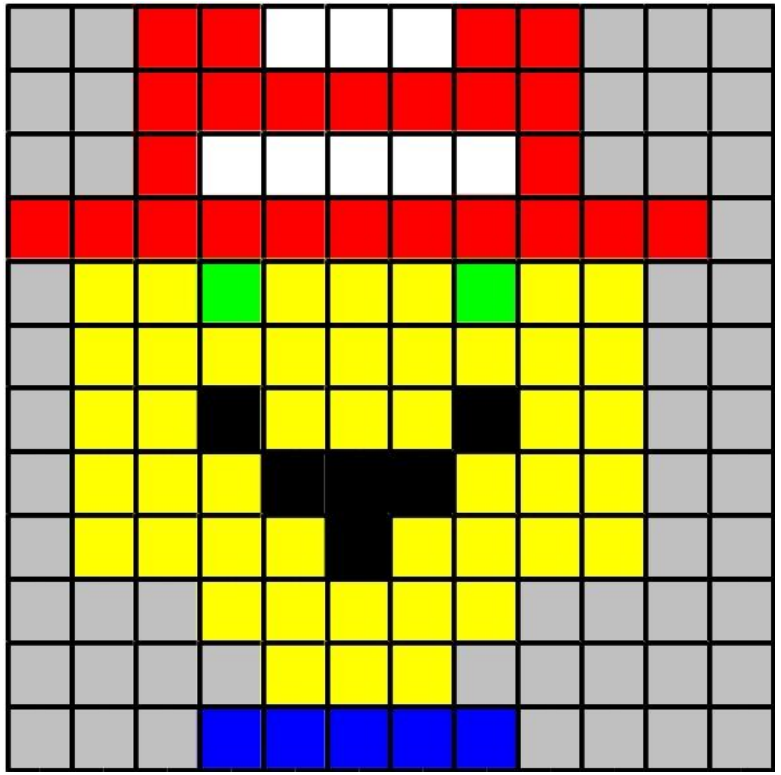
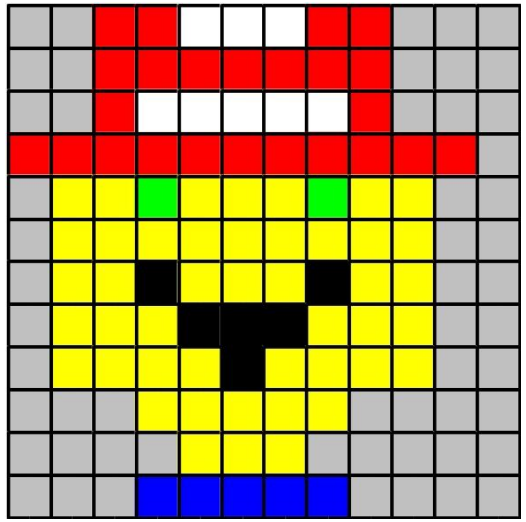
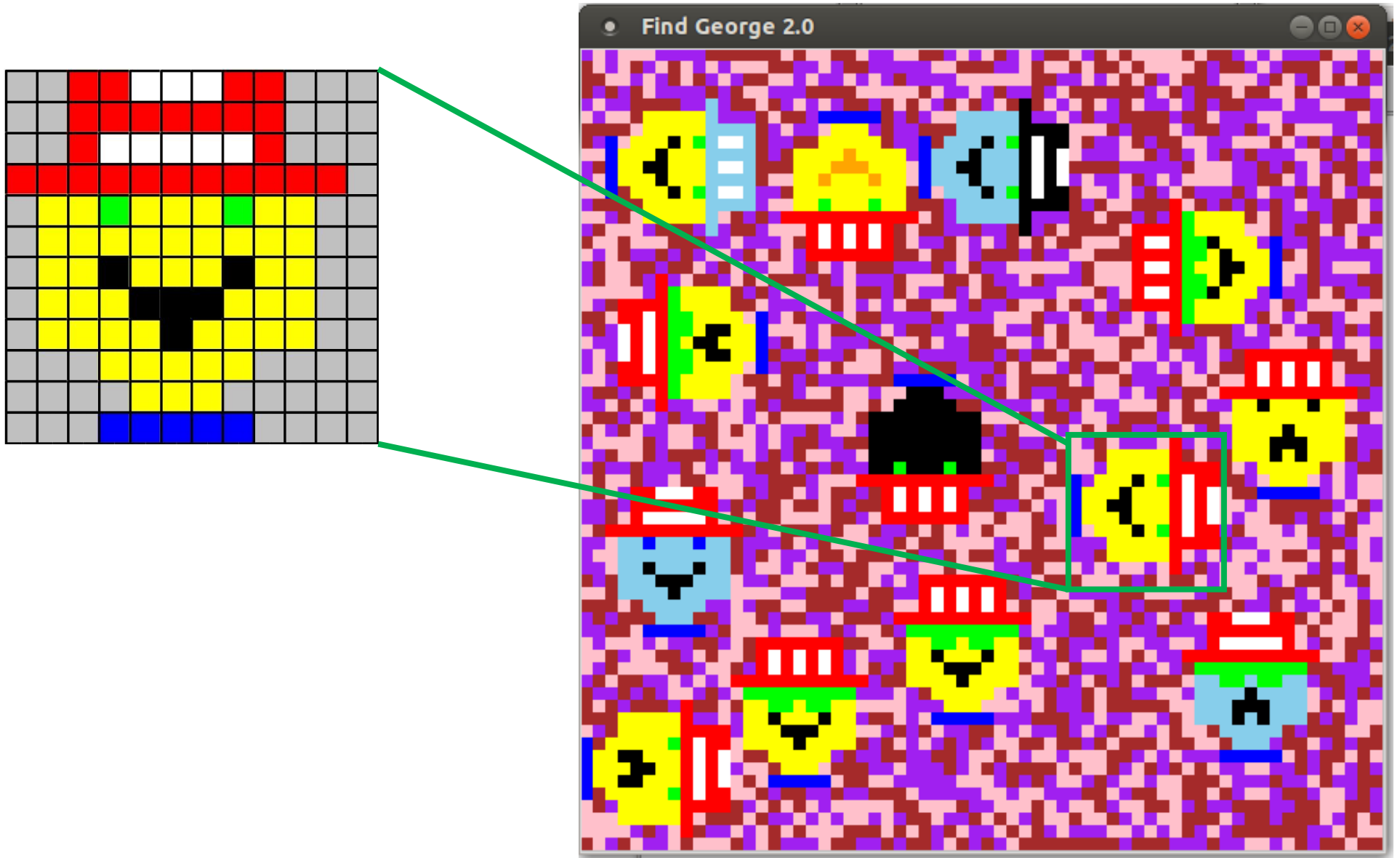


