Run Length Encoding

Uses **frequency/data** pairs to compress data

Applying an RLE algorithm is a **lossless** compression method

It is effective when there are **runs** of the same data value

Run Length Encoding

BBBBBBWWWWBBWWW

This set of data has 4 runs in it - a run of 6Bs, then 4Ws, then 2Bs and finally a run of 3Ws

It is a good candidate to be reduced in file size using RLE

Run Length Encoding

BBBBBBWWWWBBWWW

After applying RLE this becomes:

6B4W2B3W

This is shorter - so RLE has reduced the file size

(it's not quite as simple as this, but for exams this all you need to know!)

Run Length Encoding

RRRGGGGGGGGBGGGGGRGB

Will RLE reduce the file size of this data?

How many runs of data are there?

Run Length Encoding

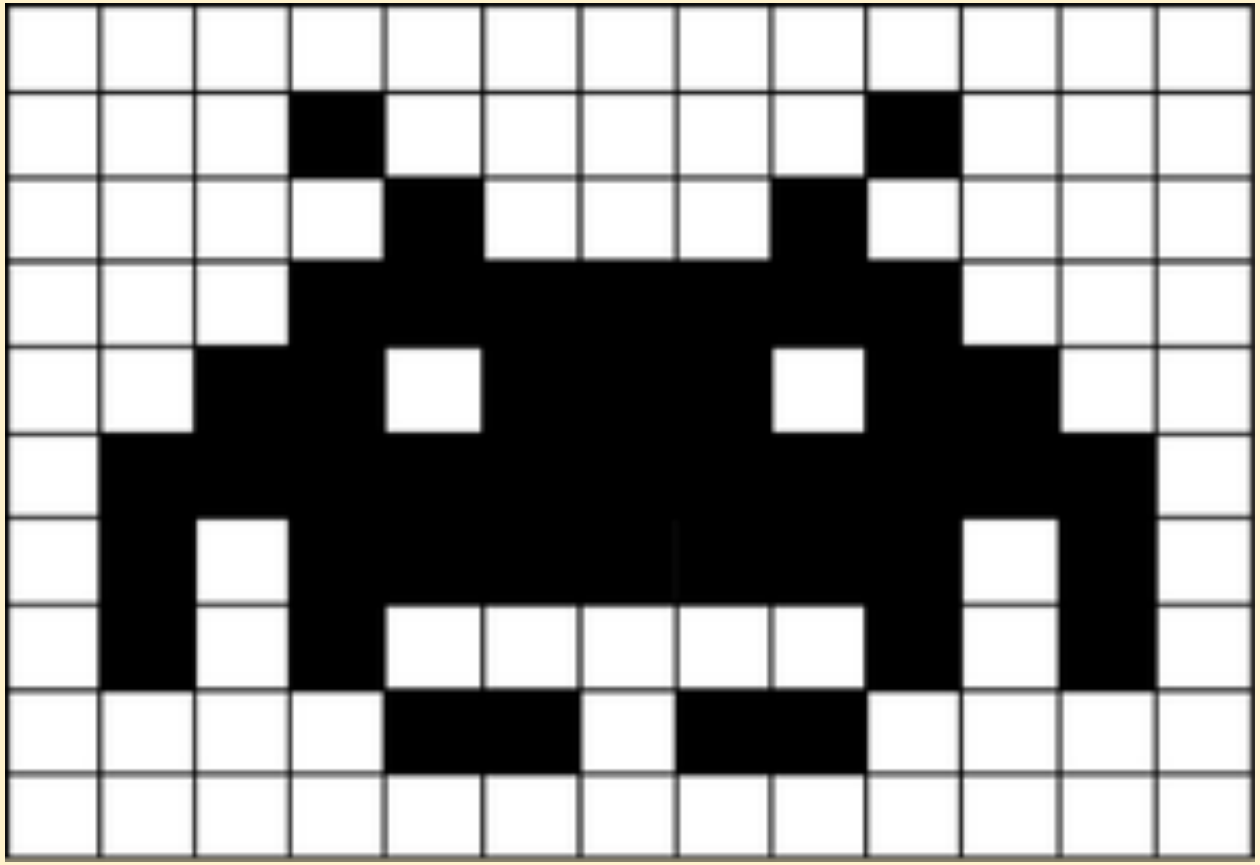
RRRGGGGGGGGBGGGGRGB

This becomes:

3R7G1B4G1R1G1B

Run Length Encoding

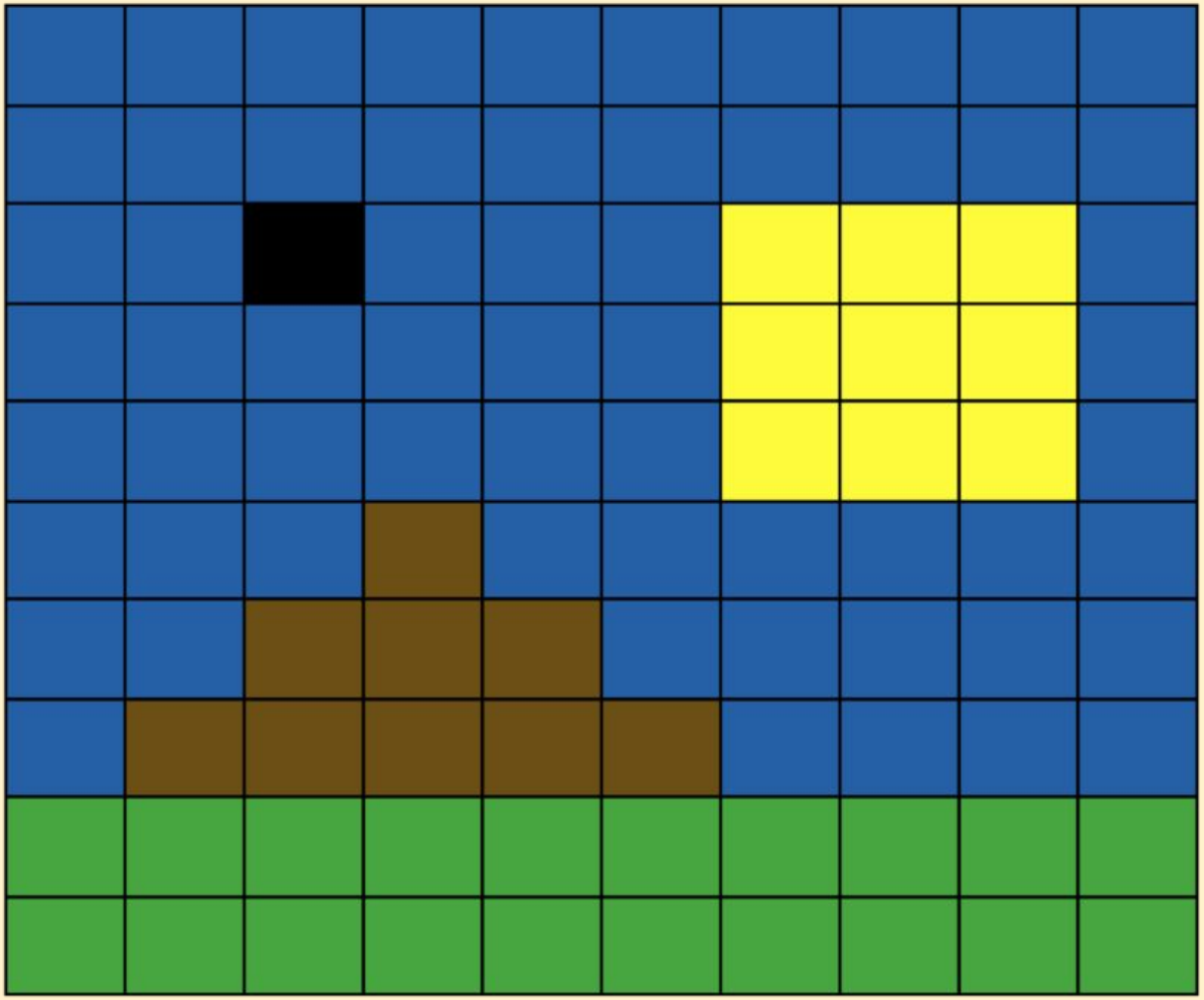
RLE works well on simple bitmap images



Bitmap 13 pixels wide and 10 pixels high using 1 bit colour depth

White is 1
Black is 0

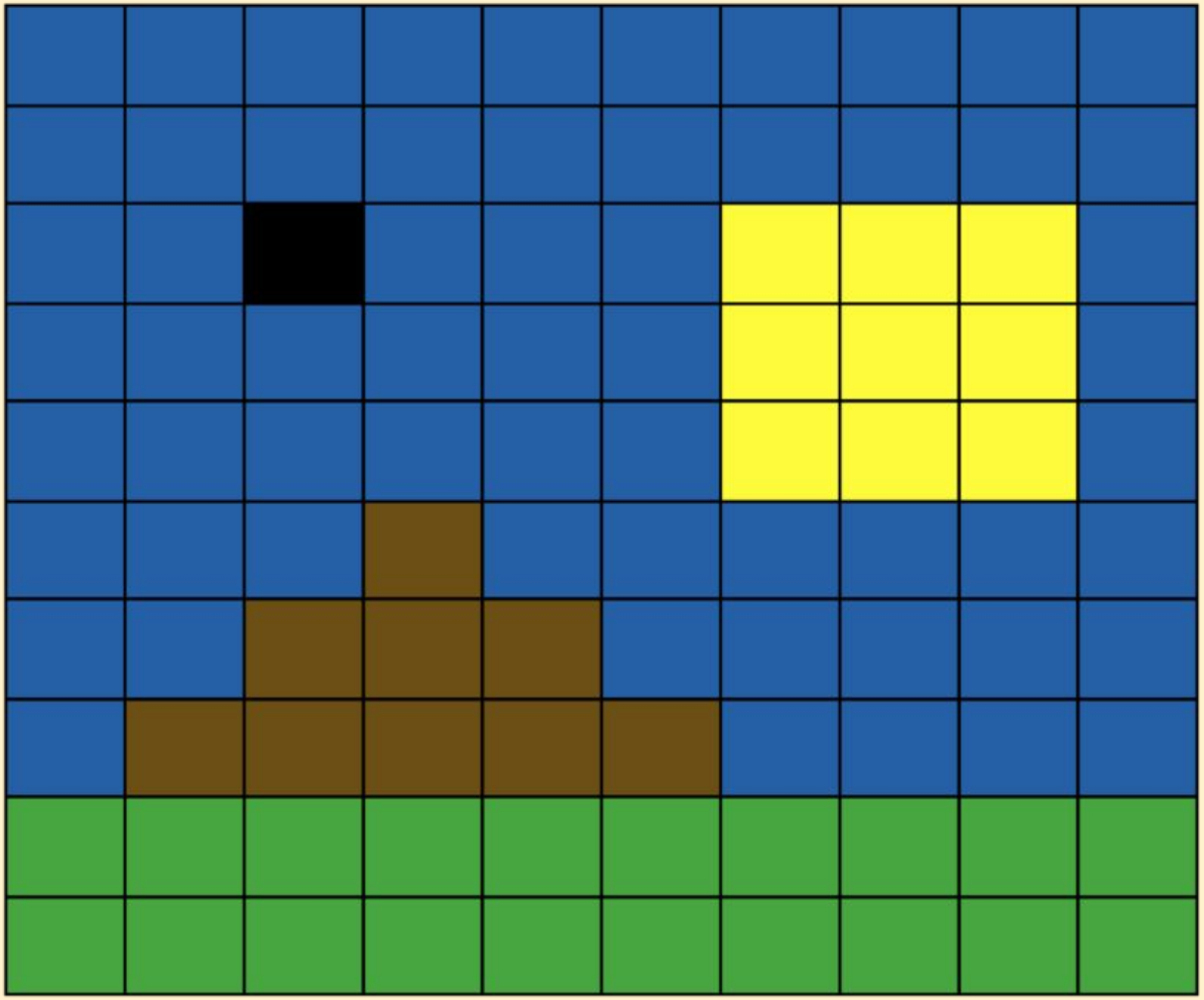
Run Length Encoding



Bitmap 10 pixels wide and 10 pixels high using 3 bit colour depth (5 colours)

Colour depth will actually be much higher than 3 because there are likely to be more colours available

Run Length Encoding



10B

10B

2B 1K 3B 3Y 1B

etc...

Run Length Encoding

RLE **usually** ends up with a smaller file size. But **not always**

If there is **repeated data** (**runs** of the same value) this almost always ends up smaller

If the data has lots of single frequency values, however, the file size won't be smaller:

```
abcdefgh

1a1b1c1d1e1f1g1h
```

Run Length Encoding

But look at the last three runs here:

RRRGGGGGGGGBGGGGRGB

You've got:

RGB

Which becomes:

1R1G1B

Run Length Encoding

If the data has lots of single frequency values the file size won't be smaller - and it can end up larger

```
abcdefgh
```

```
1a1b1c1d1e1f1g1h
```

RLE relies on **repeated runs of data** to reduce the file size