1
Devil's chessboard

Hamming codes are used to solve the problem[1] in this note.

You, your friend, and the Devil play a game. You and the Devil are in the room with a chess board with 64 tokens on it, one on each square. Meanwhile, your friend is outside of the room. The token can either be on an up position or a down position, and the difference in position is distinguishable to the eye. The Devil mixes up the positions (up or down) of the tokens on the board and chooses one of the squares and calls it the magic square. Next, you may choose one token on a square and flip its position. Then, your friend comes in and must guess what the magic square was by looking on the squares on the board.[2]

[1] Michael Tong. Devil's chessboard. 2013. URL https://brilliant.org/discussions/thread/the-devils-chessboard/

[2] Details:

1. You **may** flip a token. As in, you are not forced to flip a token; you may choose to not flip a token.
2. You can't just tell your friend what square it is. Or point to it. Or text him it. Or... you get the point.
3. Your friend knows the strategy as well (you tell him beforehand).
4. If you don't get it right, the Devil takes your soul. High stakes.

Problem

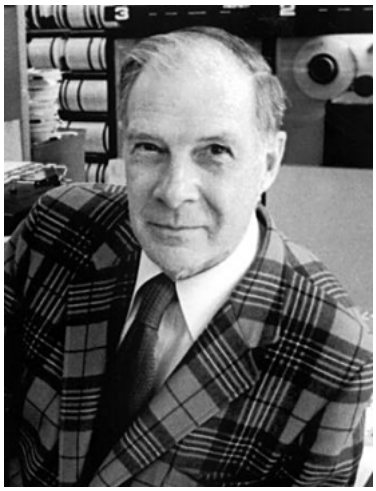
Show that there is a winning strategy such that your friend can always know what square the magic square is.

There might be solutions that exploit the chessboard geometry with its black and white fields. We will ignore the chessboard angle though and use this problem as an excuse to dive into the topic of linear codes. We will solve the problem by treating the token information as a 64-bit word and we will devise a winning strategy that involves a Hamming[3] code (a type of perfect linear code).

But first lets introduce linear codes. We operate in the field $\mathbb{F}_q$ of integers modulo a prime q.

Definition 1.1. A **linear code** C of words of length n is a subspace of the vector space $\mathbb{F}_q^n$. Let $\dim C = k$, then we say that C is a $[n, k]_q$ linear code.

Given a basis $\{\boldsymbol{c_1}, \boldsymbol{c_2}, \ldots, \boldsymbol{c_k}\}$ of C, we can build a matrix $G \in \mathbb{F}_q^{k \times n}$



[3] Richard Hamming was one of the founders of modern coding theory. http://en.wikipedia.org/wiki/Richard_Hamming

using the $\boldsymbol{c_i}$ basis vectors as rows. Then C is the row space of G and G is called a **generator matrix** of C. We have[4]

$$C = \{\boldsymbol{x}G : \boldsymbol{x} \in \mathbb{F}_q^k\},$$

so a code C is made from all linear combinations of the row vectors of its generator matrix.

Let G' be the row reduced echelon form of G. By definition G has full row rank, so G' has only nonzero rows. If $G' = \left[\, I_k \mid A_{k\times(n-k)} \,\right]$ for identity matrix I_k and some matrix A then the generator matrix G' is in **standard form**[5]. Row operations preserve the row space, so G' also generates C.

Definition 1.2. Given a $[n,k]_q$ linear code C, matrix $H \in \mathbb{F}_q^{(n-k)\times n}$ is a **parity check matrix** for C, if $C = nullspace(H) = \{\boldsymbol{c} \in \mathbb{F}_q^n : H\boldsymbol{c}^T = 0\}$.

Theorem 1.3. *Given a $[n,k]_q$ linear code C and a generator matrix $G = \left[\, I_k \mid A_{k\times(n-k)} \,\right]$ for C in standard form, then $H = \left[\, -A^T_{(n-k)\times k} \mid I_{n-k} \,\right]$ is a parity check matrix[6] for C.*

