A/B testing: what it is and how to use it

It's the technique that defines the games as a service model. Eric digs into the secrets of A/B testing

AUTHOR
ERIC MCCONNELL

Eric McConnell has been an engineer, game designer, and product manager at companies such as Rockstar Games and Zynga. He's now a product manager at Google where he works on Google Assistant games.

MODULO ARITHMETIC: EXPLAINED

Modulo arithmetic is one of the most important mathematical operations in programming. It can be thought of as the remainder after a division or the position after a wraparound computation. It is used in the form of A mod B, where the answer is the remainder of A divided by B. If A mod B = C then A = (B * k) + C where k is a constant.

Have you ever gone back and forth on multiple design decisions, wishing you could implement all of them and have players tell you which one they like best? Ever wondered how small reward changes could have a butterfly effect on major metrics? Modern game developers are implementing design experiments in live games: A/B testing is about applying scientific testing principles to game development, and provides a framework to test out design ideas. You start with a hypothesis, set up testing and control variants, run the experiment, and measure the results.

This can be used for finding which rewards increase a feature's engagement, how various level-tuning curves affect retention, and what properties to matchmake players on to ensure close skill levels. The best uses of A/B testing are for elements of your game that don't impact the player's expected behaviour. If the player expects a combination of buttons to shoot a fireball in your fighting game, then you can't change the result of those inputs to, say, an upper-cut.

Figure 1: The final results of Match-Pay-Win's two-week experiment.

Date	Control	Variant 1	Variant 2	Var 1 vs Control	Var 2 vs Control
3/1/2019	0.41	0.47	0.5	15%	22%
3/2/2019	0.63	0.69	0.72	10%	14%
3/3/2019	0.58	0.68	0.71	17%	22%
3/4/2019	0.39	0.45	0.41	15%	5%
3/5/2019	0.38	0.45	0.33	18%	-13%
3/6/2019	0.4	0.43	0.33	7%	-18%
3/7/2019	0.4	0.42	0.32	5%	-20%
3/8/2019	0.39	0.4	0.31	3%	-21%
3/9/2019	0.56	0.61	0.4	9%	-29%
3/10/2019	0.59	0.62	0.39	5%	-34%
3/11/2019	0.42	0.43	0.32	2%	-24%
3/12/2019	0.4	0.41	0.31	2%	-23%
3/13/2019	0.39	0.38	0.29	-3%	-26%
3/14/2019	0.39	0.39	0.3	0%	-23%
Avg	0.45	0.49	0.40	8%	-12%

	Control	Variant 1	Variant 2
ARPU	0.45	0.49	0.40
Delta		7.61%	-11.78%

THE SETUP

The first step to implementing and running game design experiments is to compile data from your gameplay. All the numeric values in your gameplay should be driven by a formatted data file, the most common format being JSON. This also lets you modify gameplay balance without recompiling your codebase.

The next step is identifying what you're testing and how you expect the experiment to turn out based on your current knowledge – referred to as a hypothesis. Do you think increasing daily rewards will increase session times? Are you expecting lower round times to also lower rage quit rates? As well as identifying your hypothesis, you also need to know what metrics you're using to measure success. Are you measuring daily active users? Are you looking at the average time spent in a specific game mode?

Next, identify the success threshold. If the difference between success metric variants is 1 percent, does that justify a winner or loser? Is a 10 percent delta – or change – considered a success, or 8 percent? Finally, identify how long you want to run your experiment for. The experiment length should be the same

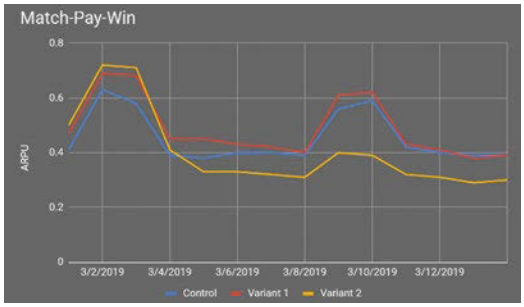

Figure 2: Variant 2 started off strong but quickly dropped against Variant 1 and the Control.

Figure 3: Variant 1 has a slow linear decline, whereas Variant 2 has a sharp delta drop.

as other, similar experiments, so that you can accurately compare results.

Let's create an imaginary game: Match-Pay-Win. It's a match-three puzzle game where the player completes levels by clearing tiles, with each level having a set move count. If the player runs out of moves, they can buy more moves and continue the level. I have a hypothesis: if I decrease the moves per level, I can increase revenue from each user. The metric I will track is Average Revenue Per User (ARPU) – the summation of revenue divided by the number of unique users. I'm going to say that, for this strategy to be considered successful, I need to see an ARPU increase of 10 percent, because anything less could equate to a rounding error. Finally, I'm willing to run this experiment for two weeks.

It's now time to designate your testing variants. A testing variant is a set of modified gameplay data that will be served to a group of players to test your hypothesis. One variant might get higher than normal rewards while another will get lower than normal rewards. While testing, you need to maintain a control variant, or a variant that maintains the status quo. For existing gameplay experiments, the control variant won't change the gameplay data, and neither will the control variant see new features.

"If I decrease the moves per level, I can increase revenue from each user"

The classic testing variant setup is 50/50, with half the players receiving the control variant and the other half receiving the testing variant. This can be expanded to support any number of scenarios. Maybe you have low, medium, and high rewards variants – if so, you could split your testing group into four, with the three subgroups each receiving low, medium, or high rewards, while the fourth subgroup serves as a control variant. For Match-Pay-Win, I want to test both my hypothesis and a more extreme version of it. Testing both at once can potentially save me an extra experiment if my hypothesis ends up being true. I'll run a control variant, my hypothesis variant (Variant 1), and a more extreme version of my hypothesis (Variant 2). The experiment will be run on a 34/33/33 percent split – the extra 1 percent going to the control variant won't hurt the results, because ARPU is normalised.

TIME TO EXECUTE

Now you can set up your gameplay data on your server according to your testing variants. How your game is programmed will dictate how this variant data reaches your players. The two most common implementations are sending gameplay data files down at the start of the game or having the gameplay logic sit server-side and dictate which variant to make gameplay calculations for.

To determine which variant a player falls in, you can use modulo arithmetic (see boxout). All modern game platforms have a user ID that you can retrieve and apply these calculations to. Take the player ID and mod it by the number of variants – this will give you the index of the variant for that player. If your player ID is 123456789 and you have three testing variants (Variant 1, Variant 2, and Control), your formula will be 123456789 % (mod) 3 = 0. The player will be in the 0th index, which is Variant 1. Now the only thing left to do is run the experiment and track your metrics of success.

In Match-Pay-Win (see **Figure 1**), the more extreme Variant 2 performed much worse than the Control variant – it started well, but quickly dropped to performing the worst of the three (see **Figure 2**). Maybe players started quitting due to the increased difficulty? We'd have to check other metrics to find out. Variant 1 performed better than the Control variant, but didn't meet our 10 percent threshold for success (see **Figure 3**). Variant 1 showed the most promise, so it may be worth retuning this experiment and running it again to find out more.

GOOD IDEAS, BAD IDEAS

One of the best uses for A/B testing is for rewards; testing loot tables, event rewards, or any place where randomised rewards are given out in a game. Reward experiments are great for measuring the short-term effects on player re-engagement and measuring long-term effects on the in-game economy and player wallets. New features are another great place to run experiments, because the player has yet to establish an expected behaviour for them compared to longer-running features.

Developer King Games makes constant use of A/B testing to balance *Candy Crush Saga*'s difficulty and player retention.