Project 1: Find Tumbling George

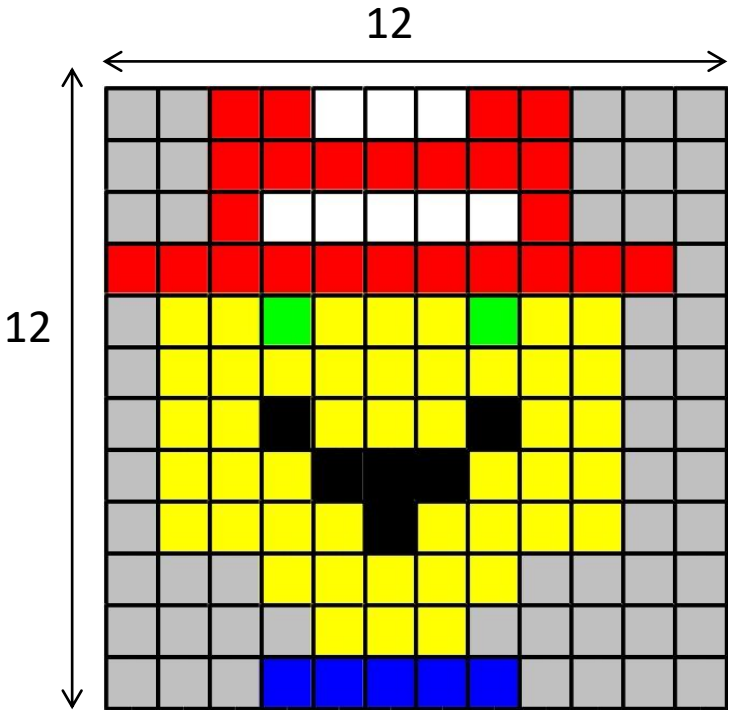
Rotation-Invariant Exact Match Puzzle

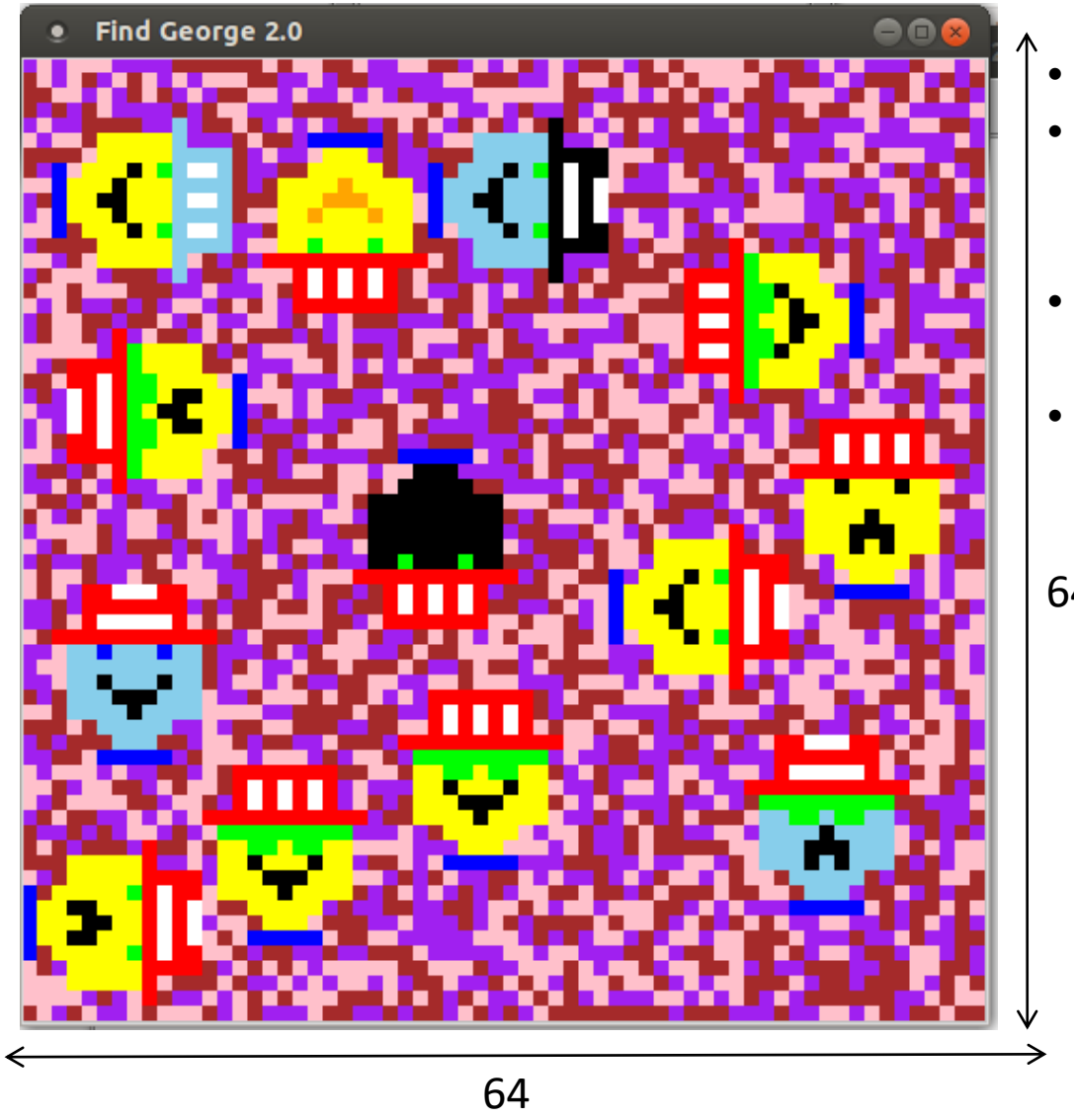


Rotation-Invariant Exact Match Puzzle



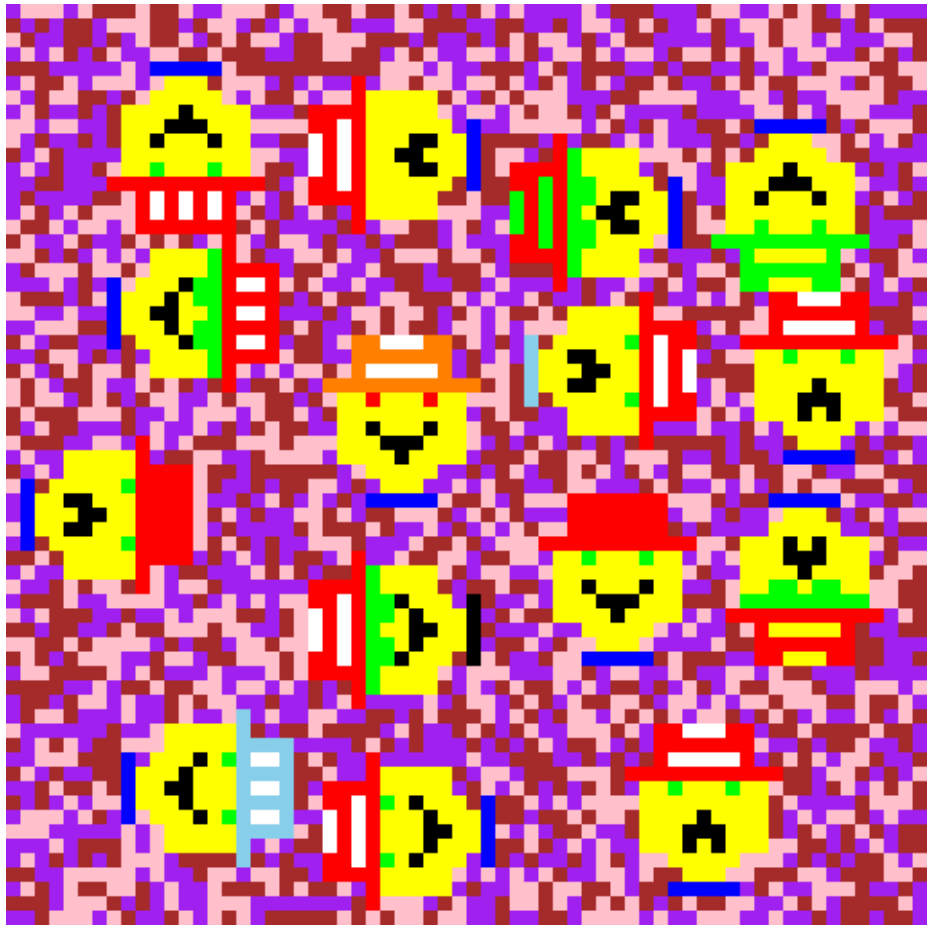
Rotation-Invariant Exact Match Puzzle





- George appears exactly once
