

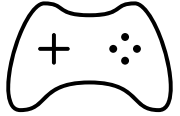
Games in Go

Wie man mit go(lang) ein Spiel baut

Julia Diers Junior Software Development Engineer

Jasmin Feldmann Software Development Engineer

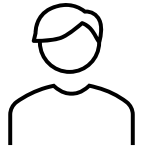
Games in Go



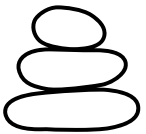
Einleitung - Videospielprinzip



Einrichtung



Spieler anlegen

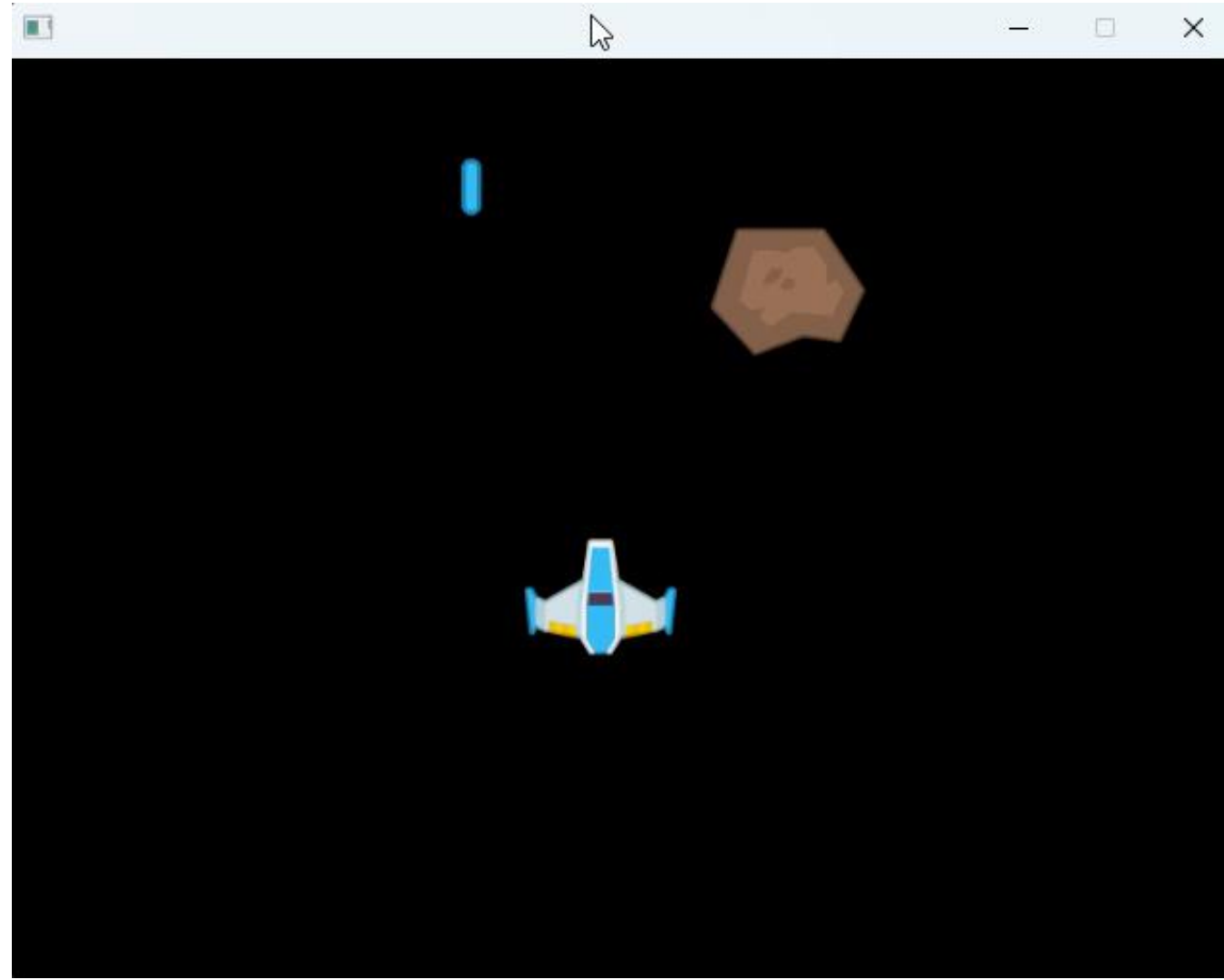


Hindernisse erstelle



Have fun ;)

Einleitung - Videospielprinzip



Einleitung - Videospielprinzip

- Ein Film/Video
 - ist eine sehr lange Abfolge von Bilder
 - die so schnell nacheinander gezeigt werden, dass unser Gehirn den Unterschied nicht erkennen kann.
 - Beim Videospiel
 - Existieren die Bilder nicht im voraus
 - Sie müssen spontan generiert werden
 - Reagieren auf Interaktion des Spielers
- Spiele sind also quasi Schleifen, die das Bild aktualisieren

```
for {  
    DrawFrame()  
}
```

main.go

```
package main
```

```
const (
```

```
    Spielbreite = 800
```

```
    Spielhoehe = 600
```

```
)
```

```
type Meteorit struct {
```

```
}
```

```
type Player struct {
```

```
}
```

```
type Game struct {
```

```
}
```

Wir brauchen:

- Die gröÙe unsers Spiels (**Spielbreite** und **Spielhoehe**)
- Eine Spielerobjekt (**Player**)
- Eine Meteoritenobjekt (**Meteorit**)
- Ein Spielobjekt (**Game**)

main.go

```
import "github.com/hajimehoshi/ebiten/v2"

func (g *Game) Layout(outsideWidth, outsideHeight int) (screenWidth, screenHeight int) {
    return Spielbreite, Spielhoehe
}

func (g *Game) Update() error {
    return nil
}

func (g *Game) Draw(screen *ebiten.Image) {
}

func main() {
    g := &Game{}

    err := ebiten.RunGame(g)
    if err != nil {
        panic(err)
    }
}
```

Hier startet unser Spiel, indem ein Spielobjekt (*Game*) erstellt wird und *RunGame* gerufen wird.

main.go

```
import "github.com/hajimehoshi/ebiten/v2"
```

```
func (g *Game) Layout(outsideWidth, outsideHeight int) (screenWidth, screenHeight int) {  
    return Spielbreite, Spielhoehe  
}
```

```
func (g *Game) Update() error {  
    return nil  
}
```

```
func (g *Game) Draw(screen *ebiten.Image) {  
}
```

```
func main() {  
    g := &Game{}  
  
    err := ebiten.RunGame(g)  
    if err != nil {  
        panic(err)  
    }  
}
```

Um Spielfunktionalität einfach nutzen zu können importieren wir ein Stück Programmcode von jemand Anderem

```
main.go

import "github.com/hajimehoshi/ebiten/v2"

func (g *Game) Layout(outsideWidth, outsideHeight int) (screenWidth, screenHeight int) {
    return Spielbreite, Spielhoehe
}

func (g *Game) Update() error {
    return nil
}

func (g *Game) Draw(screen *ebiten.Image) {
}

func main() {
    g := &Game{}

    err := ebiten.RunGame(g)
    if err != nil {
        panic(err)
    }
}
```

Um Spielfunktionalität einfach nutzen zu können importieren wir ein Stück Programmcode von jemand Anderem

Diese Funktionen werden von dem importierten Programmcode benötigt.

Sie dienen dazu, unsere Bilder darzustellen (**Draw**) und zu aktualisieren (**Update**)

Spielerbild laden

```
main.go

//go:embed assets/*
var assets embed.FS

var PlayerBild = mustLoadImage(name: "assets/player.png")

func mustLoadImage(name string) *ebiten.Image {
    f, err := assets.Open(name)
    if err != nil {
        panic(err)
    }
    defer f.Close()

    img, _, err := image.Decode(f)
    if err != nil {
        panic(err)
    }

    return ebiten.NewImageFromImage(img)
}
```

Der Ordner assets/* wird hinzugefügt

Einer Variablen *PlayerBild* wird das Bild, welches wir von *mustLoadImage* bekommen zugewiesen

Funktion, welche die entsprechende Bilddatei öffnet und in das benötigte Format umwandelt

Spieler anlegen

```
main.go

type Game struct {
    player *Player
}

type Vector struct {
    X float64
    Y float64
}

type Player struct {
    bild      *ebiten.Image
    position Vector
}

func NewPlayer() *Player {
    return &Player{
        position: Vector{X: 100, Y: 100},
        bild:     PlayerBild,
    }
}
```

Ein Spiel (*Game*) enthält einen Spieler (*Player*)
! Achtung: Die Schreibweise ist wichtig

Spieler anlegen

```
main.go

type Game struct {
    player *Player
}

type Vector struct {
    X float64
    Y float64
}

type Player struct {
    bild      *ebiten.Image
    position Vector
}

func NewPlayer() *Player {
    return &Player{
        position: Vector{X: 100, Y: 100},
        bild:     PlayerBild,
    }
}
```

Ein Spieler (*Player*) enthält:

- Sein Bild (*bild*)
- Seine aktuelle Position (*position*) (als Vektor)

Spieler anlegen

main.go

```
type Game struct {  
    player *Player  
}
```

```
type Vector struct {  
    X float64  
    Y float64  
}
```

```
type Player struct {  
    bild *ebiten.Image  
    position Vector  
}
```

```
func NewPlayer() *Player {  
    return &Player{  
        position: Vector{X: 100, Y: 100},  
        bild:      PlayerBild,  
    }  
}
```

Ein **Vector** enthält eine **X** und **Y** Koordinate

Spielerbild anlegen

```
main.go

type Game struct {
    player *Player
}

type Vector struct {
    X float64
    Y float64
}

type Player struct {
    bild      *ebiten.Image
    position Vector
}

func NewPlayer() *Player {
    return &Player{
        position: Vector{X: 100, Y: 100},
        bild:     PlayerBild,
    }
}
```

Ein neues Spilerobjekt (*Player*) wird mit *NewPlayer()* erstellt

Hier wird

- eine Anfangsposition
- unser Spielerbild (siehe Spielerbild laden) festgelegt.

Spieler anlegen

```
main.go
func (p *Player) Draw(screen *ebiten.Image) {
    op := &ebiten.DrawImageOptions{}
    op.GeoM.Translate(p.position.X, p.position.Y)
    screen.DrawImage(p.bild, op)
}

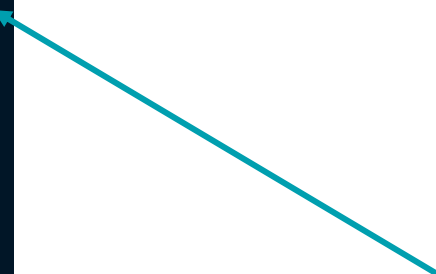
func (g *Game) Draw(screen *ebiten.Image) {
    g.player.Draw(screen)
}
```

Ein Spielobjekt (*Player*) hat auch eine Funktion *Draw*, die sich um das Anzeigen kümmert.

