight += .01f               // Grow weight each time not picked
  )
);

var lootDrop = rarityLootLookup.Pick();
print("Loot dropped: " + lootDrop);
```



WalrusCopter death loot?

Weight: 1.0

Weight: 0.5

Weight: 0.01

[dice roll]

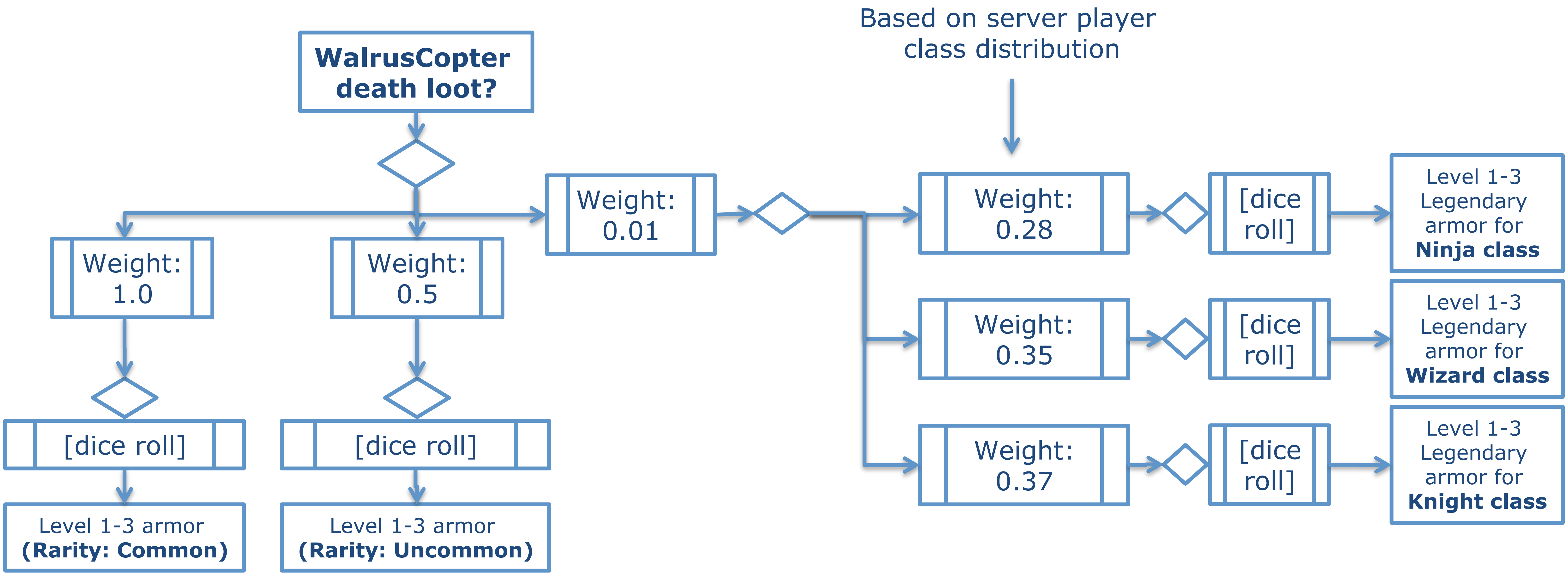
[dice roll]

[dice roll]

Level 1-3 armor **(Rarity: Common)**

Level 1-3 armor **(Rarity: Uncommon)**

Level 1-3 armor **(Rarity: Legendary)**



Based on server player class distribution

WalrusCopter death loot?

Weight: 1.0

[dice roll]

Level 1-3 armor **(Rarity: Common)**

Weight: 0.5

[dice roll]

Level 1-3 armor **(Rarity: Uncommon)**

Weight: 0.01

Weight: 0.28

[dice roll]

Level 1-3 Legendary armor for **Ninja class**

Weight: 0.35

