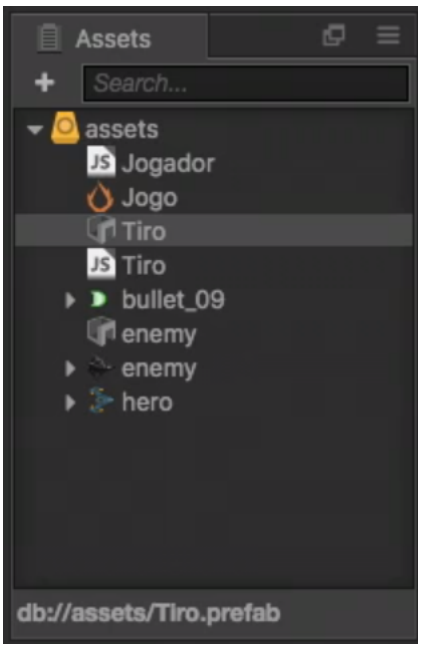
01

Gerador de inimigos

Transcrição

Chegamos ao ponto em que precisamos criar várias instâncias do nosso inimigo no jogo. Afinal, se tivermos apenas um, o jogo estará muito fácil. Para a criação de várias instâncias de um objeto, já vimos antes o que precisamos fazer. Criar um *Prefab*.

Para isso, arrastamos o objeto `enemy` da aba `Node Tree` para a aba `Assets`.



Lembrando que o nosso objeto `enemy` já possui configurações de colisão, entre outras. Os objetos criados a partir desse *prefab* também possuirão todas essas características. Ótimo, já podemos criar inimigos dinamicamente. Podemos?

Ainda não podemos criar inimigos dinamicamente por que não temos quem faça isso: O gerador de inimigos. Nosso gerador não precisará de uma imagem de representação, ele será apenas um ponto no espaço de onde os inimigos surgirão. Para criar esse ponto no espaço, clicaremos com o botão direito na aba `Node Tree` e usaremos as opções `Create -> Create Empty Node`. Este será nosso objeto vazio que criará todos os nossos inimigos. Chamaremos esse novo objeto de `GeradorInimigo`.

Sabendo que ele irá criar os inimigos, já criaremos o *script* `Gerador.js` na aba `Assets`. Algumas propriedades precisarão estar presentes no nosso gerador de inimigos. A primeira delas é o *prefab* do inimigo, a segunda, será a área onde os inimigos serão gerados e por último, um número que represente um intervalo de tempo. Com isso, poderemos gerar inimigos em um determinado espaço a cada determinado período de tempo. Assim teremos:

```
cc.Class({
    extends: cc.Component,

    properties: {
       inimigoPrefab: cc.Prefab,
       area: 10,
       tempo: 2,
    },
```

```
    // use this for initialization
    onLoad: function () {
    },

    // called every frame, uncomment this function to activate update callback
    // update: function (dt) {
    // },
});
```

Da mesma forma que fizemos com o disparo do tiro da nave, precisamos seguir alguns passos para a criação dos inimigos. Entre estes passos estão: criar a instância que se baseia no *prefab*, definir um pai para o objeto poder aparecer na cena e definir seu posicionamento. Criaremos o método `gerar` que será responsável por tudo isso. Nele teremos:

```
gerar: function(){
    let inimigo = cc.instantiate(this.inimigoPrefab);
    inimigo.parent = this.node.parent;

    let posicao = new cc.Vec2(Math.random() - .5, Math.random() - .5);
}
```

O detalhe no código que ainda não conhecemos é a criação da posição do objeto. Criaremos objetos que se posicionarão inicialmente de forma aleatória. O método `Math.random` nos ajuda com essa questão gerando valores aleatórios entre 0 e 1. Não se preocupe com a subtração de `0,5`. Com ela teremos valores entre `-0,5` e `0,5`. Adiante entenderemos o porquê.

Os demais passos para posicionar o inimigo requerem um pouco mais de cuidado, por isso, **atenção**! Note que geramos um vetor de valores aleatórios, mas que no fim, essa posição será relativa ao objeto gerador de inimigos. Repare que não temos uma posição lógica com os valores gerados. Por este motivo, transformaremos o vetor posição em uma direção. Para não termos problemas com distâncias, usaremos o método `normalize`.

O próximo passo é multiplicar a direção obtida pela área que havíamos deixado pré-configurada. Isto é, queremos que o inimigo seja gerado em uma direção em torno dessa área. Por isso, multiplicaremos os valores.

