Московский государственный технический университет им. Н. Э. Баумана

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»
Отчет по проекту

Выполнил:
Студент группы ИУ5-31Б
Баринов Егор

Проверил:
Гапанюк Ю. Е.

2025 г.

Описание

1. Название проекта: Meteor Shooter
2. Описание: Двумерная аркадная игра, разработанная на языке Go. Цель игрока — управлять космическим кораблём, уничтожать падающие метеориты и выживать как можно дольше, набирая очки.
3. Используемые технологии и инструменты:
   Язык программирования: Go (Golang)
   Система управления пакетами: Go Modules (go.mod)
   Система контроля версий: Git
   Хостинг: GitHub
4. Ключевые компоненты и архитектура:
   Игровой цикл: Реализован классический цикл (обработка ввода → обновление состояния → отрисовка).
   Сущности: Корабль игрока, метеориты, снаряды, система частиц для эффектов.
   Логика: Механика столкновений, система очков, повышение сложности со временем.
   Управление: Клавиатура (стрелки/WASD для движения, пробел/ЛКМ для стрельбы).
5. Структура проекта:

```
Project/
├── main.go                 # Точка входа, инициализация, главный цикл
├── go.mod                  # Файл зависимостей Go
├── assets/                 # Ресурсы (спрайты, звуки, шрифты)
├── internal/               # Внутренние пакеты
│   ├── game/               # Логика игры (состояние, правила)
│   ├── entities/           # Классы игровых объектов
│   └── render/             # Логика отрисовки
└── README.md               # Инструкции по сборке и запуску
```

6. Итог: Проект демонстрирует применение принципов объектно-ориентированного и процедурного программирования на Go, работу с графикой, управление состоянием приложения и организацию кода в многофайловой структуре. Весь исходный код доступен в репозитории проекта, подключённом как подмодуль к основному репозиторию с учебными работами.

