Francis' answer to the Alien Invasion Problem

How to approach

One of the approaches to solving this problem is to assume that the events of destroying cities and killing aliens happen separately. That is, the problem can be divided by two steps following:

- Step1: At the same time, all alive aliens wander around the cities as defined in the problem.
- Step2: Then, their moving are evaluated (destroyed and killed) simultaneously as defined.

This assumption is very important. If each alien wandered individually at a different time and those destructive events happened separately, this problem would have become much more complex.

With the help of this assumption, I just need to introduce only two basic data structure.

```
# world map info
world_map := {k: v}, where k: "city" and v: "route info of each city"
                     v = {p: q}, where p: "direction such as 'north', 'south', ..."
                                         q: "the adjacent cities connected with roads"

# aliens' position info: tracking aliens every turn
aliens_spot := {k: v}, where k: "city where aliens are invading"
                               v: "alien identification" in terms of v = set(alien_id)

```

I avoided OOP style coding as it can be solved with a simple data structure introduced above.
For more detail, please see the commented code below.

Note1: Individually Implemented using two languages, both *Go*, and *Python*. It seems a little funny as almost the same algorithm was used in each. Please kindly consider them.

Note2: Based on the Christopher's answer (***Q***: *Does "kill and destroy" happen when the number of aliens is exactly equal to two?* ***A***: *"kill and destroy" happens with two or more aliens.*), every city with two or more aliens was destroyed and all aliens in it were killed, when evaluating each turn right after moving of aliens. Printing out event messages is closely connected to this fundamental definition.

How to run

```
# To run with other map files, modify the variable "FILE_MAP" in the src file
### Go
$ cd go

$ go build alien_invasion.go
$ ./alien_invasion NUM_OF_ALIENS
# or
$ go run alien_invasion.go NUM_OF_ALIENS

# simple unit test
$ go test -v

### Python
$ cd python

# Python 3.6 or later is recommended
$ python3 alien_invasion.py NUM_OF_ALIENS
```

Some comments with code (based on Go implementation)

```go
/*
Note: here assumed that ONLY VALID DATA will be given as a map file.
In order to focus on the algorithm itself, I skipped all validations of
input files and command-line arguments.
*/
package main

import (
    "fmt"
    "bufio"
    "os"
    "strings"
    "strconv"
    "time"
    "math/rand"
)

type Roads map[string]string
type WorldMap map[string]Roads
type Intruders map[int]bool
type AlienSpot map[string]Intruders

func parse_world_map(mapfile string) WorldMap {
    f, err := os.Open(mapfile)
    if err != nil { panic(err) }
    defer f.Close()

    world_map := make(WorldMap)
    scanner := bufio.NewScanner(f)
    for scanner.Scan() {
        line := strings.TrimSpace(scanner.Text())
        // skipping blank line in the world map file
        if len(line) == 0 { continue }
        tokens := strings.Fields(line)
        city, roads := tokens[0], tokens[1:]
        world_map[city] = make(Roads)
        for _, road := range roads {
            split := strings.Split(road, "=")
            direction, dest := split[0], split[1]
            world_map[city][direction] = dest
        }
    }
    return world_map
}

func dump_world_map(world_map WorldMap) {
    fmt.Println("\n----- THE REMAINING WORLD ----------")
    if len(world_map) == 0 { fmt.Println("All cities are destroyed.") }
    for city, roads := range world_map {
        roads_string := ""
        for direction, dest := range roads {
            roads_string += fmt.Sprintf(" %s=%s", direction, dest)
        }
        fmt.Printf("%s%s\n", city, roads_string)
    }
}

func dump_destroy_event(city string, intruders Intruders) {
    ids := make([]string, 0, len(intruders))
    for id, _ := range intruders { ids = append(ids, fmt.Sprintf("alien %d", id)) }
```

```
    destroyer := strings.Join(ids[:len(ids)-1], ", ")
    destroyer += fmt.Sprintf(" and %s", ids[len(ids)-1])
    fmt.Printf("%s has been destroyed by %s!\n", city, destroyer)
}

func init_alien_invasion(num_aliens int, world_map WorldMap) AlienSpot {
    rand.Seed(time.Now().Unix())
    alien_spot := make(AlienSpot)
    if len(world_map) == 0 { return alien_spot }
    cities := make([]string, 0, len(world_map))
    for k := range world_map { cities = append(cities, k) }
    for id := 1; id <= num_aliens; id++ {
        invaded := cities[rand.Intn(len(cities))]
        move_alien_into_city(id, invaded, alien_spot)
    }
    return alien_spot
}

func move_alien_from_city(alien_id int, city string, alien_spot AlienSpot) {
    delete(alien_spot[city], alien_id)
    if len(alien_spot[city]) == 0 { delete(alien_spot, city) }
}

func move_alien_into_city(alien_id int, city string, alien_spot AlienSpot) {
    _, ok := alien_spot[city]
    if ok {
        alien_spot[city][alien_id] = true
    } else {
        alien_spot[city] = Intruders{alien_id: true}
    }
}

func wander_randomly(world_map WorldMap, alien_spot AlienSpot) {
    for from_city, intruders := range alien_spot {
        // No wandering when a alien gets trapped
        if roads, ok := world_map[from_city]; !ok || len(roads) == 0 { continue }

        cities := make([]string, 0, len(world_map[from_city]))
        for _, v := range world_map[from_city] { cities = append(cities, v) }
        if len(cities) == 0 { continue }
        for alien_id, _ := range intruders {
            // random selection based on roads connected to cities
            into_city := cities[rand.Intn(len(cities))]
            // pull aliens out of an existing location
            move_alien_from_city(alien_id, from_city, alien_spot)
            // place aliens into the newly selected location
            move_alien_into_city(alien_id, into_city, alien_spot)
        }
    }
}

func destroy_and_kill(world_map WorldMap, alien_spot AlienSpot) {
    for city, intruders := range alien_spot {
        // if the number of aliens at the same place is less than 2, do nothing
        if len(intruders) < 2 { continue }

        // otherwise, destroy the city and kill aliens in it
        for _, vicinity := range world_map[city] {
            // finding all related routes and destroy it
            for direction, c := range world_map[vicinity] {
                if c == city { delete(world_map[vicinity], direction) }
            }
        }
```

```
        // remove the city from world map and from the alien tracker
        delete(world_map, city)
        delete(alien_spot, city)

        // print out an event log based on the problem definition
        dump_destroy_event(city, intruders)
    }
}

func main() {
    if len(os.Args) != 2 {
        fmt.Fprintf(os.Stderr, "Usage: %s <num of aliens>\n", os.Args[0])
        return
    }
    const FILE_MAP = "../worldmap/universe.txt"
    const MAX_MOVES_ALIENS = 10000
    num_aliens, _ := strconv.Atoi(os.Args[1])

    // parse the map file given
    world_map := parse_world_map(FILE_MAP)

    // initialize the aliens position
    alien_spot := init_alien_invasion(num_aliens, world_map)
    destroy_and_kill(world_map, alien_spot)

    moves := 0
    for {
        // exit conditions
        if moves > MAX_MOVES_ALIENS { break }
        if len(world_map) == 0 { break }
        if len(alien_spot) == 0 { break }

        // each alien wanders randomly each turn
        wander_randomly(world_map, alien_spot)

        // update world map and aliens' position by definition
        // destroy cities and kill aliens if needed.
        destroy_and_kill(world_map, alien_spot)
        moves++
    }
    dump_world_map(world_map)
}
```