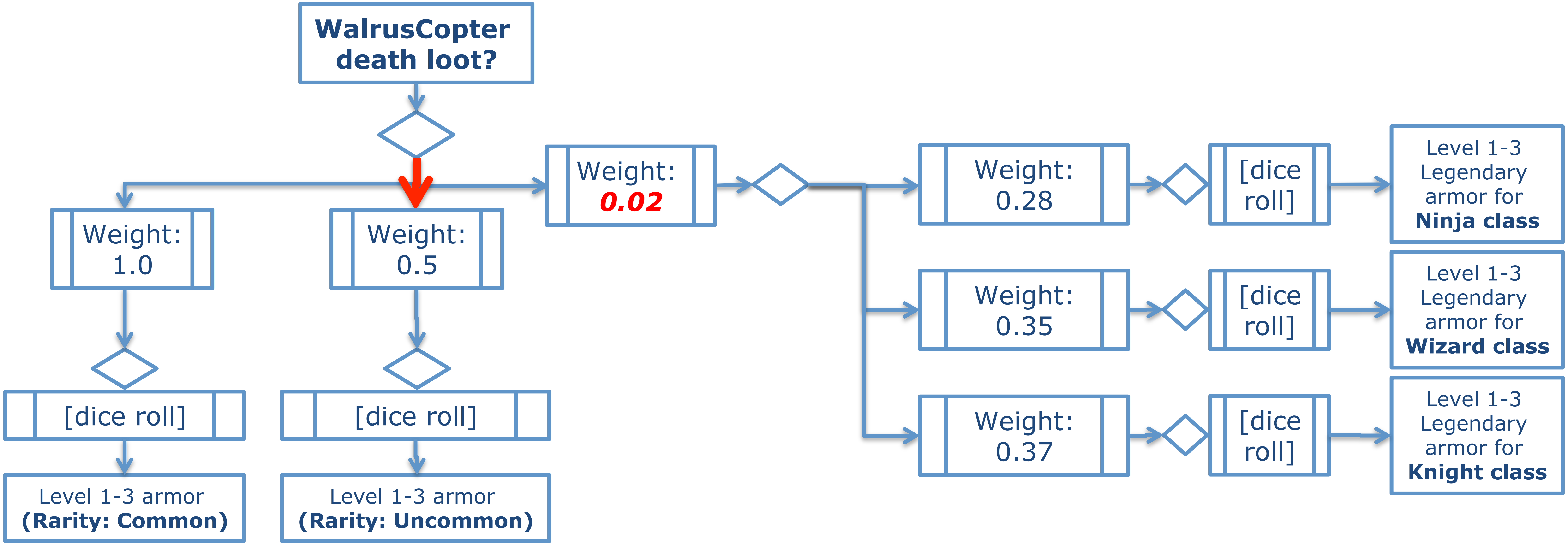
[dice roll]

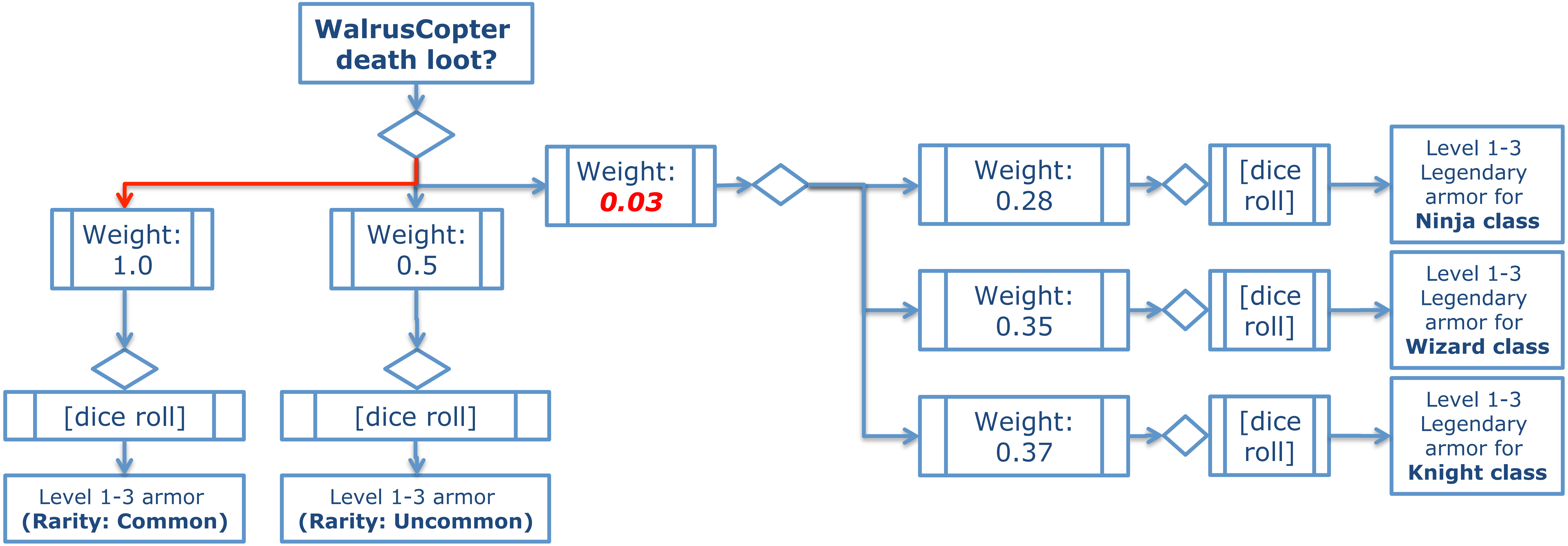
Level 1-3 Legendary armor for **Wizard class**

Weight: 0.37

[dice roll]

Level 1-3 Legendary armor for **Knight class**


WalrusCopter death loot?
Weight: 1.0
Weight: 0.5
Weight: 0.02
[dice roll]
[dice roll]
Level 1-3 armor (Rarity: Common)
Level 1-3 armor (Rarity: Uncommon)
Weight: 0.28
Weight: 0.35
Weight: 0.37
[dice roll]
[dice roll]
[dice roll]
Level 1-3 Legendary armor for Ninja class
Level 1-3 Legendary armor for Wizard class
Level 1-3 Legendary armor for Knight class



WalrusCopter death loot?

Weight: 1.0

[dice roll]

Level 1-3 armor **(Rarity: Common)**

Weight: 0.5

[dice roll]

Level 1-3 armor **(Rarity: Uncommon)**

Weight: ***0.03***

Weight: 0.28

