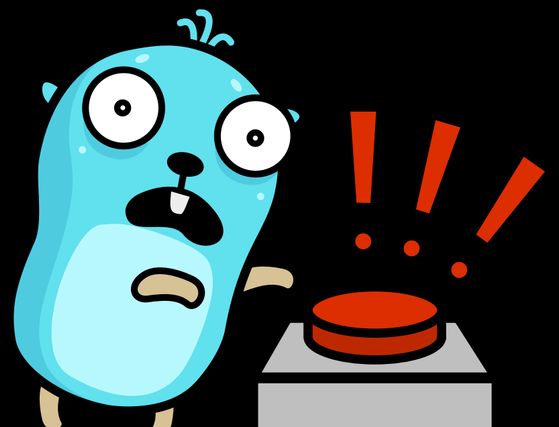
SANDGOH

SOMMAIRE

PRÉSENTATION PROJET

JEU SANDBOX "BAC À SABLE"
DÉVELOPPÉ EN GO
OBJECTIF : UN MAXIMUM DE FPS

DÉMONSTRATION



MICROS OPTIMISATIONS

RÉDUCTION DES "OBJETS" QUI COMPOSENT LA GRILLE DE JEU

```
type Cell struct {  13 usages  Paul Coignac *
    physic   func(x int, y int, g *Game)
    cellType CellType
    color    color.Color
    liquid   bool
    density  int
}
```

->

```
Implement interface
type Cell struct {  20 usages  Paul Coignac
    cellType CellType
    color    color.Color
    isActive bool
}

Implement interface
type CellData struct {  2 usages  Paul Coignac
    physic  func(x int, y int, g *Game)
    liquid  bool
    density int
}
```

RÉUTILISATION D'UN MAXIMUM D'"OBJETS" CRÉÉS

```go
for y := 0; y < gridSize; y++ {
    for x := 0; x < gridSize; x++ {
        cell := g.grid[y][x]
        if cell.cellType != Air {
            rect := ebiten.NewImage(cellSize, cellSize)
            rect.Fill(cell.color)
            op := &ebiten.DrawImageOptions{}
            op.GeoM.Translate(float64(x*cellSize), float64(y*cellSize))
            screen.DrawImage(rect, op)
        }
    }
}
```

->

```go
op := &ebiten.DrawImageOptions{}
for col, rects := range rectanglesByColor {
    for _, rectangle := range rects {
        rect := getRectImageByWidth(rectangle.w)
        rect.Fill(col)
        op.GeoM.Reset()
        op.GeoM.Translate(float64(rectangle.x*cellSize), float64(rectangle.y*cellSize))
        screenBufferImg.DrawImage(rect, op)
    }
}
```

```go
func NewWaterCell() Cell {  2 usages  Paul Coignac +2
    colors := []color.Color{...}
    return Cell{
        physic: func(x int, y int, g *Game) {
            if y+1 < gridSize {...}
        },
        cellType: Water,
        liquid:   true,
        density:  9,
        color:    colors[rand.Intn(len(colors))],
    }
}
```

->

```go
CellsTypes = map[CellType]CellData{
    Sand: {...},
    Water: {
        physic:  WaterPhysic,
        liquid:  true,
        density: 9,
    },
    Air: {...},
    Metal: {...},
    BlackHole: {...},
    WaterGenerator: {...},
}
```

