06

Build

Transcrição

Ainda podemos melhorar uma coisa em nosso código. Note que naturalmente duplicamos uma porção de código relacionada ao cálculo da direção para onde as naves devem seguir, assim como duplicamos também, o código que calcula a rotação da nave.

Uma solução prática é criar dois métodos auxiliares no `Personagem.js` para que isolem esse código repetido em um único lugar que poderá ser reaproveitado tanto no `Jogador.js`, quanto no `Inimigo.js`. Os métodos serão chamados `calculaDirecao` e `olharPara`. Os dois receberão um ponto de referência como `destino` e `direcao`, respectivamente, para que os demais cálculos possam ser feitos. Vejamos os métodos que serão adicionados ao `Personagem.js`:

```
calcularDirecao: function(destino){
    let posicao = destino.sub(this.node.position);
    posicao = posicao.normalize();
    return posicao;
},

olharPara: function(direcao){
    let angulo = Math.atan2(direcao.y, direcao.x)/
    angulo = -angulo * (180 / Math.PI);
    return angulo;
},
```

O `calcularDirecao` retorna o vetor normalizado indicando a direção para onde a nave deverá seguir, considerando um destino informado e a posição atual do objeto. Já o `olharPara` requer uma direção e retorna o ângulo de rotação do objeto.

Onde temos o código que calcula a direção e a rotação no `Jogador.js`, podemos usar estes métodos auxiliares ao invés de refazer o cálculo. O método `mudarDirecao` no `Jogador.js` ficará assim:

```
mudarDirecao: function(event){
    let posicaoMouse = event.getLocation();
    posicaoMouse = new cc.Vec2(posicaoMouse.x, posicaoMouse.y);

    this._direcao = this.calcularDirecao(posicaoMouse);
    this.node.rotation = this.olharPara(this._direcao);
},
```

O mesmo método no `Inimigo.js` fica ainda menor.

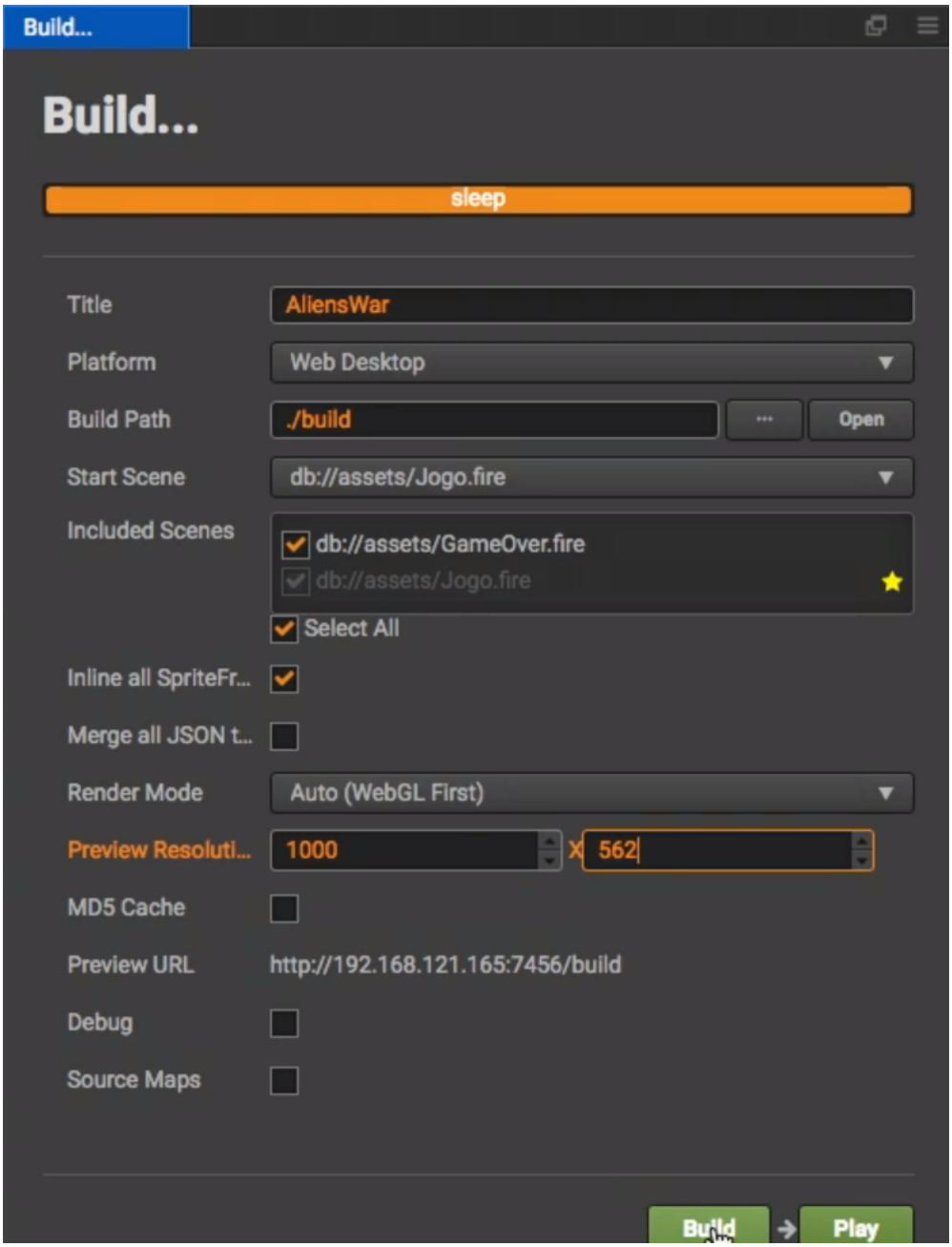
```
mudarDirecao: function(){
    this._direcao = this.calcularDirecao(this._alvo.position);
    this.node.rotation = this.olharPara(this._direcao);
},
```

E tudo continua funcionando normalmente.

Gerando o build

Agora que codificamos tudo que queríamos inicialmente, está na hora de disponibilizar nosso jogo para que os amigos joguem, avaliem, sugiram melhorias e até novas ideias para serem adicionadas.

O *build* é o conjunto de instruções e arquivos necessários para que a aplicação funcione por completo. Para gerá-lo na *Cocos*, precisamos acessar os menus `Project -> Build`. Uma janela de configurações semelhante à da imagem abaixo será aberta.



As configurações que precisamos estar de olho são: `Platform`, `Start Scene` e `Preview Resolution`.

A plataforma será `Web Desktop`, visto que nosso jogo só será jogado em navegadores de computadores de mesa e notebooks. Ou seja, não é um jogo adaptado para celulares, etc.

O `Start Scene` configura a tela inicial do jogo assim que ele for aberto, então cuidado para não deixar selecionada a tela de *game over*, por exemplo. Se não os seus amigos não conseguirão jogar.

Por último, o `Preview Resolution` informa o tamanho da tela que será desenhada para o jogo. Utilizamos os valores `1000` e `562`, que é um valor proporcional ao *FULL HD*, porém menor.

Depois disso só precisamos clicar no botão `Build`. A *Cocos* fará todo o empacotamento dos arquivos em uma pasta chamada `web-desktop` dentro da pasta `build` do seu projeto.

E para disponibilizar o jogo para seus amigos basta hospedar os arquivos da pasta `web-desktop` em algum servidor web de sua preferência!