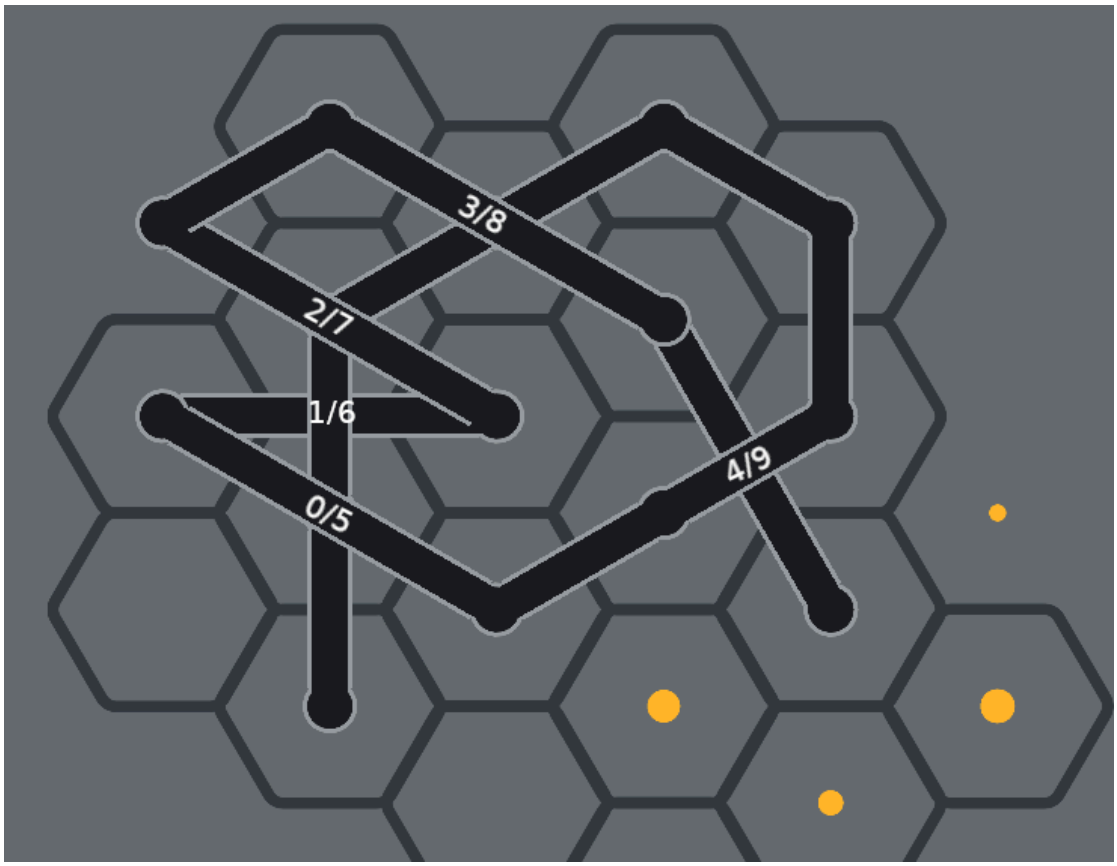
Miller Universal Notation for Computerized Knots: M.U.N.C.K.

Munck is a system for quantifying knot topologies in a way that could be recognized for a computer game. It involves storing player knots as specific munck strings and comparing them to a Platonic Munck Identifier.

Munck strings are formatted in blocks that represent places where the rope crosses over itself or other ropes. Each block has four pieces of data:

1. c_over: This stores whether the rope goes over or under the rope it crosses.
2. c_location: This stores the location (along the rope) where the intersection happened, useful for keeping intersections in order.
3. c_block: This points to the Munck block where the counterpart to this crossing is stored.
4. move: This is the move # that this crossing was added, again used for sorting.

Example knot: Numbers indicate the first and second time the intersection is used.



The Platonic Munck Identifier, P.M.I., is a simplified form of the full knot. The basic idea is to create the smallest list of correct forms of a knot without any false positives or negatives. PMIs are arrays of tuples containing two elements:

1. is_over: This states whether the rope is over or under its counterpart.
2. counterpart_index: This is the index of this crossing's counterpart.

