MATHS BEHIND ROOBET CRASH GAME

ADAM SIERAKOWSKI

Abstract. This documents discusses the maths behind the Roobet crash game. In particular we prove that on average a player gets back at most 95% of their bet.

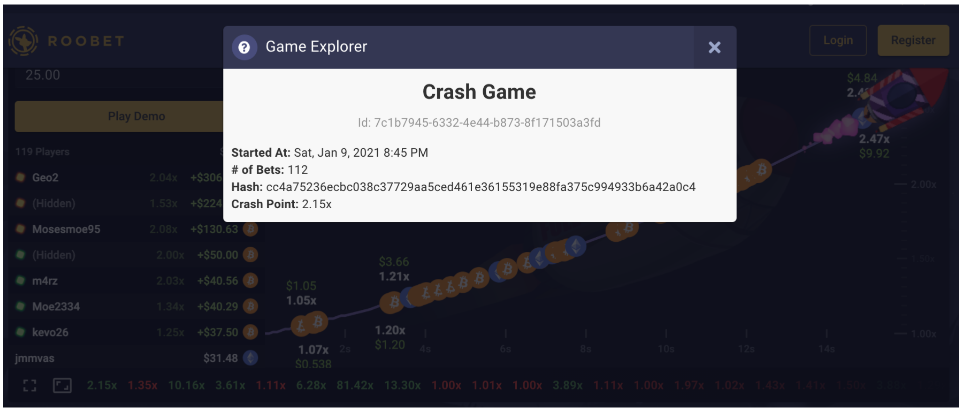
A search on roobet crash gives over half a million hits, quite popular. To see the game (you may need a VPN such as *Earth VPN*), go to

https://roobet.com/crash

1. The Crash Game

The crash game is based on you betting money and choosing your multiplayer. Then a rocket starts at multiplayer 1.00 and crashes at some (for you unknown) crash multiplayer. If your multiplayer is lower that the crash multiplayer, then you get your bet times your multiplayer back. Otherwise you loose your bet. So if you bet \$10, you choose multiplayer 1.5, and the rocket crashed at 2.00, then you win \$15, but if the rocket crashed at 1.25, you loose \$10.

The crash multipliers are all predetermined. Each game comes with a code called a Hash. The below shows a game with Hash



$cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c994933b6a42a0c4.$

With this information it is easy to compute the Hash of the game before.

$cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c994933b6a42a0c4$

$\Rightarrow fa0bd7818e238aa613426eba7422ca364369c0ec55767c8e023ba6d3ba161aeb$

The code is included in Listing 1 (final section). It is also easy to compute the crash multiplayer from the Hash code. This gives

$$cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c994933b6a42a0c4 \Rightarrow 2.15$$

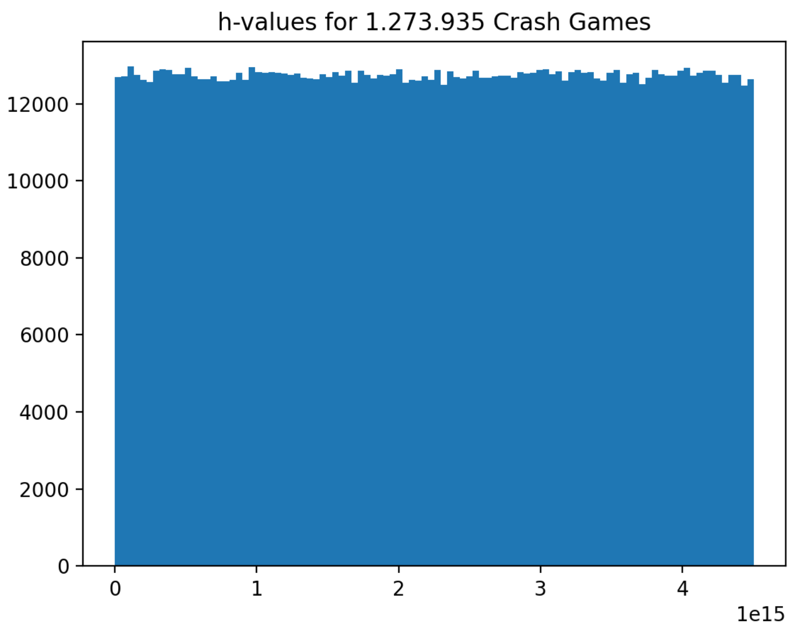
$$fa0bd7818e238aa613426eba7422ca364369c0ec55767c8e023ba6d3ba161aeb \Rightarrow 1.35$$

The code is included in Listing 2. Looking that the screen shot of the game, we see both of the numbers 2.15 and 1.35. Continuing this way we can compute all of the multipliers of the past. Also looking into the code one can see that all of the Hash codes must have been computed ahead of time, so Roobet is honest in saying that all crash multipliers are predetermined. Getting more technical, this is because of the irreversible nature of the SHA256.

