Agustín Amenábar Ramírez

∑ Nota Final	4.759201613716317
∑ Puntaje Tests	6
∑ Puntaje Clean Code	2.55
∑ Puntaje Clean Code (Escalado)	3.775
# Failed E1-BasicCombat	0
# Failed E1-InvalidTeams	0
# Failed E2	0
# Failed E2-Mix	0
# Failed E2-Random	0
# Failed E3	0
# Failed E3-Mix	0
# Failed E3-Random	0
# Cap. 2	1.5
# Cap. 3	1.95
# Cap. 4	2
# Cap. 5	1.75
# Cap. 6	1.5
# Cap. 7	1.85
# Cap. 8	2
# Cap. 10	1
# MVC	0

Clean Code

Capítulo 2: Meaningful Names 😃 `1.5/2.0`

Los siguientes números/strings/bools deberían estar encapsulados en variables, no “sueltos” en el código, de manera que se facilite su búsqueda:

- <Unit.cs, línea 86>: `<string>`. Utilizas string del tipo `Weapon == "Sword" && rivalWeapon == "Axe"` , en vez de encapsularlo en algún Enum o variable [**-0.05** puntos].
- <Skills.cs> `<número>` Tienes muchas constantes sin encapsular en este archivo (Se repiten muchos números). Podrías encapsularlos en alguna variable y después colocarla según corresponda [**-0.45** puntos].

[**-0.05** puntos cada una, **-0.5** puntos máximo, **-0.5** puntos en total]

Capítulo 3: Functions 📈 `1.95/2.0`

Los siguientes métodos hacen más de una sola cosa:

- <Battle.cs>: `<ChooseUnits>`. Por el nombre del método, sólo debería de elegir el unit, pero también lo setea (con una función). Los Set, deberían de estar en el método Round [**-0.05** puntos]

Capítulo 4: Comments 🖨️ `2.0/2.0`

Se cumple con todo 😊.

Capítulo 5: Formatting 🗃️ `1.75/2.0`

Las siguientes líneas exceden el máximo de 120 carácteres:

- <DamageManager>: línea 43, 58 y 61.
- <HpRange.cs>: línea 18.
- <ExtraDamageInFirstAttack>: línea 25.

[**-0.05** puntos cada uno, **-0.5** puntos de tope, **-0.25** puntos en total]

Capítulo 6: Objects and Data Structures 🏗️ `1.5/2.0`

Las clases deben representar un objeto, escondiendo su estructura interna y exponiendo abstracciones, o una estructura de datos, que expone su estructura interna sin implementar funcionalidades importantes, pero no ambas. Las siguientes corresponden a clases híbridas:

- `<Skill>`. Tiene lógica para setear las skills y tiene lógica para obtener las condiciones y efectos. La utilizas como EDD.
- `<Unit>` . La utilizas como una EDD y le tienes lógica del juego (Ataque, defense, etc).

  [**-0.2** puntos cada una, **-1.0** puntos máximo, **-0.4** puntos en total]

Se encontraron los siguientes trainwrecks en el código:

- <Team.cs, línea 19, 24>.

  [**-0.05** puntos cada uno, **-0.5** puntos máximo, **-0.1** puntos en total]

Capítulo 7: Error Handling ⚠ `1.85/2.0`

Los siguientes métodos retornan o reciben null:

- <NeutralizePenalty.cs>: El constructor recibe un null.
- <NeutralizeBonus.cs>: El constructor recibe un null.

  [**-0.05** puntos cada uno, **-0.5** puntos máximo, **-0.1** puntos en total]

Los bloques try/catch solo deben tener un único catch, que reciba una excepción personalizada:

- <Battle.cs>: En la función Start, el try tien dos catch.

  [**-0.05** puntos cada uno, **-0.5** puntos de tope, **-0.05** puntos en total]

Capítulo 8: Boundaries 🚧 `2.0/2.0`

Se cumple con todo 😊.

Capítulo 10: Classes 👨‍🏫 `1.0/2.0`

Las siguientes clases tienen más de una responsabilidad:

- <Battle>: Tiene las responsabilidades <setear, printear y gestionar>, cuando solo debería de hacer de Controller.
- <Game>: Tiene las responsabilidades <printear, elegir equipos, setear valores, validación de los equipos y gestiona el juego>
- <Unit>: Tiene las responsabilidades <printear y gestión de los atributos de Unit>

  [**-0.1** puntos cada responsabilidad extra, **-1.0** puntos en total]

MVC 🪟 `0.0/1.0`

- Se utiliza directamente el método view.WriteLine() en el controlador sin encapsular los anuncios en consola en sus propios métodos del proyecto vista [**-0.5** puntos].
- No se distinguen las clases qué clases pertenecen a cada tipo de proyecto. Si bien tienes tres proyectos, uno de Controller, FireEmblem (Model) y el otro de View, en los proyectos mezclas estos 3 conceptos [**-0.1** puntos].
- En el proyecto de la vista se utilizan métodos del modelo los cuales editan los datos [**-0.5** puntos].