In der Funktion wird die Position des Spielerobjektes an den importierten Programmcode weitergegeben. Dieser erledigt für uns die Anzeige.

```
main.go  
  
func (p *Player) Draw(screen *ebiten.Image) {  
    op := &ebiten.DrawImageOptions{}  
    op.GeoM.Translate(p.position.X, p.position.Y)  
    screen.DrawImage(p.bild, op)  
}  
  
func (g *Game) Draw(screen *ebiten.Image) {  
    g.player.Draw(screen)  
}
```

```
main.go  
  
type Player struct {  
    bild      *ebiten.Image  
    position Vector  
}
```



Spieler anlegen

```
main.go

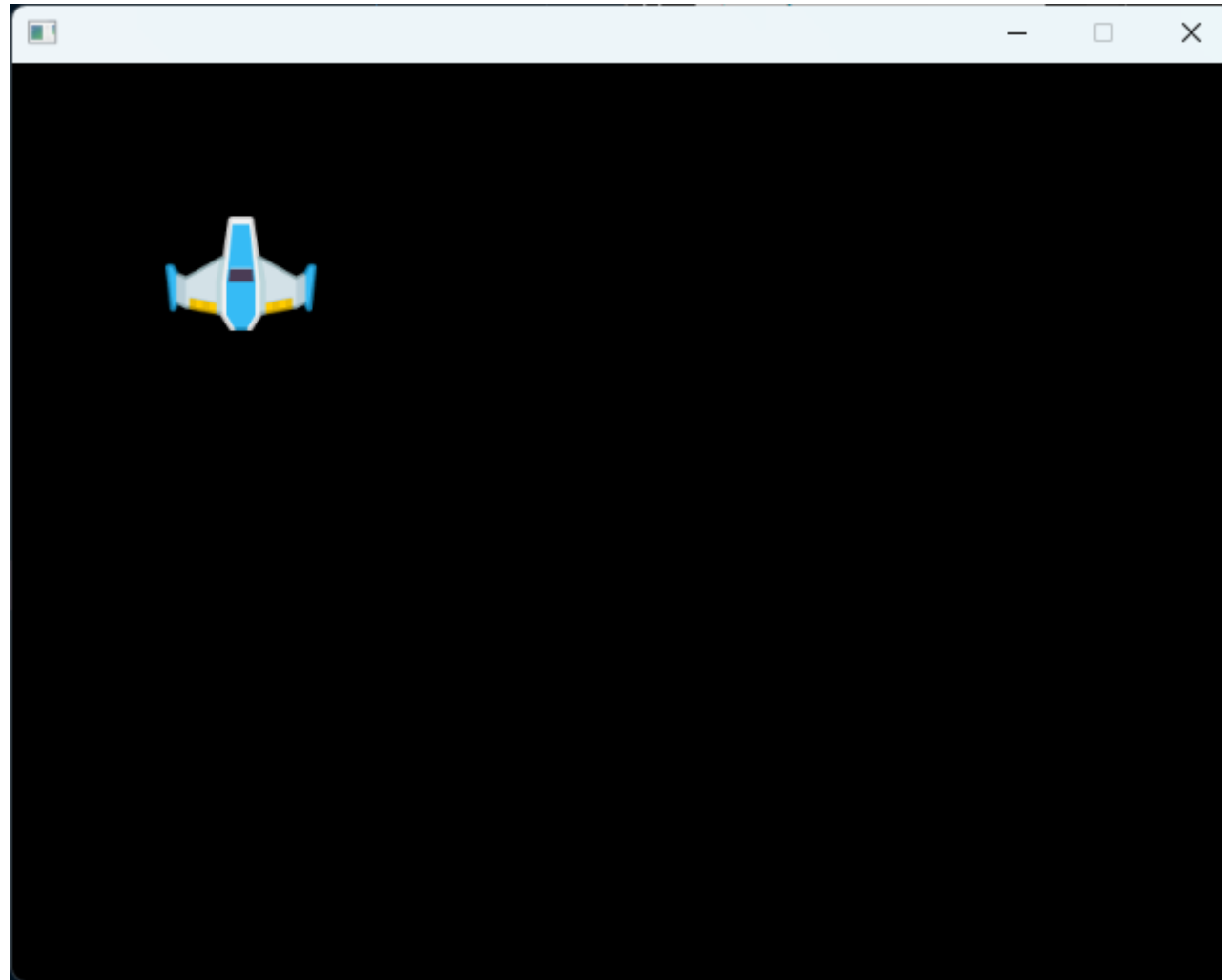
func (p *Player) Draw(screen *ebiten.Image) {
    op := &ebiten.DrawImageOptions{}
    op.GeoM.Translate(p.position.X, p.position.Y)
    screen.DrawImage(p.bild, op)
}

func (g *Game) Draw(screen *ebiten.Image) {
    g.player.Draw(screen)
}
```

Die *Draw()* Funktion des Spielobjektes (**Game**) ruft die *Draw()* Funktion des Spielerobjektes (*Player*)

Spiel starten

d.veLop



```
main.go

func NewPlayer() *Player {
    grenzen := PlayerBild.Bounds()
    halbeBreite := float64(grenzen.Dx()) / 2
    halbeHoehe := float64(grenzen.Dy()) / 2

    pos := Vector{
        X: Spielbreite/2 - halbeBreite,
        Y: Spielhoehe/2 - halbeHoehe,
    }

    return &Player{
        position: pos,
        bild:     PlayerBild,
    }
}
```

Zum Zentrieren müssen wir anhand der Grenzen des Spielerbildes (*PlayerBild.Bounds()*) die X und Y Koordinate ausrechnen.