The distribution of the crash multipliers. Roobet takes a clear cut: There is a 5% probability that a Hash automatically produces a crash multiplier equal to 1.00. For the remaining Hash codes, each one is truncated into its first 52 bits and converted to an integer h, so:

$$h \in \{0, 1, \ldots, 2^{52} - 1\}.$$

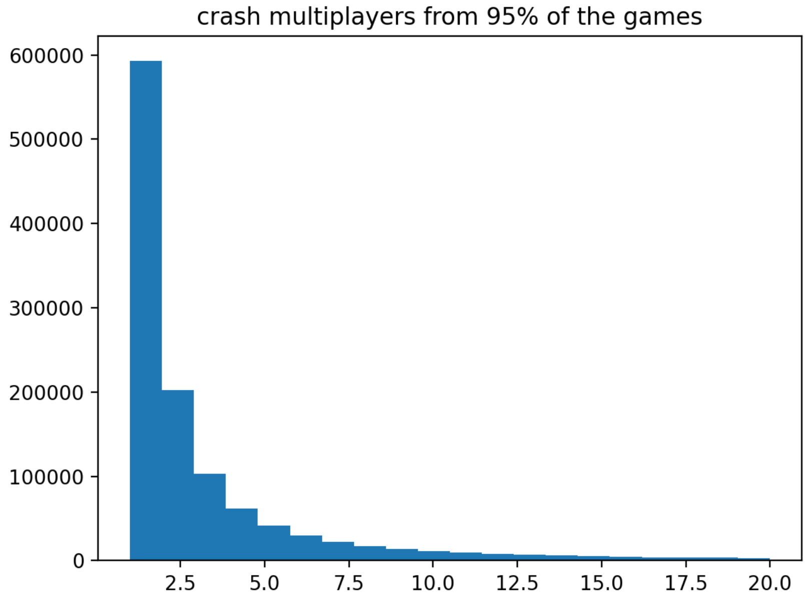
The distribution of the h's (coming form the games) is illustrated in the histogram below (the code is in Listing 3).



For each h the crash multiplier, say $cm(h)$, is given by

$$cm(h) = \frac{\left\lfloor \dfrac{100 \cdot 2^{52} - h}{2^{52} - h} \right\rfloor}{100}. \tag{1}$$

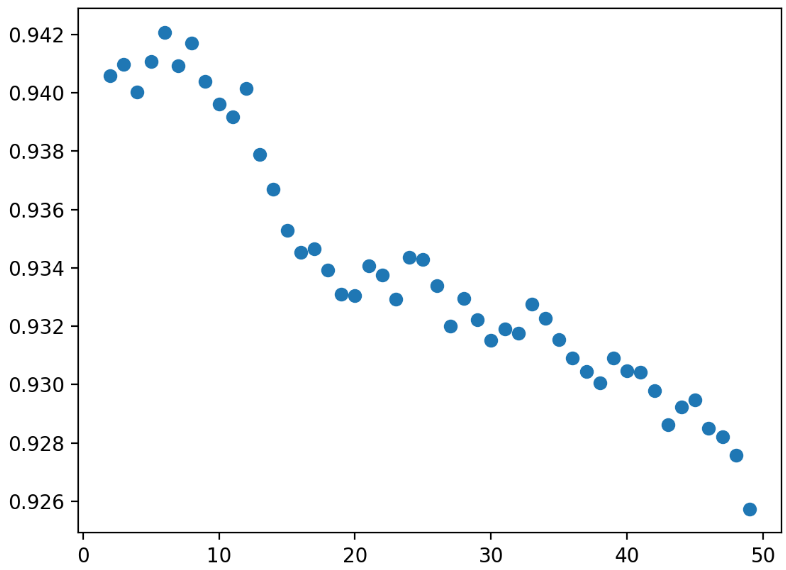
The crash multipliers (not the 5% automatically set to 1.00) are illustrated in the histogram below (the code is in Listing 4). The tail is not included.



Profit for varying strategies. Let us consider the case where someone bets \$1 on every game while consistently using the multiplayer equal to 2. The average profit per game (among all past games) would be 0.9405817408266512. On the next page the average profit is plotted as a function of a chosen multiplayer. Once again all of the (1273935) bets are \$1 each (the code is in Listing 5). It is clear that choice of the multiplayer will not make any of the strategies profitable.

Assuming that the underlying probability distribution for the values of h is a uniform (discrete) distribution and that the value of each h is independent of the previous values, one can theoretically derive that none of the strategies of the paragraph above are profitable (see Theorem 1).

