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 1.1.0

Melhorias de Versão

- Valores de X e Y do campo podem ser inseridos pelo usuário no console;
- Campo totalmente expansível mesmo não sendo matriz quadrada;
- Programa rodando sem nenhum erro ou aviso crítico;
- Código comentado para facilitar alterações.

Feature - Adição de customização de campo pelo usuário

Classe Program / Método *Setup()*

- Inicialmente o programa agora pergunta ao usuário os valores em X e Y para desenhar o campo de batalha no console.
- O campo só será desenhado se as medidas forem maiores que 1 x 1, senão o usuário deve recomeçar o processo. A nova versão alterada ficou assim:

```
30          void Setup()
31          {
32              Console.WriteLine("Bem vindo ao Auto Battle!\nEscolha a quantidade de linhas:");
33              valueLineX = Int32.Parse(Console.ReadLine());
34
35              if (valueLineX > 1)
36              {
37                  Console.WriteLine("Agora a quantidade de colunas:");
38                  valueLineY = Int32.Parse(Console.ReadLine());
39
40                  if (valueLineY > 1)
41                  {
42                      grid = new Grid(valueLineX, valueLineY);
43                      GetPlayerChoice();
44                  }
45                  else
46                  {
47                      Console.WriteLine("Quantidade insuficiente,escolha novamente...");
48                      Setup();
49                  }
50              }
51              else
52              {
53                  Console.WriteLine("Quantidade insuficiente,escolha novamente...");
54                  Setup();
55              }
56          }
```

Refatoração - Correção do Script Program.cs

Classe Program / Método *CreatePlayerCharacter()*

- Agora o método recebe os valores de skill de cada personagem e os envia diretamente para o jogador dependendo de sua escolha com os valores já definidos.
- As variáveis de skill são compartilhadas entre os personagens, facilitando a edição dos valores pelo desenvolvedor.
- O tipo do personagem (Humano ou Inimigo) também é mandado para o objeto Character para identificar seu tipo. A nova versão do código ficou assim:

```
24          var paladinSkills = new CharacterSkills("Paladino", 100, 30, 5);
25          var warriorSkills = new CharacterSkills("Guerreiro", 120, 40, 2);
26          var clericSkills = new CharacterSkills("Clerico", 90, 20, 5);
27          var archerSkills = new CharacterSkills("Arqueiro", 80, 10, 3);
```

```
92      switch (classIndex)
93      {
94          case 1:
95              PlayerCharacter.charSkillsValue = paladinSkills;
96              PlayerCharacter.playerType = "HMN";
97              classHMNSelected = 1;
98              break;
99          case 2:
100             PlayerCharacter.charSkillsValue = warriorSkills;
101             PlayerCharacter.playerType = "HMN";
102             classHMNSelected = 2;
103             break;
104         case 3:
105             PlayerCharacter.charSkillsValue = clericSkills;
106             PlayerCharacter.playerType = "HMN";
107             classHMNSelected = 3;
108             break;
109         case 4:
110             PlayerCharacter.charSkillsValue = archerSkills;
111             PlayerCharacter.playerType = "HMN";
112             classHMNSelected = 4;
113             break;
114     }
```

Classe Program / Método *CreateEnemyCharacter()*

- Foi adicionado um controle para que o inimigo não escolha o mesmo personagem do jogador. Caso isso ocorra, o método é novamente chamado.

```
125     if (randomInteger != classHMNSelected)
126     {
127         CharacterClass enemyClass = (CharacterClass)randomInteger;
128         Console.WriteLine($"Inimigo escolheu a classe: {enemyClass}");
129         EnemyCharacter = new Character(enemyClass);
130     }
131     else
132     {
133         CreateEnemyCharacter();
134     }
```

- O inimigo também recebe os valores de cada skill vindos das variáveis de controle assim como o jogador, facilitando sua edição:

```
136     switch (randomInteger)
137     {
138         case 1:
139             EnemyCharacter.charSkillsValue = paladinSkills;
140             EnemyCharacter.playerType = "ENM";
141             break;
142         case 2:
143             EnemyCharacter.charSkillsValue = warriorSkills;
144             EnemyCharacter.playerType = "ENM";
145             break;
146         case 3:
147             EnemyCharacter.charSkillsValue = clericSkills;
148             EnemyCharacter.playerType = "ENM";
149             break;
150         case 4:
151             EnemyCharacter.charSkillsValue = archerSkills;
152             EnemyCharacter.playerType = "ENM";
153             break;
154     }
```

Classe Program / Método *StartGame()*

- Cada jogador possui agora um método *DefineSkills()* dentro de *Character.cs* onde os valores vindo das variáveis do tipo **CharacterSkills** são repassados para o código de acordo com a classe escolhida.

```
25      public void DefineSkills()
26      {
27          name = charSkillsValue.name;
28          health = charSkillsValue.life;
29          baseDamage = charSkillsValue.damage;
30          damageMultiplier = charSkillsValue.damageMultiplier;
```