As an example, a twist, which looks like this:

Could be represented like this:
[[true, 1], [false, 0]]

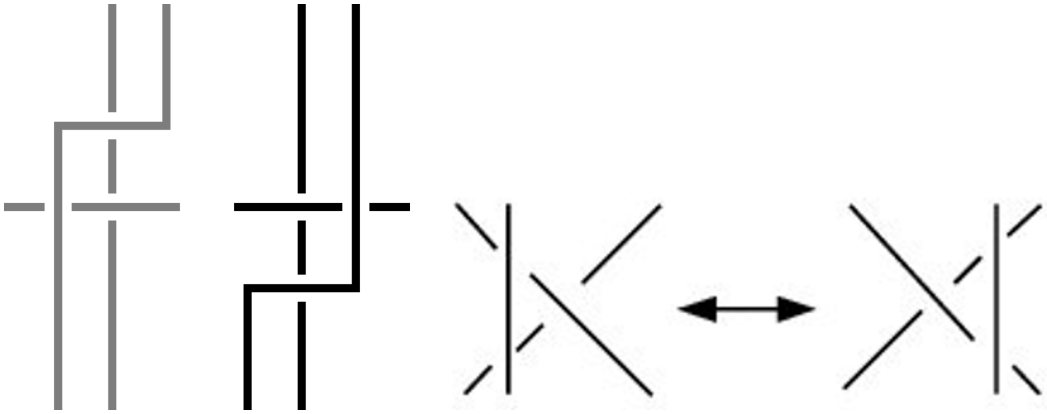
Now we have to do several more steps to reduce the ambiguity even more.

First we will remove redundant entries. Since the second element of this array is redundant, we can simply remove elements from the PMI as we find their earlier counterpart. The new array would look like this: [[true, 1]].

Then we can remove parity. Since knots are the same when flipped over, we can just assume the first element is always false, and then propagate this across the array. Our array now looks like [[false, 1]].

The rope could also be used to tie the knot from the opposite end. To account for this the checker can create a PMI for both the forward and backwards versions of a knot.

Lastly, there are a couple of times when the ordering of crossings is ambiguous. This occurs when a crossing has a loose strand passing under or over both crossing ropes. This is a clear example of a Reidemeister move (III), as shown below:



To account for this variations will have to be written out for each time these show up in a knot. Any time three ropes cross in a space they create a reidemeister (III) move, so when turning a knot into a PMI just pick an arbitrary arrangement. In my system I printed all PMIs to console so I could run through each reidemeister variation and quickly grab the representations.

Other Reidemeister moves could be accounted for as well (I and II specifically) but since some knots, such as slip knots, require them I think it is ok to leave them out of the equation.

Example database of PMIs

1 Rope PMIs:

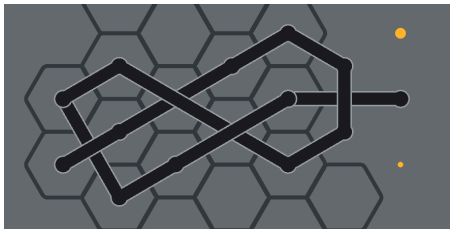
- 1 cross

  - Twist (Reidemeister (I))
    - [[false, 1]]
- 2 cross

  - Bite/Poke (Reidemeister (II))
    - [[false,2],[false,3]]
    - [[false,3],[false,2]]
- 3 cross

  - Overhand
    - [[false,3],[true,4],[false,5]]

- 4 cross



  - Figure 8
    - [[false,5],[true,4],[false,7],[true,6]]
- 5 cross

  - Slip Knot
    - [[false,5],[true,6],[false,7],[false,8],[true,9]]
    - [[false,7],[true,6],[true,5],[false,8],[true,9]]
- 7 cross

  - Bowline (this knot has a Reidemeister (III) move, often called a “slide”, so it has several alternative notations)
    - [[false,10],[true,6],[true,12],[false,13],[false,8],[true,9],[true,11]]
    - [[false,10],[true,11],[true,7],[false,13],[false,8],[true,9],[true,12]]