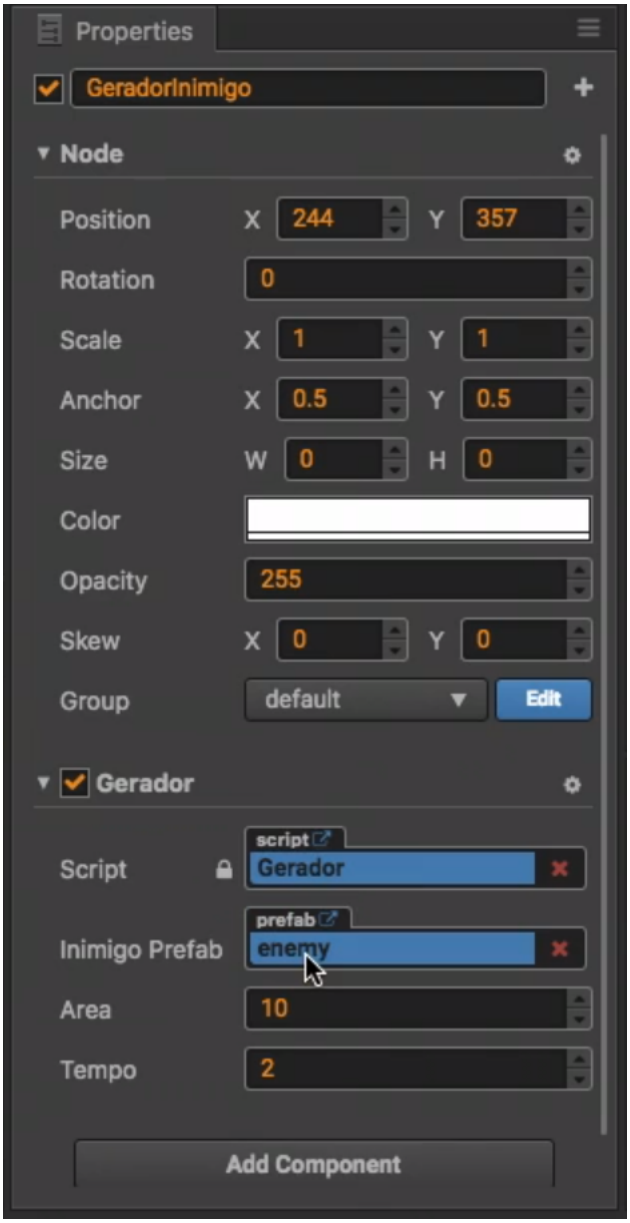
Por último, mas não menos importante, calcularemos a posição final do inimigo, somando a posição do objeto gerador ao vetor anterior. Nestes três passos nós praticamente definimos duas características do objeto inimigo: Em que direção ele será gerado e a qual distância do ponto de referência. Juntando tudo, o método `gerar` fica dessa forma:

```
gerar: function(){
    let inimigo = cc.instantiate(this.inimigoPrefab);
    inimigo.parent = this.node.parent;

    let posicao = new cc.Vec2(Math.random() - .5, Math.random() - .5);
    posicao = posicao.normalize();
    posicao = posicao.mul(this.area);
    posicao = this.node.position.add(posicao);

    inimigo.position = posicao;
}
```

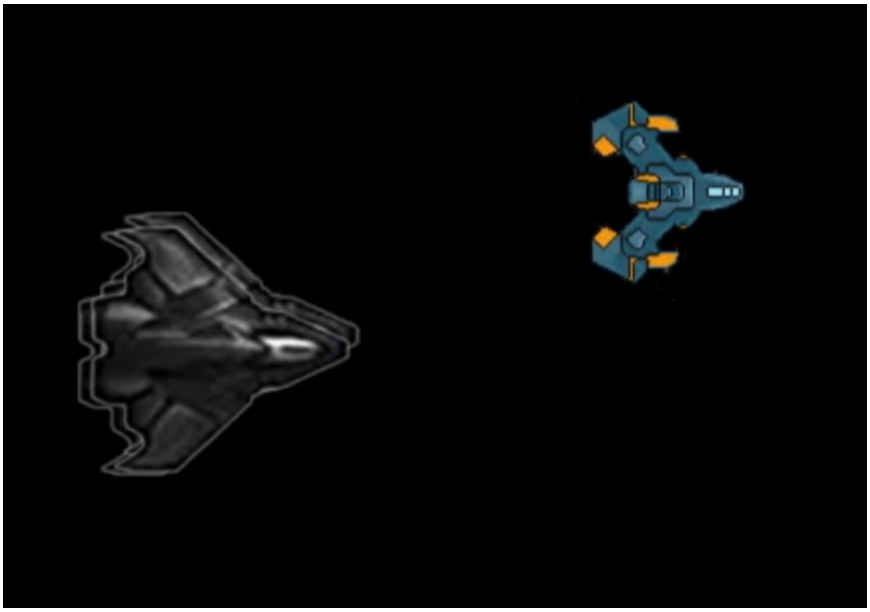
Tudo que precisamos fazer é agendar a execução desse método repetidamente. Para isso, dentro do método `onLoad` faremos uso do método `schedule` que é herdado da classe `Component`. Neste método, precisamos informar qual será o outro método a ser executado e em qual intervalo de tempo, lembrando que o valor do intervalo é em segundos.

```
onLoad: function () {
    this.schedule(this.gerar, this.tempo);
},
```

Pronto! Temos tudo pronto. Basta deixarmos tudo configurado na Cocos conforme é necessário, ou seja, relacionar o *script* `Gerador.js` ao objeto `GeradorInimigo` e também relacionar o *prefab* do inimigo ao mesmo objeto. Lembrando que precisamos apenas selecionar o objeto para a aba `Properties`, e arrastar os demais componentes.



Já podemos testar o jogo novamente e ver que a cada 2 segundos, um novo inimigo surge na cena. Só há um problema. A distância entre um inimigo gerado e outro é bem pequena, parecendo estarem no mesmo lugar. Para solucionar isso basta aumentar o valor da área para 200, por exemplo.



Entendendo os cálculos

Você lembra que a posição inicial era gerada a partir do `Math.random` e desse valor subtraíamos 0,5 ? Eis o porquê.

Vamos considerar no plano cartesiano que as coordenadas (0,0) sejam do gerador de inimigos. Como o `Math.random` gera números entre 0 e 1, temos um problema. Os inimigos seriam gerados sempre em uma área que está posicionada acima e à direita do ponto do gerador. Quando subtrairmos metade do valor gerado, estamos movendo a área para considerar o ponto onde o gerador se encontra, o centro da área disponível.