```
+--------------------+
|   CRYPTO CHUNK     |
+--------------------+
```

Welcome to this sample project!

The goal of this project was to test some ways to store a bunch of information directly on the blockchain instead of using a p2p storage.

Observations: By using a mapping(blockLocation => BlockColor) it is easy to store all the information. The problem with this storage method, there is no way to extract all the modified blocks in a single shot without using a loop or making multiple calls to a get method.

To solve this problem, the information could be stored in a single string and when the user requests the chunk information, it would come as a single string of data and then split in blocks on the off chain part. The problem with the second method is how would we remove a block from the middle of the string without having to iterate over and/or rewrite the whole thing.

The conclusion is: It is probably possible to store the information of a game map on the chain but it would need a way faster method to extract the information.

```
    -- How to start and test --
1 - start by connecting to metamask on the Bsc Testnet
2 - click load chunk to see the blocks
3 - Add some blocks to the chunk
4 - Once a few blocks have been placed, click save progress to send the changes to the blockchain
5 - To start over click reset chunk
```

made by Frederic Cote https://github.com/FredCoteMtl

```
pragma solidity ^0.4.8;

contract SimpleStorage {
    uint chunkSize;
    mapping(uint => bytes) locList;
    mapping(bytes => bytes) blocksCol;

    constructor() public {
        chunkSize = 0;
    }

    function set(bytes loc, bytes col) public {
        chunkSize = chunkSize + 1;
```

```
        locList[chunkSize] = loc;
        blocksCol[loc] = col;
    }

    function setProgress(uint size, bytes locs, bytes cols) public {
        uint j = 0;
        for(uint i=0; i<size*2; i=i+2){
            set(
                 abi.encodePacked(locs[i], locs[i+1]),
                 abi.encodePacked(cols[j], cols[j+1], cols[j+2])
            );
            j = j + 3;
        }
    }

    function getChunkSize() public view returns (uint) {
        return chunkSize;
    }

    function getColWithIndex(uint i) public view returns (bytes) {
        bytes storage loc = locList[i];
        return blocksCol[loc];
    }
    function getColWithLoc(bytes loc) public view returns (bytes) {
        return blocksCol[loc];
    }

    function getLoc(uint i) public view returns (bytes) {
        return locList[i];
    }

    function resetChunk() public {
        chunkSize = 0;
    }
}
```