Names: Jung Woo (Noel) Park, Joseph Mcmanamon, Luke Falvey, Jayde Medder.
Student ID: 1162424, 6021556, 4497820, 7305118.

Pentominoes Data Structure:

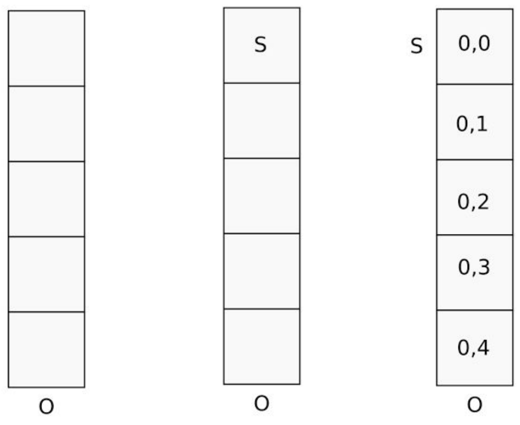
Overview:
We use the [O-Z] Conway's encoding of pentominoes for human simplicity reasons when implementing (we can always imagine the shape based on the alphabet). Our pentominoes will be represented as array of *Points.* Because there are 5 squares in a pentomino, for every pentominoes class there will be 5 *Points.* Each point will have an initial starting point of the pentominoes and 4 offset points that indicate where to draw the sequential squares to form a pentomino. We standardize the starting position to be the most top left square of the pentomino. Each pentomino will also have an enumerated type which indicates the type of pentomino we are dealing with.

- **Point Class**: represents an arbitrary point. This class encapsulates both X and Y into one instance. The X and Y will be used to hold the composition data of the pentominoes.
- **Pentomino Class**: represents a pentomino instance. This class is the data structure that holds the list of 5 points (index 0: starting point, index 1-4: offsets). The class also contains static list of all enumerated types which help label each pentominoes instance. The class has a type field the indicates one of the enumerated types labelled [O-Z].

Example:

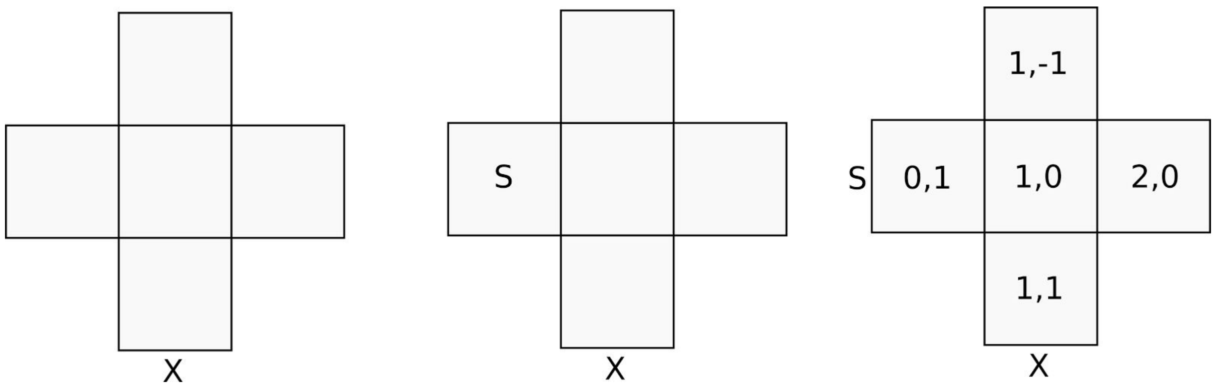
The diagram belows shows how we represent the O pentomino in our data structure implementation.

1) We first label the starting position (most top left square).
2) We then set up a starting position starting at 0,0.
3) We then find all the offsets of the rest of the squares that make up the pentomino.



Names: Jung Woo (Noel) Park, Joseph Mcmanamon, Luke Falvey, Jayde Medder.
Student ID: 1162424, 6021556, 4497820, 7305118.

Here is an example of our representation for a more complex shape. Notice this doesn't start at 0,0 because there are no square at 0,0 location:



Note: We always start at the most top left corner, if there are no squares in a corner (0,0), then we do the first most left hand side square.

Advantage:

- Our representation of the pentominoes is advantageous because with the initial starting point and the offsets, it is able to be drawn anyway on a rectangular board. For example if we swapped the starting position (indexed: 0), then by adding the offsets to this starting position, we still have the same shape of the pentominoes which differs only by its starting location.
- This representation also gives a free validity check. If the board was represented by a two dimensional array and because our positions are always working in the same positive integer space as the indices, any negative indices when computed will be invalid. So we can easily know if we can fit a pentomino starting a position on a board.
- Because the pentominoes are represented in a very abstract way, we are able to use this representation in most board representation.
- All pentomino types, starting points and offsets will be precomputed into a pentomino data file. With this data file we won't have to do unnecessary computations for rotating or reflecting pentominoes while searching our puzzle.

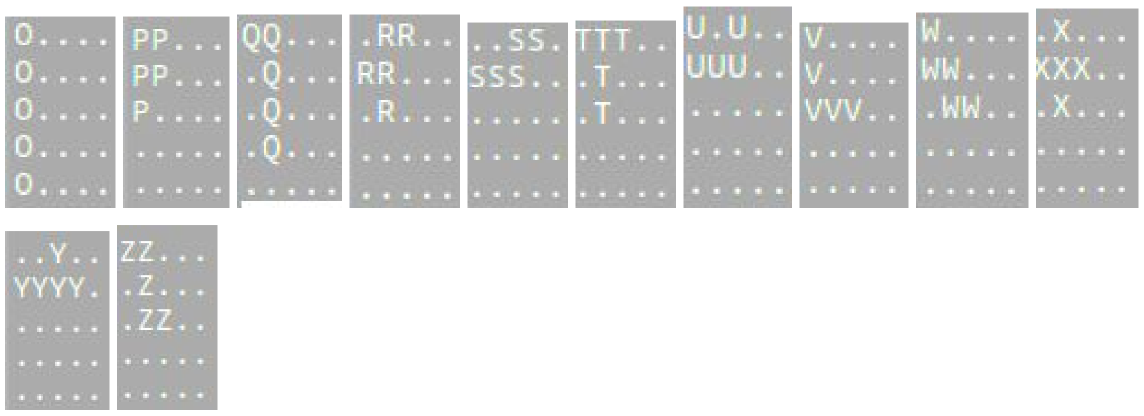
Disadvantage:

- A disadvantage of our representation is the laborious task of finding all offsets of the 63 pentominoes (including rotation and reflection). Although it is possible it's very time consuming.
- Because we always find the starting position based on a 5 x 5 matrix, we potential could run into problems when trying to use our implementation on

diagonal boards. Best way to overcome this could be to tilt the board so it's rectangular.

- Because we have a list of pentominoes each with a list of points (representing the shape), we could run into lots of O(n^2) operations for iterating all data.

Example Stringified Output:



```
O....  PP...  QQ...  .RR..  ..SS.  TTT..  U.U..  V....  W....  .X...
O....  PP...  .Q...  RR...  SSS..  .T...  UUU..  V....  WW...  XXX..
O....  P....  .Q...  .R...  .....  .T...  .....  VVV..  .WW..  .X...
O....  .....  .Q...  .....  .....  .....  .....  .....  .....  .....
O....  .....  .....  .....  .....  .....  .....  .....  .....  .....

..Y..  ZZ...
YYYY.  .Z...
.....  .ZZ..
.....  .....
.....  .....
```