Есть интерфейс `waiter` который должен:

1. параллельно запускать переданные в `run` функции с указанным контекстом.
2. количество параллельных запусков определяется параметром `maxParallel` при создании `waiter` через `newGroupWait`
3. возвращать ошибку из `wait`, если хотя бы одна функция из `run` вернула ее
4. возвращать комбинацию ошибок от вызовов `run`, если несколько задач завершились с ошибками (можно использовать `errors.Join`)

Реализуйте поля и методы структуры `waitGroup` для интерфейса `waiter`.

```
package main

import (
 "context"
 "errors"
)

type waiter interface {
 wait() error
 run(ctx context.Context, f func(ctx context.Context) error)
}

type waitGroup struct {
 // напишите ваш код здесь
}

func (g *waitGroup) wait() error {
 // напишите ваш код здесь
}

func (g *waitGroup) run(ctx context.Context, fn func(ctx context.Context) error) {
 // напишите ваш код здесь
}

func newGroupWait(maxParallel int) waiter {
 // напишите ваш код здесь
}

func main() {
 g := newGroupWait(2)

 ctx := context.Background()
 expErr1 := errors.New("got error 1")
 expErr2 := errors.New("got error 2")
 g.run(ctx, func(ctx context.Context) error {
  return nil
 })
 g.run(ctx, func(ctx context.Context) error {
  return expErr2
 })
 g.run(ctx, func(ctx context.Context) error {
  return expErr1
 })

 err := g.wait()
 if !errors.Is(err, expErr1) || !errors.Is(err, expErr2) {
  panic("wrong code")
 }
}
```