- All faces fit within image: no partial faces hanging off edges
- Faces can be upright or rotated 90, 180, 270 degrees
- Not socially distanced! … But no overlap

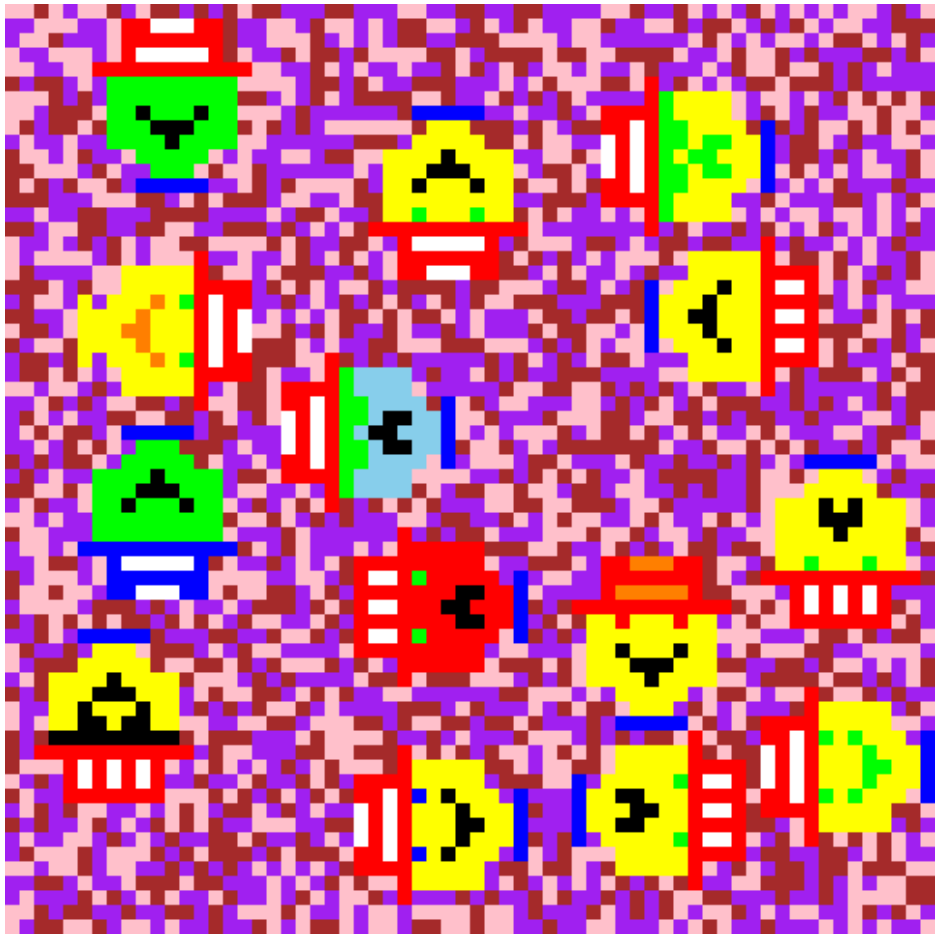
Variations



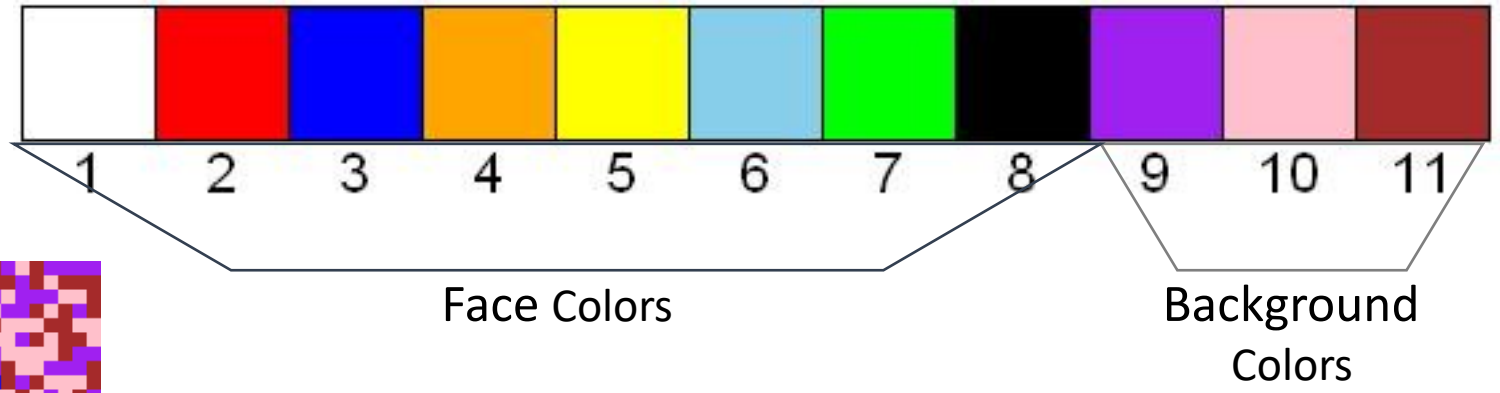
Structural differences:

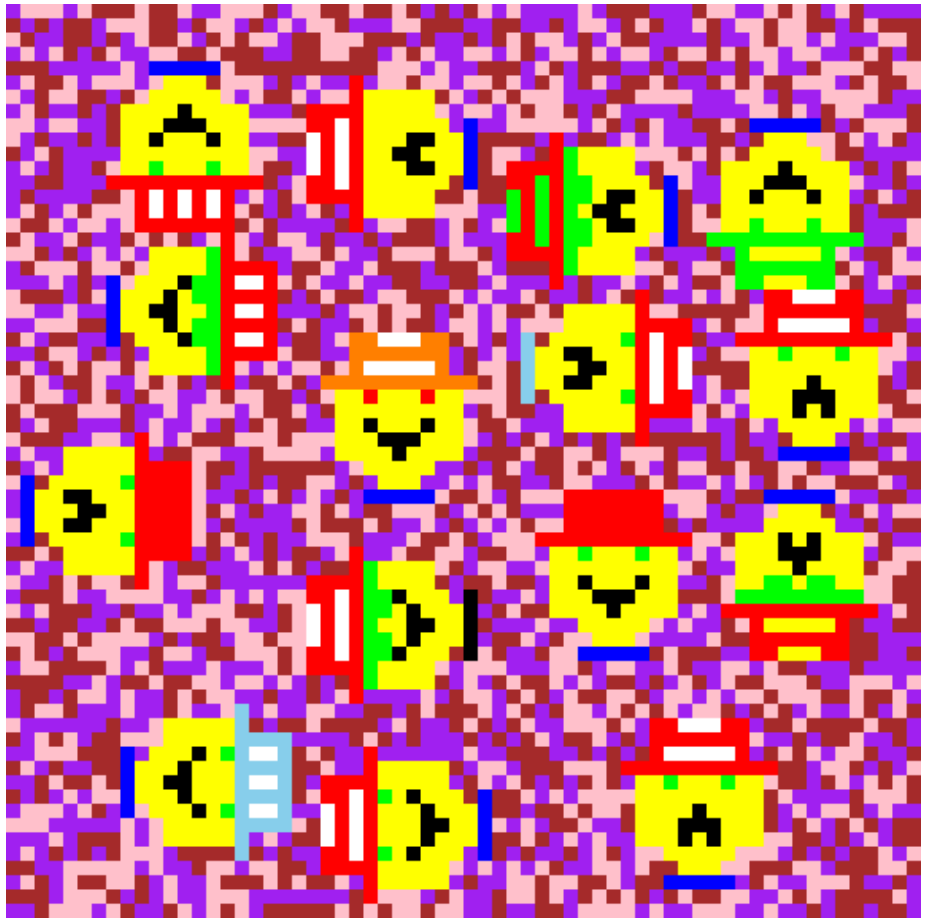
- glasses/not
- stripes
- smile

Color of features



Color Palette





Each pixel value is 1 of 11 colors: can easily fit in 8 bits (byte).

Image Array

0 1 2 3 4 5 6 7 63

64 65 66 67 68 127

“row-major” order

64x64 = 4096 pixels

Pixel 0

Pixel 4095

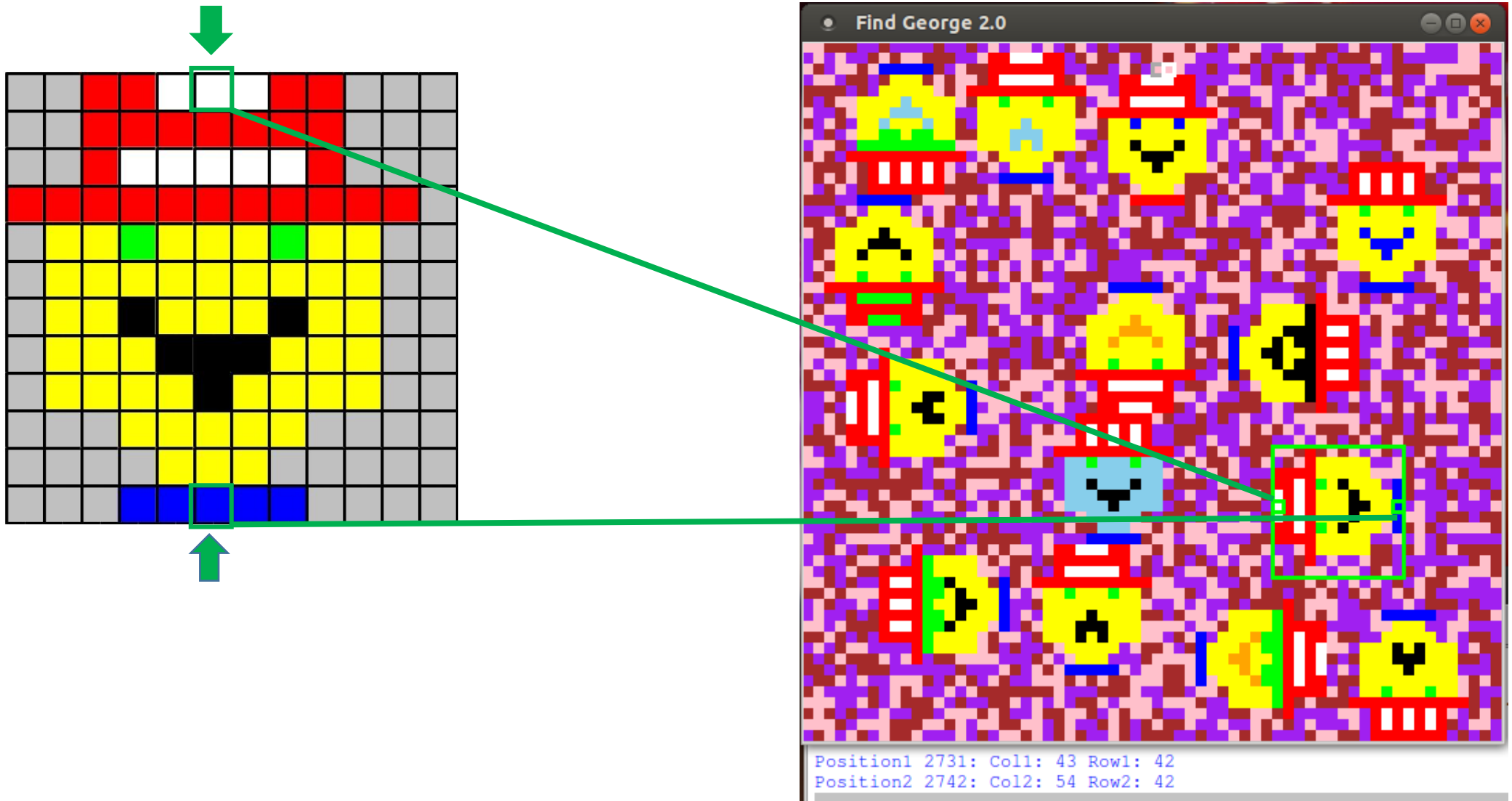
Image Array

Suppose Image base address: 4000

4000:	0x0A
4001:	0x0A
4002:	0x0B
4003:	0x0B
4004:	0x0B
4005:	0x09
4006:	0x09
4007:	0x0B
…	…
8095	0x09

