

# Pictionary App: Communicating Pixels



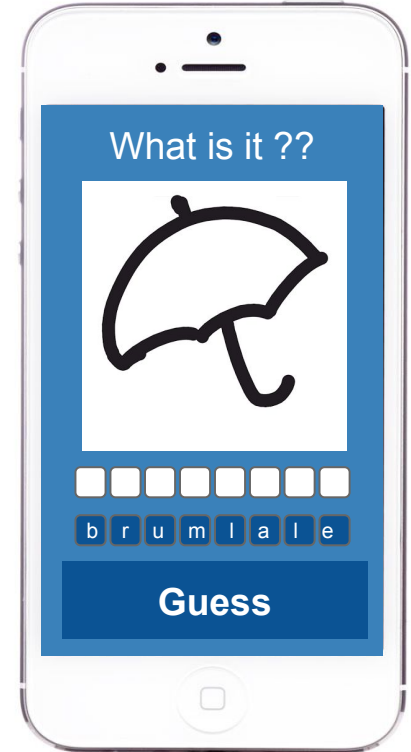
# Pictionary apps!!



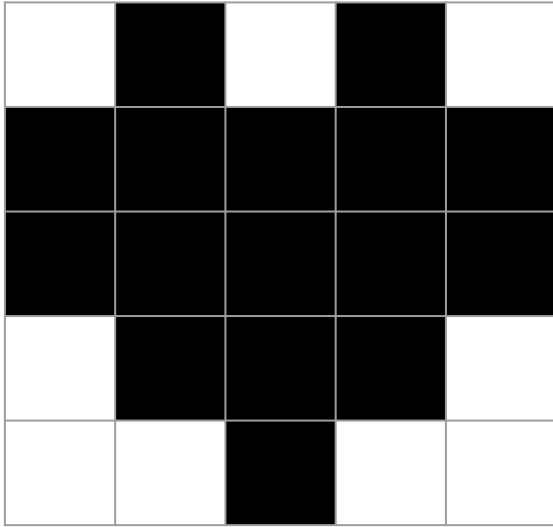
```
100001110111010000110110  
110101000001011111000000  
010010000101111100000000  
011110000010011110100000  
000010001111111011100000  
000001011101001010000101
```



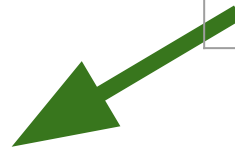
**Data transferred by the  
magic of the internet as  
ones and zeros!**



# Images represented as 1's and 0's



0	1	0	1	0
1	1	1	1	1
1	1	1	1	1
0	1	1	1	0
0	0	1	0	0



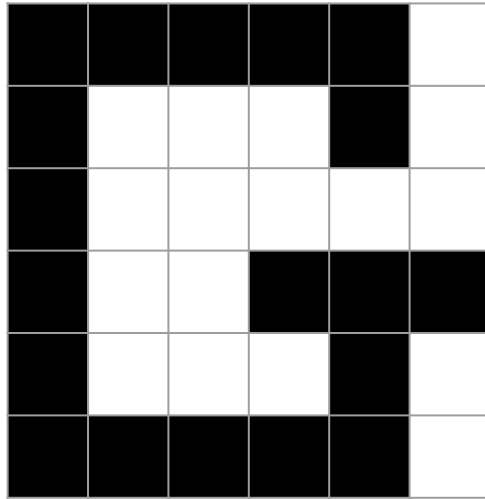
0101011111111110111000100

# Remember the Microbit and its 5X5 LED grid

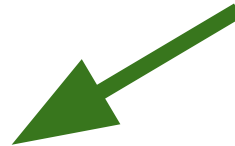
- We will learn how a computer "draws" things on a screen.
- Also how messages can be sent between our microbits
- Image:
- 



# Images represented as 1's and 0's



1	1	1	1	1	0
1	0	0	0	1	0
1	0	0	0	0	0
1	0	0	1	1	1
1	0	0	0	1	0
1	1	1	1	1	0



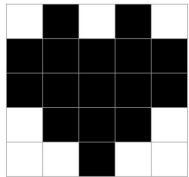
11111010001010000100111100010111110

# Recreate a drawing app without phones!

## TEAM 1

1. Create drawings
2. Turn them into 1's and 0's.

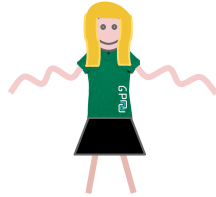
Draw a heart



```
01010111111111
110111000100
```

Find another team.

Send each other messages!!

