

# *Snake In The dark*

DEMO FINAL

FPS: 60

*PARA COMECAR O JOGO  
TECLE ENTER*

[O JOGO]

# História

Todos sabemos o que aconteceu naquele dia. Milhões de pessoas... sonhos... Explosões...

Agora você, destruído, no escuro, com três balas e vários drones te caçando, deve impedir que isso aconteça novamente.

**Local:** Profundezas do oceano - submarino terrorista - sala de lançamento automático do armamento nuclear, tamanho 21m x 21m.

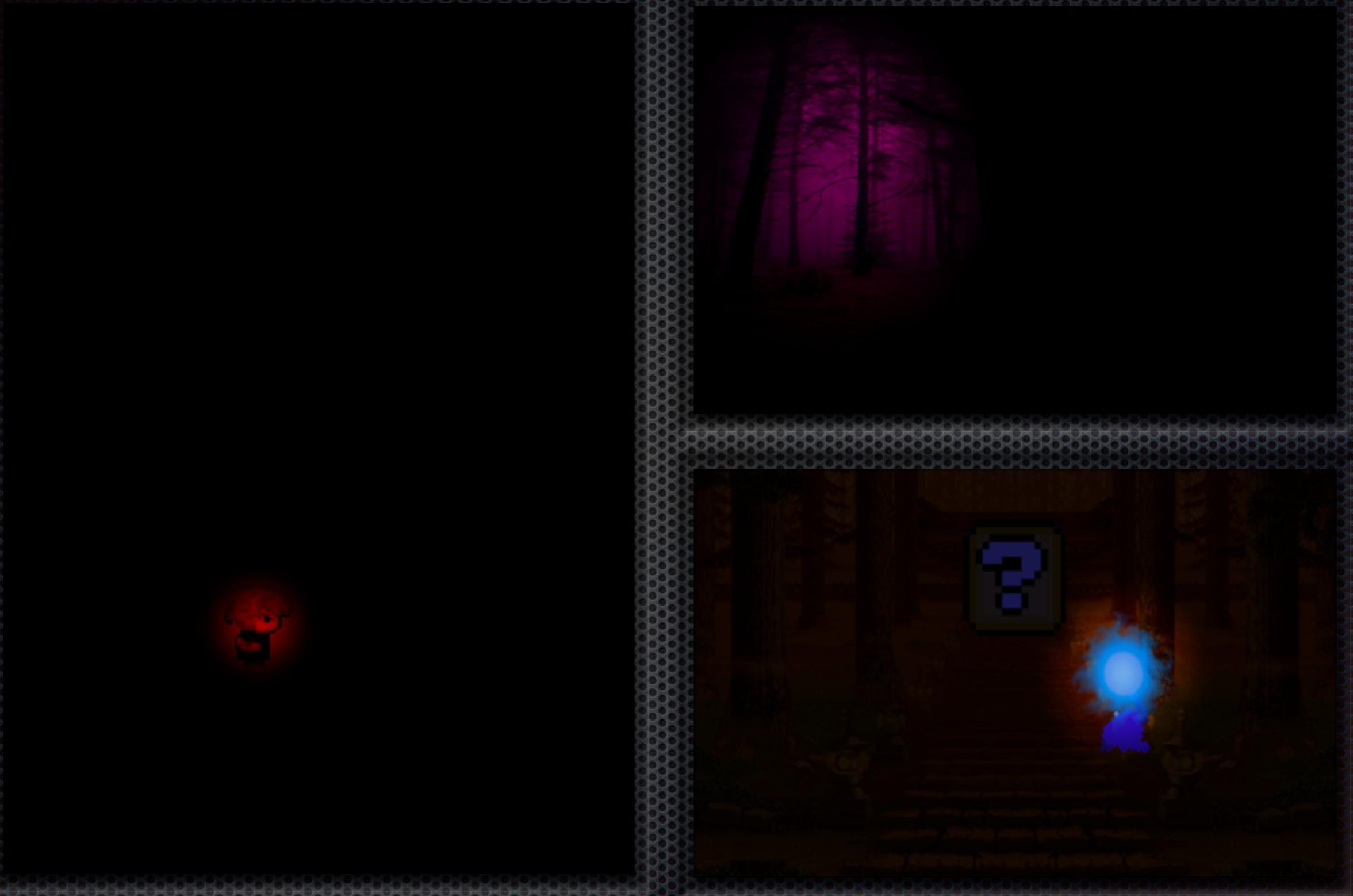
**Seu inimigo:** Hordas de drones armados, sincronizados e mortais, que atravessam a sala em linha reta e velocidade constante, saindo de todas as paredes da sala.

**Seu equipamento... restante:** Três balas de choque, sensor que detecta a presença de drones a 1m de distância

**Sua missão:** impedir o segundo lançamento de A.N.T.A (Aerial NonTripulated Armageddon). Rápido!

Você cai no centro da sala.

O computador central est à 10 metros norte e 7 metros leste. Lembrese que zzzzO zz Drzzoneszzz voz na sua zzzvelozzzdade. transmissão caiu.