4096 bytes

Goal: Find Byte Offset of Top Middle Pixel of George's Hat and Middle of George's Shirt



P1-1: C Implementation

- byte array:

```
char Crowd[4096]
```

- P1-1-shell.c

```
// your program should use this print statement to report the answer
printf("George is located at: hat pixel %d, shirt pixel %d.\n", HatLoc, ShirtLoc);
```

- Functional version: not evaluated on performance, only accuracy

P1-2: MIPS Implementation

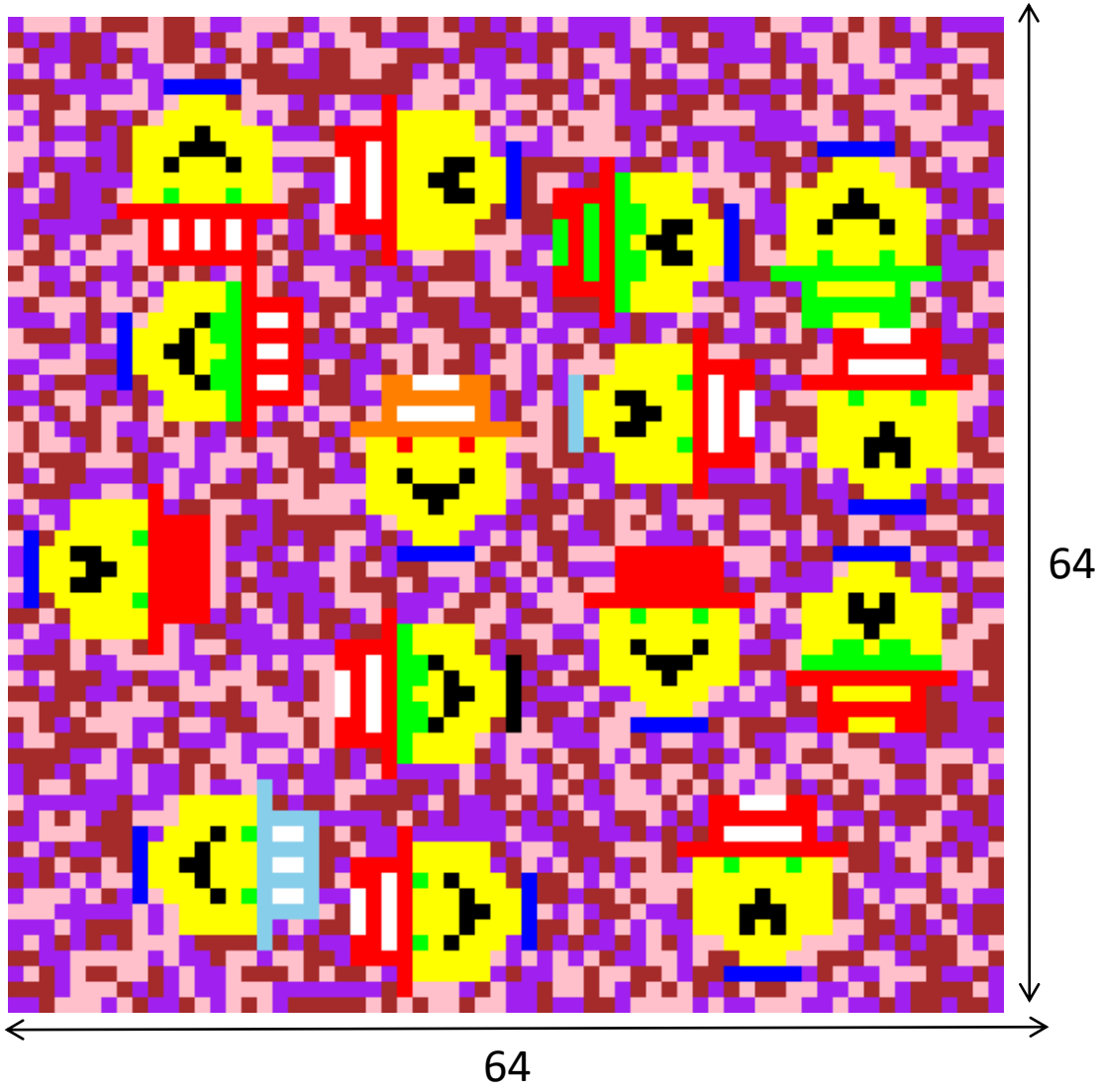
```
.data
Array:  .alloc 1024
.text
...
```

why?

Answer: 64 x 64 bytes = 2^6 x 2^6 = 2^{12} = 4096 bytes

4096bytes / 4bytes/word

= 2^{12} / 2^2 = 2^{10} = 1024 words



P1-2: Software Interrupts

Suppose Image base address: **5828**

5828:	0x09	byte 0
5829:	0x09	byte 1
5830:	0x0B	byte 2
5831:	0x0A	byte 3
5832:	0x0A	
5833:	0x09	
5834:	0x09	
5835:	0x09	
…	…	
8095	0x09	

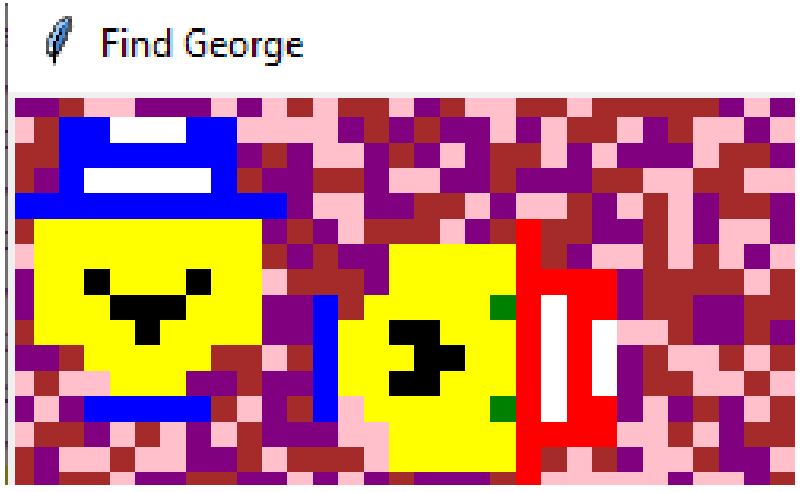
MiSaSiM uses little endian

1. SWI 570: Create Tumbling Crowd

INPUTS: $1 = base address (Image)

