Treinamento Teknisa

Boas Práticas em Programação

Rômulo A. Lousada

I. Indentação
II. Espaçamento
III. Nomenclatura de Funções e Variáveis
IV. Const/Let
V. Fail Fast - IF
VI. Comentários
VII. Reusabilidade do Código
VIII. Try...Catch

Convenções

Ao escrever um código, é importante se atentar não apenas a eficácia da solução que está sendo desenvolvida, mas também se atentar a alguns pontos importantes para deixar o código mais limpo e legível para outros desenvolvedores.

Alguns pontos importantes serão mencionados, mas a boa prática não se resume a apenas estes. É necessário que o profissional estude e fique por dentro das tendências e os guias/normas existentes para expandir o entendimento e conhecimento.

Código de Exemplo

A imagem a seguir contém um trecho de código onde nenhuma boa prática é aplicada. No decorrer deste treinamento, a cada conceito, vamos aplicar ao código, até chegar ao resultado final.

JS exemplo.js > playerAtaque

```
1   var playerAtaque=async(personagem,inimigo)=>{
2   if(getMouseBotaoPressionado()){
3   var movimento=getMovimento(personagem.movement);
4   if(!isCorrendo(movimento)){
5   var acao=getAcao(personagem.action);
6   if(!isBloqueando(acao)){
7   personagem.status.dano=personagem.offense*personagem.level;
8   personagem.status.defesa=personagem.defense*personagem.level;
9   personagem.status.critico=personagem.critChance;
10  inimigo.status.dano=inimigo.offense*inimigo.level;
11  inimigo.status.defesa=inimigo.defense*inimigo.level;
12  inimigo.status.critico=inimigo.critChance;
13  var dano=await ataque(personagem,inimigo);
14  console.log('Personagem '+personagem.name+' causou '+dano+' de dano ao inimigo '+inimigo.name);
15  }else{
16  console.log('Não pode atacar enquanto está bloqueando!');
17  }
18  }else{
19  console.log('Não pode atacar enquanto está correndo!');
20  }
21  }
22  }
```

I. Indentação

Indentação

Extremamente importante para a legibilidade do código.

Um código não indentado ou mal-indentado pode dificultar muito o entendimento dele.

É uma prática comum, e que deve ser treinada diariamente para que seja cada vez mais comum a escrita de códigos dessa forma.

A cada “bloco”, dar um tab para gerar um espaçamento para dentro do código, indicando que ele está contido dentro de uma estrutura externa.

Indentação

exemplo.js > playerAtaque

```
var playerAtaque=async(personagem,inimigo)=>{
if(getMouseBotaoPressionado()){
var movimento=getMovimento(personagem.movement);
if(!isCorrendo(movimento)){
var acao=getAcao(personagem.action);
if(!isBloqueando(acao)){
personagem.status.dano=personagem.offense*personagem.level;
personagem.status.defesa=personagem.defense*personagem.level;
personagem.status.critico=personagem.critChance;
inimigo.status.dano=inimigo.offense*inimigo.level;
inimigo.status.defesa=inimigo.defense*inimigo.level;
inimigo.status.critico=inimigo.critChance;
var dano=await ataque(personagem,inimigo);
console.log('Personagem '+personagem.name+' causou '+dano+' de dano ao inimigo '+inimigo.name);
}else{
console.log('Não pode atacar enquanto está bloqueando!');
}
}else{
console.log('Não pode atacar enquanto está correndo!');
}
}
}
```

Indentação

identacao.js > playerAtaque

```
1  var playerAtaque=async(personagem,inimigo)=>{
2    if(getMouseBotaoPressionado()){
3      var movimento=getMovimento(personagem.movement);
4      if(!isCorrendo(movimento)){
5        var acao=getAcao(personagem.action);
6        if(!isBloqueando(acao)){
7          personagem.status.dano=personagem.offense*personagem.level;
8          personagem.status.defesa=personagem.defense*personagem.level;
9          personagem.status.critico=personagem.critChance;
10         inimigo.status.dano=inimigo.offense*inimigo.level;
11         inimigo.status.defesa=inimigo.defense*inimigo.level;
12         inimigo.status.critico=inimigo.critChance;
13         var dano=await ataque(personagem,inimigo);
14         console.log('Personagem '+personagem.name+' causou '+dano+' de dano ao inimigo '+inimigo.name);
15       }else{
16         console.log('Não pode atacar enquanto está bloqueando!');
17       }
18     }else{
19       console.log('Não pode atacar enquanto está correndo!');
20     }
21   }
22 }
```

II. Espaçamento

Espaçamento

Não é obrigatório, porém, dar um espaço de linha no código em alguns momentos torna muito melhor a visibilidade do que está acontecendo.