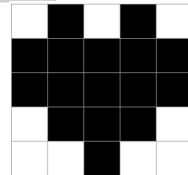


## TEAM 2

1. Transform them back to pictures
2. Guess what they are!

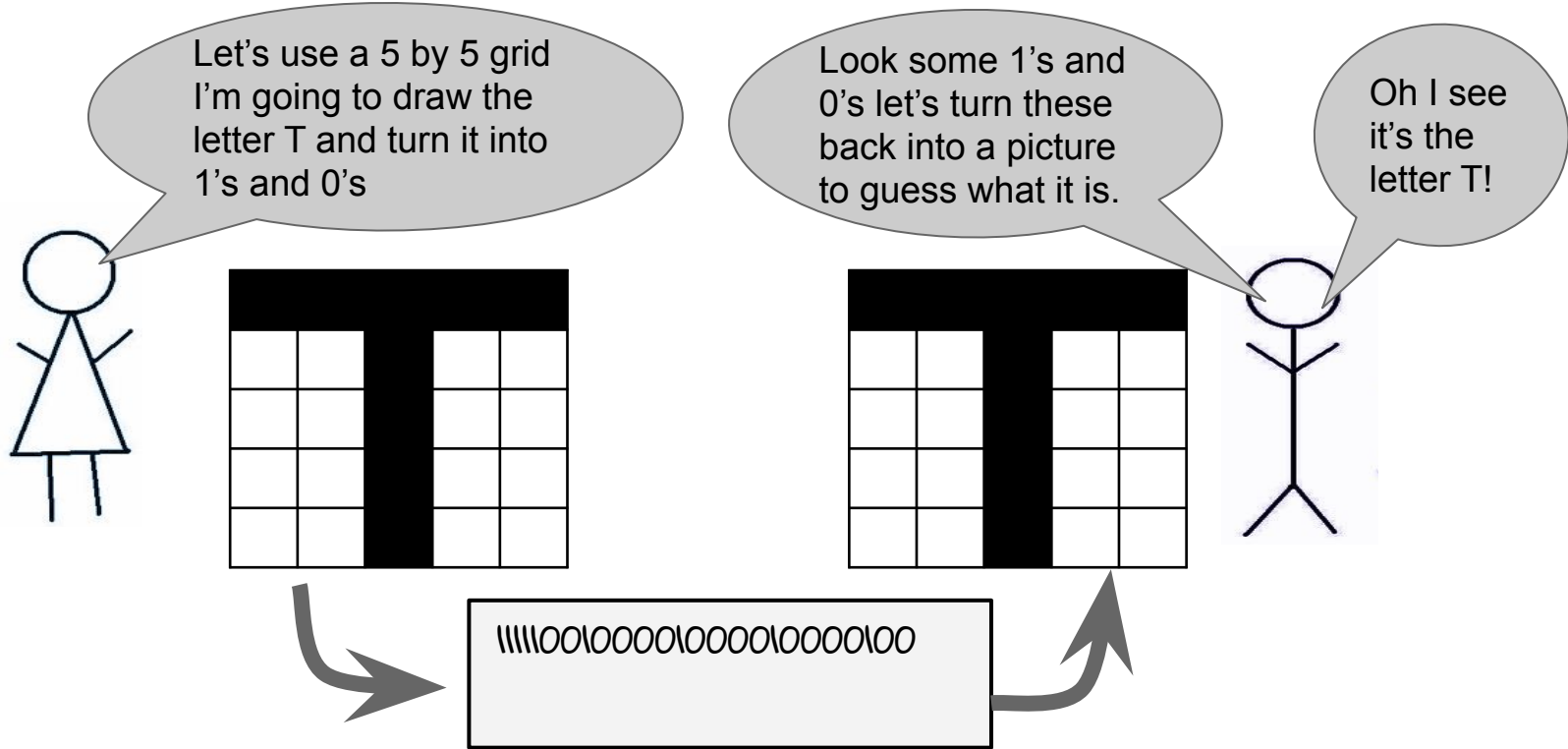
```
01010111111111
110111000100
```

It's a heart!!!