[dice roll]

Level 1-3 Legendary armor for **Ninja class**

Weight: 0.35

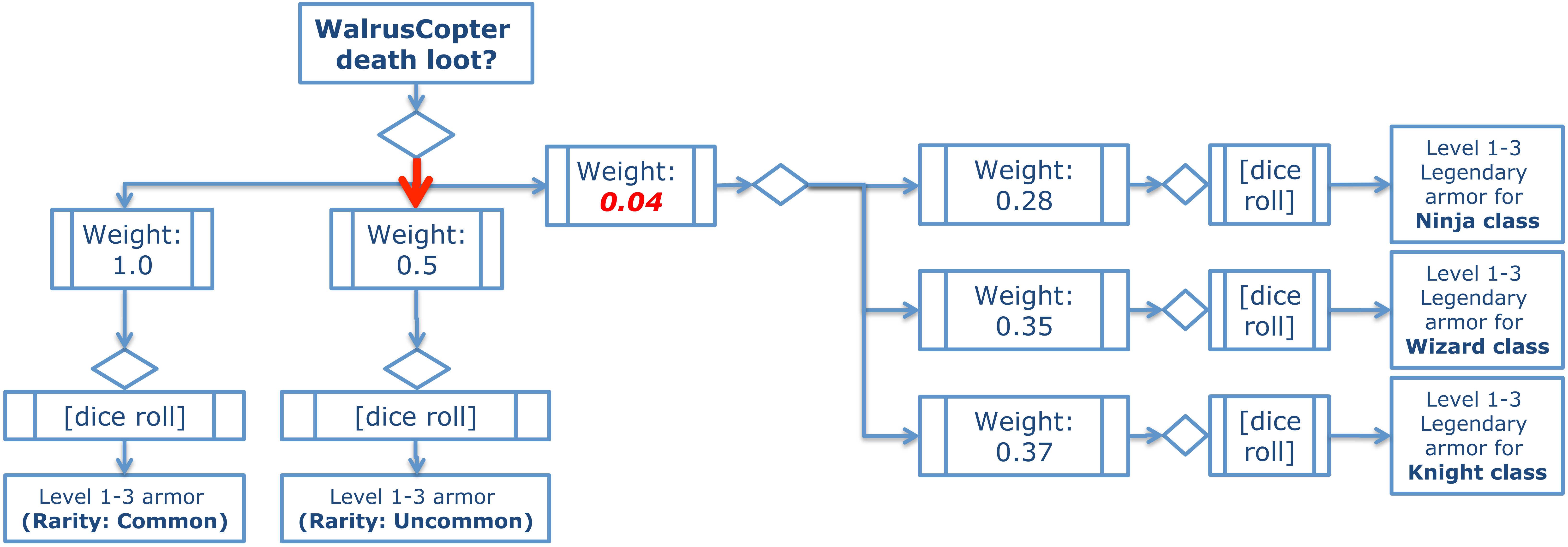
[dice roll]

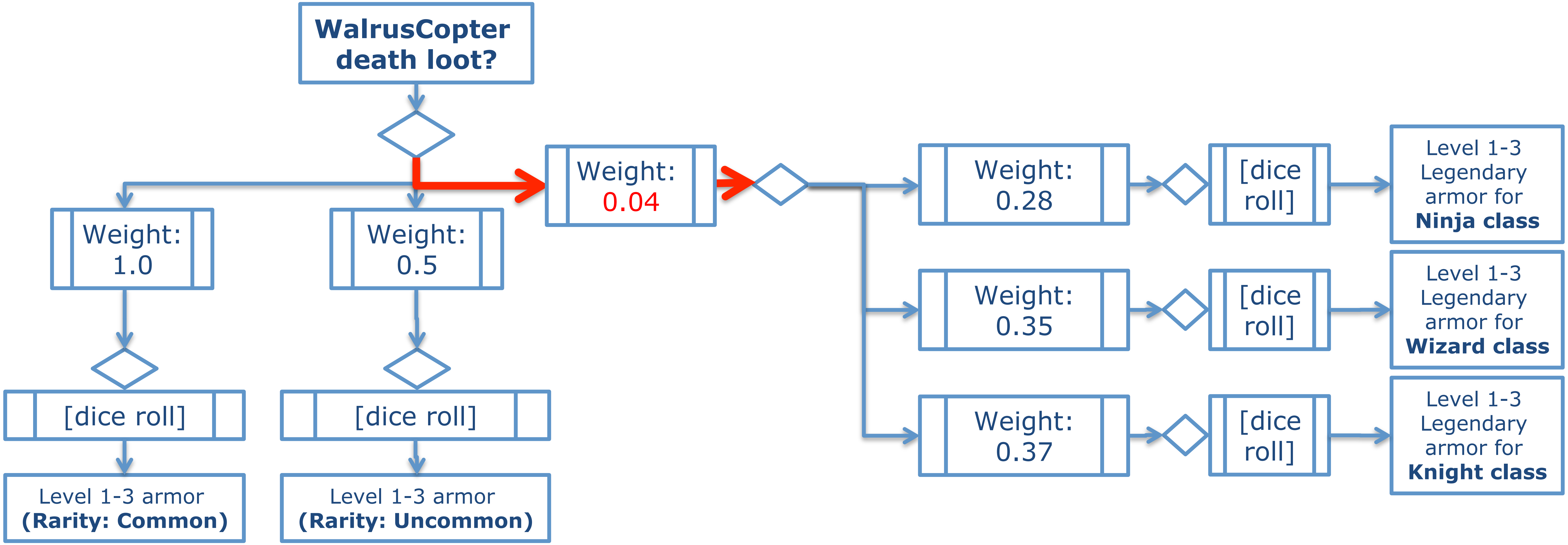
Level 1-3 Legendary armor for **Wizard class**

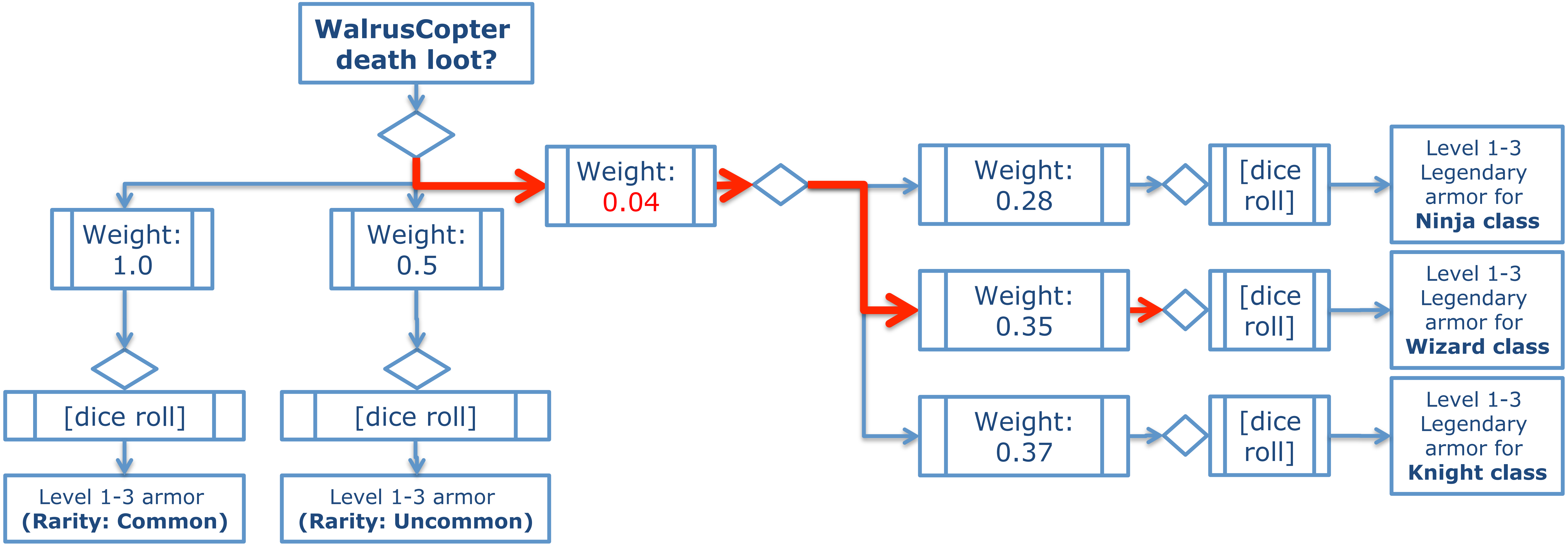
Weight: 0.37

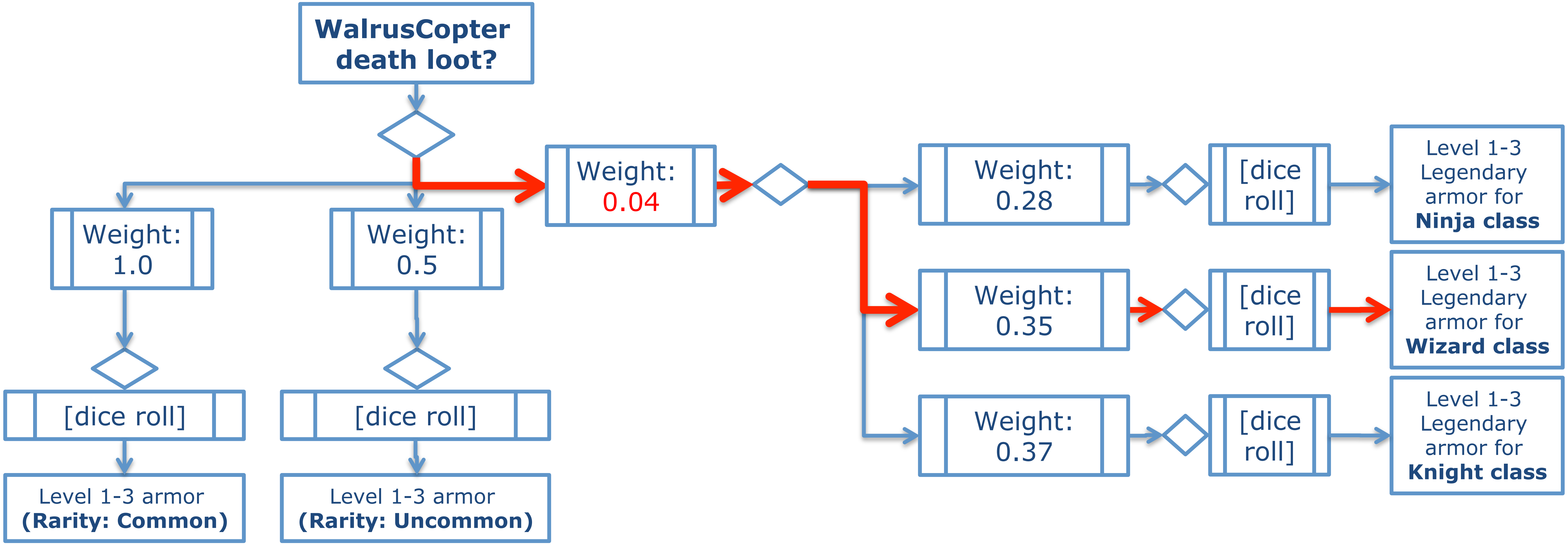
[dice roll]

Level 1-3 Legendary armor for **Knight class**


WalrusCopter death loot?
Weight: 1.0
[dice roll]
Level 1-3 armor (Rarity: Common)
Weight: 0.5
[dice roll]
Level 1-3 armor (Rarity: Uncommon)
Weight: 0.04
Weight: 0.28
[dice roll]
Level 1-3 Legendary armor for Ninja class
Weight: 0.35
[dice roll]
Level 1-3 Legendary armor for Wizard class
Weight: 0.37
[dice roll]
Level 1-3 Legendary armor for Knight class


WalrusCopter death loot?
Weight: 1.0
[dice roll]
Level 1-3 armor (Rarity: Common)
Weight: 0.5
[dice roll]
Level 1-3 armor (Rarity: Uncommon)
Weight: 0.04
Weight: 0.28
[dice roll]
Level 1-3 Legendary armor for Ninja class
Weight: 0.35
[dice roll]
Level 1-3 Legendary armor for Wizard class
Weight: 0.37
[dice roll]
Level 1-3 Legendary armor for Knight class


WalrusCopter death loot?
Weight: 1.0
[dice roll]
Level 1-3 armor (Rarity: Common)
Weight: 0.5
[dice roll]
Level 1-3 armor (Rarity: Uncommon)
Weight: 0.04
Weight: 0.28
[dice roll]
Level 1-3 Legendary armor for Ninja class
Weight: 0.35
[dice roll]
Level 1-3 Legendary armor for Wizard class
Weight: 0.37
[dice roll]
Level 1-3 Legendary armor for Knight class



WalrusCopter death loot?

