SQLillo Royale

Your task is to program a bot that survives for as long as possible.

The bot is implemented in the Lua programming language. At the most basic idea, you need to implement two functions

```
def bot_init(me)
    ...
end

def bot_main(me)
    ...
end
```

The `bot_init` function will be called once at the start of a match, and after that `bot_main` will be called on each tick.

You can use globals to save information between each tick and you can use any Lua data structure and language feature.

Both functions receive a `me` which is the type you use to query your surroundings, move or cast a skill. The available functions are

- `me:pos()` returns a vec type with your player position
- `me:move(v)` given a vector moves in that direction this tick
- `me:health()` returns your current health
- `me:visible()` returns a list of visible entities
- `me:cod()`
- `me:cast(s, v)` casts a skill spell `s` in the `v` direction. `s` can be 0 - small projectile, damage is 10, cooldown is 1. 1 - dash in the given direction 2 - melee attack, distance is 2, damage 20 and cooldown 50.

The `visible()` call returns an `entity` type which has the following methods

- `:pos()` returns a vec type with the entity position
- `:type()` returns a string that represents the entity type, it can be `player` or `bullet`
- `:id()` returns the id of the entity as an integer. You can use it to check if it's the same entity. Bullets don't have a stable id, players do.

The `vec` type has the following methods

- `:add(v)` returns a new vector that is the result of adding both of them.
- `:sub(v)` returns a new vector that is the result of subtracting both of them.
- `:x()` returns the x component of the vector
- `:y()` returns the x component of the vector
- `:neg()` returns a new vector that is the result of negating the vector

Sample code

```
-- Global variables
local target = nil
local cooldowns = {0, 0, 0}

-- Initialize bot
function bot_init(me)
end

-- Main bot function
function bot_main(me)
    local me_pos = me:pos()

    -- Update cooldowns
    for i = 1, 3 do
        if cooldowns[i] > 0 then
            cooldowns[i] = cooldowns[i] - 1
        end
    end

    -- Find the closest visible enemy
    local closest_enemy = nil
    local min_distance = math.huge

    for _, player in ipairs(me:visible()) do
        local dist = vec.distance(me_pos, player:pos())

        if dist < min_distance then
            min_distance = dist
            closest_enemy = player
        end
    end

    -- Set target to closest visible enemy
    local target = closest_enemy

    if target then
        local direction = target:pos():sub(me_pos)

        -- If target is within melee range and melee attack is not on cooldown, use melee a
        if min_distance <= 2 and cooldowns[3] == 0 then
            me:cast(2, direction)
            cooldowns[3] = 50
        -- If target is not within melee range and projectile is not on cooldown, use proje
        elseif cooldowns[1] == 0 then
```

```
            me:cast(0, direction)
            cooldowns[1] = 1
        end

        -- Move towards the target
        me:move(direction)
    end
end
```