MÉMOISATION

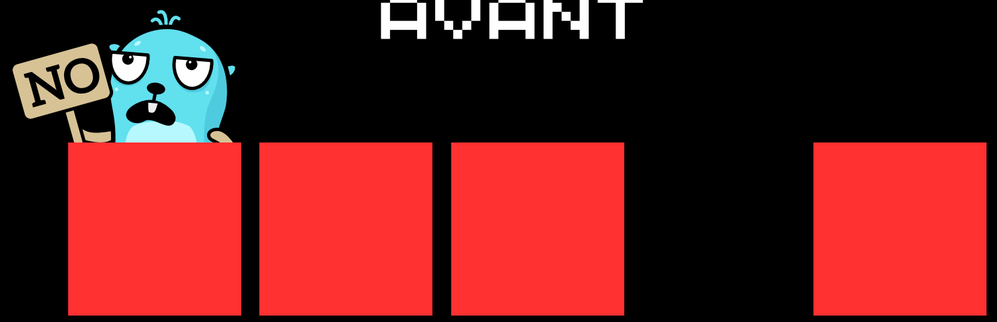
```
var cachedRects = make([]*ebiten.Image, gridSize)  3 usag

func getRectImageByWidth(width int) *ebiten.Image {  1 us
    index := width - 1
    if cachedRects[index] != nil {
        return cachedRects[index]
    }
    rect := ebiten.NewImage(width*cellSize, cellSize)
    cachedRects[index] = rect
    return rect
}
```

MACROS
OPTIMISATIONS

REDUCTION DU NOMBRE DE FORMES À DESSINER

AVANT



4 FORMES

APRÈS

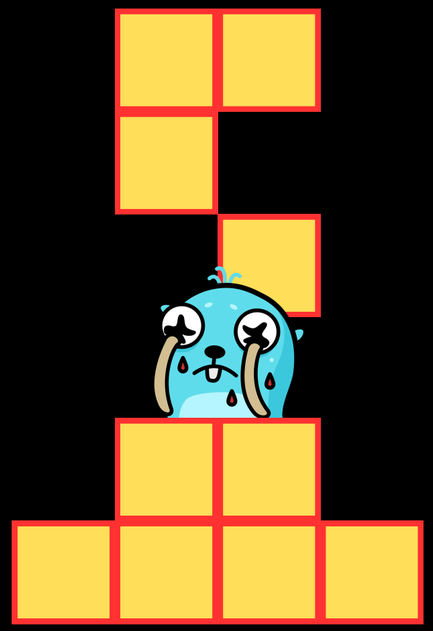
2 FORMES

REDUCTION DU NOMBRE DE FORMES À DESSINER

```
func drawCells(g *Game, screen *ebiten.Image) {  2 usages
    updatedCellsByColor := groupUpdatedCellsByColor(g)
    rectanglesByColor := groupRectanglesHorizontallyByColor(updatedCellsByColor)
    drawRectangles(rectanglesByColor, screen)
}
```