Proof. Let $\boldsymbol{c} \in C$ be a code word from C. Then there exists an $\boldsymbol{x} \in \mathbb{F}_q^k$ such that $\boldsymbol{c} = \boldsymbol{x}G$. We have

$$\begin{aligned}
H\boldsymbol{c}^T &= H(\boldsymbol{x}G)^T \\
&= \left[-A^T_{(n-k)\times k} | I_{n-k}\right] \left(\boldsymbol{x}\left[I_k | A_{k\times(n-k)}\right]\right)^T \\
&= \left[-A^T_{(n-k)\times k} | I_{n-k}\right] \left[\frac{I_k}{A^T_{(n-k)\times k}}\right] \boldsymbol{x}^T \\
&= (-A^T + A^T)\boldsymbol{x}^T \\
&= 0
\end{aligned}$$

This means that $C \subseteq nullspace(H)$. We have $\dim C = k$ and

$$\dim nullspace(H) = n - rank(H) = n - n + k = k,$$

so $C = nullspace(H)$ and H is a parity check matrix for C.

□

What can we do if the generator matrix is not in standard form? Swapping columns in the generator matrix does not preserve the row space, so the linear code generated with the modified matrix is clearly not the same as the original code, but it is an equivalent code[7].

Definition 1.4. The **Hamming distance** $d(\boldsymbol{x},\boldsymbol{y})$ between two vectors $\boldsymbol{x},\boldsymbol{y} \in \mathbb{F}_q^n$ is the number of positions in which the vectors differ. With $\boldsymbol{x} = x_1x_2\ldots x_n$ and $\boldsymbol{y} = y_1y_2\ldots y_n$ we have

$$d(\boldsymbol{x},\boldsymbol{y}) = |\{i : 1 \le i \le n : x_i \neq y_i\}|$$

[4] We treat vectors as row vectors in this section. That means that $\boldsymbol{x} \in \mathbb{F}_q^k$ is a matrix $\mathbb{F}_q^{1\times k}$.

[5] Not every generator matrix can be row reduced to the standard form. For example

$$\begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

cannot.

[6] With G in standard form this theorem let's us construct a parity check matrix very easily. Also worth noting that in standard form we generate a code word from a message $\boldsymbol{x} \in \mathbb{F}_q^k$ by appending $n-k$ parity check bits to the message with $\boldsymbol{x}G$. We check if the transmitted and received word $\boldsymbol{y} \in \mathbb{F}_q^n$ is a valid code word by verifying $H\boldsymbol{y}^T = 0$. If true then the first k positions of $\boldsymbol{y}$ are the original message $\boldsymbol{x}$.

[7] A $[n,k_1]_q$ linear code C_1 is equivalent to a $[n,k_2]_q$ linear code C_2 if there is a permutation $\pi \in S_n$ such that when π is applied to the coordinate indices of all the code words from C_1, it produces all the code words from C_2. Equivalent linear codes have the same dimension $k_1 = k_2$.

The **Hamming weight** $w(\boldsymbol{x})$ is the number of positions that differ from zero:

$$w(\boldsymbol{x}) = |\{i : 1 \leq i \leq n : x_i \neq 0\}| = d(\boldsymbol{x}, \boldsymbol{0})$$

We will use the following properties of Hamming distances:

Lemma 1.5.

$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{x}) \geq 0$$
$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{y}) = 0 \Leftrightarrow x = y$$
$$\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{y}) = d(\boldsymbol{y}, \boldsymbol{x})$$
$$\forall \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z} \in \mathbb{F}_q^n : d(\boldsymbol{x}, \boldsymbol{y}) \leq d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y})$$

Proof of Lemma 1.5
The first three properties are obvious from the definition of Hamming distance. For the last property let i be an index where $\boldsymbol{x}$ and $\boldsymbol{y}$ differ, so $x_i \neq y_i$. For vector $\boldsymbol{z}$ we can have the following cases for position i:

$$z_i = x_i \Rightarrow z_i \neq y_i$$
$$z_i = y_i \Rightarrow z_i \neq x_i$$
$$z_i \neq x_i \wedge z_i \neq y_i$$

In each of these cases the contribution of z_i to $d(\boldsymbol{x}, \boldsymbol{z}) + d(\boldsymbol{z}, \boldsymbol{y})$ is at least one, whereas on the left side position i contributes one to $d(\boldsymbol{x}, \boldsymbol{y})$. A similar analysis holds for indices i where $x_i = y_i$. □

Definition 1.6. The **minimum distance** of C is:

$$d(C) = \min\{d(\boldsymbol{x}, \boldsymbol{x'}) : \boldsymbol{x}, \boldsymbol{x'} \in C \wedge \boldsymbol{x} \neq \boldsymbol{x'}\} = \min\{w(\boldsymbol{x}) : \boldsymbol{x} \in C\}$$

The minimum distance is important enough that we add it to the characteristic notation of a linear code: $[n, k, d]_q$ is a linear code over field $\mathbb{F}_q$ with bit strings of length n, code dimension k and minimum distance between code words d.

