**Equivalence writeup**

(1,0,1) and (2,0,2) are distinct points in R2 but are equal in a projective finite field Zm with m = 3 by the third property. Letting p = (1,0,1) and q = (2,0,2), the cross product is computed as

By substituting in each coordinate for the given formula.

(1,0,0) and (0,1,0) are distinct points in both R2 and the projective finite field Zm with m = 3 by the third property. Letting p = (1,0,0) and q = (0,1,0), the cross product is computed as

To double check and see if p and q are on the line, we need to calculate the multiplication of each coordinate in [0,0,1] with each coordinate in (1,0,0) and (0,1,0). If the result is 0 for one of the points, then the point is on the line.

Each equation is true, thus each point is on the line.

**Generating all points recipe**

Recipe name: generate\_all\_points

Inputs:

* mod, the modulus equal to m for Zm, the finite field in which the projective geometry is taking place

Outputs:

* points\_list, a sequence of all the points generated by the function

Steps:

1. Create points\_list as an empty sequence.
2. Create a Boolean variable is\_in\_list that will represent whether or not a newly generated point is already in points\_list. Initialize it to False.
3. For each x\_coord in 0, 1, 2,…, m-1, do
   1. For each y\_coord in 0, 1, 2,…, m-1, do
      1. For each z\_coord in 0, 1, 2,…, m-1, do
         1. Check to see if the new point would be (0,0,0). If it is, then set the new point to (0,0,1). Since the next iteration would be this anyways and thus will be removed, this skips (0,0,0) without using bad programming techniques like using break or too much nested code.
         2. Create a tuple new\_point that is assigned the value (x\_coord, y\_coord, z\_coord)
         3. Set is\_in\_list to false.
         4. For each point in points\_list, do
            1. Check to see if new\_point is equal to point using the function equivalent with the arguments (new\_point, point, mod)
            2. If it is, set is\_in\_list equal to True
         5. If is\_in\_list is false, then create a new sequence to replace points\_list that is essentially points\_list but with new\_point at the end of the sequence.
4. Return points\_list

Output:

* points\_list, a sequence of all the points generated by the function

**Create cards recipe**

Recipe name: create\_cards

Inputs:

* points, a sequence of unique points, each represented by a tuple of 3 integers
* lines, a sequence of unique lines, each represented by a tuple of 3 integers
* mod, an integer representing the prime modulus

Outputs:

* cards\_list, a sequence of sequences where each subsequence represents a card in the game

Steps:

1. Create a new sequence, cards\_list, that will store every generated card
2. for each line in lines, do
   1. initialize a new sequence to an empty sequence called card
   2. for each point\_idx in the indices of points, do
      1. Use the function incident with the arguments (points[point\_idx] which is the element of points that is at the index point\_idx, line, mod). If it returns true, then
         1. Add the point\_idx to card
   3. Append card to the end of cards\_list
3. Return cards\_list

Outputs:

* cards\_list, a sequence of sequences where each subsequence represents a card in the game

**Discussion**

It is not possible to create a valid deck of 40 “Spot it!” cards because there is no space Zm where m is prime (and thus a valid projective geometrical space) that would produce 40 spot it cards. When m = 5, 31 cards are generated. When m = 7, 57 cards are generated. Since there’s no prime number between 5 and 7, and the number of generated cards is proportional to the number of spot it cards, 40 isn’t possible.