# Organização da Apresentação

- Componentes:
  - Engine e Visuals
  - Mapa
  - Player e Drones
  - Equipamentos

# Design Patterns Usados

- Abstract factory
- Composite
- Observer
- Singleton

**DEMO**

[a **ENGINE** e **VISUALS**]

# Funcionamento

- Objetivo: controle de performance, recursos, transição de telas e interatividade.
- Base libGDX: manteve, em sua essência, o padrão de versatilidade e eficiência

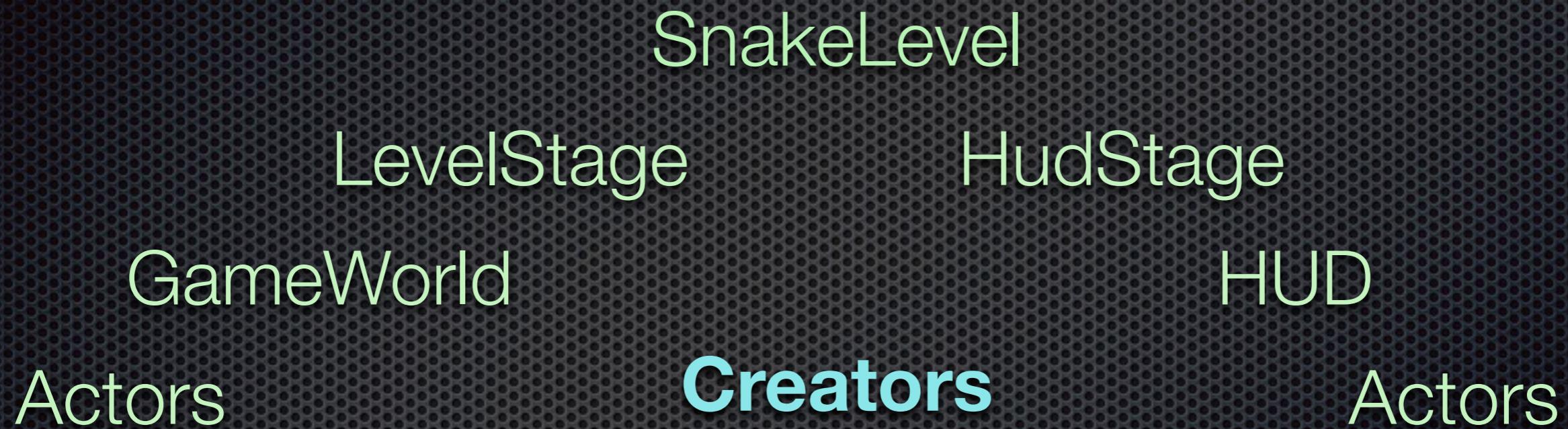


# Funcionamento

- Executados em diversas plataformas:
  - sistemas **Android** e **iOS**,
  - desktop **Windows**, **Linux** e **OSX**,
  - além de portabilidade para **web**, com base no Google GWT



# Funcionamento



# Estrutura

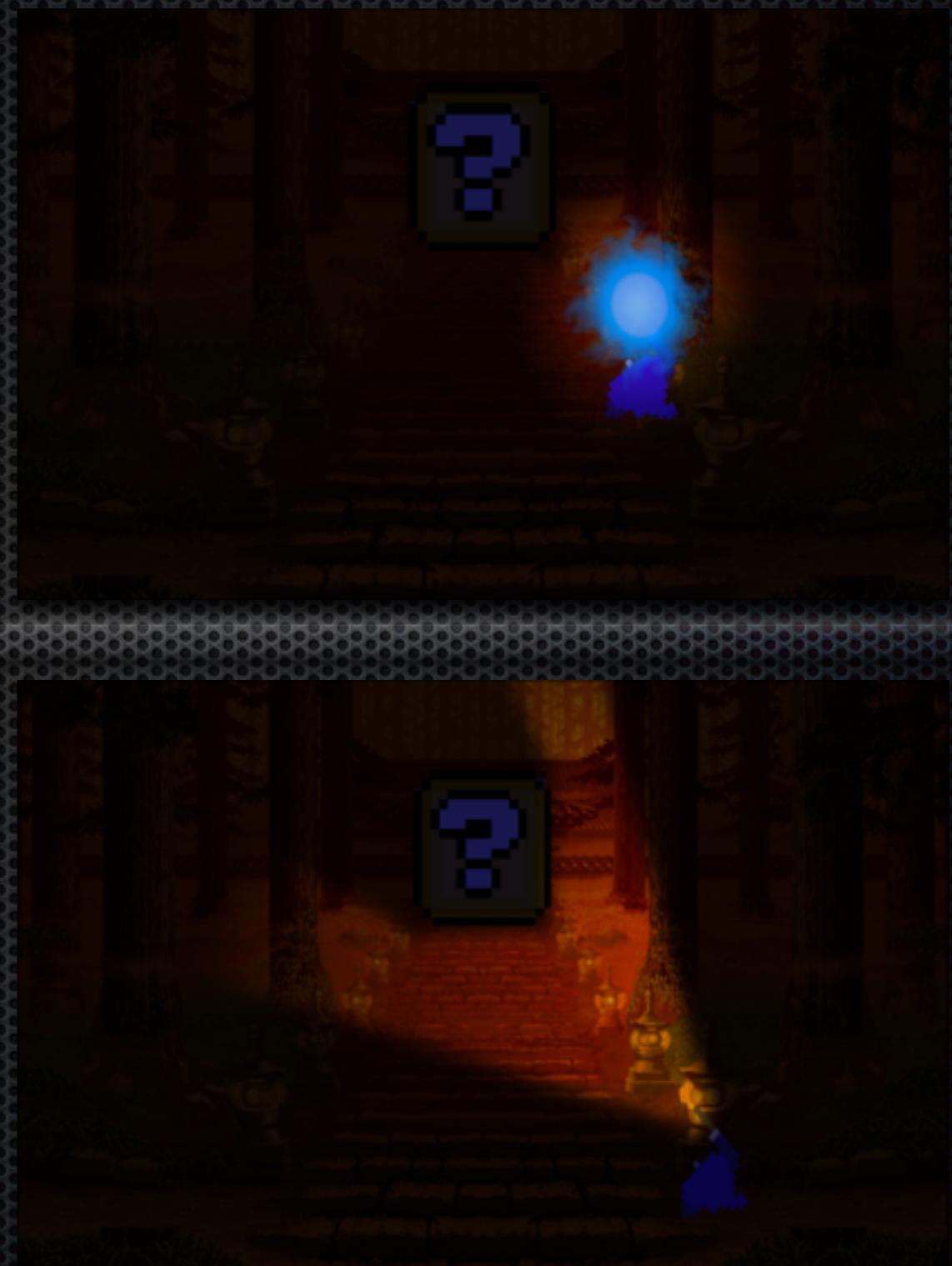
- Plano de acesso em que o “mundo”
- Extende a classe abstrata **GameWorld** (Factory) para inserção no loop principal do jogo (**GameLevel**).
- Possibilidade de inserir cutscenes implementando a **interface Cutscene**
- Menu de pausa com a **interface PauseMenu**

