Esse documento tem a única e exclusiva finalidade de mostrar o progresso de criação e alteração de código envolvendo o projeto AUTO BATTLE. Para consulta do GDD do projeto, acesse o link abaixo:

https://docs.google.com/document/d/15OolJKZ_w-G3_eCipQ59GLVsxFXyGatnxMR1QsjsyS8/edit?usp=sharing

Versão 1.0.1

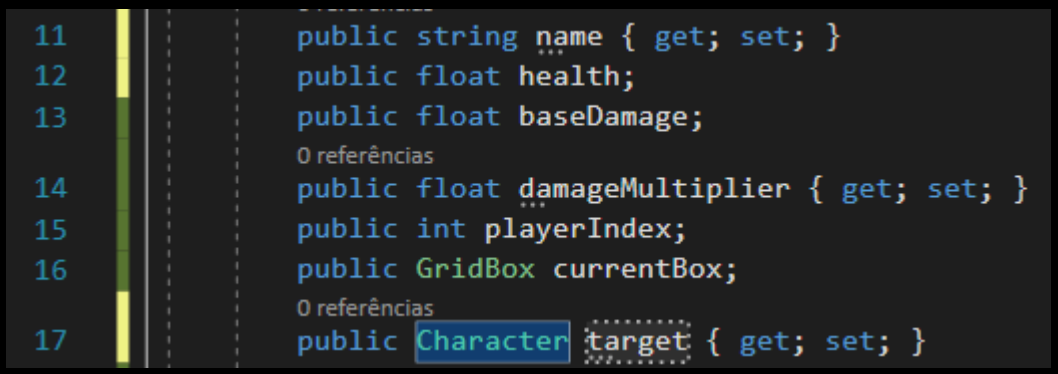
Melhorias de Versão

- Personagens possuem identificação nas Boxes (H para humano e I para Inimigo);
- Posições são definidas de forma aleatória no campo;
- Ordem de quem começa primeiro é também aleatória;
- Feedbacks mais claros de vida e dano causados;
- Finalização indicando quem ganhou a batalha.

Refatoração - Correção do Script Character.cs

Classe Character / Linhas 11 a 17

- Nome das variáveis fora da norma padrão de escrita. Variáveis começam com letras minúsculas e seus sobrenomes compostos devem ter letra maiúscula. A nova versão alterada ficou:



Classe Character / Método *TakeDamage()*

- O método possui um parâmetro *amount*, mas nunca é usado dentro da função.
- Também não é necessário comparar se o jogador morreu aqui, a função *HandleTurn()* da classe Program já determina se o próximo turno será chamado ou finalizado de acordo com a vida dos personagens. A nova versão alterada ficou:

```
24      public bool TakeDamage(float amount)
25      {
26          health -= amount;
27          return false;
28      }
```

Classe Character / Método *Die()*

- O método *Die()* teve seu nome alterado para *EndGame()*, já que nesse local seria mais efetivo dar as recompensas do jogador caso ganhasse ou realizasse outras ações além de morrer, como receber XP, pontos, etc.

```
30      public void EndGame()
31      {
32          Console.WriteLine($"Player {target.name} venceu o {name}!");
33      }
```

Classe Character / Método *WalkTO()*

- Esse método está incorreto, nomes de funções e métodos possuem apenas letra maiúscula no ínicio do nome.
- O método está obsoleto no código, já que a função *Start Turn()* regula o movimento de cada personagem. Dessa forma foi excluída do código.

Classe Character / Método *StartTurn()*

- A comparação com função anônima abaixo não é necessária, já que os personagens dentro do campo já serão inseridos na range delimitada e irão buscar a menor rota para batalharem, além de que sua posição já é definida na grade inicialmente.
- Parênteses também estão duplicadas, apenas 1 par é necessário.

```
if ((battlefield.grids.Exists(x => x.Index == currentBox.Index - 1)))
```

- Na linha 76 o comando “return” está em posição indevida. O método deve executar todo o comando antes de retornar algum valor mesmo sendo nulo. A nova versão ficou assim:

```
Console.WriteLine($"Player {name} andou pra cima");
currentBox.ocupied = false;
battlefield.grids[currentBox.Index] = currentBox;
currentBox = battlefield.grids.Find(x => x.Index == currentBox.Index - battlefield.xLenght);
currentBox.ocupied = true;
currentBox.charType = name;
battlefield.grids[currentBox.Index] = currentBox;
battlefield.DrawBattlefield(5, 5);
return;
```

- As variáveis currentBox que usam o comando “Find” estão em aspas duplas. Foram removidas já que é necessário apenas 1 par.

Classe Character / Método *CheckCloseTargets()*

- A comparação da linha 116 é errada: ele deve comparar se APENAS uma posição está sendo ocupada pelo inimigo próximo para começar a batalha e não todas as 4 ao mesmo tempo. A nova versão alterada ficou:

```
105         if (left || right || up || down)
106         {
107             Console.WriteLine("Hora da batalha!");
108             return true;
109         }
```

Classe Character / Método Attack*()*

- Foi adicionado uma variável que armazena o valor de dano causado ao inimigo, dando ao jogador o feedback de ação do personagem que antes não estava claro, já que o valor do dano é aleatório.

```
1 referência
public void Attack (Character target)
{
    var rand = new Random();
    int damageValue = rand.Next(0, (int)baseDamage);
    target.TakeDamage(damageValue);
    Console.WriteLine($"Player {name} está atacando {target.name} e deu {damageValue} de dano!\n");
    Console.WriteLine($"Health {name} | {health} --- Health {target.name} | {target.health}");
}
```