Existem regras onde esse espaçamento deve ocorrer, e o programador pode escolher seguir elas a risca, ou apenas como achar necessário.

Espaçamento

identacao.js > playerAtaque

```
1  var playerAtaque=async(personagem,inimigo)=>{
2    if(getMouseBotaoPressionado()){
3      var movimento=getMovimento(personagem.movement);
4      if(!isCorrendo(movimento)){
5        var acao=getAcao(personagem.action);
6        if(!isBloqueando(acao)){
7          personagem.status.dano=personagem.offense*personagem.level;
8          personagem.status.defesa=personagem.defense*personagem.level;
9          personagem.status.critico=personagem.critChance;
10         inimigo.status.dano=inimigo.offense*inimigo.level;
11         inimigo.status.defesa=inimigo.defense*inimigo.level;
12         inimigo.status.critico=inimigo.critChance;
13         var dano=await ataque(personagem,inimigo);
14         console.log('Personagem '+personagem.name+' causou '+dano+' de dano ao inimigo '+inimigo.name);
15       }else{
16         console.log('Não pode atacar enquanto está bloqueando!');
17       }
18     }else{
19       console.log('Não pode atacar enquanto está correndo!');
20     }
21   }
22 }
```

Espaçamento

espacamento.js > playerAtaque

```
var playerAtaque = async(personagem, inimigo) => {
  if (getMouseBotaoPressionado()) {
    var movimento = getMovimento(personagem.movement);

    if (!isCorrendo(movimento)) {
      var acao = getAcao(personagem.action);

      if (!isBloqueando(acao)) {
        personagem.status.dano = personagem.offense * personagem.level;
        personagem.status.defesa = personagem.defense * personagem.level;
        personagem.status.critico = personagem.critChance;

        inimigo.status.dano = inimigo.offense * inimigo.level;
        inimigo.status.defesa = inimigo.defense * inimigo.level;
        inimigo.status.critico = inimigo.critChance;

        var dano = await ataque(personagem, inimigo);
        console.log('Personagem ' + personagem.name + ' causou ' + dano + ' de dano ao inimigo ' + inimigo.name);
      } else {
        console.log('Não pode atacar enquanto está bloqueando!');
      }
    } else {
      console.log('Não pode atacar enquanto está correndo!');
    }
  }
}
```

III. Nomenclatura de Funções e Variáveis

Nomenclatura de Funções e Variáveis

Na hora de colocar um nome, seja um arquivo, função ou variável, é importante seguir algumas regras ou modelos já existentes.

No caso do arquivo, seguir o modelo já adotado no projeto, caso exista. Se não existir, definir com a equipe a forma que os arquivos vão ser nomeados.

Funções e variáveis devem ser nomeados respeitando os padrões definidos da linguagem, porém, existem alguns fatores a se atentar.

