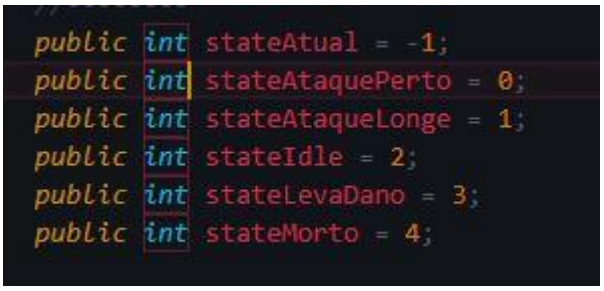
Refatoração da máquina de estados

Acadêmicos: Diego Hartmann; Philipp Altendorf
Disciplina: Inteligência Artificial
Professor: Ricardo Cherobin

Código Antigo

A configuração do código anteriormente estava da seguinte forma:

Os dados de controle do estado eram todos variáveis públicas, tendo uma para o **levando dano** e outra para o **ataque perto.**.



```
public int stateAtual = -1;
public int stateAtaquePerto = 0;
public int stateAtaqueLonge = 1;
public int stateIdle = 2;
public int stateLevaDano = 3;
public int stateMorto = 4;
```

Além disso, para setar o estado atual, era feita uma checagem para identificar tanto o estado de **levar dano** quanto o estado de **atacar perto**, dentro de um ‘if’ maior que checa se tem distância o suficiente para um combate *melee*.

```
public void ChecarEstadoAtual(Heroi _Jogador, long diffTime){

    if (!TemVida()){
        EstadoAtual(this.stateMorto);
        return;
    }

    if (DistanciaAteJogador() < this.distanciaParaCombateMelee){
        levaDanoContador += diffTime / 1000.0f;
        ataquePertoContador += diffTime / 1000.0f;

        if(levaDanoContador > levaDanoFrequencia) {
            EstadoAtual(this.stateLevaDano);
            levaDanoContador = 0;
            //return;
        }
        if(ataquePertoContador > ataquePertoFrequencia) {
            EstadoAtual(this.stateAtaquePerto);
            ataquePertoContador = 0;
            //return;
        }
        return;
    }
```

Código Novo

Depois, os dados de controle do estado foram atualizados para constantes privadas, e a do **estado atual** continuou uma variável. Foram criadas também variáveis de controle de uma sub-máquina de estados, a **melee**, que pode ser tanto ***levando dano*** quanto ***atacando de perto***.

```
private int stateAtual;
private final int STATE_ATAQUE_LONGE = 0;
private final int STATE_IDLE = 1;
private final int STATE_MORTO = 2;
private final int STATE_MELEE = 3;
private int stateMeleeCurrent;
private final int STATE_ATAQUE_PERTO = 10;
private final int STATE_LEVA_DANO = 11;
```

Agora, a maquina de estados geral apenas se preocupa em setar os 4 estados básicos.

```
public void ChecarEstadoAtual(Heroi _Jogador, long diffTime){

    if (!TemVida()){
        EstadoAtual(this.STATE_MORTO);
        return;
    }

    if (DistanciaAteJogador() < this.distanciaParaCombateMelee){
        EstadoAtual(this.STATE_MELEE);
        return;
    }

    if(DistanciaAteJogador() < this.distanciaParaAtaqueLonge) {
        EstadoAtual(this.STATE_ATAQUE_LONGE);
        return;
    }
        heroi.vel = 40;
        return;
    }
    EstadoAtual(this.STATE_IDLE);
}
```

E a mudança dos sub-estados do melee são atualizados dentro da máquina principal que atualiza as ações.

```
public void AplicarAcaoParaEstadoAtual(long _diffTime){
    switch (stateAtual) {

        case STATE_ATAQUE_PERTO:
            AtacarPerto();
            break;

        case STATE_MELEE:
            ChecarEstadoMelee();
            AplicatEstadoMelee();
            break;

        case STATE_IDLE:
            Idle();
            break;

        case STATE_MORTO:
            Morrer();
            break;

        default:
            break;
    }
}
```

As duas funções destacadas na imagem acima estão escritas da seguinte forma, para atualizar qual o sub-estado do **melee** (ataque ou levar dano):

```
private void ChecarEstadoMelee(){
    levaDanoContador += diffTime / 1000.0f;
    ataquePertoContador += diffTime / 1000.0f;

    if(levaDanoContador > levaDanoFrequencia) {
        EstadoMelee(this.STATE_LEVA_DANO);
        levaDanoContador = 0;
    }
    if(ataquePertoContador > ataquePertoFrequencia) {
        EstadoMelee(this.STATE_ATAQUE_PERTO);
        ataquePertoContador = 0;
    }


}
private void AplicatEstadoMelee(){
    switch (stateMelee) {
        case STATE_LEVA_DANO:
            LevaDano();
            break;
        case STATE_ATAQUE_PERTO:
            AtacarPerto();
            break;
        default:
            break;
    }
}
```