Листинг программы:

```
package game

import (
    "math"

    "github.com/hajimehoshi/ebiten/v2"

    "github.com/ThreeDotsLabs/meteors/assets"
)

const (
    bulletSpeedPerSecond = 350.0
)

type Bullet struct {
    position Vector
    rotation float64
    sprite   *ebiten.Image
}

func NewBullet(pos Vector, rotation float64) *Bullet {
    sprite := assets.LaserSprite

    bounds := sprite.Bounds()
    halfW := float64(bounds.Dx()) / 2
    halfH := float64(bounds.Dy()) / 2

    pos.X -= halfW
    pos.Y -= halfH

    b := &Bullet{
        position: pos,
        rotation: rotation,
        sprite:   sprite,
    }

    return b
}

func (b *Bullet) Update() {
    speed := bulletSpeedPerSecond / float64(ebiten.TPS())

    b.position.X += math.Sin(b.rotation) * speed
    b.position.Y += math.Cos(b.rotation) * -speed
}

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

    op := &ebiten.DrawImageOptions{}
```

```
    op.GeoM.Translate(-halfW, -halfH)
    op.GeoM.Rotate(b.rotation)
    op.GeoM.Translate(halfW, halfH)

    op.GeoM.Translate(b.position.X, b.position.Y)

    screen.DrawImage(b.sprite, op)
}

func (b *Bullet) Collider() Rect {
    bounds := b.sprite.Bounds()

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

package game

import (
    "fmt"
    "image/color"
    "time"

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

    "github.com/ThreeDotsLabs/meteors/assets"
)

const (
    screenWidth  = 800
    screenHeight = 600

    meteorSpawnTime = 1 * time.Second

    baseMeteorVelocity  = 0.25
    meteorSpeedUpAmount = 0.1
    meteorSpeedUpTime   = 5 * time.Second
)

type Game struct {
    player           *Player
    meteorSpawnTimer *Timer
    meteors          []*Meteor
    bullets          []*Bullet

    score int

    baseVelocity  float64
```

```
    velocityTimer *Timer
}

func NewGame() *Game {
    g := &Game{
        meteorSpawnTimer: NewTimer(meteorSpawnTime),
        baseVelocity:     baseMeteorVelocity,
        velocityTimer:    NewTimer(meteorSpeedUpTime),
    }

    g.player = NewPlayer(g)

    return g
}

func (g *Game) Update() error {
    g.velocityTimer.Update()
    if g.velocityTimer.IsReady() {
        g.velocityTimer.Reset()
        g.baseVelocity += meteorSpeedUpAmount
    }

    g.player.Update()

    g.meteorSpawnTimer.Update()
    if g.meteorSpawnTimer.IsReady() {
        g.meteorSpawnTimer.Reset()

        m := NewMeteor(g.baseVelocity)
        g.meteors = append(g.meteors, m)
    }

    for _, m := range g.meteors {
        m.Update()
    }

    for _, b := range g.bullets {
        b.Update()
    }

    // Check for meteor/bullet collisions
    for i, m := range g.meteors {
        for j, b := range g.bullets {
            if m.Collider().Intersects(b.Collider()) {
                g.meteors = append(g.meteors[:i], g.meteors[i+1:]...)
                g.bullets = append(g.bullets[:j], g.bullets[j+1:]...)
                g.score++
            }
        }
    }

    // Check for meteor/player collisions
    for _, m := range g.meteors {
```

```
        if m.Collider().Intersects(g.player.Collider()) {
            g.Reset()
            break
        }
    }

    return nil
}

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

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

    for _, b := range g.bullets {
        b.Draw(screen)
    }

    text.Draw(screen, fmt.Sprintf("%06d", g.score), assets.ScoreFont, screenWidth/2-100,
50, color.White)
}

func (g *Game) AddBullet(b *Bullet) {
    g.bullets = append(g.bullets, b)
}

func (g *Game) Reset() {
    g.player = NewPlayer(g)
    g.meteors = nil
    g.bullets = nil
    g.score = 0
    g.meteorSpawnTimer.Reset()
    g.baseVelocity = baseMeteorVelocity
    g.velocityTimer.Reset()
}

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

package game

import (
    "math"
    "math/rand"

    "github.com/hajimehoshi/ebiten/v2"

    "github.com/ThreeDotsLabs/meteors/assets"
)
```

```
const (
    rotationSpeedMin = -0.02
    rotationSpeedMax = 0.02
)

type Meteor struct {
    position      Vector
    rotation      float64
    movement      Vector
    rotationSpeed float64
    sprite        *ebiten.Image
}

func NewMeteor(baseVelocity float64) *Meteor {
    target := Vector{
        X: screenWidth / 2,
        Y: screenHeight / 2,
    }

    angle := rand.Float64() * 2 * math.Pi
    r := screenWidth / 2.0

    pos := Vector{
        X: target.X + math.Cos(angle)*r,
        Y: target.Y + math.Sin(angle)*r,
    }

    velocity := baseVelocity + rand.Float64()*1.5

    direction := Vector{
        X: target.X - pos.X,
        Y: target.Y - pos.Y,
    }
    normalizedDirection := direction.Normalize()

    movement := Vector{
        X: normalizedDirection.X * velocity,
        Y: normalizedDirection.Y * velocity,
    }

    sprite := assets.MeteorSprites[rand.Intn(len(assets.MeteorSprites))]

    m := &Meteor{
        position:      pos,
        movement:      movement,
        rotationSpeed: rotationSpeedMin + rand.Float64()*(rotationSpeedMax-
rotationSpeedMin),
        sprite:        sprite,
    }
    return m
}

func (m *Meteor) Update() {
```

```
    m.position.X += m.movement.X
    m.position.Y += m.movement.Y
    m.rotation += m.rotationSpeed
}

func (m *Meteor) Draw(screen *ebiten.Image) {
    bounds := m.sprite.Bounds()
    halfW := float64(bounds.Dx()) / 2
    halfH := float64(bounds.Dy()) / 2

    op := &ebiten.DrawImageOptions{}
    op.GeoM.Translate(-halfW, -halfH)
    op.GeoM.Rotate(m.rotation)
    op.GeoM.Translate(halfW, halfH)

    op.GeoM.Translate(m.position.X, m.position.Y)

    screen.DrawImage(m.sprite, op)
}

func (m *Meteor) Collider() Rect {
    bounds := m.sprite.Bounds()

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

```
package game

import (
    "math"
    "time"

    "github.com/hajimehoshi/ebiten/v2"

    "github.com/ThreeDotsLabs/meteors/assets"
)

const (
    shootCooldown     = time.Millisecond * 500
    rotationPerSecond = math.Pi

    bulletSpawnOffset = 50.0
)

type Player struct {
    game *Game

    position Vector
```

```
    rotation float64
    sprite   *ebiten.Image

    shootCooldown *Timer
}

func NewPlayer(game *Game) *Player {
    sprite := assets.PlayerSprite

    bounds := sprite.Bounds()
    halfW := float64(bounds.Dx()) / 2
    halfH := float64(bounds.Dy()) / 2

    pos := Vector{
        X: screenWidth/2 - halfW,
        Y: screenHeight/2 - halfH,
    }

    return &Player{
        game:          game,
        position:      pos,
        rotation:      0,
        sprite:        sprite,
        shootCooldown: NewTimer(shootCooldown),
    }
}

func (p *Player) Update() {
    speed := rotationPerSecond / float64(ebiten.TPS())

    if ebiten.IsKeyPressed(ebiten.KeyLeft) {
        p.rotation -= speed
    }
    if ebiten.IsKeyPressed(ebiten.KeyRight) {
        p.rotation += speed
    }

    p.shootCooldown.Update()
    if p.shootCooldown.IsReady() && ebiten.IsKeyPressed(ebiten.KeySpace) {
        p.shootCooldown.Reset()

        bounds := p.sprite.Bounds()
        halfW := float64(bounds.Dx()) / 2
        halfH := float64(bounds.Dy()) / 2

        spawnPos := Vector{
            p.position.X + halfW + math.Sin(p.rotation)*bulletSpawnOffset,
            p.position.Y + halfH + math.Cos(p.rotation)*-bulletSpawnOffset,
        }

        bullet := NewBullet(spawnPos, p.rotation)
        p.game.AddBullet(bullet)
    }
```