Nomenclatura de Funções e Variáveis

I. Evitar nomes muito longos. Ele precisa ser conciso e claro.

I. Tomar cuidado para não misturar idiomas e criar algo em “portuglês”.

I. Respeitar as regras de letras minúsculas e maiúsculas:
   A. PascalCase
   B. camelCase
   C. snake_case

Nomenclatura de Funções e Variáveis

espacamento.js > playerAtaque

```
1   var playerAtaque = async(personagem, inimigo) => {
2     if (getMouseBotaoPressionado()) {
3       var movimento = getMovimento(personagem.movement);
4
5       if (!isCorrendo(movimento)) {
6         var acao = getAcao(personagem.action);
7
8         if (!isBloqueando(acao)) {
9           personagem.status.dano = personagem.offense * personagem.level;
10          personagem.status.defesa = personagem.defense * personagem.level;
11          personagem.status.critico = personagem.critChance;
12
13          inimigo.status.dano = inimigo.offense * inimigo.level;
14          inimigo.status.defesa = inimigo.defense * inimigo.level;
15          inimigo.status.critico = inimigo.critChance;
16
17          var dano = await ataque(personagem, inimigo);
18          console.log('Personagem ' + personagem.name + ' causou ' + dano + ' de dano ao inimigo ' + inimigo.name);
19        } else {
20          console.log('Não pode atacar enquanto está bloqueando!');
21        }
22      } else {
23        console.log('Não pode atacar enquanto está correndo!');
24      }
25    }
26  }
```

Nomenclatura de Funções e Variáveis

JS nomenclatura.js > playerAttack

```
1   var playerAttack = async(character, enemy) => {
2     if (getMouseButtonDown()) {
3       var movement = getMovement(character.movement);
4
5       if (!isSprinting(movement)) {
6         var action = getAction(character.action);
7
8         if (!isBlocking(action)) {
9           character.status.offense = character.offense * character.level;
10          character.status.defense = character.defense * character.level;
11          character.status.crit = character.critChance;
12
13          enemy.status.offense = enemy.offense * enemy.level;
14          enemy.status.defense = enemy.defense * enemy.level;
15          enemy.status.crit = enemy.critChance;
16
17          var damage = await attack(character, enemy);
18          console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
19        } else {
20          console.log('Cannot attack while blocking!');
21        }
22      } else {
23        console.log('Cannot attack while sprinting!');
24      }
25    }
26  }
```

IV. Const/Let

Const/Let

Se manter atualizado é um papel importante de um desenvolvedor.

Mudanças acontecem em uma frequência alta, e é necessário se adaptar para utilizar os melhores conceitos disponíveis.

Um dos exemplos de atualização, nem tão recente assim, é a mudança da forma de declaração de variável no javascript.

Antes era utilizada a palavra *var* para fazer a declaração, porém essa forma apresentava problemas relacionados ao escopo de variáveis declaradas dessa forma.

Para isso, foi criado o *const* e o *let*, que permitem declarar variáveis sem este problema.

Const/Let

JS nomenclatura.js > playerAttack

```
1   var playerAttack = async(character, enemy) => {
2     if (getMouseButtonDown()) {
3       var movement = getMovement(character.movement);
4
5       if (!isSprinting(movement)) {
6         var action = getAction(character.action);
7
8         if (!isBlocking(action)) {
9           character.status.offense = character.offense * character.level;
10          character.status.defense = character.defense * character.level;
11          character.status.crit = character.critChance;
12
13          enemy.status.offense = enemy.offense * enemy.level;
14          enemy.status.defense = enemy.defense * enemy.level;
15          enemy.status.crit = enemy.critChance;
16
17          var damage = await attack(character, enemy);
18          console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
19        } else {
20          console.log('Cannot attack while blocking!');
21        }
22      } else {
23        console.log('Cannot attack while sprinting!');
24      }
25    }
26  }
```

Const/Let

JS constlet.js > playerAttack

```
const playerAttack = async(character, enemy) => {
  if (getMouseButtonDown()) {
    const movement = getMovement(character.movement);

    if (!isSprinting(movement)) {
      const action = getAction(character.action);

      if (!isBlocking(action)) {
        character.status.offense = character.offense * character.level;
        character.status.defense = character.defense * character.level;
        character.status.crit = character.critChance;

        enemy.status.offense = enemy.offense * enemy.level;
        enemy.status.defense = enemy.defense * enemy.level;
        enemy.status.crit = enemy.critChance;

        const damage = await attack(character, enemy);
        console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
      } else {
        console.log('Cannot attack while blocking!');
      }
    } else {
      console.log('Cannot attack while sprinting!');
    }
  }
}
```

V. Fail Fast - IF

Fail Fast - IF

É uma forma de deixar o código mais claro, principalmente quando o mesmo possui diversas condições *if/else*.

Ele tenta verificar os eventuais “problemas” primeiro, e caso caia nessas condições, já para o código por ali.

