Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
IIC2113 Diseño Detallado de Software

Entrega 1: Fire Emblem

Francisco Ignacio Gazitúa Requena
Cristian Andrés Hinostroza Espinoza
Estefania Mata Uri-Uri Pakarati Cofré

Introducción

En esta entrega debes implementar el flujo básico del juego, que abarca la totalidad del juego sin considerar las habilidades (desde la Sección 1 hasta la Sección 6 del enunciado del proyecto). Esto incluye implementar:

1. La validación del juego.
2. El flujo de turnos de los jugadores.
3. Todas las unidades.
4. Cálculos de daño considerando tipos de armas, tipos de daño y ventaja del triángulo de armas.
5. Flujo completo del combate, considerando todos los ataques (ataque, contraataque y Follow-Up) y cuando termina un combate.

Test cases

El código base incluye todos los tests que debes completar hasta el final del proyecto. Sin embargo, en esta entrega solo debes pasar los siguientes tests:

- E1-BasicCombat.
- E1-InvalidTeams.

Cada test consiste en un archivo de texto que contiene el input y output esperado de tu programa. Todos los test cases se encuentran en la carpeta `data.zip`. Por ejemplo, el siguiente es una versión ligeramente modificada del test case ubicado en `data/E1-BasicCombat-Tests/000.txt`:

```
Elige un archivo para cargar los equipos
0: 000.txt
1: 001.txt
2: 002.txt
3: 003.txt
4: 004.txt
5: 005.txt
INPUT: 0
Player 1 selecciona una opción
0: Sigurd
INPUT: 0
Player 2 selecciona una opción
0: Celica
1: Veronica
2: Robin
INPUT: 2
Round 1: Sigurd (Player 1) comienza
```

```
Ninguna unidad tiene ventaja con respecto a la otra
Sigurd ataca a Robin con 27 de daño
Robin ataca a Sigurd con 24 de daño
Robin ataca a Sigurd con 24 de daño
Sigurd (6) : Robin (27)
Player 2 selecciona una opción
0: Celica
1: Veronica
2: Robin
INPUT: 1
Player 1 selecciona una opción
0: Sigurd
INPUT: 0
Round 2: Veronica (Player 2) comienza
Ninguna unidad tiene ventaja con respecto a la otra
Veronica ataca a Sigurd con 29 de daño
Veronica (49) : Sigurd (0)
Player 2 ganó
```

Tu programa debe generar el mismo output que aparece en el test para estar correcto. Notar que el test incluye el keyword "`INPUT: `" en algunas líneas (e.j., en la línea 8). Esto indica que hay que pedir un input al usuario (en vez de escribir algo) y el input ingresado por el usuario será el número que aparece luego de "`INPUT: `". Por ejemplo, si "`INPUT: 2`" entonces el número ingresado por el usuario será `2`.

Los tests de esta entrega comprueban que se cumplan dos escenarios.

El primer escenario consiste en verificar que el equipo elegido por cada jugador sea válido. Recordar que un equipo es inválido cuando tiene unidades repetidas, una unidad con más de dos habilidades, una unidad con habilidades repetidas, equipos vacíos o equipos de más de 3 unidades.

Si un equipo es inválido, se muestra un mensaje indicando que la selección es inválida y termina el programa:

```
Elige un archivo para cargar los equipos
0: 01.txt
1: 02.txt
2: 03.txt
3: 04.txt
INPUT: 3
Archivo de equipos no válido
```

El segundo escenario es que ambos equipos ingresados sean válidos. Si esto ocurre, el juego comenzará, momento en el que los jugadores podrán escoger qué unidades utilizarán para atacar y defender según sea el caso. El juego continuará normalmente hasta que uno de los jugadores quede sin unidades vivas en su equipo. Este caso será explicado con más detalle en las siguientes secciones.

Formato equipos

Lo primero que debe hacer tu programa es pedirle al usuario que seleccione un equipo. Cada grupo de test cases tiene un conjunto de equipos posibles diseñados para verificar el correcto funcionamiento de tu programa. Los equipos también se encuentran en `data.zip`. Por ejemplo, los equipos usados en los test cases `E1-BasicCombat` están en `data/E1-BasicCombat/`.