Códigos...

# Visuals

- Ferramentas avançadas de controle gráfico do jogo
- Projeções e controle de camera
- um sistema dinâmico de Luz e Sombra.





Códigos...

**DEMO**

# [Abstract Factory]

- É utilizada na **package Creators**, de modo a tornar as Screens independentes do tipo de mundo (top-down, plataforma, 3D), Stages e HUDs utilizados.

```
// Creates GameWorld  
  
world = WorldCreator.createWorld(levelType, levelDataID);  
  
// Creates HUD  
  
hud = HUDCreator.createHUD(levelType, levelDataID);
```

# [Observer]

- Serve para separar e determinados eventos dos objetos  
*(objetos que assinam outros objetos e os assinantes são informados quando um evento ocorre, de modo que ele pode ou não tomar uma ação para aquele evento)*
- Muito usado na interface gráfica, por exemplo, quando o usuário interage no jogo.
- Relacionado ao **Stage** - com sistema de inscrições.

```
// Adds world and HUD to the stages
stageWorld.addActor(world);
stageHUD.addActor(hud);
```

# [Composite]

- Apresenta-se como uma feature suportada pelo componente
- Temos na interface do jogo uma hierarquia recursiva de objetos que criar o mundo e as screens. (já explicado)
- Atores inscritos no **Stage** são compostos de outros atores (a exemplo de um mapa que contém inimigos), possibilitando a criação de formas complexas e mundos extensos facilmente.

# [Composite]

- Componentes menores para adicionar características ao mundo - totalmente independentes e seguindo a ideia de transparência
- exemplo: **Visuals** são projeções e controle da camera

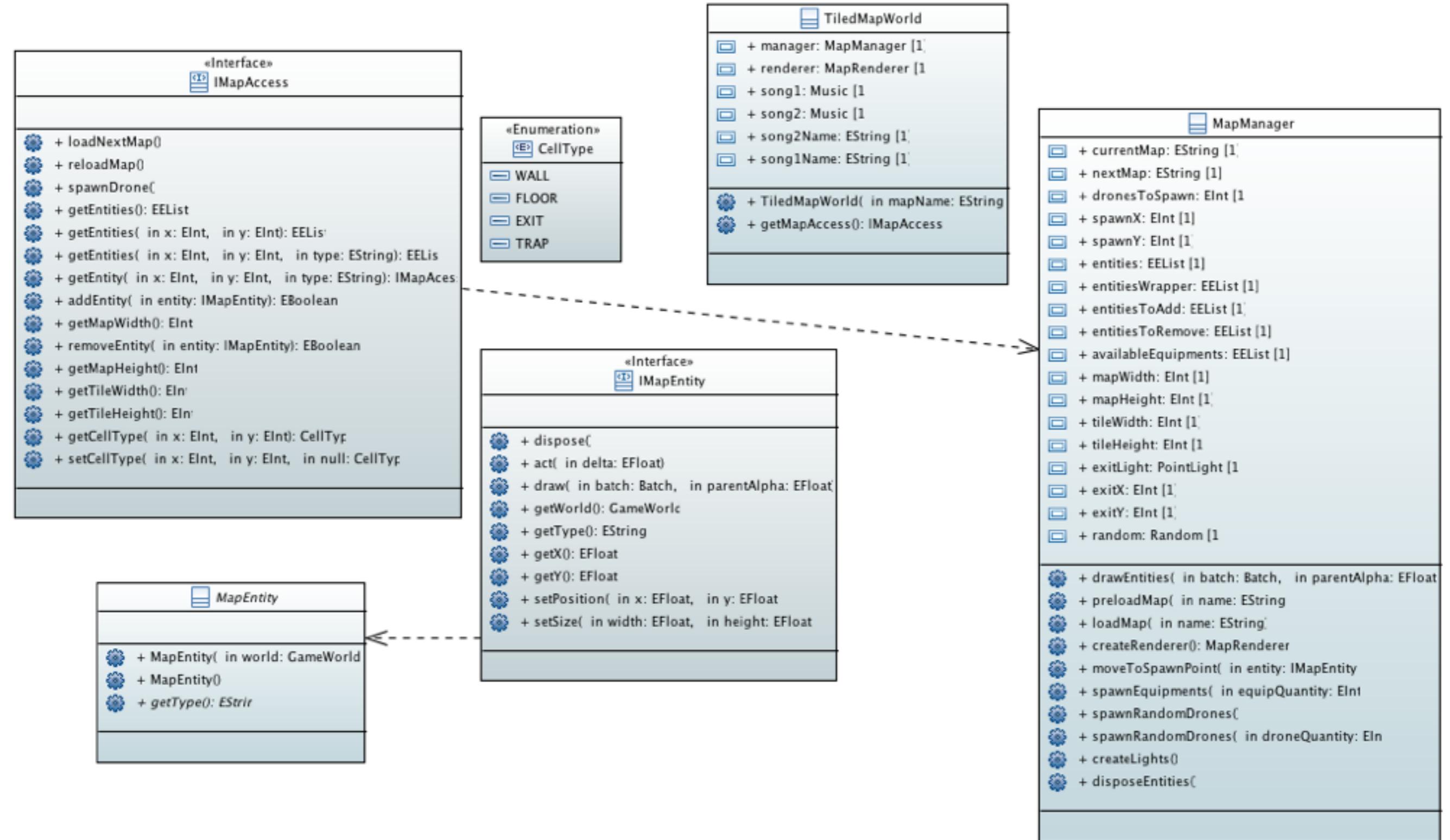
[o MAPA]