Für X nehmen wir:

- die Hälfte der **Breite des Spielerbildes**
- die Hälfte der **gesamten Spielbreite**
- Und subtrahieren die **Breite des Spielerbildes** von der **gesamten Spielbreite**

Für Y machen wir das gleiche mit der Höhe

main.go

```
func (p *Player) Update() error {  
    geschwindigkeit := 5.0  
    if ebiten.IsKeyPressed(ebiten.KeyDown) {  
        p.position.Y += geschwindigkeit  
    }  
    if ebiten.IsKeyPressed(ebiten.KeyUp) {  
        p.position.Y -= geschwindigkeit  
    }  
    if ebiten.IsKeyPressed(ebiten.KeyLeft) {  
        p.position.X -= geschwindigkeit  
    }  
    if ebiten.IsKeyPressed(ebiten.KeyRight) {  
        p.position.X += geschwindigkeit  
    }  
    return nil  
}
```

Die Update() Funktion des Spielerobjektes (Player) ist dafür zuständig auf unsere Steuerung zu reagieren.

```
main.go

func (p *Player) Update() error {
    geschwindigkeit := 5.0
    if ebiten.IsKeyPressed(ebiten.KeyDown) {
        p.position.Y += geschwindigkeit
    }
    if ebiten.IsKeyPressed(ebiten.KeyUp) {
        p.position.Y -= geschwindigkeit
    }
    if ebiten.IsKeyPressed(ebiten.KeyLeft) {
        p.position.X -= geschwindigkeit
    }
    if ebiten.IsKeyPressed(ebiten.KeyRight) {
        p.position.X += geschwindigkeit
    }
    return nil
}
```

Die Update() Funktion des Spielerobjektes (Player) ist dafür zuständig auf unsere Steuerung zu reagieren.

Mit *ebiten.IsKeyPressed()* bekommen wir eine Information darüber, ob die angegebene Taste gedrückt wurde.

main.go

```
func (p *Player) Update() error {  
    geschwindigkeit := 5.0  
    if ebiten.IsKeyPressed(ebiten.KeyDown) {  
        p.position.Y += geschwindigkeit  
    }  
    if ebiten.IsKeyPressed(ebiten.KeyUp) {  
        p.position.Y -= geschwindigkeit  
    }  
    if ebiten.IsKeyPressed(ebiten.KeyLeft) {  
        p.position.X -= geschwindigkeit  
    }  
    if ebiten.IsKeyPressed(ebiten.KeyRight) {  
        p.position.X += geschwindigkeit  
    }  
    return nil  
}
```

Für Pfeiltasten

- runter (*ebiten.KeyDown*)
- rauf (*ebiten.KeyUp*)
- links (*ebiten.KeyLeft*)
- rechts (*ebiten.KeyRight*)

wird die jeweilige Koordinate mit einer festen *geschwindigkeit* addiert/subtrahiert.

Spieler bewegen

```
main.go

func (p *Player) Update() error {
    geschwindigkeit := 5.0
    if ebiten.IsKeyPressed(ebiten.KeyDown) {
        p.position.Y += geschwindigkeit
    }
    if ebiten.IsKeyPressed(ebiten.KeyUp) {
        p.position.Y -= geschwindigkeit
    }
    if ebiten.IsKeyPressed(ebiten.KeyLeft) {
        p.position.X -= geschwindigkeit
    }
    if ebiten.IsKeyPressed(ebiten.KeyRight) {
        p.position.X += geschwindigkeit
    }
    return nil
}
```

Update wird 60 mal die Sekunde ausgeführt!

Das Bild wird 300 mal die Sekunde aktualisiert.

Spieler bewegen

```
main.go  
  
func (g *Game) Update() error {  
    return g.player.Update()  
}
```

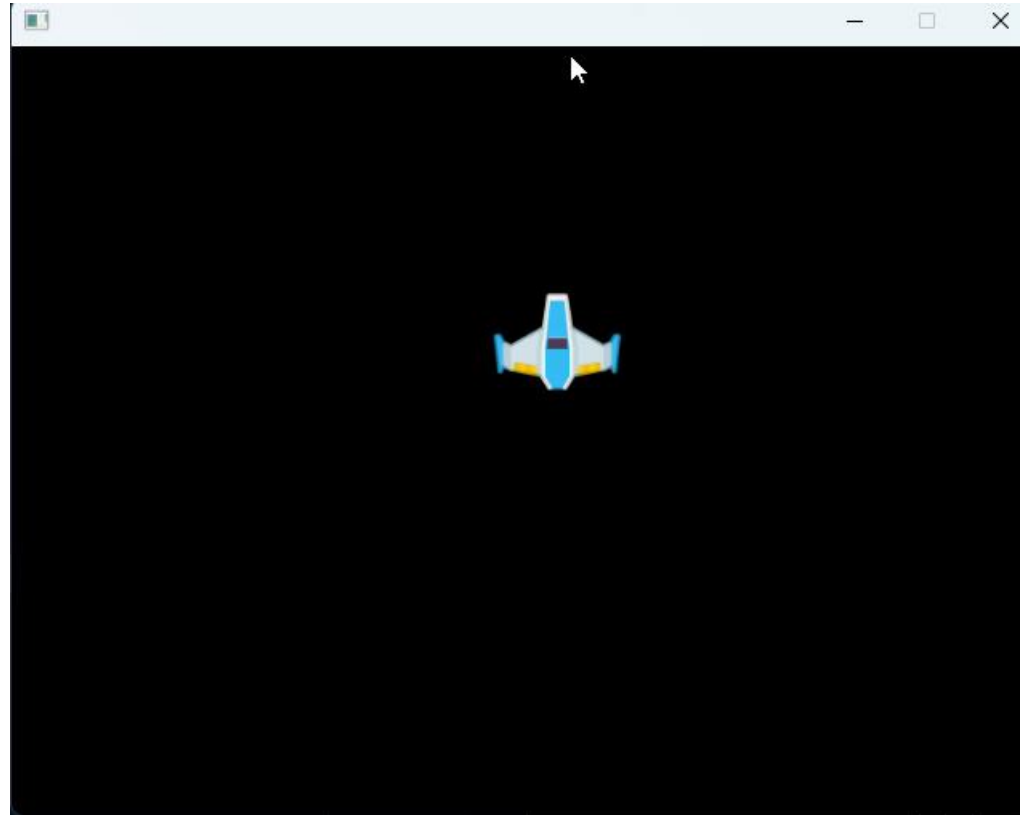
Die *Update*-Funktion des Spielobjektes (*Game*) führt die *Update*-Funktion des Spielerobjektes (*Player*) aus.

