Q1)

The function “teleport” (Perl on the left, Python on the right)

```perl
sub teleport{
    my ($self, $maze) = @_;
    my $row_rand = int(rand $maze->getHeight());
    my $col_rand = int(rand $maze->getWidth());
    my $row_cur = $self->getPos()->getR();
    my $col_cur = $self->getPos()->getC();

    my $row_dif = $row_rand - $row_cur;
    my $col_dif = $col_rand - $col_cur;

    local @rshift = (0, 0, 0, 0, $row_dif);
    local @cshift = (0, 0, 0, 0, $col_dif);
    $self->move(4, $maze);
}
```

```python
def teleport(self,maze):
    import random
    row = random.randint(0,maze.getHeight() - 1)
    col = random.randint(0,maze.getWidth() - 1)
    pos = Position()
    pos.setR(row)
    pos.setC(col)
    if(maze.getCellContent(pos) == '*'):
        self.leave(maze)
        self.curPos.setR(row)
        self.curPos.setC(col)
        self.occupy(maze)
    else:
        maze.explore(pos)
```

-The Perl code makes use of dynamic scoping. It makes the arrays “rshift” and “cshift” ‘local variables’ and computes the difference in columns and rows that would let the function “move” make the player move to the desired location by looking at the “row_dif” and “col_dif” variables in dynamically scoped arrays “cshift” and “rshift”. On the other hand, the Python code is unable to have such an option. Therefore, the Python code just picks a random legitimate position and just switches positions of the current player.

-The same method of modifying dynamically scoped arrays “rshift” and “cshift” and passing them to “move” function is used in other special move functions as well. So, making use of dynamic scoping provides you the convenience of having a persistent method of using the function “move” in all the special move functions such as “teleport”, “throughBlocked” and “rush”.

-As can be seen the code on the left provides convenience in the way that we can always make use of the move function by merely modifying the dynamically scoped array variables cshift and rshift. On the other hand, this decreases the readability of the code. As can be seen when comparing the Python code to Perl code. So, there is always a tradeoff between convenience and readability.

Q2)

Local keyword temporarily changes the value of global variables. And it makes use of the idea of dynamic scoping when doing so. In other words, when we try to access the value of a variable, unlike statically scoped variables, we do not go out one level but check the variable’s value in the place it was previously called. The keyword local is usually used when we want the value of a variable to be visible to called subroutines. Although “my” has a wider usage, sometimes using “local” is a wiser option.

In scenarios which we want to give a global variable a temporary value, or in scenarios in which we want to temporarily change some elements, if not only one element of an array, using local could be

considered. Just like the way I localized cshift and rshift global variables and changed their value before calling the subroutine “move”, inside special functions “teleport”, “rush” and “throughBlocked”.

When it comes to my opinion, I think that local should be avoided if possible. Although it could cause conveniences in the scenarios listed above, it could cause a lot of confusion to both the person coding the program and the one reading it. And I think that this is the reason most of the contemporary used programming languages do not have the option of dynamic scoping to have cleaner codes and less confusion. Also, I had to think so much to find a way of using dynamic scoping in these functions when using “my” is just sufficient in solving the issue in a clean way.