OUTPUTS: none - memory populated with image array

```
.data
Image: .alloc 1024
.text
FindGeorge: addi  $1, $0, Image
            swi   570
            . . .
```



byte 3 2 1 0

```
5828: 0XA0B0909
5832: 0X909090A
5836: 0XB0A090A
5840: 0XA0B0B0A
5844: 0XA0A0B09
5848: 0XB0A0B0B
5852: 0XB0B0B0B
5856: 0XA090A09
5860: 0XB090B09
```

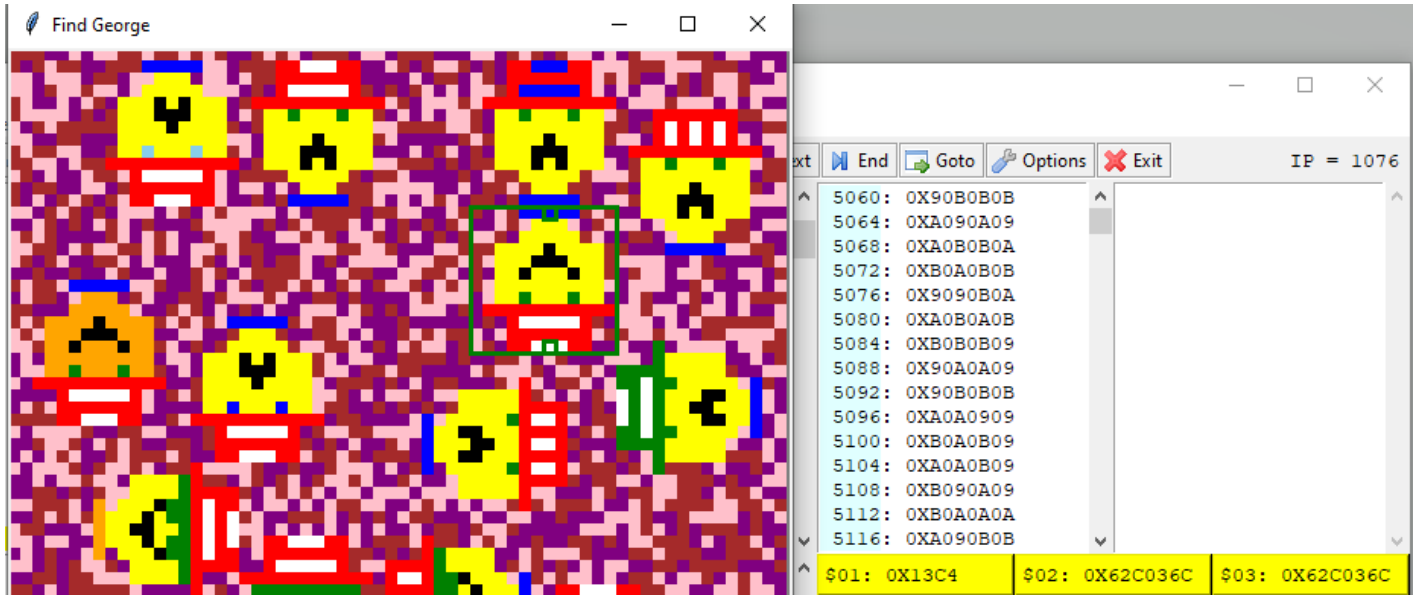
P1-2: Software Interrupts

2. SWI 571: Locate Tumbling George – report middle pixel at the top of George's hat and middle shirt pixel

INPUTS: $2 should contain two packed numbers: in the upper 16 bits, the hat pixel position and in the lower 16 bits, the shirt pixel position. Each pixel position must be a number between 0 and 4095, inclusive.

OUTPUTS: $3 gives the correct answer from oracle

incorrect answer in yellow; oracle's in green



correct answer and oracle's in green

P1-2: Software Interrupts

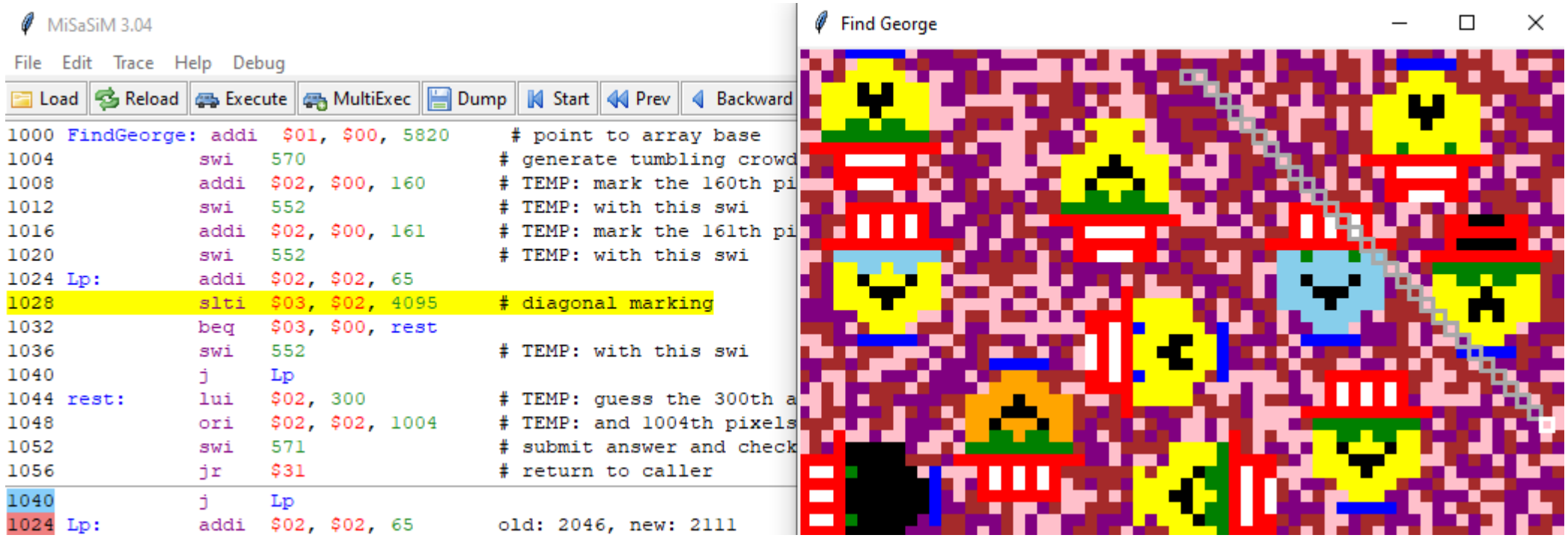
For debugging only

3. SWI 552: Highlight Position

INPUTS: $2 offset into Image; draws a white outline around the cell at that offset.

