Below is a function used in a sample videogame, which contains a lot of repeated code:

```haskell
# next-state-tick : GameState -> GameState
fun next-state-tick(g):
    if g.playerx < -50: game(700, g.playery, g.dangerx, g.targetx, g.score)
    else if g.dangerx < -50: game(g.playerx, g.playery, 700, g.targetx, g.score)
    else if g.targetx < -50: game(g.playerx, g.playery, g.dangerx, 700, g.score)
    else: g
end
end
```

Write the contract and purpose statement for a helper function you could write in order to make the code shorter and easier to read.

```haskell
# is-off-left : Number -> Boolean

Consumes an x-coordinate and asks if it is less than -50
```
Detecting Helper Functions

Below is a function used in a sample videogame, which contains a lot of repeated code:

```plaintext
# draw-state : GameState -> Image

fun draw-state(g):
    if g.level == 1: put-image(PLAYER-IMG1,
                                g.playerx, g.playery,
                                BACKGROUND)
    else if g.level == 2: put-image(PLAYER-IMG2,
                                     g.playerx, g.playery,
                                     BACKGROUND)
    else if g.level == 3: put-image(PLAYER-IMG3,
                                     g.playerx, g.playery,
                                     BACKGROUND)
    else: g
end
```

Write the contract and purpose statement for a helper function you could write in order to make the code shorter and easier to read.

```plaintext
# __draw-char__ : __Number__________ -> __Image_____

Consumes a level and produces the correct character image based on that level.
```