The next lemma establishes a connection between the minimum distance of a linear code and one of its parity check matrix.

Lemma 1.7. *The minimum distance of a code C equals the minimum number of linearly dependent columns in one of its parity check matrices.*

Proof of Lemma 1.7
Let H be a parity check matrix of $[n, k, d]_q$ linear code C. There must be a code word $\boldsymbol{c}$ with $w(\boldsymbol{c}) = d$. $\boldsymbol{c}$ belongs to the nullspace of H, so

$$H\boldsymbol{c}^T = \boldsymbol{0}$$

But $H\boldsymbol{c}^T$ is a linear combination of column vectors of H, with d nonzero coefficients, so the column vectors in this linear combination are linearly dependent. □

So far we have worked with fields $\mathbb{F}_q$ of any prime q. Now we switch to the binary world $q = 2$ and $\mathbb{F}_2$. Our vectors are bit strings. We transmit these bit strings over a binary symmetric channel.

Definition 1.8. In a **binary symmetric channel** each bit sent has the same probability $p < \frac{1}{2}$ of being received incorrectly.

We send a code word $\boldsymbol{x} \in C$ from a $[n, k]_2$ linear code C over a binary symmetric channel and receive a bit string $\boldsymbol{y}$. If there were no transmission errors, then $y = x$. If there were errors, we want to find the most likely code word $\boldsymbol{x}$ that was transmitted given the errors in $\boldsymbol{y}$.

One decoding strategy[8] would be to choose a code word $\boldsymbol{x}$ with minimum Hamming distance over all code words from C to received bit string $\boldsymbol{y}$. This type of decoding is called *nearest neighbor decoding*. The chosen $\boldsymbol{x}$ is not always unique.

[8] Finding an appropriate code word for the transmitted bit string is called *decoding*. Finding the most likely code word is called *maximum likelihood decoding*.

Theorem 1.9. *In a binary symmetric channel with error probability $p < \frac{1}{2}$ the nearest neighbor decoding is a maximum likelihood decoding.*

Proof. Given a bit string $\boldsymbol{y} \in \mathbb{F}_2^n$ received through the channel, let $P_{\boldsymbol{y}}(\boldsymbol{x})$ be the probability that the code word $\boldsymbol{x}$ was sent when $\boldsymbol{y}$ was received. Because the channel is a binary symmetric channel, we have

$$P_y(x) = p^{d(x,y)}(1-p)^{n-d(x,y)}$$

Consider two code words x and x' such that $d(x,y) \leq d(x',y)$. Because $p < \frac{1}{2}$, we then have $P_y(x) \geq P_y(x')$. It follows that

$$\max_{x \in C} P_y(x) = \min_{x \in C} d(x,y)$$

so the likeliest code word is the nearest neighbor to y. □

For the rest of this section we use nearest neighbor decoding. We want to know if we can detect and possibly correct a transmission with errors. Let's define clearly what we mean by that. A transmission is a pair $(x,y) \in C \times \mathbb{F}_2^n$, where a code word x was sent and a bit strings y was received. It has $d(x,y)$ transmission errors. The nearest neighbor decoding $nnd(y)$ finds a code word (not necessarily unique) closest to y. The following holds by definition:

$$d(y, nnd(y)) = \min_{c \in C} d(y,c)$$

If no errors occurred in the transmission, then $x = y$ and also $d(x,y) = 0$ and $nnd(y) = x$. If errors in the transmission occurred we want to:

E.1 detect that errors happened, i.e. establish that $y \notin C$.

E.2 correct the errors, i.e. establish $nnd(y) = x$.

The next theorem describes the conditions for **E.1**.