Update (des Spielobjektes) wird automatisch 60 mal die Sekunde ausgeführt!

Das Bild wird 300 mal die Sekunde aktualisiert.

Spieler bewegen - ausprobieren

d.velop



Hindernisse (Meteoriten) einfügen

main.go

```
var MeteoritenBilder = mustLoadImages(path: "assets/meteors/*.png")
```

```
type Meteorit struct {  
    bild      *ebiten.Image  
    position  Vector  
    bewegung  Vector  
}
```

```
func mustLoadImages(path string) []*ebiten.Image {  
    matches, err := fs.Glob(assets, path)  
    if err != nil {  
        panic(err)  
    }  
  
    images := make([]*ebiten.Image, len(matches))  
    for i, match := range matches {  
        images[i] = mustLoadImage(match)  
    }  
  
    return images  
}
```

Ein Meteorit enthält:

- Sein Bild
- Seine aktuelle Position
- Seine Bewegung (um wie viel die X und Y sich verändern sollen)

Hindernisse (Meteoroiten) laden

main.go

```
var MeteoritenBilder = mustLoadImages( path: "assets/meteors/*.png")
```

```
type Meteorit struct {  
    bild      *ebiten.Image  
    position  Vector  
    bewegung  Vector  
}
```

```
func mustLoadImages(path string) []*ebiten.Image {  
    matches, err := fs.Glob(assets, path)  
    if err != nil {  
        panic(err)  
    }  
  
    images := make([]*ebiten.Image, len(matches))  
    for i, match := range matches {  
        images[i] = mustLoadImage(match)  
    }  
  
    return images  
}
```

mustLoadImages() nimmt nun einen Pfad entgegen und kann eine Liste an Bilder (*[]*ebiten.Image*) zurückgeben.

Das nutzen wir um verschiedene Meteoriten anzeigen zu können.

!Achtung: *mustLoadImage()* benötigen wir immer noch

Hindernisse (Meteoroiten) anlegen

main.go

```
type Game struct {  
    player          *Player  
    meteoriten      []*Meteorit  
    meteoritenSpawnTimer *timer.Timer  
}
```

Ein Spielobjekt bekommt

- eine Liste an Meteoriten (*meteoriten* *[]*Meteorit*)
- Eine Stoppuhr, nach wie viel Zeit ein Meteorit erstellt werden soll (*meteoritenSpawnTimer*)

Hindernisse (Meteroiten) anlegen

```
main.go

func main() {
    g := &Game{
        meteoritenSpawnTimer: timer.NewTimer(2 * time.Second),
        player:                 NewPlayer(),
    }

    err := ebiten.RunGame(g)
    if err != nil {
        panic(err)
    }
}
```

Im Spielobjekt wird mit *timer.NewTimer(2 * time.Second)* ein neuer Spawn Timer erstellt.

Dieser gibt alle 2 Sekunden das Signal (*IsReady()*), um einen neuen Meteoriten anzulegen.

Hindernisse (Meteroiten) anlegen

```
main.go
func NewMeteorit() *Meteorit {
    return &Meteorit{
        bild: MeteoritenBilder[rand.Intn(len(MeteoritenBilder))],
        position: Vector{
            X: float64(rand.Intn(Spielbreite)),
            Y: 0,
        },
        bewegung: Vector{
            X: 0,
            Y: float64(rand.Intn(3) + 1), // Random speed between 1 and 3
        },
    }
}
```

Ein neuer Meteroit wird mit *NewMeteorit()* erstellt.

Er enthält:

- Ein zufällig ausgewähltes MeteoritenBild
rand.Intn(len(MeteroitenBilder)) wählt eine zufällige Zahl zwischen 0 und Anzahl MeteoritenBilder
- Eine Anfangsposition
Die Meteoriten sollen alle am oberen Rand starten (Y: 0) und eine zufällige X Position bekommen
- Eine Position in der sich der Meteorit bewegen soll
Die Meteoriten sollen sich gerade nach unten bewegen, weswegen nur Y sich erhöht.