# Funcionamento

- Projetado para realizar toda a parte de armazenamento físico
- Representação lógica dos níveis do jogo
- tiles = geometria de níveis - obstáculos ou entidades (itens, inimigos etc.)

# Funcionamento

- Responsável por manter a lista de entidades vivas no mapa
- Checar os movimentos realizados pelas mesmas.



Códigos...

# [Abstract Factory]

- Especificamente na **package Creators**, que é uma factory de GameWorlds, HUDs e as próprias Screens.

[o Player e Drones]

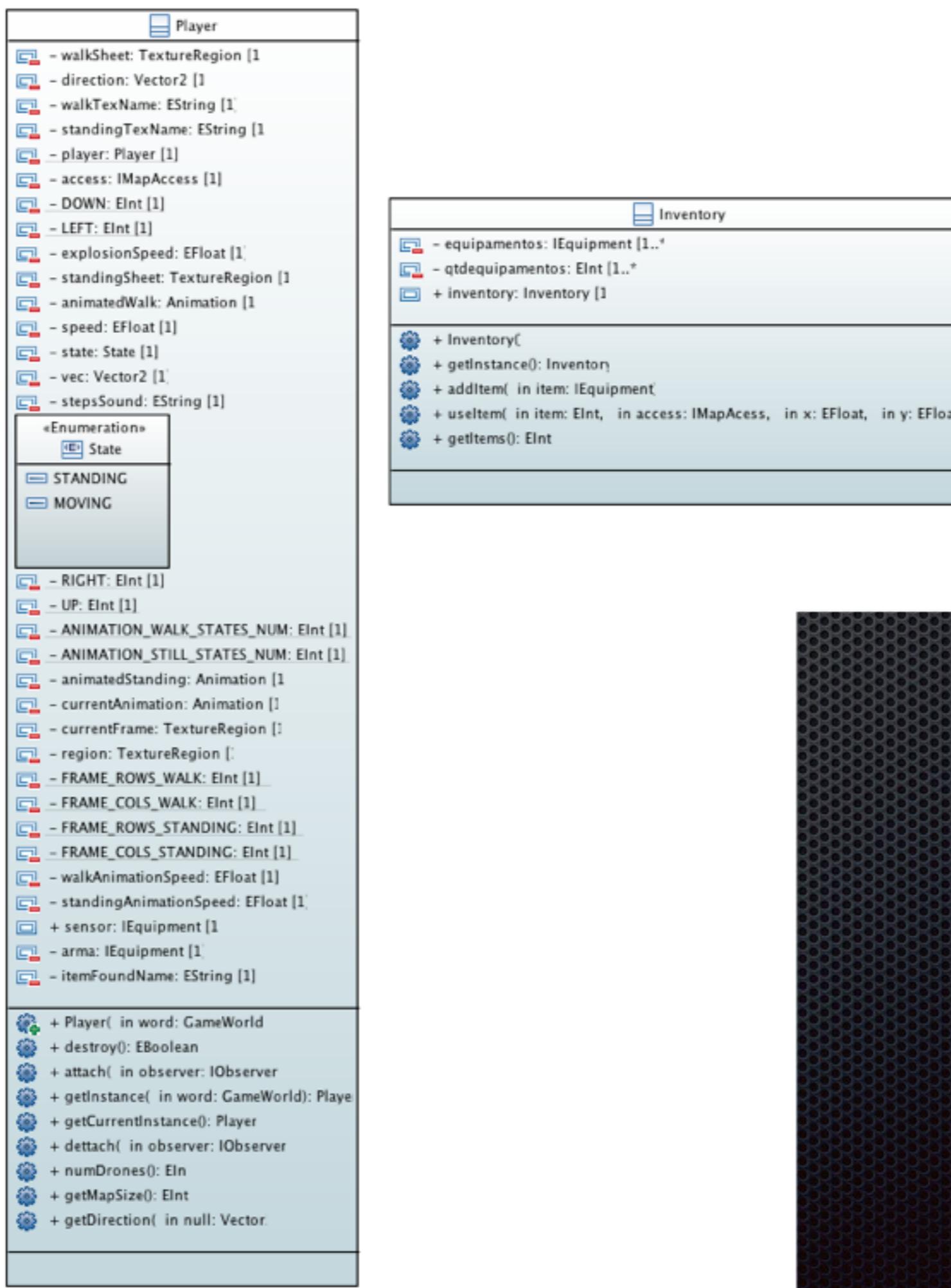
# Funcionamento

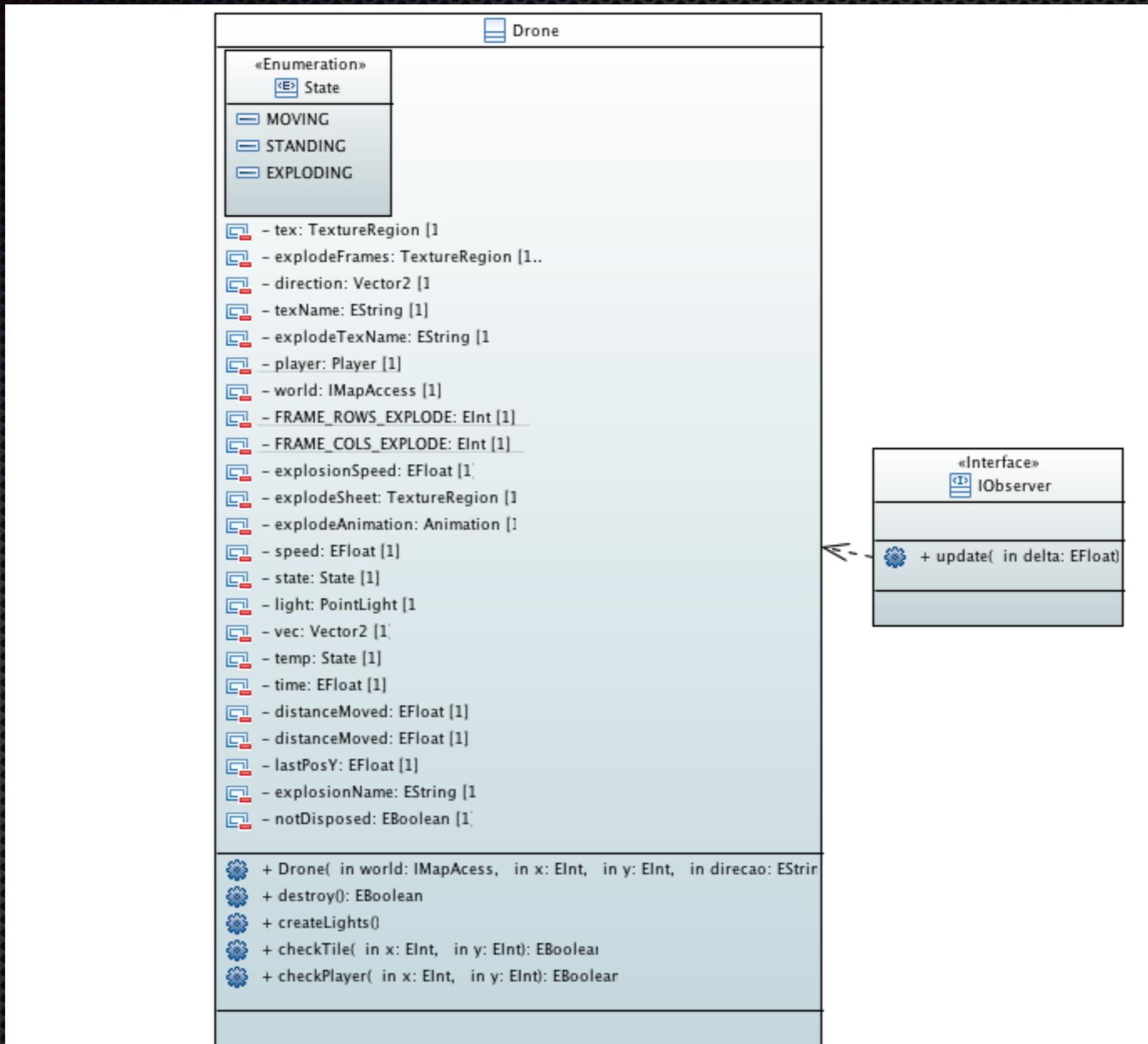
- **PLAYER:** Encarrega de receber o input do usuário e controlar o player
- **DRONE:** Só é capaz de andar para a frente uma vez por turno e ao se chocar com a parede ou com uma trap ele explode.



# Funcionamento

- act - atualização contínua do Player
- Get instance - pattern singleton
- update - pattern observer
- O Drone - usa o pattern observer e assina a engine para receber o update.





# [Singleton]

- O PLAYER tem a função guardar e executar tudo aquilo que tem relação com o player. (**Get instance**)
- Utilizado para garantir que uma classe tenha uma única instância durante toda a execução do programa, mantendo um ponto global de acesso ao seu objeto.
- O número de instâncias do Player deve ser limitado a apenas uma, pois o jogo não possui modo multiplayer.