Theorem 1.10. *Given a $[n,k,d]_2$ linear binary code C, we can detect that any transmission with up to e errors was erroneous if and only if $d > e$.*

Proof. ($\Rightarrow$) Let (x,y) be a transmission with $d(x,y) \leq e < d$ errors. Assume $y \in C$. Then $d(x,y) \leq e < d$ is a contradiction to d being the minimal distance of C. It follows that $y \notin C$.
($\Leftarrow$) We can detect that any transmission with up to e errors was erroneous. Assume $d \leq e$. Then there exist two code words $x \neq x'$ such that $d(x,x') \leq e$. Now consider transmission (x,x'). It's impossible to detect that it had errors because x' is a code word. This is a contradiction with the fact that we can detect that any transmission with up to e errors was erroneous. So $d > e$. □

For **E.2** we have this theorem:

Theorem 1.11. *Given a $[n,k,d]_2$ linear binary code C, we can correct any transmission with up to e errors if $d > 2e$.*

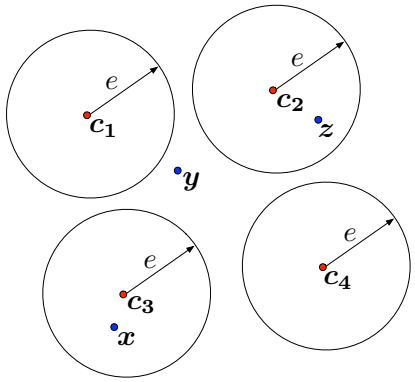


Figure 1.1: A Hamming sphere for code word c with radius e is the set $\{x : d(x,c) \leq e\}$. In this figure the spheres don't overlap, so vectors (blue dots) that fall within a sphere can be error-corrected to code words (red dots).

Proof. Let $(\boldsymbol{x}, \boldsymbol{y})$ be a transmission with $d(\boldsymbol{x}, \boldsymbol{y}) \leq e$ errors and $d > 2e$. Assume $nnd(\boldsymbol{y}) \neq \boldsymbol{x}$. Then $d(\boldsymbol{y}, nnd(\boldsymbol{y})) \leq e$ (otherwise $\boldsymbol{x}$ would be closer than $nnd(\boldsymbol{y})$ to $\boldsymbol{y}$). We have

$$d(\boldsymbol{x}, nnd(\boldsymbol{y})) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, nnd(\boldsymbol{y})) \leq e + e = 2e$$

which contradicts $d > 2e$. So $nnd(\boldsymbol{y}) = \boldsymbol{x}$.

□

Theorems 1.10 and 1.11 tell us that a large minimum distance $d(C)$ allows us to detect and correct more errors. But a large minimum distance between code words also limits the number of code words. The following theorem puts an upper bound on the number of code words given a minimum distance.

Theorem 1.12. *Given a $[n, k, 2t+1]_2$ linear binary code C, we have*

$$|C| \leq \frac{2^n}{\sum_{i=0}^{t} \binom{n}{i}}$$

*This upper bound is called **Hamming bound**.*

Proof. Given a bit string $\boldsymbol{x}$ and an integer $i \leq n$, there are $\binom{n}{i}$ ways to choose the i positions at which $\boldsymbol{x}$ and and another bit string $\boldsymbol{y}$ differ. So there are $\binom{n}{i}$ bit strings $\boldsymbol{y}$ with $d(\boldsymbol{x}, \boldsymbol{y}) = i$. This means there are

$$\sum_{i=0}^{t} \binom{n}{i}$$

bit strings $\boldsymbol{y}$ with $d(\boldsymbol{x}, \boldsymbol{y}) \leq t$.

On the other hand, a bit string $\boldsymbol{y}$ with $d(\boldsymbol{y}, \boldsymbol{x}) \leq t$ to a code word $\boldsymbol{x}$ cannot have the same $d(\boldsymbol{y}, \boldsymbol{x'}) \leq t$ to a different code word $\boldsymbol{x'}$ because then

$$d(\boldsymbol{x}, \boldsymbol{x'}) \leq d(\boldsymbol{x}, \boldsymbol{y}) + d(\boldsymbol{y}, \boldsymbol{x'}) \leq t + t \leq 2t$$

which is a contradiction to $d(C) = 2t + 1$.

So for each code word, we have at most $\sum_{i=0}^{t} \binom{n}{i}$ bit strings with Hamming distance $\leq t$ and we cannot have the same bit strings near two different code words. We have 2^n bit strings, so

$$|C| \sum_{i=0}^{t} \binom{n}{i} \leq 2^n$$

□

A binary linear code that achieves equality in the Hamming bound 1.12 is called a **perfect code**.

We are now ready to define Hamming codes.

