Code Correction – errors found

1. Namespaces missing or variable types not correctly declared (c2061)
2. The Character class was missing some properties, so I added them manually.
   1. Name was missing.
   2. Duplicated gridbox property fixed, removed one of them and fixed the missing Types class reference.
   3. Removed duplicated reference to target Property.
   4. Cleaned unused #includes
3. Inconsistent code formatting, there was no standardization on case type, sometimes camelCase was used, other times Pascal case.
   1. Referred to this [documentation](https://lefticus.gitbooks.io/cpp-best-practices/content/03-Style.html) to make the code refactor to camelCase and snake\_case.
   2. There were also spelling mistakes:Texto

      Descrição gerada automaticamente com confiança média
   3. I have gone on every class fixing spelling mistakes and code conventions.
   4. BattfleField class was misspelled, the right name should be Battlefield.
4. Removed namespace refs from header files, it is a good practice to avoid unwanted conflicts.
5. Removed the unnecessary properties of the battlefield class, there is no need to have a pointer to each player when we have a list of all players that we could iterate as needed.
6. The character function TakeDamage was receiving a variable that was not being used, I fixed the issue by using that variable as the value of damage the player will be inflicted.
7. Repeated includes on the character class, I removed the repeated ones as there was no need for 2 includes of the same class.
8. Each class should have responsibilities only to itself, so there is no sense in the Battlefield class having a return random integer to other classes, this is a principle of clean code.
   1. That is also valid for the Create Character and create enemy character, also both of them will be refactored in a create player function se we can re-use the same function every time.
   2. Created a turn handler so we can segregate responsibilities from each class and organize better how the game will flow.
   3. Removed Drawn Battlefield from Grid class, as that should be a function of battlefield.
   4. Removed turn handling from battlefield, so we can give that responsibility to a class that is designed to handle turns.
9. Refactored how the player management work, so it is simpler and more efficient.
10. There were many functions that could be simplified and deleted, the allocate enemy character and allocate player character on battlefield were deleted and I created a AlocatePlayer, this made the code cleaner, and we didn’t repeat code unnecessarily.
11. In the character class, the switch used to define what was going to be the char class was compering strings, that is not optimal because the switch case needs a number to work properly.
12. Updated some recursive functions to improve performance, recursive functions usually create a stack that while repetition loops don’t.
13. One of the things that were not being used is pass by reference, this way we can greatly reduce the memory cost and it is easier to read and use than pointers
14. Used references whenever possible, they help with reducing memory costs and make the application faster.
15. I tried to refactor all the classes utilizing SOLID and OOP, the main refactors on the code revolve around setting the responsibilities of each class to methods and properties that make sense in the context of the class.

Code Review – Changes

I believe that I refactored the whole project, mainly because there where so many errors and malpractices that only fixing them would not be worth it, the code would continue to be wrong on the basic concepts of OOP and SOLID, even if it works after the fixes, it would be a hard to maintain and understand code.

Documenting every change on the code proved to be chaotic and not intuitive, so I will explain everything on this document.

1. The Character class:
   1. Had a lot of commented out code, missing variables and code that should no be there.
   2. I added all the variables requested by the challenge.
   3. I divided the move function, so we have a easier time making changes on movement.
   4. I added the enum to the character class as it is something related to this class.
   5. I am also planning on implementing the strategy pattern to create each character, I want to use the Character class as an interface where we define the properties and methods that should be implemented by the child classes, in this case one for each character class. This way I could make each class unique and interesting while making all interchangeable.
   6. I implemented a factory method to create character so we can centralize the creation and have a more streamlined process.