# [Observer]

- O DRONE assina o Player para receber o **update** (já explicado)
- A classe só tem sua posição e direção em que se move e o método que faz isso.

```
//Singleton
static public Inventory getInstance(){
    if (inventory==null)
        inventory = new Inventory();
    return inventory;
}

public void addItem(IEquipment item){

    switch(item.getName().toLowerCase()){

        case "flashlight":
            if(equipamentos[0]==null)
                equipamentos[0] = item;
            qtdequipamentos[0]++;
            break;

        case "emp":
            if(equipamentos[1]==null)
                equipamentos[1] = item;
            qtdequipamentos[1]++;
            break;

        case "trap":
            if(equipamentos[2]==null)
                equipamentos[2] = item;
            qtdequipamentos[2]++;
            break;
    }
}
```

```
public void attach(IObserver observer){  
    observersToAdd.add(observer);  
}  
  
public void detach (IObserver observer) {  
    observersToRemove.remove(observer);  
}  
  
private void update(float delta){  
    observers.removeAll(observersToRemove);  
    observers.addAll(observersToAdd);  
    observersToRemove.clear();  
    observersToAdd.clear();  
  
    for (IObserver observer : observers)  
        observer.update(delta);  
}
```

```
static public Player getInstance(GameWorld world){  
    if(player == null || player.getWorld() != world) {  
        player = new Player(world);  
    }  
    return player;  
}  
  
static public Player getCurrentInstance(){  
    if(player!=null)  
        return player;  
    return null;  
}
```

```
public void useItem(int item, IMapAccess access, float x, float y){  
    if(qtdequipamentos[item]>0){  
        equipamentos[item].activateOnMap(access);  
        qtdequipamentos[item]--;  
        if(qtdequipamentos[item]>0){  
            switch(item){  
                case 0:  
                    equipamentos[item] = EquipmentCreator.createFactory("flashlight").create(x, y, false, access);  
                    break;  
                case 1:  
                    equipamentos[item] = EquipmentCreator.createFactory("emp").create(x, y, false, access);  
                    break;  
                case 2:  
                    equipamentos[item] = EquipmentCreator.createFactory("trap").create(x, y, false, access);  
                    break;  
            }  
        }  
    }  
}
```

```
IMapEntity entity = access.getEntity((int)getX(), (int)getY(), "equipment");
if (entity != null) {
    itemFound.play(.3f);
    IEquipment equipment = (IEquipment) entity;
    access.removeEntity(equipment); //Retira do mundo
    equipment.setPosition(.5f, .5f);
    equipment.setOnMap(false);
    equipment.onPickup(this);
    addActor((Actor) equipment); //Adiciona ao player
    inventory.addItem(equipment);
}

if (state == State.STANDING) {

    if (Gdx.input.isKeyPressed(Input.Keys.LEFT) && !CellType.WALL.equals(access.getCellType((int)getX() - 1, (int)getY()))){
        distanceMoved = 0;
        direction.set(-speed, 0);
        currentAnimation = animatedWalk[LEFT];
        lastPosX = getX(); lastPosY = getY();
        state = State.MOVING;
        stateTime = 0;
        update(delta);
        steps.play(.4f);
    }
}
```

```
public void update(float delta){

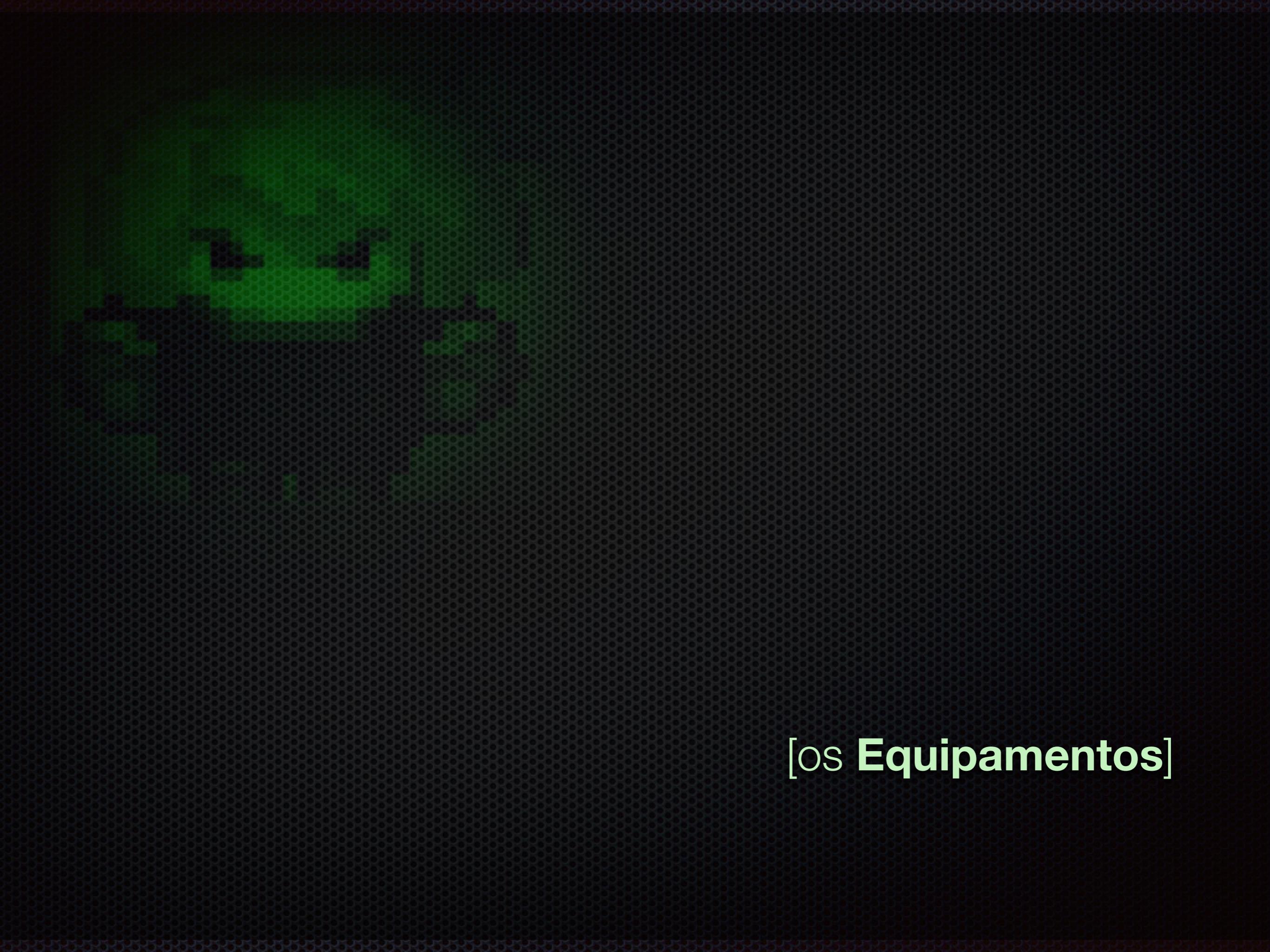
    if(direction.x < 0) {
        if (checkTile((int) getX() - 1, (int) getY()) && state != State.EXPLODING ) {
            state = State.EXPLODING;
            time = 0;
        }
        else {

            if (state != State.EXPLODING && checkPlayer((int)getX(), (int)getY())){
                state = State.MOVING;
                lastPosX = getX();
                lastPosY = getY();
                distanceMoved = 0;
            }
        }
    }
}
```

```
public boolean checkTile(int x, int y){
    if ((CellType.WALL.equals(world.getCellType(x, y))
        || (CellType.TRAP.equals(world.getCellType(x, y)))))

        return true;
    else
        return false;
}

//confere se ha um player na posicao x y
//Se houver explode o drone
public boolean checkPlayer(int x, int y){
    IMapEntity entity = world.getEntity((int)getX(), (int)getY(), "player");
    if (entity != null){
        if (((Player) entity).destroy()){
            world.reloadMap();
            try {
                ScreenCreator.addAndGo("SnakeScreen", "gameOver", "");
            }
            catch (Exception e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        else {
            this.destroy();
        }
        return false;
    }
    else return true;
}
```