Definition 1.13. A Hamming code $\mathfrak{H}_r$ of order r (where r is a positive integer) is a binary linear code with the parity check matrix with columns that are all the $2^r - 1$ nonzero bit strings of length r.

Changing the order of the columns in the parity check matrix produces equivalent codes with the same minimum distance. So for easier analysis we now consider Hamming codes with parity check matrix in standard form, ie the last r columns form the identity matrix I_r, so $H = \left[A_{r\times(n-r)} \mid I_r \right]$, with $n = 2^r - 1$. From theorem 1.3 we then know the generator matrix is $G = \left[I_{n-r} \mid -A^T_{(n-r)\times r} \right] = \left[I_{n-r} \mid A^T_{(n-r)\times r} \right]$, since we operate in $\mathbb{F}_2$. We can see that $\dim \mathfrak{H}_r = n - r$. What is the minimum distance of $\mathfrak{H}_r$? All columns are nonzero and distinct, so no two columns are linearly dependent[9]. But consider the linear combination of the three columns

$$[1, 1, 0, \ldots, 0]^T + [1, 0, 0, \ldots, 0]^T + [0, 1, 0, \ldots, 0]^T = \mathbf{0}^T$$

They are linearly dependent. From lemma 1.7 it follows that $d(\mathfrak{H}_r) = 3$, so $\mathfrak{H}_r$ is a $[2^r - 1, 2^r - 1 - r, 3]_2$ binary linear code. According to theorem 1.11 it can correct transmissions with one error.

[9] Again, this is in $\mathbb{F}_2$. The sum of two distinct columns is always nonzero, so a linear combination that is zero has to have coefficients zero, hence linearly independent.

Theorem 1.14. *$\mathfrak{H}_r$ is a perfect code.*

Proof. The generator matrix has full row rank, so we need all linear combinations of the rows to get all the code words. This are binary words, so there are 2^{n-r} distinct linear combinations. It means $|\mathfrak{H}_r| = 2^{n-r}$.

Inserting into formula of theorem 1.12, we get[10]

[10] With $t = 1$, because $d(\mathfrak{H}_r) = 3$.

$$2^{n-r} \sum_{i=0}^{1} \binom{n}{i} = 2^{n-r}(1 + n) = 2^{n-r}(1 + 2^r - 1) = 2^{n-r}2^r = 2^n$$

□

This concludes our dive into linear codes and Hamming codes. Let's return to our problem and solve it using Hamming codes. The state of the chessboard is a binary word of length 64. We use $r = 6$, so Hamming code $\mathfrak{H}_6$. The word length is $2^6 - 1 = 63$. We agree that the devil choosing bit 64 is a special case which we handle later. For now imagine the chessboard as a 63-bit binary word and the devil only choosing a magic field between 1 and 63.

The winning strategy can be summarized as follows: the first player needs to modify the 63-bit word (by flipping at most one bit) in such a way that the magic field is the one bit error of a code word in $\mathfrak{H}_6$. Then the second player only has to come in, decode[11] the modified chessboard and point to the corrected error which is the same magic field.

Is this always possible? We know that Hamming codes are perfect codes, so any 63-bit word is at most one bit away from a code word. We have the following cases for the initial state of the chessboard:

- It happens to be a code word in $\mathfrak{H}_6$. Then the first player flips the magic field bit, producing an error there.
- It happens to be a 63-bit word that is a one bit error at the magic field. The first player doesn't flip any bit in this case.
- It happens to be a 63-bit word with a one bit error different from the magic field.

The last case needs a little thinking. Assume H is the parity check matrix for our $\mathfrak{H}_6$ Hamming code and assume the state of the chessboard is $\boldsymbol{x}$, which is one bit error from a code word $\boldsymbol{c_1}$. Also let $1 \leq m \leq 63$ be the magic field bit and $\boldsymbol{e_m}$ the unit vector with bit m set. The one bit error is different from the magic field, so $\boldsymbol{x} - \boldsymbol{c_1} \neq \boldsymbol{e_m}$. Let $\boldsymbol{y} = \boldsymbol{x} - \boldsymbol{e_m}$, which is also one bit away from a code word $\boldsymbol{c_2}$, with error bit k. So $\boldsymbol{y} = \boldsymbol{c_2} + \boldsymbol{e_k}$.