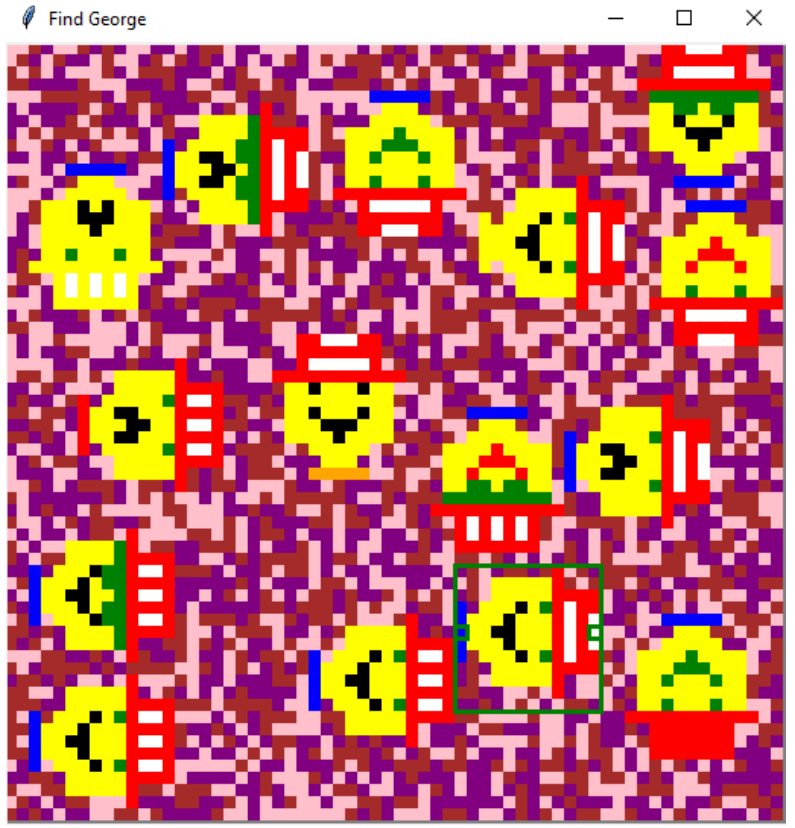
Cells that have been highlighted previously in the trace are drawn with a gray outline.

OUTPUTS: none.



P1-2: Accuracy *and Efficiency* Evaluated

```
static I= 123, dynamic I= 4682, reg data= 16, static data= 1024, stack data= 0
Arith: 41.5% Jump: 0.2% Load: 19.7% Branch: 38.5% Shift: 0.0% Logic: 0.0%
```




```
static I= 123, dynamic I= 8261, reg data= 16, static data= 1024, stack data= 0
Arith: 44.8% Jump: 0.1% Load: 18.8% Branch: 36.2% Shift: 0.0% Logic: 0.0%
```

P1-2: Efficiency Evaluation

- Your Goal: Beat the baseline

  static code size: 123 instructions, dynamic instruction length: 4300 instructions (avg.), total register and memory storage: 16 words (not including dedicated registers $0, $31, or the 1024 words for the input crowd array)

- Run MultiExecute to get average DI, storage over multiple runs

```
static I= 30, dynamic I= 327, reg data= 7, static data= 1024, stack data= 0
Arith: 34.3% Jump: 0.9% Load: 32.4% Branch: 32.4%

static I= 30, dynamic I= 1179, reg data= 7, static data= 1024, stack data= 0
Arith: 33.6% Jump: 0.3% Load: 33.1% Branch: 33.1% ic I= 30, dynamic I= 327, reg
```

. . .

We'll run 100 trials (same trials for all programs).

Score

	Part	Description	Points
Accuracy Metrics	P1-1	Find George (C code)	25
	P1-2	Find George (MIPS code)	
		correct operation, proper commenting/style	25
Performance Metrics		static code size	15
		dynamic execution length	25
		storage requirements (registers, memory)	10
		Total	100

For each Performance Metric:

Points = PercentCredit x (Performance Metric Points)
where

$$PercentCredit = 2 - \frac{Metric_{Your\ Program}}{Metric_{Baseline\ Program}}$$

Baseline Metrics:
static code size: 123 instructions,
dynamic inst length: 4300 instructions (avg.),
total register & memory storage: 16 words

E.g., if your program has a **3010** avg dynamic execution length,

Points for dynamic: (2-3010/4300) x 25 = (2-0.7)x25 = 1.3 x 25 = 32.5 (out of 25)

If your program uses 24 words, storage points: (2-24/16) x 10 = (2-1.5)x10 = 5 (out of 10)