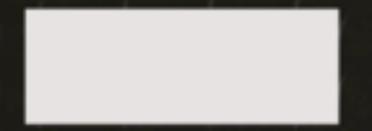
[os **Equipamentos**]

# Funcionamento

- Oferece itens que se espalham pelo mapa e podem ser coletados e utilizados pelo jogador.
- Cada item (ou equipamento) provoca diferentes efeitos sobre o mapa
- Equipments é um componente bastante versátil, podendo produzir quaisquer tipos de objetos de natureza semelhante.

# Estrutura

- Os equipamentos serão utilizados a partir de um **método activate**, que receberá como parâmetro **IMapAccess**, para poder realizar as operações desejadas no mapa



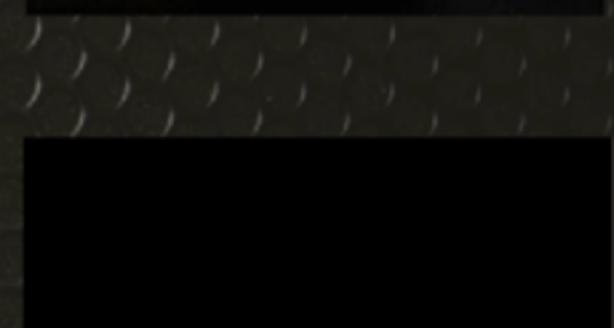
x 1



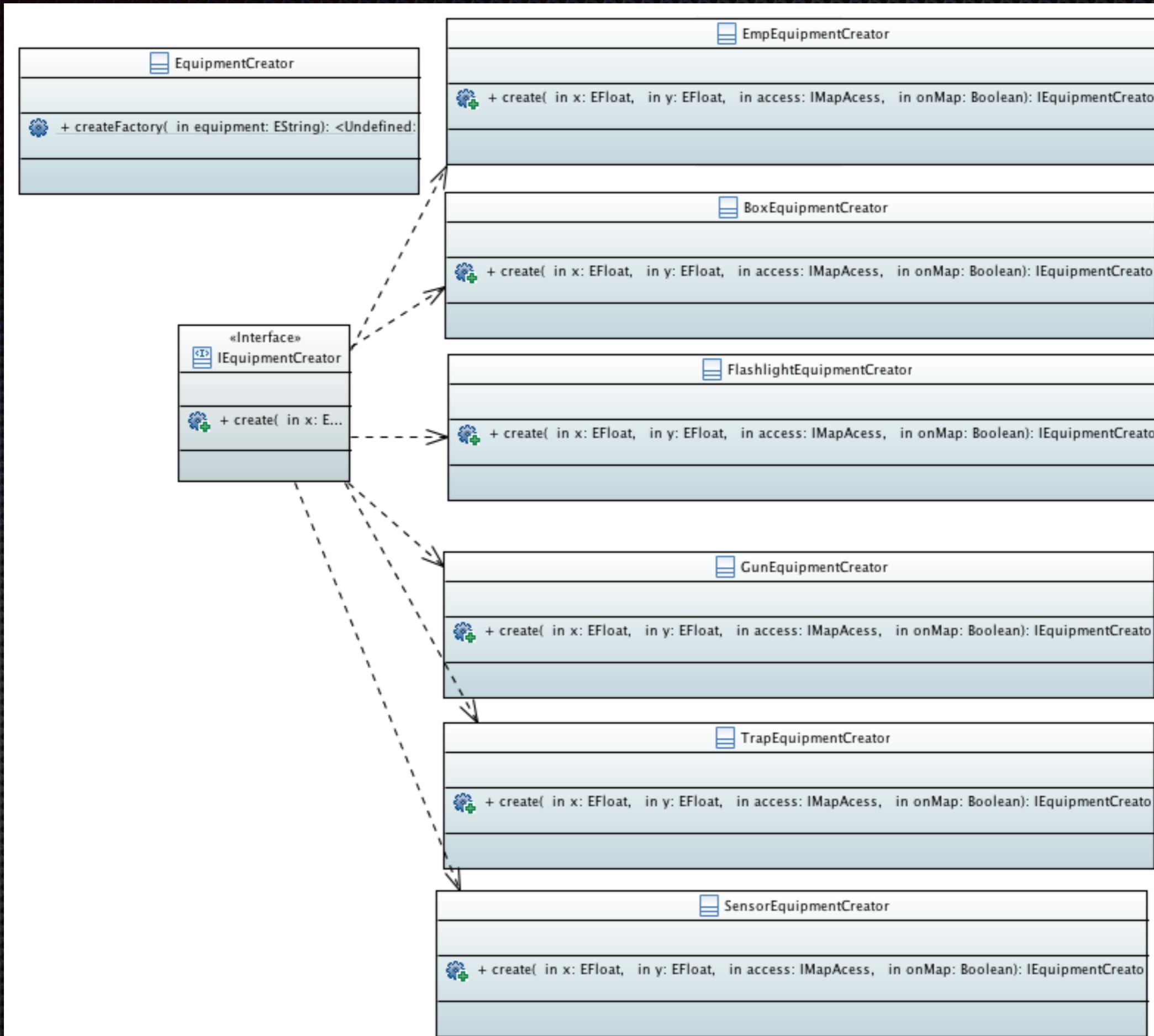