Now consider $\boldsymbol{x} - \boldsymbol{e_k}$:

$$H(\boldsymbol{x} - \boldsymbol{e_k}) = H(\boldsymbol{y} + \boldsymbol{e_m} - \boldsymbol{e_k}) = H(\boldsymbol{y} - \boldsymbol{e_k}) + H\boldsymbol{e_m} = H\boldsymbol{c_2} + H\boldsymbol{e_m} = H\boldsymbol{e_m}$$

So $\boldsymbol{x} - \boldsymbol{e_k}$ has one bit error at the magic field, which is what we want. Flipping bit k on the initial chessboard $\boldsymbol{x}$ achieves that.

In all three cases the modified chessboard is one bit away from a code word with the error at the magic field and the chessboard was modified by flipping at most one bit. The players agree that if the chessboard is a code word instead, then the devil chose bit 64 as the magic field, which handles the special case. Modifying the chessboard to get a code word can also be done by flipping at most one bit. This scales to any chessboard with size a power of two.

[11] Decoding is done as follows: $\boldsymbol{x}$ needs to be decoded. It is one bit away from a code word $\boldsymbol{c}$ with error at bit k. Let $\boldsymbol{e_k}$ be the unit vector with bit k set. So $\boldsymbol{x} = \boldsymbol{c} + \boldsymbol{e_k}$ and

$$H\boldsymbol{x} = H(\boldsymbol{c} + \boldsymbol{e_k}) = H\boldsymbol{e_k}$$

Since $\boldsymbol{e_k}$ is be the unit vector with bit k set, $H\boldsymbol{e_k}$ is column k from the parity check matrix H. To decode we calculate $H\boldsymbol{x}$ and look to see which column in H the result is. To save the lookup step we can be even more elegant. Instead of the parity check matrix in standard form, we choose a parity check matrix where column k is the bit representation of k. Instead of lookup we just reverse the bit representation back to the integer k.

What follows is a Mathematica session illustrating the strategy. We use a parity check matrix with column k the bit representation of integer k. This simplifies decoding as remarked in the side note 11 above.

The function *hamming* generates the parity check matrix for a Hamming code with a given r.

```
In[1]:= hamming[r_Integer] := Transpose[Table[IntegerDigits[i, 2, r], {i, 1, 2^r - 1}]]
```

For example

```
In[2]:= hamming[4] // MatrixForm
```

Out[2]=

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{pmatrix}$$

We define r and the corresponding Hamming code h for our chessboard

```
In[3]:= r = 6;
```

```
In[4]:= h = hamming[6];
```

The function *pos* returns the unit vector with the error bit set from decoding the specified word.

```
In[5]:= pos[w_] := With[{s = Mod[h.w, 2]}, UnitVector[2^Length[s] - 1, FromDigits[s, 2]]]
```

Function *friendOne* implements the strategy part for the first friend. Given the initial state of the chessboard *cb* and a magic field *mf*, it returns a modified chessboard.

```
In[6]:= friendOne[cb_List, mf_Integer] := Module[{em, y, ey},
        em = UnitVector[2^r - 1, mf]; y = Mod[cb - em, 2]; ey = pos[y];
        z = Mod[cb - ey, 2]
        ]
```

Function *friendTwo* implements the strategy part for the second friend: decoding the specified chessboard and returning the index of the error bit which is also the magic field.

```
In[7]:= friendTwo[cb_List] := Position[pos[cb], 1][[1,1]]
```

This next function is returning random initial states for the chessboard.

```
In[8]:= rw := RandomInteger[1, {2^r - 1}]
```

We can now simulate one game with the devil.
cb is the initial (random) state of the chessboard.

```
In[9]:= cb = rw;
```

The magic field is some integer, the devil chose 23.

```
In[10]:= mf = 23;
```

The first friend enters the room, modifies the chessboard according to *friendOne*. The returned value is the modified chessboard.

```
In[11]:= cb2 = friendOne[cb, mf];
```

Let's check that the Hamming distance between initial and modified chessboard is at most one.

```
In[12]:= HammingDistance[cb, cb2]

Out[12]= 1
```

The second friend comes in and decodes with *friendTwo*, getting 23.

```
In[13]:= friendTwo[cb2]

Out[13]= 23
```

Bibliography

Michael Tong. Devil's chessboard. 2013. URL https://brilliant.org/discussions/thread/the-devils-chessboard/.