Matching

Removing the go statement from `Seek` will turn the program into a synchronous one, where all steps can be predicted. Anna is going to send a message to Bob, Cody to Dave, and Eva to no one.

Changing wait group declarations will throw an error because we don't pass the wait group by reference to `Seek` method and so the wait group inside `Seek` isn't able to communicate with the one in `main`. Two messages will be sent/received and the last one will fail as all go routines are asleep.

Removing the buffer will thrown an error because unbuffered channels accept sends only if there is a corresponding receive. There doesn't seem to be a clear receiver of the sent information through the channel.

> By default channels are unbuffered, meaning that they will only accept sends (chan ¡-) if there is a corresponding receive (¡- chan) ready to receive the sent value. Buffered channels accept a limited number of values without a corresponding receiver for those values.[1]

Removing default will not cause any problems in the current set up as it is constructed in such a way that one will be left unmatched. However, decreasing the size of the array of people, for example by removing *Eva* will throw a an error (a deadlock) because there is no one in the match channel left and it is still open. Thus, we are waiting for something that never arrives, which leads to a deadlock. A solution would be to add `close(match)` right after `wg.Wait()`.

Julia

The program uses all CPUs. The speed improved in the order of 20s. It seems like some might argue that we should have go routines for each pixel, but I think that we should have four that can be spread out on 4 processors.

```
// Stefan Nilsson 2013-02-27

// This program creates pictures of Julia sets
    (en.wikipedia.org/wiki/Julia_set).
package main

import (
    ”fmt”
    ”image”
    ”image/color”
    ”image/png”
    ”log”
    ”math/cmplx”
    ”os”
    ”runtime”
    ”strconv”
    ”sync”
    ”time”
)

type ComplexFunc func(complex128) complex128

var Funcs []ComplexFunc = []ComplexFunc{
    func(z complex128) complex128 { return z*z - 0.61803398875 },
    func(z complex128) complex128 { return z*z + complex(0, 1) },
```

[1] https://gobyexample.com/channel-buffering, accessed 2015.04.14

```go
    func(z complex128) complex128 { return z*z + complex(-0.835, -0.2321) },
    func(z complex128) complex128 { return z*z + complex(0.45, 0.1428) },
    func(z complex128) complex128 { return z*z*z + 0.400 },
    func(z complex128) complex128 { return cmplx.Exp(z*z*z) - 0.621 },
    func(z complex128) complex128 { return (z*z+z)/cmplx.Log(z) + complex(0.268, 0.060) },
    func(z complex128) complex128 { return cmplx.Sqrt(cmplx.Sinh(z*z)) + complex(0.065, 0.122) },
}

func main() {
    // use all the power!
    numcpu := runtime.NumCPU()
    runtime.GOMAXPROCS(numcpu)

    start := time.Now()

    ch := make(chan error, len(Funcs)) // new channel

    wg := new(sync.WaitGroup)
    wg.Add(len(Funcs)) // the number of functions = no. of pictures.
    for n, fn := range Funcs {
        go CreatePng("picture-"+strconv.Itoa(n)+".png", fn, 1024, wg, ch)
    }

    wg.Wait()
    close(ch)

    for k := range ch {
        if k != nil {
            log.Fatal(k)
        }
    }
    fmt.Println(time.Since(start))
}

// CreatePng creates a PNG picture file with a Julia image of size n x n.
func CreatePng(filename string, f ComplexFunc, n int, wg *sync.WaitGroup, ch chan error) { // need pointer to wg.
    file, err := os.Create(filename)
    if err != nil {
        ch <- err
        return
    }
    defer file.Close()
    ch <- png.Encode(file, Julia(f, n))

    wg.Done()
    return
}

// Julia returns an image of size n x n of the Julia set for f.
func Julia(f ComplexFunc, n int) image.Image {
    bounds := image.Rect(-n/2, -n/2, n/2, n/2)
    img := image.NewRGBA(bounds)
    s := float64(n / 4)

    // code duplication... not good.

    for i := bounds.Min.X; i < bounds.Max.X; i++ {
        wg := new(sync.WaitGroup)
        wg.Add(4)
        go func() {
            for j := bounds.Min.Y; j < bounds.Max.Y/4; j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }

            wg.Done()
        }()

        go func() {
            for j := bounds.Max.Y / 4; j < 2*(bounds.Max.Y/4); j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }
            wg.Done()
        }()

        go func() {
            for j := 2 * (bounds.Max.Y / 4); j < 3*(bounds.Max.Y/4); j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }
            wg.Done()
        }()

        go func() {
            for j := 3*(bounds.Max.Y/4) + (bounds.Max.Y % 4); j < bounds.Max.Y; j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }
            wg.Done()
        }()

        wg.Wait()
    }

    return img
}

// Iterate sets z_0 = z, and repeatedly computes z_n = f(z_{n-1}), n 1,
// until |z_n| > 2  or n = max and returns this n.
func Iterate(f ComplexFunc, z complex128, max int) (n int) {
    for ; n < max; n++ {
        if real(z)*real(z)+imag(z)*imag(z) > 4 {
            break
        }
        z = f(z)
    }
    return
}
```