Hindernisse (Meteroiten) anzeigen

```
main.go

func (g *Game) Update() error {
    g.player.Update()

    g.meteoritenSpawnTimer.Update()
    if g.meteoritenSpawnTimer.IsReady() {
        g.meteoritenSpawnTimer.Reset()
        g.meteoriten = append(g.meteoriten, NewMeteorit())
    }

    return nil
}
```

Ein neuer *Meteorit* wird nicht, wie der *Player* in der main() Methode erstellt, sondern wir entsprechend des *meteoritenSpawnTimers* in der Spielobjekt (*Game*) *Update()* Funktion erstellt.

Der neue Meteroit wird mit *append* zu der Meteoritenliste im Spieleobjekt (*Game*) hinzugefügt

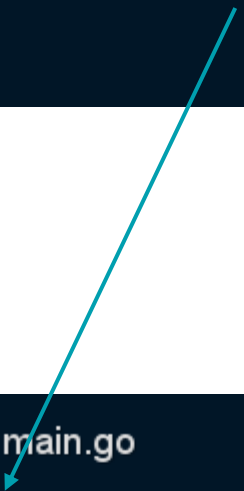
Hindernisse (Meteroiten) animieren

```
main.go  
  
func (m *Meteorit) Update() {  
    m.position.X += m.bewegung.X  
    m.position.Y += m.bewegung.Y  
}
```

Die Bewegung eines Meteroits ist in der *Update*-Funktion am Meteoritenobjekt festgelegt.

Die X Koordinate wird mit dem Bewegungsfaktor m.bewegung.X addiert.

Die Y Koordinate wird mit dem Bewegungsfaktor m.bewegung.Y addiert.



```
main.go  
  
bewegung: Vector{  
    X: 0,  
    Y: float64(rand.Intn(3) + 1), // Random speed between 1 and 3  
},
```

Hindernisse (Meteoriten) animieren

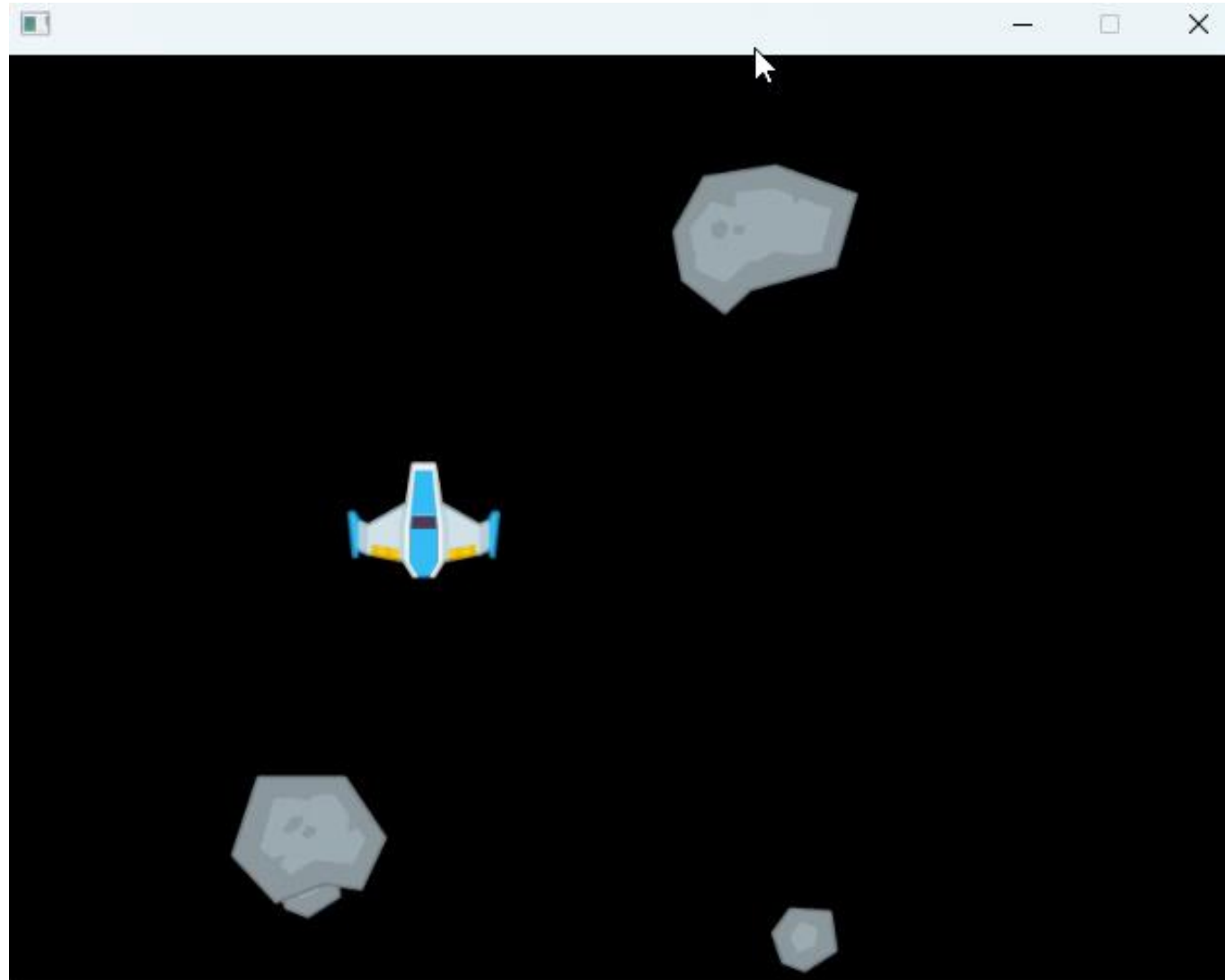
main.go

```
func (g *Game) Update() error {  
    g.player.Update()  
  
    g.meteoritenSpawnTimer.Update()  
    if g.meteoritenSpawnTimer.IsReady() {  
        g.meteoritenSpawnTimer.Reset()  
        g.meteoriten = append(g.meteoriten, NewMeteorit())  
    }  
  
    for _, m := range g.meteoriten {  
        m.Update()  
    }  
  
    return nil  
}
```