```
}

func (p *Player) Draw(screen *ebiten.Image) {
    bounds := p.sprite.Bounds()
    halfW := float64(bounds.Dx()) / 2
    halfH := float64(bounds.Dy()) / 2

    op := &ebiten.DrawImageOptions{}
    op.GeoM.Translate(-halfW, -halfH)
    op.GeoM.Rotate(p.rotation)
    op.GeoM.Translate(halfW, halfH)

    op.GeoM.Translate(p.position.X, p.position.Y)

    screen.DrawImage(p.sprite, op)
}

func (p *Player) Collider() Rect {
    bounds := p.sprite.Bounds()

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

```
package game

type Rect struct {
    X      float64
    Y      float64
    Width  float64
    Height float64
}

func NewRect(x, y, width, height float64) Rect {
    return Rect{
        X:      x,
        Y:      y,
        Width:  width,
        Height: height,
    }
}

func (r Rect) MaxX() float64 {
    return r.X + r.Width
}

func (r Rect) MaxY() float64 {
    return r.Y + r.Height
}
```

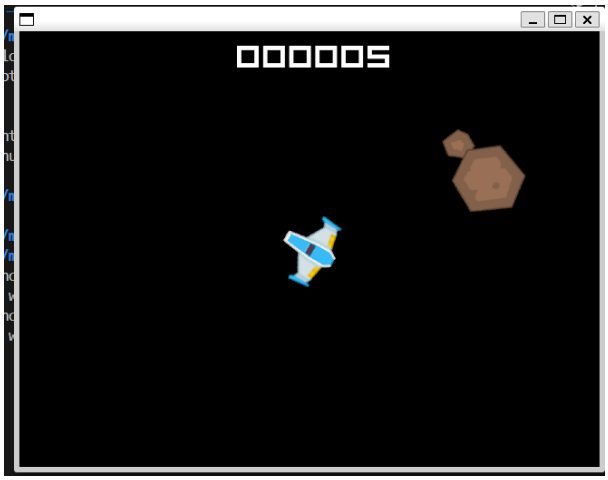
```go
func (r Rect) Intersects(other Rect) bool {
    return r.X <= other.MaxX() &&
        other.X <= r.MaxX() &&
        r.Y <= other.MaxY() &&
        other.Y <= r.MaxY()
}

package game

import (
    "time"

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

type Timer struct {
    currentTicks int
    targetTicks  int
}

func NewTimer(d time.Duration) *Timer {
    return &Timer{
        currentTicks: 0,
        targetTicks:  int(d.Milliseconds()) * ebiten.TPS() / 1000,
    }
}

func (t *Timer) Update() {
    if t.currentTicks < t.targetTicks {
        t.currentTicks++
    }
}

func (t *Timer) IsReady() bool {
    return t.currentTicks >= t.targetTicks
}

func (t *Timer) Reset() {
    t.currentTicks = 0
}

package game

import "math"

type Vector struct {
    X float64
    Y float64
}

func (v Vector) Normalize() Vector {
    magnitude := math.Sqrt(v.X*v.X + v.Y*v.Y)
```

```
    return Vector{v.X / magnitude, v.Y / magnitude}
}

package assets

import (
    "embed"
    "image"
    _ "image/png"
    "io/fs"

    "github.com/hajimehoshi/ebiten/v2"
    "golang.org/x/image/font"
    "golang.org/x/image/font/opentype"
)

//go:embed *
var assets embed.FS

var PlayerSprite = mustLoadImage("player.png")
var MeteorSprites = mustLoadImages("meteors/*.png")
var LaserSprite = mustLoadImage("laser.png")
var ScoreFont = mustLoadFont("font.ttf")

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)
}

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
}
```

```go
func mustLoadFont(name string) font.Face {
    f, err := assets.ReadFile(name)
    if err != nil {
        panic(err)
    }

    tt, err := opentype.Parse(f)
    if err != nil {
        panic(err)
    }

    face, err := opentype.NewFace(tt, &opentype.FaceOptions{
        Size:    48,
        DPI:     72,
        Hinting: font.HintingVertical,
    })
    if err != nil {
        panic(err)
    }

    return face
}
```

```go
package main

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

    "github.com/ThreeDotsLabs/meteors/game"
)

func main() {
    g := game.NewGame()

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

Результат выполнения:


000005