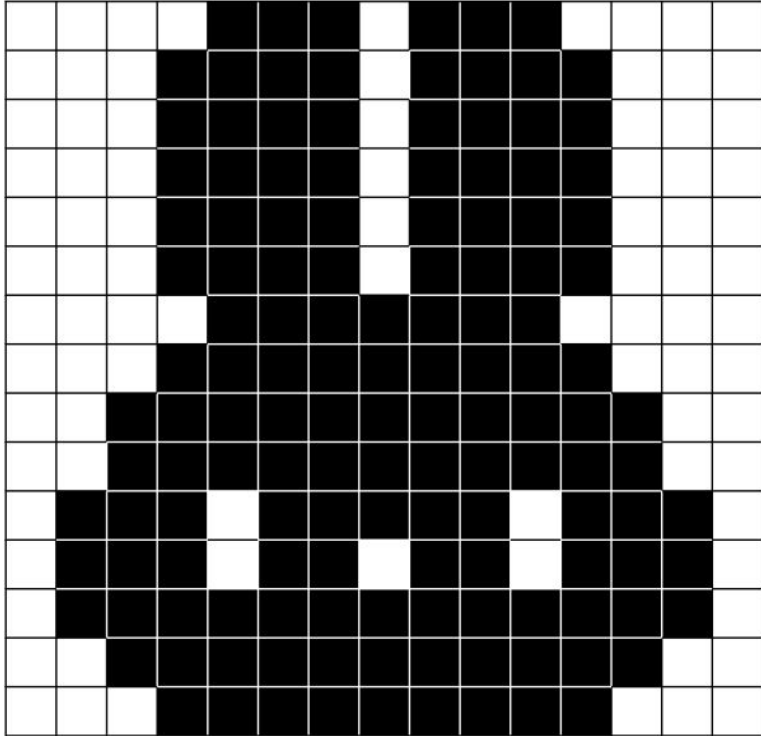


Transfer to the other team using the magic of internet tutors!





# But what if we need more squares?



The string for this bunny is 225 characters long:

```
000011101110000000111101111000000111101
111000000111101111000000111101111000000
111101111000000011111110000000111111111
000001111111111100001111111111100011101
111101110011101101101110011111111111110
0011111111111000001111111111000
```

That's too many to deal with. We're going to make a mistake!



# We can compress our messages!

The bottom row of the bunny looks like this:



We can represent it with 15 bits: 000111111111000

Or we can represent it like this: **031903**

This means put 0 3 times, then put 1 9 times, then put 0 3 times



# But we have a problem!

The second bottom row of the bunny looks like this:



We can represent it like this: **0211102**

This means put 0 2 times, then put 1 11 times, then put 0 2 times

But this is ambiguous and confusing. Does it mean that or does it mean put a 0 2 times, 1 once, 0 once and a 2? Whaaaaaaa?

# Solution! (Well partly)

We know that we are only dealing with 0s and 1s.

Assume that we always start with a 0, just write how long the string of zeroes is and then the next length will be 1s and so on. Let's look at the last example:



Starting with zeroes we have 2 then we have 11 for 1s then we go back to 2 for zeros. So we represent this line with 2112

Problem! This is still ambiguous, is it 2-11-2 or is it 2-1-1-2?

# Full solution!

We're only going to use 1 digit to represent the number of 1s or zeroes ever.

But we have to represent up to the number 15. We're going to cheat! We're going to use letters.

So our numbers are like this

0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---





So this 

--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--

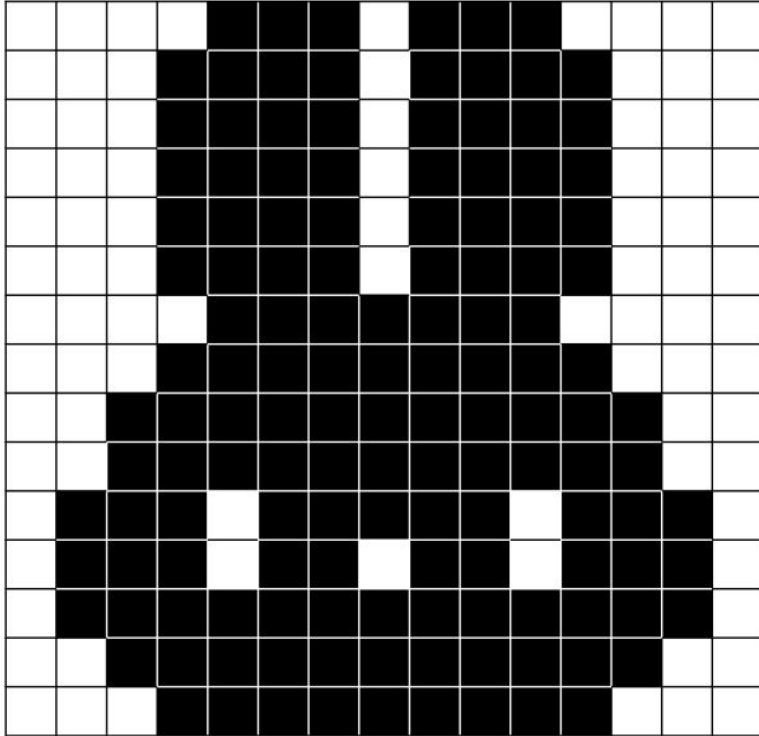
Is represented as 2B2 - which can no longer mean anything else!