Weight: 1.0

[dice roll]

Level 1-3 armor **(Rarity: Common)**

Weight: 0.5

[dice roll]

Level 1-3 armor **(Rarity: Uncommon)**

Weight: 0.04

Weight: 0.28

[dice roll]

Level 1-3 Legendary armor for **Ninja class**

Weight: 0.35

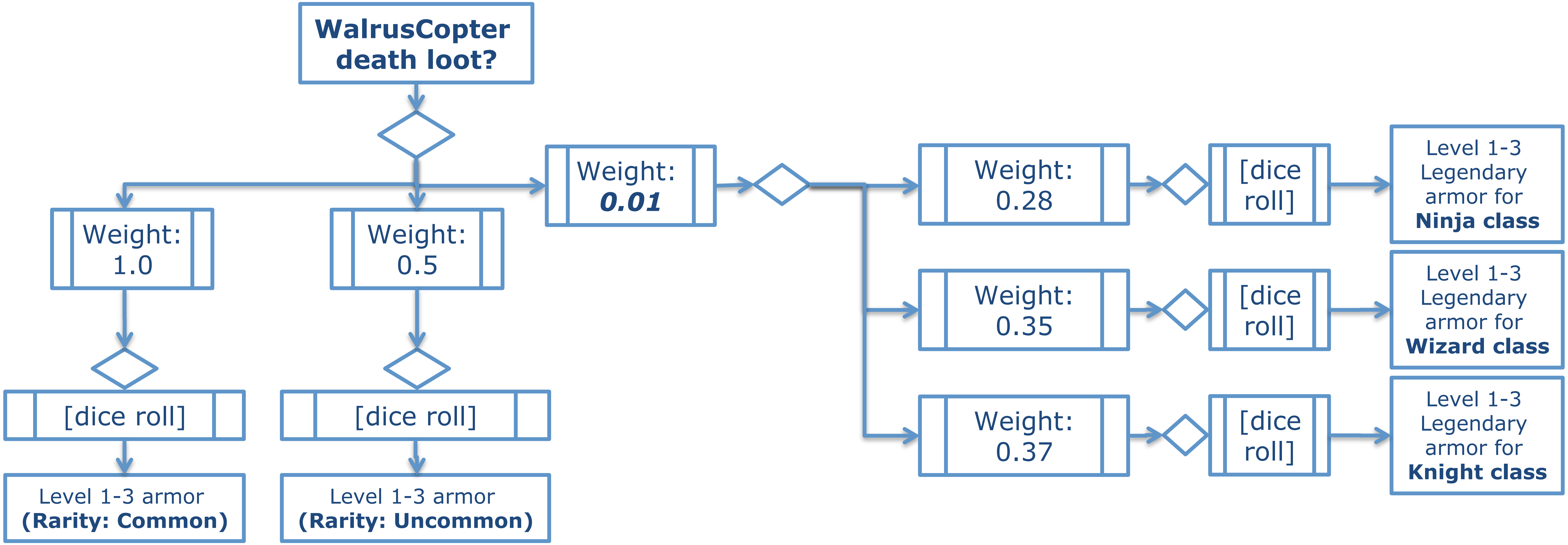
[dice roll]

Level 1-3 Legendary armor for **Wizard class**

Weight: 0.37

[dice roll]

Level 1-3 Legendary armor for **Knight class**


WalrusCopter death loot?
Weight: 1.0
Weight: 0.5
Weight: 0.01
[dice roll]
[dice roll]
Level 1-3 armor (Rarity: Common)
Level 1-3 armor (Rarity: Uncommon)
Weight: 0.28
Weight: 0.35
Weight: 0.37
[dice roll]
[dice roll]
[dice roll]
Level 1-3 Legendary armor for Ninja class
Level 1-3 Legendary armor for Wizard class
Level 1-3 Legendary armor for Knight class

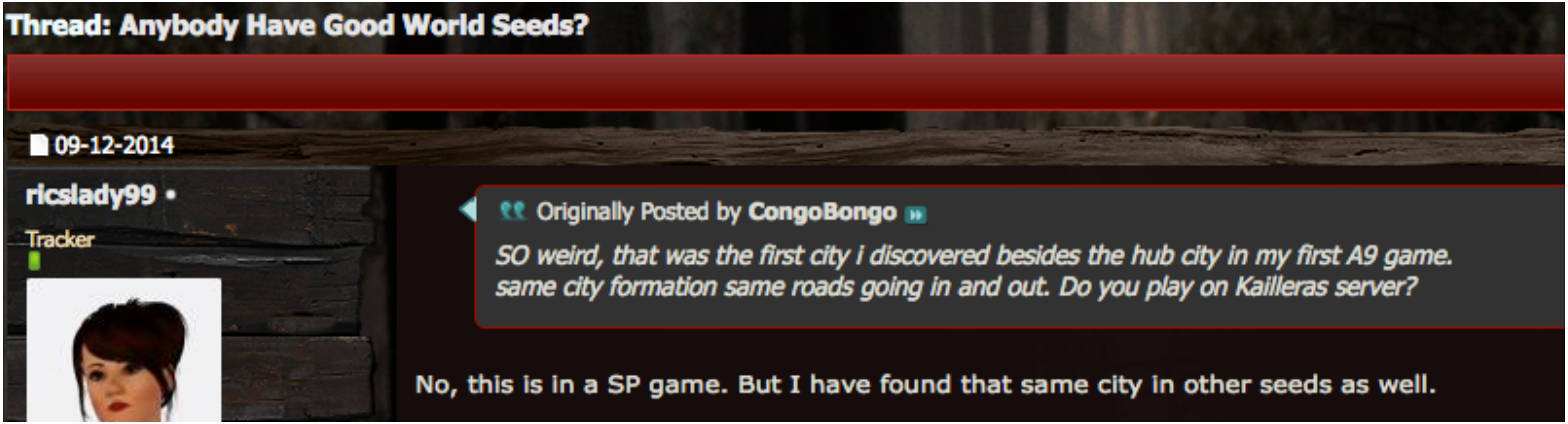
Dark Secret of RNG #3: *Random Hashing*

See Squirrel’s earlier talk (“SquirrelNoise”).

…Let’s see how Random Hashing can be used to solve a deep dark problem: “Deep Echoes”.

A Tale of Two Cities

Each Procedural City in your world is generated from a seed. So they're unique… right?



Thread: Anybody Have Good World Seeds?