Other strategies focus on changing size of the bet or multiplayer or a combination. But changing the size of bet will just get you to loose money faster of slower (on average) and changing size of the multiplayer will just shift you from one loosing strategy to the next. Consequently no such strategy is profitable.



2. The Proof

To showcase a bit math I have written a poof why the average reward per game is at most \$0.95.

Theorem 1. *Suppose all $h \in \{0, \dots, 2^{52}-1\}$ come from identical independent uniform discrete distributions. Then for any fixed multiplier $x > 1$ and bets of size \$1, the average reward per game is at most \$0.95.*

Proof. By construction of the get_multiplier() 5% of all games automatically crash at multiplier 1.00. It therefore suffices to show that for the remaining 95% of all games, the player at most breaks even (meaning the average profit is at most 0). Moreover, since all h's are independent and from identical uniform distributions,

$$E(P) := \text{average Profit over } 2^{52} \text{ games} = \sum_{h=0}^{2^{52}-1} \text{profit for a game}(h).$$

For h such that the chosen player multiplier x is less than the crash multiplier $cm(h)$ the profit for a game$(h) = x - 1$. For h such that $x \geq cm(h)$ the profit for a game$(h) = -1$ (loss of \$1). We get

$$\begin{aligned} E(P) &= \sum_{h:x<cm(h)} \text{profit for a game}(h) + \sum_{h:x\geq cm(h)} \text{profit for a game}(h) \\ &= \sum_{h:x<cm(h)} (x-1) + \sum_{h:x\geq cm(h)} (-1) \end{aligned}$$

Let N be the set of $h \in \{0, \ldots, 2^{52}-1\}$ such that $x < cm(h)$ so

$$N := \{h : x < cm(h)\},$$

and $e := 2^{52}$. Let $|N|$ denote the number of elements in N. We get

$$\begin{aligned} E(P) &= \sum_{h:x<cm(h)} (x-1) + \sum_{h:x\geq cm(h)} (-1) \\ &= |N|(x-1) + (e-|N|)(-1) = |N|x - e. \end{aligned}$$

Now suppose $E(P) > 0$. Then the above gives $|N|x - e > 0$ or $|N| > \frac{e}{x}$. But the following show that $|N| > \frac{e}{x}$ is impossible: Let $h = e - \frac{e}{x}$ (for now assume h is an integer). Then

$$100 \cdot cm(h) = 100 \cdot \frac{\left\lfloor \dfrac{100 \cdot e - h}{e-h} \right\rfloor}{100} = \left\lfloor \frac{100 - (1-\frac{1}{x})}{1-(1-\frac{1}{x})} \right\rfloor = \lfloor 99x + 1 \rfloor$$

Such h does not satisfy $x < cm(h)$, because if it did, then

$$100x < 100 \cdot cm(h) = \lfloor 99x+1 \rfloor \leq 99x + 1 \quad (\Rightarrow x < 1).$$

Therefore $h \notin N$. Let $h = 0$. Then $cm(h) = 1$. For this h, $x < cm(h)$ also fails so $h \notin N$. It follows that for all $h \in \{0, \ldots, 2^{52}-1\}$ such that $0 \leq h \leq e - \frac{e}{x}$, we get $h \notin N$. Consequently,

$$|N| \leq e - (e - \frac{e}{x} + 1) = \frac{e}{x} - 1,$$

so $|N| > \frac{e}{x}$ is impossible as claimed. Now if $h = e - \frac{e}{x}$ is not an integer, the same computation gives at last that $|N| \leq \frac{e}{x}$. Either way $|N| > \frac{e}{x}$ is impossible. Since the assumption that $E(P) > 0$ leads to a contradiction (gives both $|N| > \frac{e}{x}$ and $|N| \leq \frac{e}{x}$), we conclude that $E(P) \leq 0$ for any choice of $x > 1$. The average profit is at most 0, so the average reward per game is at most \$0.95. $\square$

3. The Code

Here I have included the Python code used or discussed earlier.