# Examples

## Representing different things

<p>A whole white line of length 15</p> 	<p>We always start by representing white so this would just be F</p>
<p>A whole black line of length 15</p> 	<p>We always start by representing white so this would be 0F</p>
<p>A checkerboard starting with white</p> 	<p>111111111111111</p>
<p>A checkerboard starting with black</p> 	<p>011111111111111 - this is the most annoying thing to represent but we're not going to use it if we can help it</p>

# Run length encoding the bunny



43134 - 34143 - 34143 - 34143 - 34143 -  
34143 - 474 - 393 - **2B2** - **2B2** - 131513 -  
13121213 - **1D1** - **2B2** - 393

Isn't that easier than

```
000011101110000000111101111000000111101
111000000111101111000000111101111000000
111101111000000011111110000000111111111
0000011111111111100001111111111100011101
111101110011101101101110011111111111110
0011111111111110000011111111111000
```

# The Grids

- Explain the different grid sizes: 5x5, 10x10, 15x15.
- Show examples of simple, medium, and complex images.
- Explain that larger grids mean more 1s and 0s, which is a great metaphor for larger data files



# The Rules of the Game

- Get a secret word.
- Draw it on the grid.
- Encode it into a binary string.
  - If it's bigger than 5 x 5 convert with **Run Length Encoding**
- Hand the binary string to the other team via a tutor.
- Decode and guess.
- Tutor checks that you've encode, decoded guessed correctly and awards points





# How to win

Here's the points table that will show you how to score yourself (keep track of these points!)

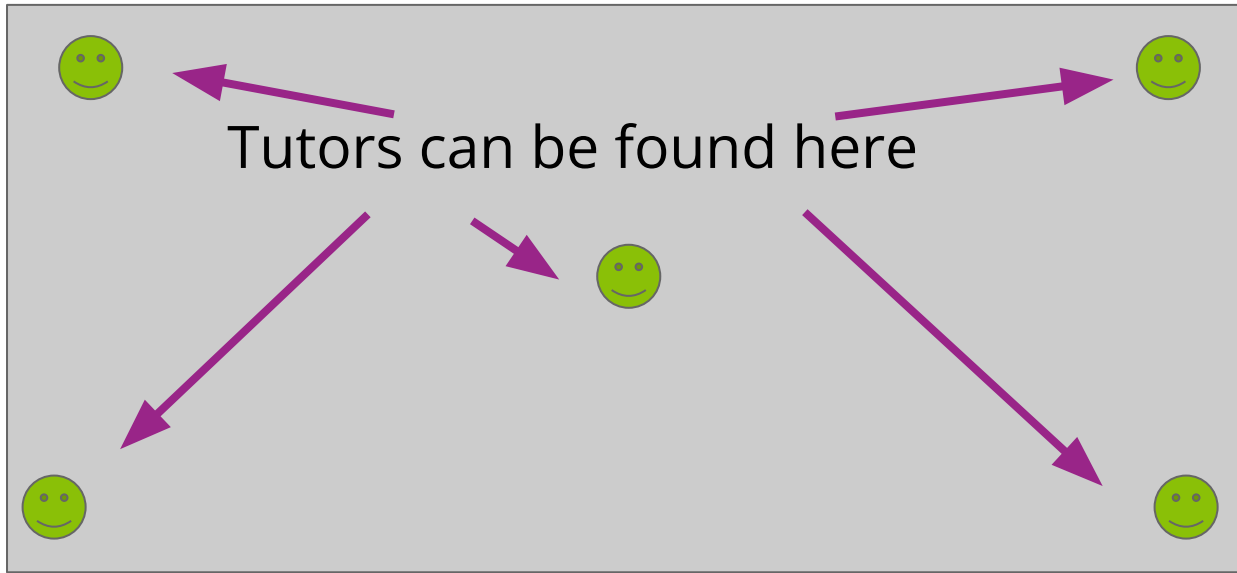
Action	5x5 Grid	10x10 Grid	15x15 Grid
<b>Successful Encoding</b> (correct binary)	5 points	10 points	15 points
<b>Successful Decoding</b> (correctly drawn)	5 points	10 points	15 points
<b>Successful Guess</b> (correctly identified)	10 points	10 points	10 points

Notice!! points are awarded for both **encoding** and **decoding** correctly.



# Let's Play!

Tutor data transfer stations:



# Posters

If you forget any of those instructions posters are around the room to refer to.



# The Rules of the Game

- Get a secret word.
- Draw it on the grid.
- Encode it into a binary string.
  - If it's bigger than 5 x 5 convert with **Run Length Encoding**
- Hand the binary string to the other team via a tutor.
- Decode and guess.
- Tutor checks that you've encode, decoded guessed correctly and awards points
- Teams of 3-4