09-12-2014

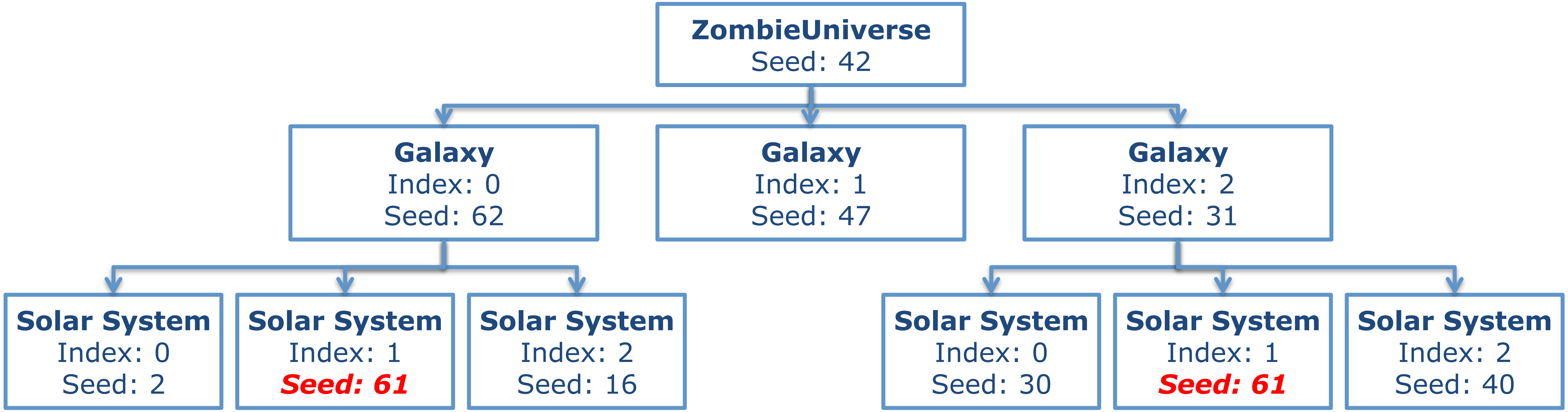
ricslady99 •

Tracker

Originally Posted by **CongoBongo**

> *SO weird, that was the first city i discovered besides the hub city in my first A9 game. same city formation same roads going in and out. Do you play on Kailleras server?*

No, this is in a SP game. But I have found that same city in other seeds as well.



ZombieUniverse
Seed: 42

Galaxy
Index: 0
Seed: 62

Galaxy
Index: 1
Seed: 47

Galaxy
Index: 2
Seed: 31

Solar System
Index: 0
Seed: 2

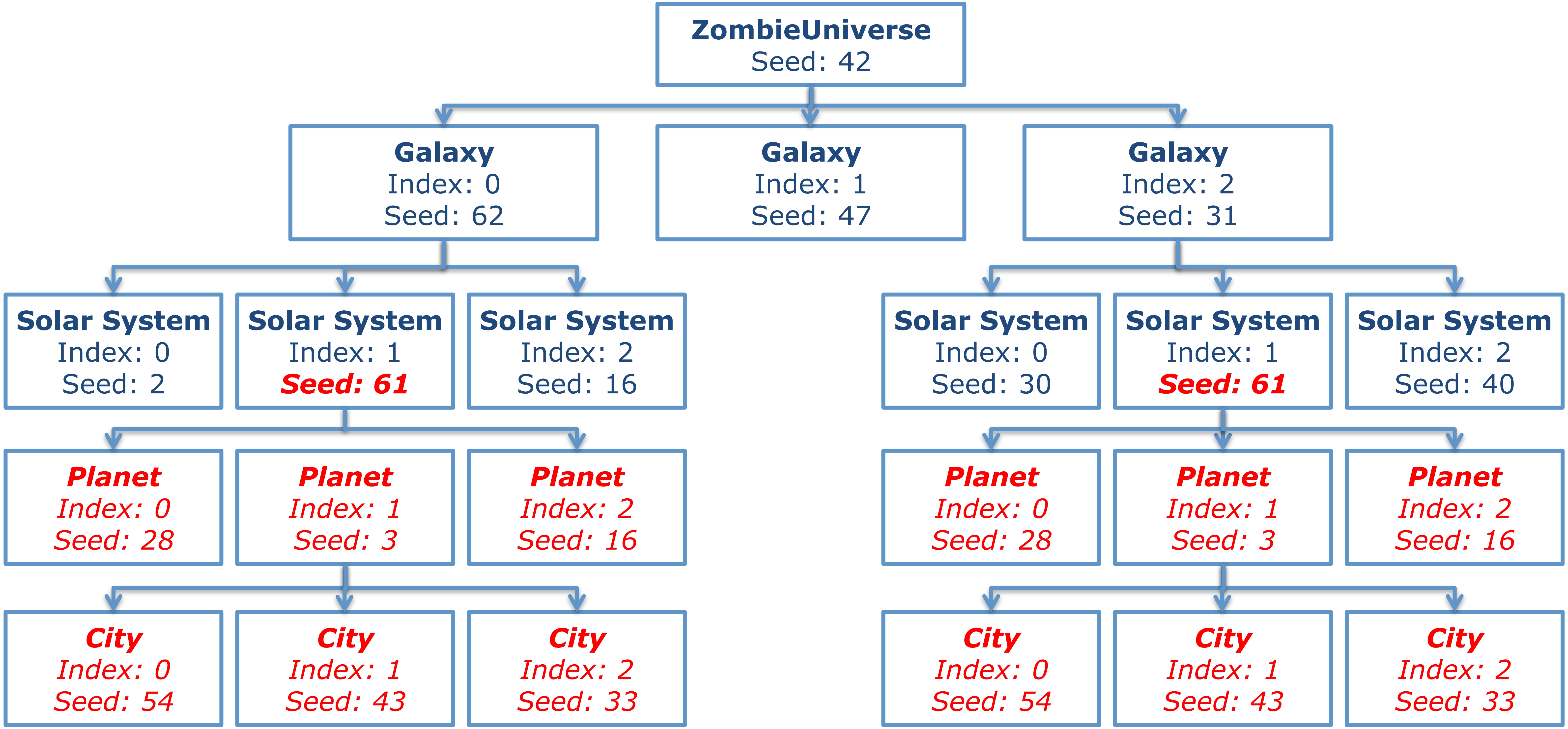
Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 16

Solar System
Index: 0
Seed: 30

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 40



ZombieUniverse
Seed: 42

Galaxy
Index: 0
Seed: 62

Galaxy
Index: 1
Seed: 47

Galaxy
Index: 2
Seed: 31

Solar System
Index: 0
Seed: 2

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 16

Planet
Index: 0
Seed: 28

Planet
Index: 1
Seed: 3

Planet
Index: 2
Seed: 16

City
Index: 0
Seed: 54

City
Index: 1
Seed: 43

City
Index: 2
Seed: 33

Solar System
Index: 0
Seed: 30

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 40

Planet
Index: 0
Seed: 28

Planet
Index: 1
Seed: 3

Planet
Index: 2
Seed: 16

City
Index: 0
Seed: 54

City
Index: 1
Seed: 43

City
Index: 2
Seed: 33

Solving Deep Echoes:

- Generate element content (e.g. name) based on element.Seed
- Generate *child elements* using some other seed…
  - Should be *unique to that element*