Para saber qué equipos se pueden utilizar, la clase `Game.cs` (que es la entrada a tu programa) recibe en su constructor el parámetro `teamsFolder`. Este parámetro contiene la ruta a la carpeta con todos los equipos disponibles según el test case que se esté ejecutando. Esta carpeta tendrá un archivo `.txt` por cada equipo posible. Los equipos deben ser mostrados al usuario y luego se le debe pedir como input que elija alguno de ellos. En el siguiente ejemplo, hay 6 equipos posibles y el usuario elige el equipo `0`:

```
Elige un archivo para cargar los equipos
0: 000.txt
1: 001.txt
2: 002.txt
3: 003.txt
4: 004.txt
5: 005.txt
INPUT: 0
```

Luego de elegido el equipo hay que leer el archivo y ver si el equipo es válido. Los archivos de equipos tienen el siguiente formato. La primera línea indica que comienza la sección del primer jugador, seguida inmediatamente de los nombres de sus unidades. Luego de las unidades del primer jugador habrá una línea indicado el inicio de la sección del segundo jugador, seguida por el nombre de las unidades de su equipo. Por ejemplo, este es un posible equipo:

```
Player 1 Team
Lyn (Def/Res Push)
Player 2 Team
Edelgard (Swift Sparrow,Eclipse Brace)
Veronica
Sigurd
```

En caso que una unidad tenga habilidades, estas se encuentran en la misma línea del nombre de la unidad entre paréntesis. Si una unidad tiene más de una habilidad, estas se encontrarán dentro del paréntesis delimitadas por una coma sin espacio entre ellas. En el ejemplo anterior, Edelgard tiene las habilidades Swift Sparrow y Eclipse Brace, Lyn tiene la habilidad Def/Res Push y Veronica con Sigurd no tienen habilidades.

Formato Habilidades

La información de las habilidades del juego está en el archivo: `skills.json`. Por cada habilidad se indica su nombre y su descripción. Sin embargo, en este entrega solo usaremos el nombre de la habilidad para verificar si un equipo es válido. La lógica para implementar cada habilidad será agregada en las siguientes entregas.

```
[
    {...},
    {
        "Name": "Sword Agility",
        "Description": "Si la unidad usa espada, otorga Spd+12 a un costo de Atk-6 durante
    el combate"
    },
    {
        "Name": "Tome Precision",
        "Description": "Otorga Atk/Spd+6 al usar magia."
    },
    {
        "Name": "Resonance",
        "Description": "Si la unidad usa magia, si HP de la unidad >=2, la unidad pierde 1
    HP al inicio del combate e inflige +3 de daño por ataque durante el combate."
    },
    {
        "Name": "Lance Power",
        "Description": "Otorga Atk+10 a un costo de Def-10 al usar una lanza"
    },
    {...}
]
```

Formato Unidades

La información de las unidades del juego se encuentra en el archivo `characters.json`. Los atributos de cada unidad te permitirán computar cuánto daño realizan al rival durante sus batallas (entre otras cosas).

```
[
  {...},
  {
    "Name": "Marth",
    "Weapon": "Sword",
    "Gender": "Male",
    "DeathQuote": "Forgive me, my friends...",
    "HP": "54",
    "Atk": "62",
    "Spd": "53",
    "Def": "43",
    "Res": "37"
  },
  {
    "Name": "Tiki",
    "Weapon": "Magic",
    "Gender": "Female",
    "DeathQuote": "No...I don't want to be...alone...",
    "HP": "59",
    "Atk": "61",
    "Spd": "46",
    "Def": "47",
    "Res": "39"
  },
  {
    "Name": "Alm",
    "Weapon": "Bow",
    "Gender": "Male",
    "DeathQuote": "No... Not when we were...so close... Forgive me, Celica...",
    "HP": "54",
    "Atk": "62",
    "Spd": "51",
    "Def": "36",
    "Res": "32"
  },
  {...}
]
```

Output del juego

El programa siempre comienza mostrando los equipos que se pueden elegir dentro de la carpeta `teamsFolder`. Luego de ello, si el equipo es inválido, se notifica al usuario y termina el programa:

```
Elige un archivo para cargar los equipos
0: 01.txt
1: 02.txt
2: 03.txt
3: 04.txt
INPUT: 3
Archivo de equipos no válido
```

En el caso contrario, comenzará una batalla entre los equipos del *Player 1* y del *Player 2*, donde siempre parte atacando *Player 1*. Se le pregunta a este último qué unidad desea usar para atacar y luego se le pregunta al segundo jugador qué unidad usará para defenderse. En el siguiente ejemplo, *Player 1* ataca con *Hector* y *Player 2* defiende con *Marth*:

```
Player 1 selecciona una opción
0: Hector
INPUT: 0
Player 2 selecciona una opción
0: Marth
1: Eliwood
INPUT: 0
```

Notar que solo deben aparecer como unidades seleccionables, tanto para atacar como defender, las unidades del equipo que siguen vivas. Luego de seleccionar las unidades, el combate ocurre de forma automática.