Weather Station

```
// Stefan Nilsson 2013-02-27

// This program creates pictures of Julia sets (en.wikipedia.org/wiki/Julia_set).
package main

import (
    "fmt"
    "image"
    "image/color"
    "image/png"
    "log"
    "math/cmplx"
    "os"
    "runtime"
    "strconv"
    "sync"
    "time"
)

type ComplexFunc func(complex128) complex128

var Funcs []ComplexFunc = []ComplexFunc{
    func(z complex128) complex128 { return z*z - 0.61803398875 },
    func(z complex128) complex128 { return z*z + complex(0, 1) },
    func(z complex128) complex128 { return z*z + complex(-0.835, -0.2321) },
    func(z complex128) complex128 { return z*z + complex(0.45, 0.1428) },
    func(z complex128) complex128 { return z*z*z + 0.400 },
    func(z complex128) complex128 { return cmplx.Exp(z*z*z) - 0.621 },
    func(z complex128) complex128 { return (z*z+z)/cmplx.Log(z) +
        complex(0.268, 0.060) },
    func(z complex128) complex128 { return cmplx.Sqrt(cmplx.Sinh(z*z)) +
        complex(0.065, 0.122) },
}

func main() {
    // use all the power!
    numcpu := runtime.NumCPU()
    runtime.GOMAXPROCS(numcpu)

    start := time.Now()

    ch := make(chan error, len(Funcs)) // new channel

    wg := new(sync.WaitGroup)
    wg.Add(len(Funcs)) // the number of functions = no. of pictures.
    for n, fn := range Funcs {
        go CreatePng("picture-"+strconv.Itoa(n)+".png", fn, 1024, wg, ch)
    }

    wg.Wait()
    close(ch)

    for k := range ch {
        if k != nil {
            log.Fatal(k)
        }
    }
    fmt.Println(time.Since(start))
}

// CreatePng creates a PNG picture file with a Julia image of size n x n.
func CreatePng(filename string, f ComplexFunc, n int, wg *sync.WaitGroup,
    ch chan error) { // need pointer to wg.
    file, err := os.Create(filename)
    if err != nil {
        ch <- err
        return
    }
    defer file.Close()
    ch <- png.Encode(file, Julia(f, n))

    wg.Done()
    return
}

// Julia returns an image of size n x n of the Julia set for f.
func Julia(f ComplexFunc, n int) image.Image {
    bounds := image.Rect(-n/2, -n/2, n/2, n/2)
    img := image.NewRGBA(bounds)
    s := float64(n / 4)

    // code duplication... not good.

    for i := bounds.Min.X; i < bounds.Max.X; i++ {
        wg := new(sync.WaitGroup)
        wg.Add(4)
        go func() {
            for j := bounds.Min.Y; j < bounds.Max.Y/4; j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }

            wg.Done()
        }()

        go func() {
            for j := bounds.Max.Y / 4; j < 2*(bounds.Max.Y/4); j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }
            wg.Done()
        }()

        go func() {
            for j := 2 * (bounds.Max.Y / 4); j < 3*(bounds.Max.Y/4); j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }
            wg.Done()
        }()

        go func() {
            for j := 3*(bounds.Max.Y/4) + (bounds.Max.Y % 4); j <
                bounds.Max.Y; j++ {

                n := Iterate(f, complex(float64(i)/s, float64(j)/s), 256)
                r := uint8(0)
                g := uint8(0)
                b := uint8(n % 32 * 8)
                img.Set(i, j, color.RGBA{r, g, b, 255})

            }
            wg.Done()
        }()

        wg.Wait()
    }

    return img
}

// Iterate sets z_0 = z, and repeatedly computes z_n = f(z_{n-1}), n 1,
// until |z_n| > 2  or n = max and returns this n.
func Iterate(f ComplexFunc, z complex128, max int) (n int) {
    for ; n < max; n++ {
        if real(z)*real(z)+imag(z)*imag(z) > 4 {
            break
        }
        z = f(z)
    }
    return
}
```