x 1

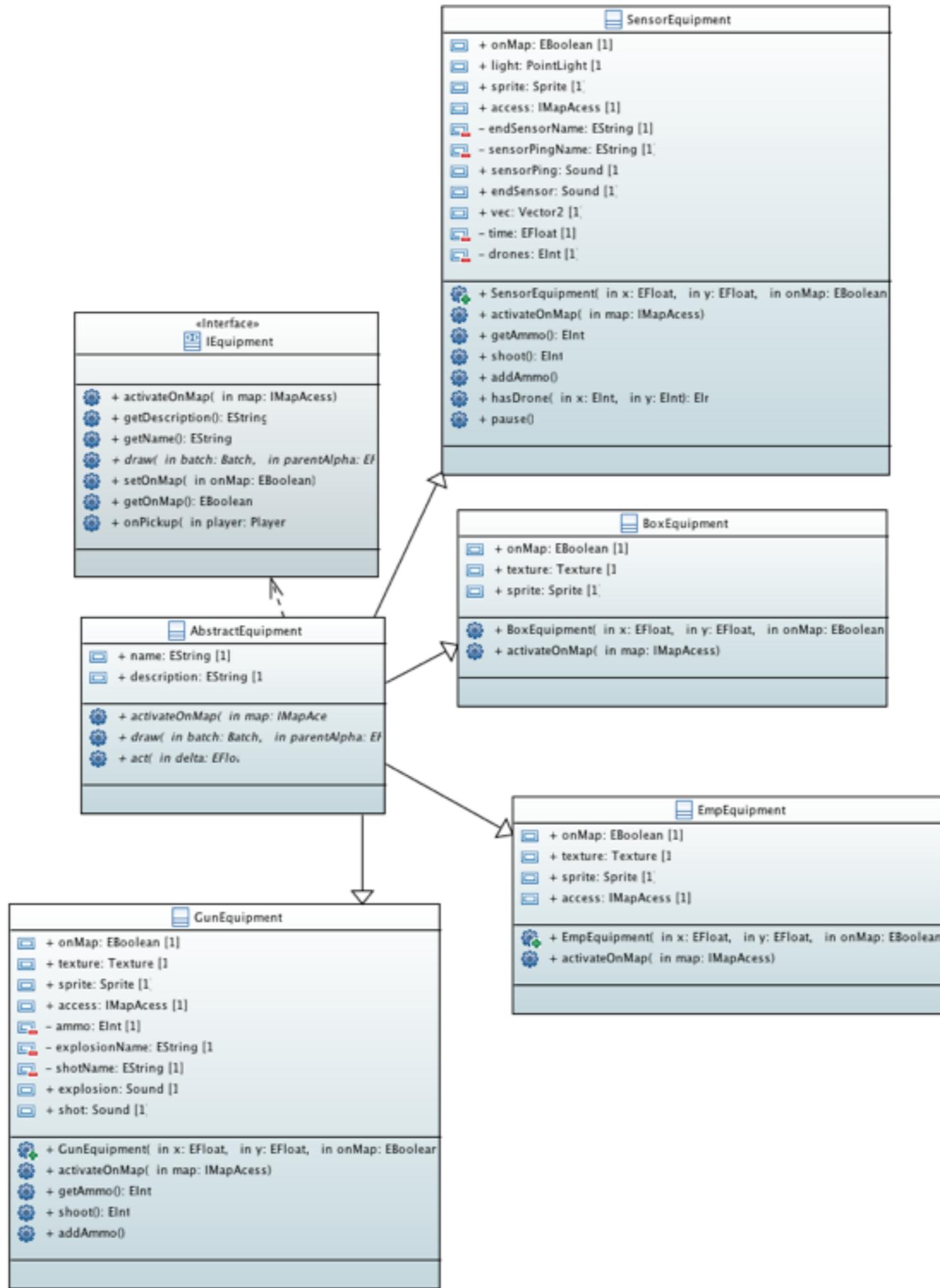


x 2









Códigos...

# [Abstract Factory]

- Criação de objetos diferentes, porém relacionados por suas funções e características dentro do jogo.
- Criar objetos de forma dinâmica, conforme a demanda do módulo cliente.

# Duvidas?

## [a **ENGINE**]

Pedro Henrique Ferreira Stringhini  
Jessica do Prado Oliveira

RA 156983  
RA 146584

## [o **MAPA**]

Gabriel Souza Franco

RA 155477

## [o **PLAYER** e **DRONE**]

Guilherme Higa  
Agustina Diamint

RA 160162  
RA 157637

## [os **EQUIPAMENTOS**]

Beatriz Sechin Zazulla  
Gabriel Jorge Gimenez

RA 154779  
RA 155449