Dessa forma, além do código ficar mais limpo, a escalabilidade e manutenção também ficam fáceis, permitindo adicionar mais condições, sem tornar a estrutura do código mais complexa do que deveria.

Fail Fast - IF

JS constlet.js > playerAttack

```
1   const playerAttack = async(character, enemy) => {
2     if (getMouseButtonDown()) {
3       const movement = getMovement(character.movement);
4
5       if (!isSprinting(movement)) {
6         const action = getAction(character.action);
7
8         if (!isBlocking(action)) {
9           character.status.offense = character.offense * character.level;
10          character.status.defense = character.defense * character.level;
11          character.status.crit = character.critChance;
12
13          enemy.status.offense = enemy.offense * enemy.level;
14          enemy.status.defense = enemy.defense * enemy.level;
15          enemy.status.crit = enemy.critChance;
16
17          const damage = await attack(character, enemy);
18          console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
19        } else {
20          console.log('Cannot attack while blocking!');
21        }
22      } else {
23        console.log('Cannot attack while sprinting!');
24      }
25    }
26  }
```

Fail Fast - IF

failfast.js > playerAttack

```
const playerAttack = async (character, enemy)=>{
  if (!getMouseButtonDown()) {
    return;
  }

  const movement = getMovement(character.movement);

  if (isSprinting(movement)) {
    console.log('Cannot attack while sprinting!');
    return;
  }

  const action = getAction(character.action);

  if (isBlocking(action)) {
    console.log('Cannot attack while blocking!');
    return;
  }

  character.status.offense = character.offense * character.level;
  character.status.defense = character.defense * character.level;
  character.status.crit = character.critChance;

  enemy.status.offense = enemy.offense * enemy.level;
  enemy.status.defense = enemy.defense * enemy.level;
  enemy.status.crit = enemy.critChance;

  const damage = await attack(character, enemy);
  console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
}
```

VI. Comentários

Comentários

Existem opiniões distintas, onde alguns defendem o uso de comentários no código para facilitar o entendimento do mesmo, e outros defendem que se o código precisa de comentário, o código deveria ser reescrito para que não seja tão confuso.

O ideal é não necessitar de comentários explicando o código, mas caso seja muito importante comentar um trecho do código, que seja feito de maneira sucinta, direto ao ponto.

Comentários

JS failfast.js > playerAttack

```
1  const playerAttack = async (character, enemy)=>{
2    if (!getMouseButtonDown()) {
3      return;
4    }
5
6    const movement = getMovement(character.movement);
7
8    if (isSprinting(movement)) {
9      console.log('Cannot attack while sprinting!');
10     return;
11   }
12
13   const action = getAction(character.action);
14
15   if (isBlocking(action)) {
16     console.log('Cannot attack while blocking!');
17     return;
18   }
19
20   character.status.offense = character.offense * character.level;
21   character.status.defense = character.defense * character.level;
22   character.status.crit = character.critChance;
23
24   enemy.status.offense = enemy.offense * enemy.level;
25   enemy.status.defense = enemy.defense * enemy.level;
26   enemy.status.crit = enemy.critChance;
27
28   const damage = await attack(character, enemy);
29   console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
30 }
```

Comentários

comentario.js > playerAttack

```
const playerAttack = async (character, enemy)=>{
  if (!getMouseButtonDown()) {
    //Attack can't occur if mouse isn't pressed.
    return;
  }

  const movement = getMovement(character.movement);

  if (isSprinting(movement)) {
    console.log('Cannot attack while sprinting!');
    return;
  }

  const action = getAction(character.action);

  if (isBlocking(action)) {
    console.log('Cannot attack while blocking!');
    return;
  }

  //Character Status Calculation
  character.status.offense = character.offense * character.level;
  character.status.defense = character.defense * character.level;
  character.status.crit = character.critChance;

  //Enemy Status Calculation
  enemy.status.offense = enemy.offense * enemy.level;
  enemy.status.defense = enemy.defense * enemy.level;
  enemy.status.crit = enemy.critChance;

  const damage = await attack(character, enemy);
  console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
}
```

VII. Reusabilidade

Reusabilidade

É um conceito muito importante, pois evita reescrever códigos semelhantes na mesma ou em diferentes funções.