  - Should *not* be *a function of its parentage*
  - In our example we use *its N-dimensional "address" based on hierarchy indices*, which we then *hash.*

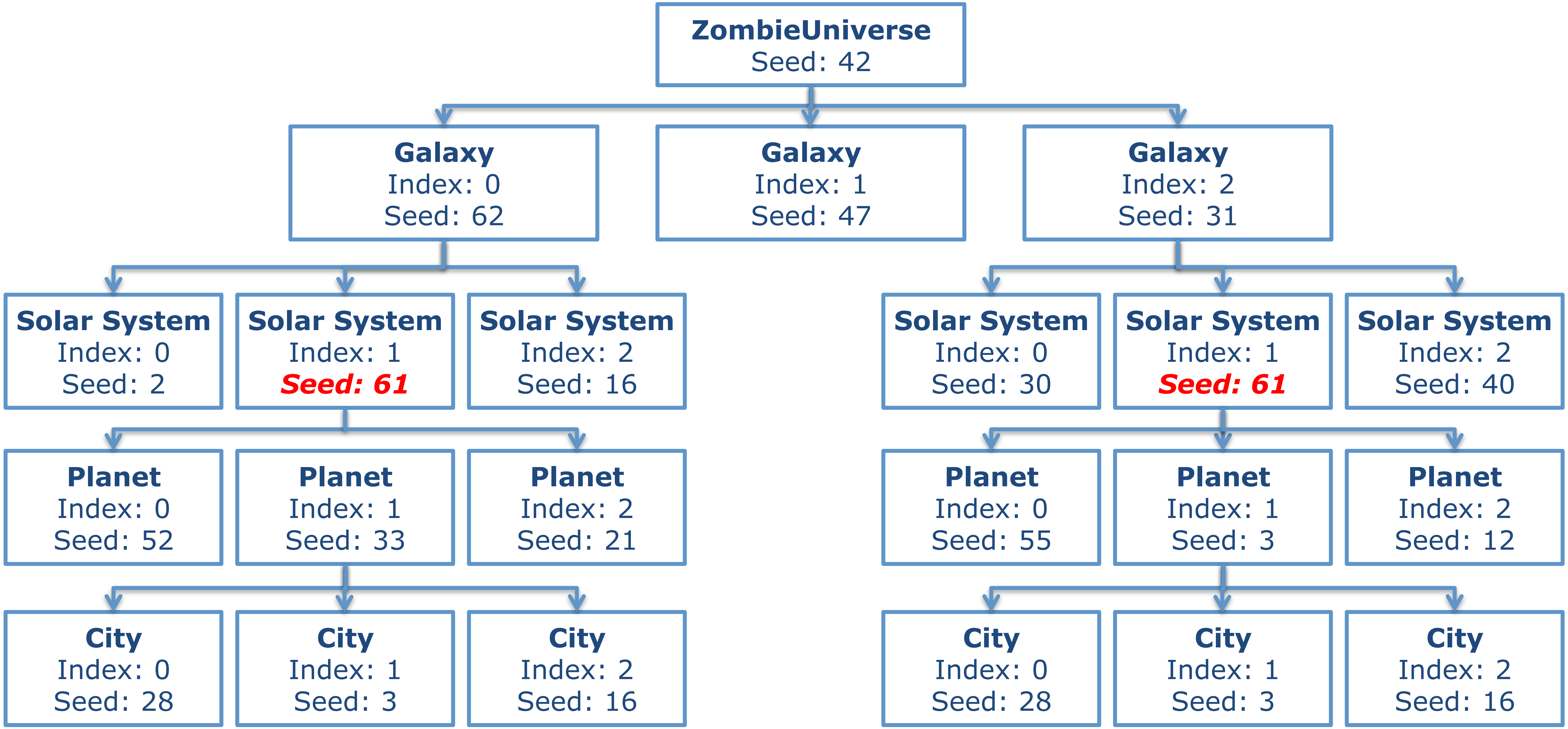
```
public int GetSeedByIndexAddress(
    Type type,
    IEnumerator<UniverseLayer> layers
  )
{
  return SquirrelNoise.Hash(
     ZombieUniverse.Instance.Seed,
     layers.Select(l => l.Index).ToArray()
  );
}
```

```
void InitializeChildren() {
  var rng = new RNG(
    ZombieUniverse.Instance.GetSeedByIndexAddress(this.GetType(), GetHierarchy())
  );

  int numChildren = rng.Next(3, 5);

  var usedChildSeeds = new HashSet<int>();
  for (int i = 0; i < numChildren; i++)
  {
    int childSeed;
    do
      childSeed = rng.NextInRange(ZombieUniverse.GlobalSeedRange);
    while (usedChildSeeds.Contains(childSeed));
    usedChildSeeds.Add(childSeed);

    TChild child = ZombieUniverse.Generate<TChild>(this, i, childSeed, transform);
  }
}
```



ZombieUniverse
Seed: 42

Galaxy
Index: 0
Seed: 62

Galaxy
Index: 1
Seed: 47

Galaxy
Index: 2
Seed: 31

Solar System
Index: 0
Seed: 2

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 16

Solar System
Index: 0
Seed: 30

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 40

Planet
Index: 0
Seed: 52

Planet
Index: 1
Seed: 33

Planet
Index: 2
Seed: 21

Planet
Index: 0
Seed: 55

Planet
Index: 1
Seed: 3

Planet
Index: 2
Seed: 12

City
Index: 0
Seed: 28

City
Index: 1
Seed: 3