In der *Update*-Funktion des Spielobjektes (*Game*) wird für jeden Meteoriten *m.Update()* gerufen

Hindernisse (Meteroiten) - ausprobieren

d.veLop



```
main.go

func (p *Player) KollisionsRechteck() rect.Rect {
    bounds := p.bild.Bounds()
    return rect.NewRect(
        p.position.X,
        p.position.Y,
        float64(bounds.Dx()),
        float64(bounds.Dy()),
    )
}

func (m *Meteorit) KollisionsRechteck() rect.Rect {
    bounds := m.bild.Bounds()
    return rect.NewRect(
        m.position.X,
        m.position.Y,
        float64(bounds.Dx()),
        float64(bounds.Dy()),
    )
}
```

Player und *Meteorit* erhalten ein *KollisionsRechteck()*

Dieses gibt die Grenzen eines Objektes an.

```
main.go

func (g *Game) Update() error {
    g.player.Update()

    g.meteoritenSpawnTimer.Update()
    if g.meteoritenSpawnTimer.IsReady() {
        g.meteoritenSpawnTimer.Reset()
        g.meteoriten = append(g.meteoriten, NewMeteorit())
    }

    for _, m := range g.meteoriten {
        m.Update()
        if m.KollisionsRechteck().IstKollidiert(g.player.KollisionsRechteck()) {
            g.GameOver()
        }
    }

    return nil
}
```

In der *Update*-Funktion des Spieleobjektes (*Game*) prüfen wir für jeden Meteoriten, ob es eine Kollision gab (*IstKollidiert*)

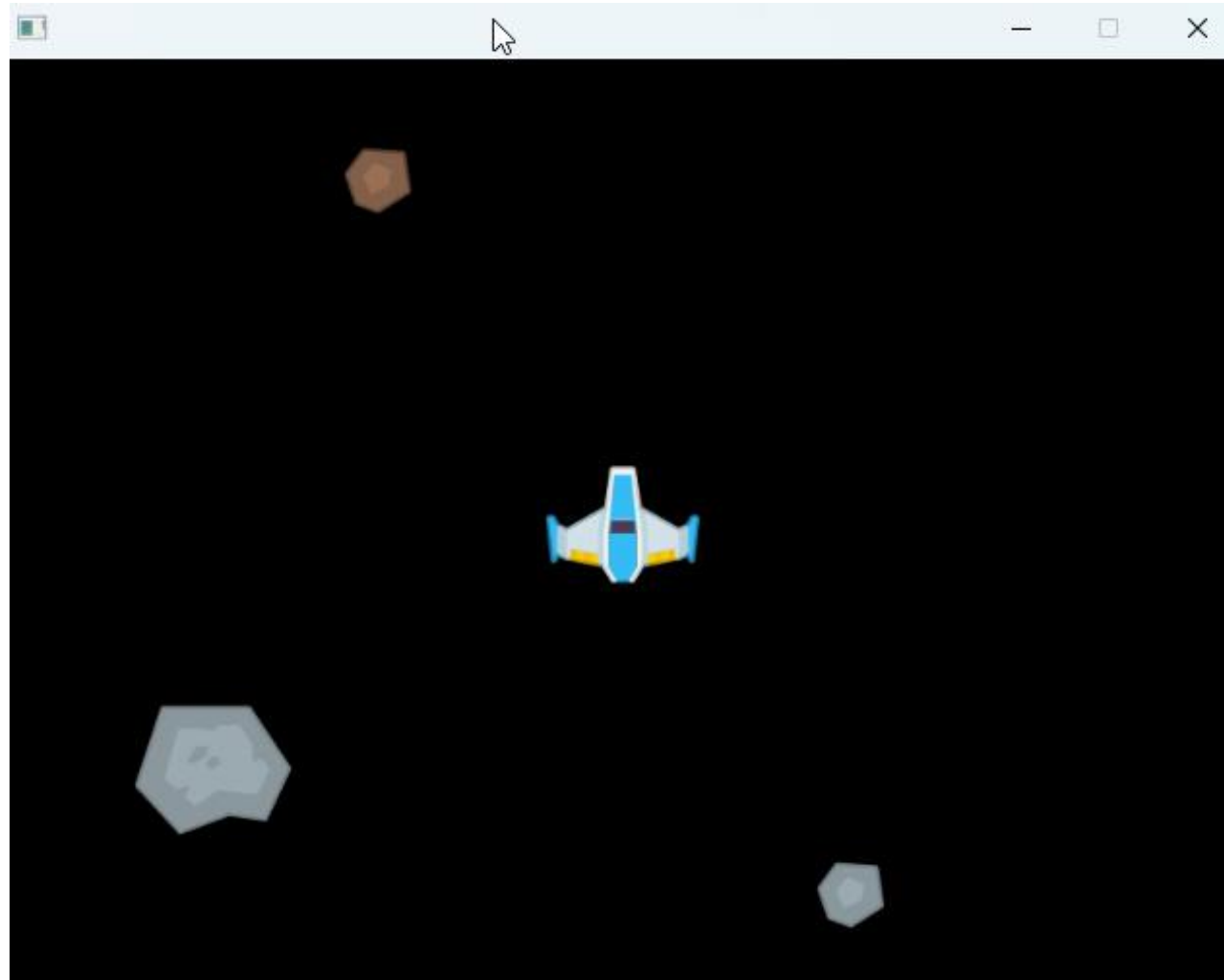
Kollision - GameOver

```
main.go  
  
func (g *Game) GameOver() {  
    g.player = NewPlayer()  
    g.meteoriten = nil  
}
```