A reutilização facilita caso seja necessário alterar algo posteriormente, fazendo que apenas uma função sendo alterada, reflita para todos os outros lugares que utilizam aquela função.

Ganho de espaço no código.

Reusabilidade

comentario.js > playerAttack

```
const playerAttack = async (character, enemy)=>{
  if (!getMouseButtonDown()) {
    //Attack can't occur if mouse isn't pressed.
    return;
  }

  const movement = getMovement(character.movement);

  if (isSprinting(movement)) {
    console.log('Cannot attack while sprinting!');
    return;
  }

  const action = getAction(character.action);

  if (isBlocking(action)) {
    console.log('Cannot attack while blocking!');
    return;
  }

  //Character Status Calculation
  character.status.offense = character.offense * character.level;
  character.status.defense = character.defense * character.level;
  character.status.crit = character.critChance;

  //Enemy Status Calculation
  enemy.status.offense = enemy.offense * enemy.level;
  enemy.status.defense = enemy.defense * enemy.level;
  enemy.status.crit = enemy.critChance;

  const damage = await attack(character, enemy);
  console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
}
```

Reusabilidade

reusabilidade.js > statusCalculation

```
const playerAttack = async (character, enemy)=>{
  if (!getMouseButtonDown()) {
    //If mouse is not button is not pressed.
    return;
  }

  const movement = getMovement(character.movement);

  if (isSprinting(movement)) {
    console.log('Cannot attack while sprinting!');
    return;
  }

  const action = getAction(character.action);

  if (isBlocking(action)) {
    console.log('Cannot attack while blocking!');
    return;
  }

  character.status = statusCalculation(character);
  enemy.status = statusCalculation(enemy);

  const damage = await attack(character, enemy);
  console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
}

const statusCalculation = (entity) => {
  const offense = entity.offense * entity.level;
  const defense = entity.defense * entity.level;
  const crit = entity.crit;

  return {
    offense,
    defense,
    crit
  }
}
```

VIII. Try...Catch

Try...Catch

É uma cláusula que especifica o que deve acontecer caso uma exceção seja disparada durante a execução do código.

Tratamento de erro eficaz, que evita que a mensagem de erro padrão retorne e apareça para o usuário.

Permite que o programador defina o melhor procedimento caso um erro ocorra durante a execução do código.

Try...Catch

reusabilidade.js > statusCalculation

```
const playerAttack = async (character, enemy)=>{
  if (!getMouseButtonDown()) {
    //If mouse is not button is not pressed.
    return;
  }

  const movement = getMovement(character.movement);

  if (isSprinting(movement)) {
    console.log('Cannot attack while sprinting!');
    return;
  }

  const action = getAction(character.action);

  if (isBlocking(action)) {
    console.log('Cannot attack while blocking!');
    return;
  }

  character.status = statusCalculation(character);
  enemy.status = statusCalculation(enemy);

  const damage = await attack(character, enemy);
  console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
}

const statusCalculation = (entity) => {
  const offense = entity.offense * entity.level;
  const defense = entity.defense * entity.level;
  const crit = entity.crit;

  return {
    offense,
    defense,
    crit
  }
}
```

Try...Catch

trycatch.js > playerAttack

```
1  const playerAttack = async (character, enemy)=>{
2    try {
3      if (!getMouseButtonDown()) {
4        //If mouse is not button is not pressed.
5        return;
6      }
7
8      const movement = getMovement(character.movement);
9
10     if (isSprinting(movement)) {
11       console.log('Cannot attack while sprinting!');
12       return;
13     }
14
15     const action = getAction(character.action);
16
17     if (isBlocking(action)) {
18       console.log('Cannot attack while blocking!');
19       return;
20     }
21
22     //Character Status Calculation
23     character.status = statusCalculation(character);
24
25     //Enemy Status Calculation
26     enemy.status = statusCalculation(enemy);
27
28     const damage = await attack(character, enemy);
29     console.log('Character ' + character.name + ' caused ' + damage + ' damage on the enemy ' + enemy.name);
30   } catch(error) {
31     rollbackGame();
32     saveGame();
33     throw error;
34   }
35 }
```

Dúvidas