P1-1: C Implementation

- byte array:

```
char Crowd[4096]
```

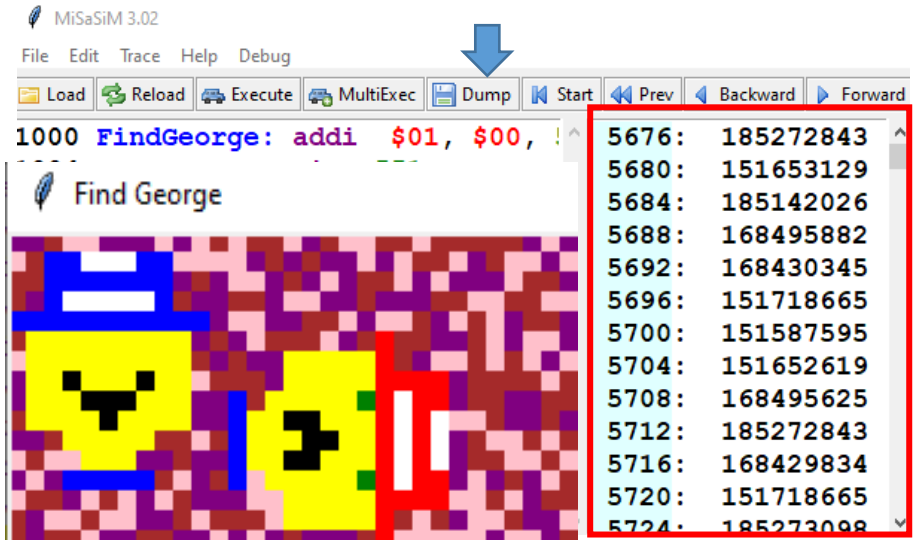
- P1-1-shell.c

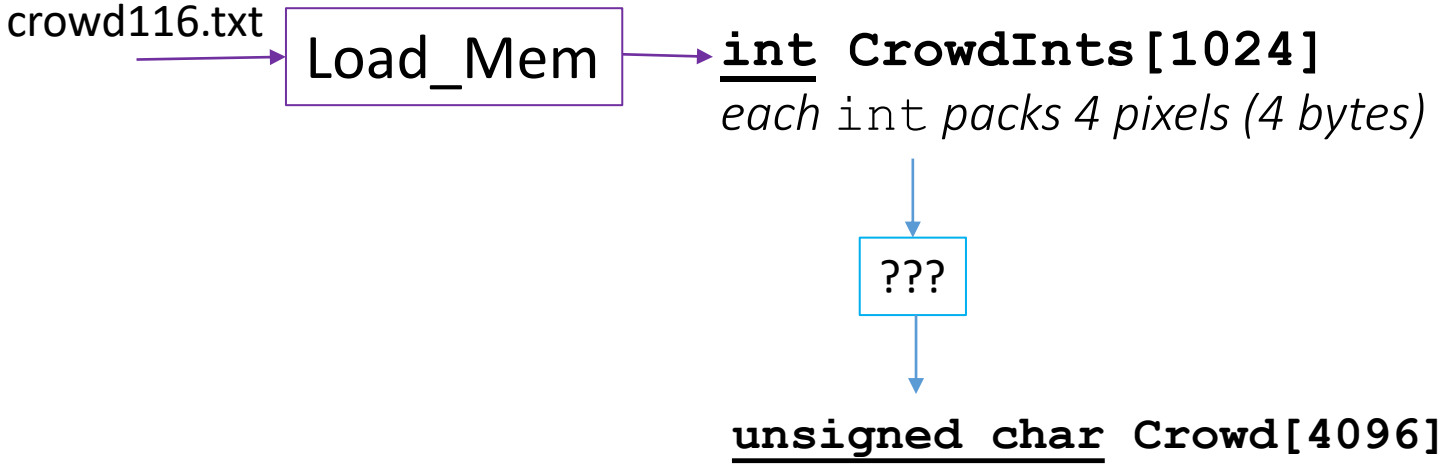
```
// your program should use this print statement to report the answer
printf("George is located at: hat pixel %d, shirt pixel %d.\n", HatLoc, ShirtLoc);
```

- Functional version: not evaluated on performance, only accuracy

P1-1 How to turn `int` array to byte array access

- Test cases for P1-1 can be generated using MiSaSiM Dump command
- P1-1-shell.c contains Load_Mem function





We'd like to treat the test data as a byte array `Crowd`.

P1-1-shell.c uses a “type cast” to do this

Data:

```
5676:   185272843
5680:   151653129
5684:   185142026
5688:   168495882
5692:   168430345
5696:   151718665
5700:   151587595
5704:   151652610
5708:   16849
5712:   18527
5716:   16842
5720:   15171
5724:   18527
```

```
int CrowdInts[1024]; // value of identifier CrowdInts is
                     // base address (5676) of array of ints
```

```
char *Crowd = (char *)CrowdInts;
```

nts

The compiler nc

Crowd

differently from

Crowd

Time for quick intro to type casting…

Quick Intro to Type Cast

```
int x = 3;
printf("int: %d, float: %f\n", x, x);
```

⇒ ERROR

```
int x = 3;
printf("int: %d, float: %f\n", x, (float) x);
```

P1-1-shell.c uses a “type cast” to do this

Data:

```
5676:   185272843
5680:   151653129
5684:   185142026
5688:   168495882
5692:   168430345
5696:   151718665
5700:   151587595
5704:   151652619
5708:   168495625
5712:   185272843
5716:   168429834
5720:   151718665
5724:   185273098
```

```
int CrowdInts[1024]; // value of identifier CrowdInts is
                     // base address (5676) of array of ints
```

```
char *Crowd = (char *)CrowdInts;
```

This tells the compiler to treat the value `(5676)` of `CrowdInts` as a base address of an *array of chars* and let `Crowd` refer to that base address.

The compiler now knows how to translate:

`CrowdInts[i]  -- using lw/sw`

differently from

`Crowd[i] -- using lbu/sb`

```
#define DEBUG 1 // RESET THIS TO 0 BEFORE SUBMI

int main(int argc, char *argv[]) {
    int                 CrowdInts[1024];
    // This allows you to access the pixels (inc
    // as byte array accesses (e.g., Crowd[25] c
    char *Crowd = (char *)CrowdInts;
    ...
    NumInts = Load_Mem(argv[1], CrowdInts);
    if (DEBUG){
      printf("Crowd[0] is Pixel 0: 0x%02x\n", Crowd[0]);
      printf("Crowd[107] is Pixel 107: 0x%02x\n", Crowd[107]);

      printf("CrowdInts[211] packs 4 Pixels: 0x%08x\n", CrowdInts[211]);
      printf("Crowd[211*4] is Pixel 844: 0x%02x\n", Crowd[844]);
      printf("Crowd[211*4+1] is Pixel 845: 0x%02x\n", Crowd[845]);
      printf("Crowd[211*4+2] is Pixel 846: 0x%02x\n", Crowd[846]);
      printf("Crowd[211*4+3] is Pixel 847: 0x%02x\n", Crowd[847]);
    }
    ...
```

```
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents
se-to-students$ gcc -g -Wall P1-1-shell.c -o demo
linda@Sassafras:/mnt/c/Users/Linda Wills/Documents
se-to-students$ ./demo tests/crowd-1439-2143.txt
Crowd[0] is Pixel 0: 0x0a
Crowd[107] is Pixel 107: 0x09
CrowdInts[211] packs 4 Pixels: 0x090a0b09
Crowd[211*4] is Pixel 844: 0x09
Crowd[211*4+1] is Pixel 845: 0x0b
Crowd[211*4+2] is Pixel 846: 0x0a
Crowd[211*4+3] is Pixel 847: 0x09
George is located at: hat pixel 0, shirt pixel 0.
```

```
addi $2, $0, 844
lw   $6, CrowdInts($2)

lb   $7, Crowd($2)
# CrowdInts and Crowd
# have same value
# (base address)
```

Project 1: Submit 3 parts

- P1-1.c – functional version in C
- P1-2-int.asm – intermediate version in MIPS, includes change log
  - not graded for accuracy/efficiency
  - must have substantial changes to P1-2-shell.asm
  - not graded, but 10 points off Project 1 if missing

Example

```
#=============================================================
# CHANGE LOG: brief description of changes made from P1-2-shell.asm
# to this version of code.
# Date  Modification
# 02/12 Looping through pixels to find one w/ color $3
# 02/15 Reduced avg DI by only looking at pixels starting at row 10
#=============================================================
```

- P1-2.asm – final version in MIPS, includes change log
  - graded for accuracy and efficiency

Honor Code

In all programming assignments, you should design, implement, and test your own code. Any submitted assignment containing non-shell code that is not fully created and debugged by the student constitutes academic misconduct. You should not share code, debug code, or discuss its performance with anyone. *Once you begin implementing your solution, you must work alone.*