GameOver() setzt das Spiel auf den Ursprung zurück

Kollision - ausprobieren

d.velop



Have fun ;)

d.velop

- Wähle einen eigenen main character
 - Aus dem assets Ordner
 - <https://kenney.nl/assets/category:2D?sort=update>

Have fun ;)

- Verändere die Farbe des Players bei einer Kollision

```
● ● ● main.go  
  
op := &colorm.DrawImageOptions{}  
op.GeoM.Translate(p.position.X, p.position.Y)  
  
cm := colorm.ColorM{}  
cm.Translate(r: 1.0, g: 1.0, b: 1.0, a: 0.0)  
colorm.DrawImage(screen, p.bild, cm, op)
```

Have fun ;)

- Laser!!!

```
const (
    laserSpeedPerSecond = 350 = 350.0
)

var laser = MustLoadImage(name: "assets/laser.png")

type Laser struct {
    position Vector
    bild      *ebiten.Image
}

func NewLaser(playerPos Vector, playerBounds image.Rectangle) *Laser {
    halfW := float64(playerBounds.Dx()) / 2
    halfH := float64(playerBounds.Dy()) / 2

    playerPos.X += halfW
    playerPos.Y -= halfH

    laserBounds := laser.Bounds()
    halfW = float64(laserBounds.Dx()) / 2
    halfH = float64(laserBounds.Dy()) / 2

    playerPos.X -= halfW
    playerPos.Y -= halfH

    b := &Laser{
        position: playerPos,
        bild:     laser,
    }

    return b
}
```

```
func (b *Laser) Update() {
    speed := laserSpeedPerSecond / float64(ebiten.TPS())
    b.position.Y += -speed
}

func (b *Laser) Draw(screen *ebiten.Image) {
    bounds := b.bild.Bounds()
    halfW := float64(bounds.Dx()) / 2
    halfH := float64(bounds.Dy()) / 2

    op := &ebiten.DrawImageOptions{}

    op.GeoM.Translate(b.position.X+halfW, b.position.Y+halfH)

    screen.DrawImage(b.bild, op)
}

func (b *Laser) KollisionsRechteck() Rect {
    bounds := b.bild.Bounds()

    return NewRect(
        b.position.X,
        b.position.Y,
        float64(bounds.Dx()),
        float64(bounds.Dy()),
    )
}
```


Have fun ;)

- Laser!!!

```
main.go

type Player struct {

    bild          *ebiten.Image
    position      util.Vector
    shootCooldown *util.Timer
    lasers        []*util.Laser
}
```

```
main.go

return &Player{
    position:      pos,
    bild:         PlayerBild,
    shootCooldown: util.NewTimer(1 * time.Second),
}
```

```
main.go

func (g *Game) Update() error {
    g.player.Update()

    g.meteoritenSpawnTimer.Update()
    if g.meteoritenSpawnTimer.IsReady() {
        g.meteoritenSpawnTimer.Reset()
        g.meteoriten = append(g.meteoriten, NewMeteorit())
    }

    for _, laser := range g.player.lasers {
        laser.Update()
    }

    for _, m := range g.meteoriten {
        m.Update()
        if m.KollisionsRechteck().IstKollidiert(g.player.KollisionsRechteck()) {
            // Handle collision with player
            g.GameOver()
        }
        for _, laser := range g.player.lasers {
            if m.KollisionsRechteck().IstKollidiert(laser.KollisionsRechteck()) {
                // Remove the meteorite and the laser
                g.meteoriten = append(g.meteoriten[:0], g.meteoriten[1:]...) // Remove the first meteorite
            }
        }
    }

    return nil
}
```

Have fun ;)

- Laser!!!

d.velop

```
main.go

func (p *Player) Update() error {
    geschwindigkeit := 5.0

    if ebiten.IsKeyPressed(ebiten.KeyDown) {
        p.position.Y += geschwindigkeit
    }

    if ebiten.IsKeyPressed(ebiten.KeyUp) {
        p.position.Y -= geschwindigkeit
    }

    if ebiten.IsKeyPressed(ebiten.KeyLeft) {
        p.position.X -= geschwindigkeit
    }

    if ebiten.IsKeyPressed(ebiten.KeyRight) {
        p.position.X += geschwindigkeit
    }

    p.shootCooldown.Update()

    if p.shootCooldown.IsReady() && ebiten.IsKeyPressed(ebiten.KeySpace) {
        p.shootCooldown.Reset()
        p.lasers = append(p.lasers, util.NewLaser(p.position, p.bild.Bounds()))
    }

    return nil
}
```

Have fun ;)

d.velop

- Laser!!!

```
main.go

func (g *Game) Draw(screen *ebiten.Image) {
    g.player.Draw(screen)

    for _, laser := range g.player.lasers {
        laser.Draw(screen)
    }

    for _, m := range g.meteoriten {
        m.Draw(screen)
    }
}
```

Code:

<https://github.com/jasfeld/bitsAndBurgers/tree/setup>