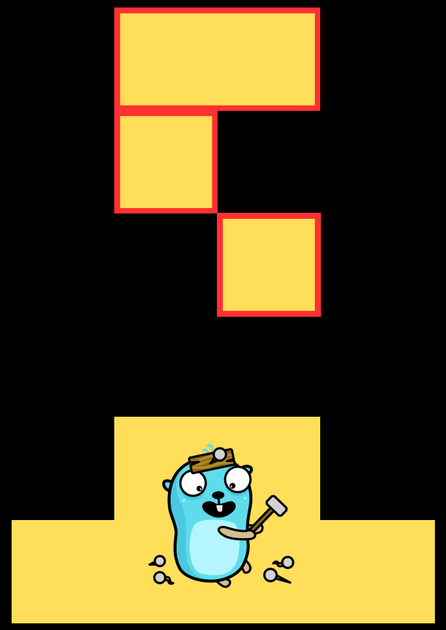
AJOUT DE LA NOTION DE CELLULES ACTIVES

AVANT



10 FORMES

APRÈS

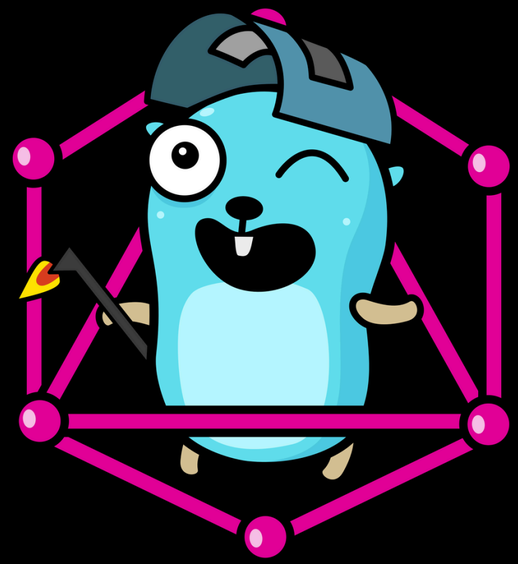


3 FORMES

AJOUT DE LA NOTION DE CELLULES ACTIVES

```go
func switchPlace(Ax int, Ay int, Bx int, By int, g *Game) {
    cellA := g.grid[Ay][Ax]
    cellB := g.grid[By][Bx]
    cellA.isActive = true
    cellB.isActive = true
    g.grid[By][Bx] = cellA
    g.grid[Ay][Ax] = cellB
}
```

```go
op := &ebiten.DrawImageOptions{}
for col, rects := range rectanglesByColor {
    for _, rectangle := range rects {
        rect := getRectImageByWidth(rectangle.w)
        rect.Fill(col)
        op.GeoM.Reset()
        op.GeoM.Translate(float64(rectangle.x*cellSize), float64(rectangle.y*cellSize))
        screenBufferImg.DrawImage(rect, op)
    }
}
```



BENCHMARK

DANS LA FONCTION MAIN:

```
func main() {
    initFlags()
    initWindow()
    initCellsTypes()
    game := getGame()
```

BENCHMARK

DANS LA FONCTION DE
RECUPÉRATION DES FLAGS

```go
func initFlags() {  1 usage
    benchmarkModeUnparsed := flag.Bool( name: "benchmark",  value: false,  usage: "benchmark mode")
    flag.Parse()
    benchmarkMode = *benchmarkModeUnparsed
}
```

./MAIN.GO -BENCHMARK TRUE

BENCHMARK

DANS LA FONCTION D'INITIALISATION DE LA GRILLE

```
if benchmarkMode && y < 10 {
    grid[y][x] = NewSandCell()
} else if benchmarkMode && y > 80 && x > 40 && x < 60 {
    grid[y][x] = NewWaterCell()
} else if benchmarkMode && y == 50 && x > 20 && x < 40 {
    grid[y][x] = NewMetalCell()
} else if benchmarkMode && y == 50 && x > 60 && x < 95 {
    grid[y][x] = NewBlackHoleCell()
} else if benchmarkMode && y == 30 && x > 74 && x < 78 {
    grid[y][x] = NewWaterGeneratorCell()
} else {
    grid[y][x] = NewAirCell()
}
```

BENCHMARK

PUIS DANS LA FONCTION UPDATE

```
if benchmarkMode {
    benchmarkCheck()
}
```

```
func benchmarkCheck() {  1 usage
    countUpdate++
    if countUpdate >= 100 {
        os.Exit( code: 0)
    }
}
```

BENCHMARK

RÉSULTATS

```
Benchmark 1: ./main -benchmark true
  Time (mean ± σ):      8.353 s ±  0.507 s    [User: 8.425 s, System: 1.914 s]
  Range (min … max):    7.932 s …  9.701 s    10 runs

Benchmark 2: ../v1/main -benchmark true
  Time (mean ± σ):     31.072 s ±  0.663 s    [User: 48.415 s, System: 3.889 s]
  Range (min … max):   30.405 s … 32.160 s    10 runs

Summary
  ./main -benchmark true ran
    3.72 ± 0.24 times faster than ../v1/main -benchmark true
```

CONCLUSION

GAIN CONSÉQUENTS À LA FOIS LORSQUE DES PIXELS SONT EN MOUVEMENT ET LORSQU'ILS SONT FIGÉS

UTILISATION DE SOLUTIONS TEL QUE LA MÉMOISATION

AXES D'AMÉLIORATIONS FUTURS :
UTILISATION DU THREADING