- Assim que o jogo é iniciado, o método de cada jogador é chamado dessa forma:

```
167         foreach(var chars in AllPlayers)
168         {
169             chars.DefineSkills();
170         }
```

Classe Program / Método *HandleTurn()*

- Agora a estrutura if compara se um jogador morreu e se o outro está vivo para evitar que o jogo continue com um novo round mesmo com um dos jogadores já derrotados.

```
202  void HandleTurn()
203  {
204      if(PlayerCharacter.health <= 0 && EnemyCharacter.health > 0)
205      {
206          Console.Write(Environment.NewLine + Environment.NewLine);
207          EnemyCharacter.EndGame();
208          Console.Write(Environment.NewLine + Environment.NewLine);
209          return;
210      }
211      else if (EnemyCharacter.health <= 0 && PlayerCharacter.health > 0)
212      {
213          Console.Write(Environment.NewLine + Environment.NewLine);
214          PlayerCharacter.EndGame();
215          Console.Write(Environment.NewLine + Environment.NewLine);
216          return;
217      }
218      else
219      {
220          Console.Write(Environment.NewLine + Environment.NewLine);
221          Console.WriteLine("Clique em qualquer tecla para começar o próximo turno...\n");
222          Console.Write(Environment.NewLine + Environment.NewLine);
223          ConsoleKeyInfo key = Console.ReadKey();
224          StartTurn();
225      }
226  }
```

Classe Program / Método *AlocatePlayers()*

- O método foi excluído, já que ele apenas chamava outro método que pode ser chamado diretamente dentro de *StartGame()*.

Classe Program / Método *AlocatePlayerCharacter()* e *AlocateEnemyCharacter()*

- O valor **charType** agora recebe o atributo playerType dentro do objeto **Character** ao invés do nome do personagem.

```
264  RandomLocation.charType = EnemyCharacter.playerType;
```

```
244  RandomLocation.charType = PlayerCharacter.playerType;
```

Refatoração - Correção do Script Character.cs

Classe Character / Linhas 15 e 17

- Foram adicionadas duas novas variáveis:
  - **playerType**: identifica se o jogador é humano ou máquina;
  - **charSkillsValue**: recebe os atributos da struct **CharacterSkills** do script *Types.cs* para armazenar os valores de skill da classe escolhida no script *Program.cs.*

Classe Character / Método *StartTurn()*

- Agora o método *DrawBattlefield()* de cada condição não precisa mais de parâmetros para ser desenhado: os valores de X e Y já são passados e armazenados dentro do script *Grid.cs* para facilitar o uso e alterações no programa.

```
63      battlefield.DrawBattlefield();
```

- As linhas 83 e 95 comparavam erroneamente a index atual da current box no eixo X do battlefield, sendo que o correto é comparar as posições da box no eixo Y de cada condição:

```
83      currentBox = battlefield.grids.Find(x => x.Index == currentBox.Index - battlefield.yLength);
95      currentBox = battlefield.grids.Find(x => x.Index == currentBox.Index + battlefield.yLength);
```

Classe Character / Método *CheckCloseTargets()*

- As variáveis up e down recebiam o valor **ocupied** comparado as boxes de **Xlength**, ou seja, do eixo X. Mas o certo é usar o **YLength** já que elas trabalham com o eixo Y para verificar a posição do personagem no campo:

```
110     bool up = battlefield.grids.Find(x => x.Index == currentBox.Index + battlefield.yLength).ocupied;
111     bool down = battlefield.grids.Find(x => x.Index == currentBox.Index - battlefield.yLength).ocupied;
```

Classe Character / Método *Attack()*

- O método *TakeDamage()* agora recebe como parâmetro o multiplicador do dano da classe escolhida:

```
125     target.TakeDamage(damageValue * damageMultiplier);
```

Refatoração - Correção do Script Types.cs

Classe Types / Método *CharacterClassSpecific()*

- Método excluído já que o outro método **CharacterSkills** já identificava os atributos necessários do personagem.

Classe Types / Método *CharacterSkills()*

- Adicionado todos os atributos necessários para as classes do jogo. Com essa struct é possível remover e/ou adicionar mais atributos se necessário aos personagens e depois apenas alterar seus valores dentro das variáveis no script *Program.cs*:

```
27  public struct CharacterSkills
28  {
29      public string name;
30      public float life;
31      public float damage;
32      public float damageMultiplier;
33
        4 referências
34      public CharacterSkills(string nameChar, float lifeChar, float damageChar, float damageMultChar)
35      {
36          name = nameChar;
37          life = lifeChar;
38          damage = damageChar;
39          damageMultiplier = damageMultChar;
40      }
41  }
```