Refatoração - Correção do Script Grid.cs

———

Classe Grid / Método *Grid()*

- A função *grids.Add (newBox)* está no lugar indevido. O loop desse método adiciona as colunas primeiro e depois as linhas do campo. Essa função deve ficar no loop onde as colunas estão sendo inseridas:

```
20          for (int i = 0; i < Lines; i++)
21          {
22              for(int j = 0; j < Columns; j++)
23              {
24                  GridBox newBox = new GridBox(j, i, false, (Columns * i + j),"");
25                  grids.Add(newBox);
26              }
27          }
```

Classe Grid / Método *DrawBattlefield()*

- O nome do método *drawBattlefield()* está errado. Nomes de métodos devem ter letras maiúsculas no início.

```
            0 referências
37          public void DrawBattlefield(int Lines, int Columns)
```

- Foi adicionado um novo sistema para o método checar onde estão posicionados os personagens. A variável local **checkListGrid** recebe a index total da quantidade de elementos dentro da lista, e usando ela é possível percorrer todas as boxes com o loop **for** de forma mais prática.

```
33              int checkListGrid = 0;
```

- Dentro do loop **for** criador de colunas, cada Box é verificada se possui algum elemento **Character**. Na classe *Type / Grid Box* foi adicionado um novo atributo **charType,** que recebe o valor “HMN” ou “ENM” para classificar o jogador. No loop, se o parâmetro é obedecido, uma identificação é adicionada a Box do jogador.

```
35          for (int i = 0; i < Lines; i++)
36          {
37              for (int j = 0; j < Columns; j++)
38              {
39                  if (grids[checkListGrid].ocupied)
40                  {
41                      if (grids[checkListGrid].charType == "HMN")
42                      {
43                          Console.Write("[H]\t");
44                      }
45                      else
46                      {
47                          Console.Write("[I]\t");
48                      }
49
50                  }
51                  else
52                  {
53                      Console.Write($"[ ]\t");
54                  }
```

Refatoração - Correção do Script Program.cs

Linha 22

- Variável **numberOfPossibleTiles** sem utilização no momento, foi removida do código.

Classe Program / Método *CreatePlayerCharacter()*

- A variável name recebe o valor “HMN” para identificação do tipo do jogador.

```
66          PlayerCharacter.name = "HMN";
```

- Atributos do personagem não foram definidos ainda, estão usando valores padrões como teste.

```
63          PlayerCharacter.health = 100;
64          PlayerCharacter.baseDamage = 20;
65          PlayerCharacter.playerIndex = 0;
```

Classe Program / Método *CreateEnemyCharacter()*

- Variáveis **baseDamage** e **playerIndex** estavam referenciando o objeto **PlayerCharacter** estando no método **EnemyCharacter**. Foram trocadas para o objeto **EnemyCharacter.**
- A variável name recebe o valor “ENM” para identificação do tipo do jogador.

```
79          EnemyCharacter.baseDamage = 20;
80          EnemyCharacter.playerIndex = 1;
81          EnemyCharacter.name = "ENM";
```

Classe Program / Método *StartTurn()*

- A ação da condição *currentTurn == 0* está incorreta: não é possível organizar uma lista utilizando o comando Sort() sem um Interface de comparação **IComparable** previamente criada na classe.
- Nesse comparador if devemos não organizar os players, mas sim embaralhar todos eles para que suas ordens mudem no jogo. O comando Order By com uma função anônima consegue o mesmo resultado mas de forma mais simplificada:

```
100         Random rand = new Random();
101         var shuffle = AllPlayers.OrderBy(item => Guid.NewGuid()).ToList();
102         AllPlayers.Clear();
```

- Depois de armazenar e embaralhar todos os players, a lista All **Players** é limpa para que seja adicionados os Characters na ordem sorteada com um **foreach**:

```
100         Random rand = new Random();
101         var shuffle = AllPlayers.OrderBy(item => Guid.NewGuid()).ToList();
102         AllPlayers.Clear();
103
104         foreach (var value in shuffle)
105         {
106             AllPlayers.Add(value);
107         }
```

- Feedback de turno atual foi adicionado ao final do método mostrado ao jogador na tela:

```
117     currentTurn++;
118     Console.WriteLine($"---- FIM DO {currentTurn}° TURNO ----");
119     HandleTurn();
```

Classe Program / Método *HandleTurn()*

- Ao comparar se a vida do jogador “HMN” for igual a 0 o método *EndGame()* é chamado do elemento PlayerCharacter:

```
124                 if(PlayerCharacter.health <= 0)
125                 {
126                     PlayerCharacter.EndGame();
127                     return;
128                 }
```

- Ao comparar se a vida do jogador “HMN” for igual a 0 o método *EndGame()* é chamado do elemento EnemyCharacter:

```
129     else if (EnemyCharacter.health <= 0)
130     {
131         Console.Write(Environment.NewLine + Environment.NewLine);
132         EnemyCharacter.EndGame();
133         Console.Write(Environment.NewLine + Environment.NewLine);
134         return;
135     }
```

Classe Program / Método *AlocatePlayerCharacter()*

- A variável random não estava recebendo o valor do método *GetRandomInt*, fazendo o jogador nascer sempre na mesma posição. O código alterado ficou:

```
160             int random = GetRandomInt(0,grid.grids.Count - 1);
```

Classe Program / Método *AlocateEnemyCharacter()*

- A variável random não estava recebendo o valor do método *GetRandomInt*, fazendo o jogador nascer sempre na mesma posição. O código alterado ficou:

```
180        int random = GetRandomInt(0, grid.grids.Count - 1);
```

Refatoração - Correção do Script Types.cs

Classe Types / Método *GridBox()*

- Adicionado a variável charTtype para identificação do jogador: HUMANO ou INIMIGO:

```
19  public struct GridBox
20  {
21      public int xIndex;
22      public int yIndex;
23      public bool ocupied;
24      public int Index;
25      public string charType;
```