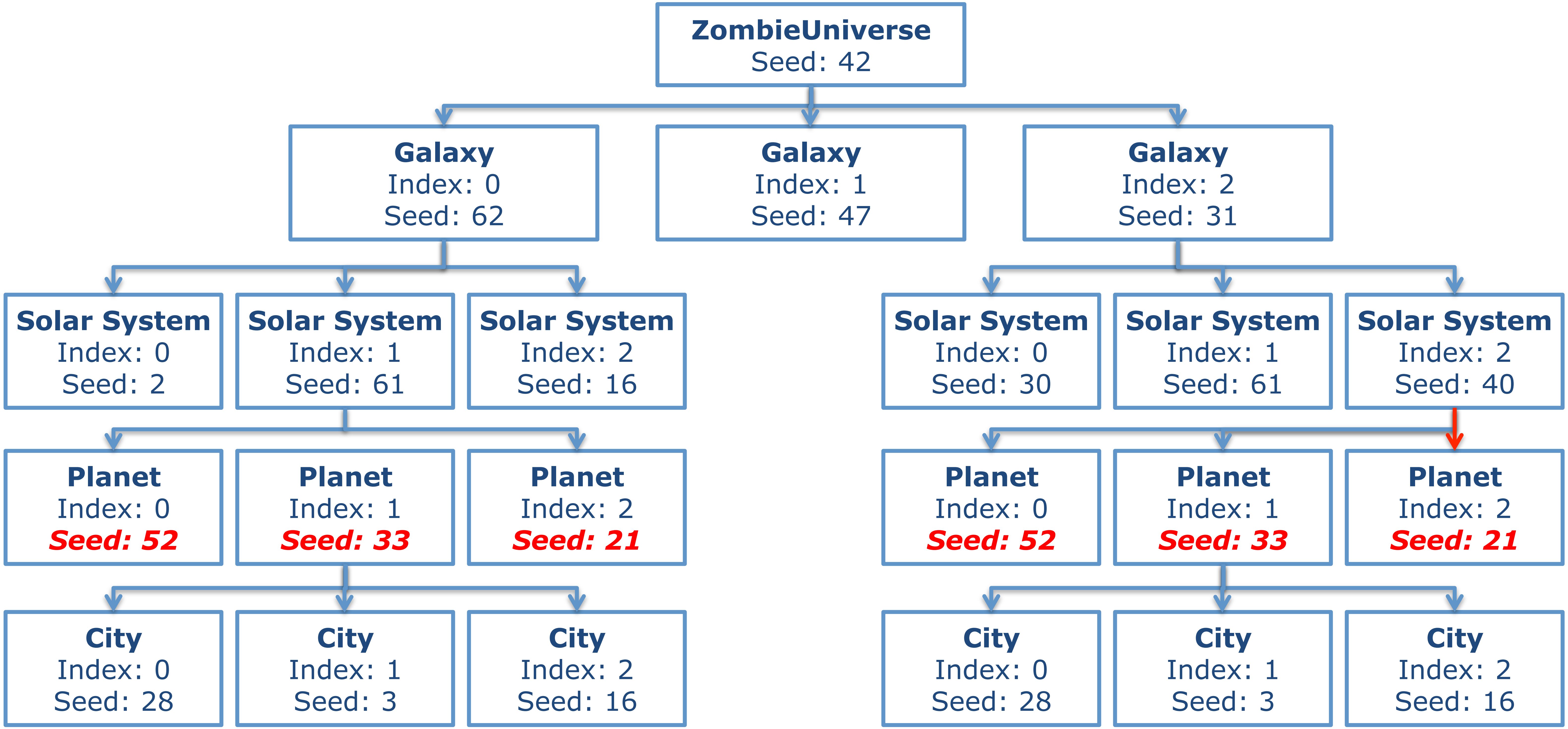
City
Index: 2
Seed: 16

City
Index: 0
Seed: 28

City
Index: 1
Seed: 3

City
Index: 2
Seed: 16

But What About “Broad Echoes”?



ZombieUniverse
Seed: 42

Galaxy
Index: 0
Seed: 62

Galaxy
Index: 1
Seed: 47

Galaxy
Index: 2
Seed: 31

Solar System
Index: 0
Seed: 2

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 16

Solar System
Index: 0
Seed: 30

Solar System
Index: 1
Seed: 61

Solar System
Index: 2
Seed: 40

Planet
Index: 0
Seed: 52

Planet
Index: 1
Seed: 33

Planet
Index: 2
Seed: 21

Planet
Index: 0
Seed: 52

Planet
Index: 1
Seed: 33

Planet
Index: 2
Seed: 21

City
Index: 0
Seed: 28

City
Index: 1
Seed: 3

City
Index: 2
Seed: 16

City
Index: 0
Seed: 28

City
Index: 1
Seed: 3

City
Index: 2
Seed: 16

Solving “Broad Echoes”:

- I let the solution as an exercise for the viewer.
- …By which I mean, I ran out of time .
- At some point the Pigeonhole Problem makes collisions and repetition inevitable.

Recommended Reading

Dan Cook on Loot Tables:

http://www.lostgarden.com/2014/12/loot-drop-tables.html

Unity Blog on Repeatable Random Numbers (by "runevision"):
https://blogs.unity3d.com/2015/01/07/a-primer-on-repeatable-random-numbers/

Image attributions:

Photo of die: By Ana - Flickr: Luck., CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=12530733

Deck of cards image:

By Christian Gidlöf – Photo taken by Christian Gidlöf, Public Domain, https://commons.wikimedia.org/w/index.php?curid=597083