About DartConnect's Random Draw Methodology

Let's Talk About Randomness

In computer science, creating truly random numbers is surprisingly hard. Luckily for us a lot of smart people did a lot of hard work for us to make our lives easier.

Let's start with the basics:

"The problem is that to humans, truly random does not *feel* random,"
Mattias Petter Johansson - Spotify

Randomness doesn't seem random to a normal person. Our brains are trained from birth to recognise patterns and see coincidences; this is because evolution-wise, it's a beneficial trait. The caveman who saw what appeared to be a face in the bushes and ran away was also the caveman who didn't get eaten by a tiger; no matter how many times he was wrong[1].

True randomness is so problematic to the average user that both iTunes and Spotify had to change their shuffle algorithm from true randomness to something that seems to be random to the average human[2], but actually isn't random at all.

The reality is that in a tournament situation the draw must be 100% random; even if people see patterns and coincidences that makes them suspect it may not be:

"Hey that guy signed up right before me and now I'm playing him, how can it be random?"

Creating an algorithm that makes things appear random but isn't would actually open up registration to gamification and do our customers a real disservice.

---

[1] Superior pattern processing is the essence of the evolved human brain
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC4141622/

[2] How to shuffle songs?
https://labs.spotify.com/2014/02/28/how-to-shuffle-songs/

So what is the DartConnect solution?

The bracket program uses the Javascript V8 engines built in random number generator that's based on the xorshift128+ algorithm[3]. Whilst not cryptographically secure, it has a period length of 2^{128} - 1 which means the original seed number won't repeat itself for a very, very, **VERY** long time. For a bracket generated with DartConnect we use the **Fisher-Yates Shuffle**[4] which produces an unbiased permutation (every permutation is equally likely) and is extremely efficient in conjunction with the xorshift128+ random number generation to produce a random draw.

Here's an example of that function in action:

```
var testArray = [1,2,3,4,5,6,7,8];
var shuffleArr = function(array) {
  var m = array.length, t, i;
  while (m) {
    i = Math.floor(Math.random() * m--);
    t = array[m];
    array[m] = array[i];
    array[i] = t;
  }
  return array;
}
for (var i=0; i<8; i++) {
  console.log(shuffleArr(testArray));
}
```

And here's the output:

```
[1, 7, 5, 8, 3, 6, 4, 2]
[4, 5, 3, 8, 1, 2, 7, 6]
[7, 6, 2, 4, 5, 3, 8, 1]
[3, 4, 7, 8, 1, 5, 2, 6]
[2, 1, 4, 5, 6, 8, 3, 7]
[3, 7, 8, 6, 4, 5, 1, 2]
[2, 5, 8, 3, 6, 7, 1, 4]
[7, 1, 3, 4, 8, 5, 6, 2]
```

[3] Further scramblings of Marsaglia's xorshift generators
http://vigna.di.unimi.it/ftp/papers/xorshiftplus.pdf

[4] Fisher–Yates Shuffle visualised
https://bost.ocks.org/mike/shuffle/

Any questions?

If you're that way minded and have any thoughts, suggestions or possible improvements; feel free to contact me and have a chat - I'm always open to new ideas!

Richard Barrington-Hill
http://barringtonmedia.co.uk/contact.html