Listing 1. Code for prev_hash.

```
import hashlib


def prev_hash(hash_code):
    return hashlib.sha256(hash_code.encode()).hexdigest()


def main():
    game_hash = 'cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c\
994933b6a42a0c4'
    print(prev_hash(game_hash))


main()
```

OUTPUT:
fa0bd7818e238aa613426eba7422ca364369c0ec55767c8e023ba6d3ba161aeb

Listing 2. Code for get_multiplier.

```
import hmac
import hashlib


def hmac_hash(hash_code):
    key = "0000000000000000000fa3b65e43e4240d71762a5bf397d5304b259\
6d116859c"
    return hmac.new(hash_code.encode(), key.encode(),
                    digestmod=hashlib.sha256).hexdigest()


def get_multiplier(hash_code):
    hash_hex = hmac_hash(hash_code)
    if (int(hash_hex, 16) % 20 == 0):
        return 1
    h = int(hash_hex[:13], 16)
    e = 2 ** 52
    return (((100 * e - h) / (e - h)) // 1) / 100.0


def main():
    game_hash = 'cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c\
994933b6a42a0c4'
    print(get_multiplier(game_hash))
    game_hash = 'fa0bd7818e238aa613426eba7422ca364369c0ec55767c8e0\
23ba6d3ba161aeb'
    print(get_multiplier(game_hash))
```

```
main()
```

OUTPUT:
2.15
1.35

Listing 3. Code for h_distribution

```
import hmac
import hashlib
import matplotlib.pyplot as plt


def prev_hash(hash_code):
    return hashlib.sha256(hash_code.encode()).hexdigest()


def hmac_hash(hash_code):
    key = "0000000000000000000fa3b65e43e4240d71762a5bf397d5304b259\
6d116859c"
    return hmac.new(hash_code.encode(), key.encode(),
                    digestmod=hashlib.sha256).hexdigest()


def get_h(hash_code):
    return int(hmac_hash(hash_code)[:13], 16)


def h_distribution():
    game_hash = 'cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c9\
94933b6a42a0c4'  # the 1273934th game
    results = [get_h(game_hash)]
    for i in range(1273934):
        game_hash = prev_hash(game_hash)
        results.append(get_h(game_hash))
    plt.hist(results, bins=100)
    plt.title('h-values for 1.273.935 Crash Games')
    plt.show()


def main():
    h_distribution()

main()
```

OUTPUT:
Historgram (on p. 2)

Listing 4. Code for multiplayer_distribution

```
import hmac
import hashlib
import matplotlib.pyplot as plt


def prev_hash(hash_code):
    return hashlib.sha256(hash_code.encode()).hexdigest()


def hmac_hash(hash_code):
    key = "0000000000000000000fa3b65e43e4240d71762a5bf397d5304b259\
6d116859c"
    return hmac.new(hash_code.encode(), key.encode(),
                    digestmod=hashlib.sha256).hexdigest()


def get_multiplier_modified(hash_code):
    hash_hex = hmac_hash(hash_code)
    if (int(hash_hex, 16) % 20 == 0):
        return 20  # will not be shown
    h = int(hash_hex[:13], 16)
    e = 2 ** 52
    return (((100 * e - h) / (e - h)) // 1) / 100.0


def multiplayer_distribution():
    game_hash = 'cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c9\
94933b6a42a0c4'  # the 1273935th game
    results = [get_multiplier_modified(game_hash)]
    for i in range(1273934):
        game_hash = prev_hash(game_hash)
        mult = get_multiplier_modified(game_hash)
        if mult < 20:
            results.append(mult)
    plt.hist(results, bins=20)
    plt.title('crash multiplayers from 95% of the games')
    plt.show()


def main():
    multiplayer_distribution()


main()
```

OUTPUT:
Historgram (on p. 3)

Listing 5. Code for average_return

```
import hmac
import hashlib
```

```
# import numpy
import matplotlib.pyplot as plt


def prev_hash(hash_code):
    return hashlib.sha256(hash_code.encode()).hexdigest()


def hmac_hash(hash_code):
    key = "0000000000000000000fa3b65e43e4240d71762a5bf397d5304b259\
6d116859c"
    return hmac.new(hash_code.encode(), key.encode(),
                    digestmod=hashlib.sha256).hexdigest()


def get_multiplier(hash_code):
    hash_hex = hmac_hash(hash_code)
    if (int(hash_hex, 16) % 20 == 0):
        return 1.00
    h = int(hash_hex[:13], 16)
    e = 2 ** 52
    return (((100 * e - h) / (e - h)) // 1) / 100.0


def average_return(your_multiplyer):
    game_hash = 'cc4a75236ecbc038c37729aa5ced461e36155319e88fa375c9\
94933b6a42a0c4'  # the 1273934th game
    results = [get_multiplier(game_hash)]
    for i in range(1273934):
        game_hash = prev_hash(game_hash)
        results.append(get_multiplier(game_hash))
    return (sum(your_multiplyer < crash_multiplayer for crash_multiplayer
                in results) / len(results) * your_multiplyer)


def main():
    print(average_return(2))
    li = [[x, average_return(x)] for x in range(2, 50)]
    plt.scatter(*zip(*li))
    plt.show()


main()
```

OUTPUT:
0.9405817408266512
Scatter plot (on p. 4)