En cada ronda, se anuncia el número de la ronda, quién inicia el combate, qué unidad tiene ventaja del triángulo de armas, cada uno de los ataques y, finalmente, el HP restante de cada unidad:

```
Player 1 selecciona una opción
0: Hector
INPUT: 0
Player 2 selecciona una opción
0: Marth
1: Eliwood
INPUT: 0
Round 1: Hector (Player 1) comienza
Marth (Sword) tiene ventaja con respecto a Hector (Axe)
Hector ataca a Marth con 10 de daño
Marth ataca a Hector con 22 de daño
Marth ataca a Hector con 22 de daño
Hector (17) : Marth (44)
```

Al momento de anunciar la ventaja del triángulo de armas, las opciones de texto son las siguiente:

- "*Ninguna unidad tiene ventaja con respecto a la otra*".
- "Unidad1 (Arma1) *tiene ventaja con respecto a* Unidad2 (Arma2)".

Además, para esta entrega, existen 3 ataques posibles por combate: ataque, contraataque y follow-up. En el ejemplo anterior, el último ataque de *Marth* es un follow-up. En caso de que ninguna de las unidades puede hacer un follow-up se debe mostrar el siguiente mensaje:

- "*Ninguna unidad puede hacer un follow up*".

El siguiente ejemplo muestra un caso donde no hay ventajas del triángulo de armas ni Follow-Ups:

```
Round 2: Dimitri (Player 2) comienza
Ninguna unidad tiene ventaja con respecto a la otra
Dimitri ataca a Claude con 35 de daño
Claude ataca a Dimitri con 14 de daño
Ninguna unidad puede hacer un follow up
Dimitri (39) : Claude (19)
```

Luego de que finaliza un combate, se muestra el HP restante de cada unidad y se vuelven a elegir unidades. Esta vez, el segundo jugador escoge la unidad que ataca y el primer jugador la unidad que defiende. El combate continuará intercalando quién ataca y quién defiende hasta que un jugador pierda a todas sus unidades. Al terminar el juego se anuncia al ganador – tal como muestra el siguiente ejemplo:

```
Player 1 selecciona una opción
0: Hector
INPUT: 0
Player 2 selecciona una opción
0: Marth
1: Eliwood
INPUT: 0
```

```
Round 1: Hector (Player 1) comienza
Marth (Sword) tiene ventaja con respecto a Hector (Axe)
Hector ataca a Marth con 10 de daño
Marth ataca a Hector con 22 de daño
Marth ataca a Hector con 22 de daño
Hector (17) : Marth (44)
Player 2 selecciona una opción
0: Marth
1: Eliwood
INPUT: 0
Player 1 selecciona una opción
0: Hector
INPUT: 0
Round 2: Marth (Player 2) comienza
Marth (Sword) tiene ventaja con respecto a Hector (Axe)
Marth ataca a Hector con 22 de daño
Marth (44) : Hector (0)
Player 2 ganó
```

Finalmente, si una unidad es derrotada antes que se completen todos los ataques, se interrumpirá el combate y se mostrará de inmediato el estado final de la ronda. Esto de la siguiente manera:

```
Round 2: Dimitri (Player 2) comienza
Ninguna unidad tiene ventaja con respecto a la otra
Dimitri ataca a Claude con 35 de daño
Dimitri (39) : Claude (0)
```

En el ejemplo anterior, el HP de *Claude* llegó a 0 en el primer ataque de *Dimitri*, por lo que no sucede el contraataque de *Claude* y no se anuncia que ninguna unidad puede hacer un Follow-Up. Además, notar que el daño anunciado es el daño total realizado – independiente de la vida restante del oponente. Es decir, aunque el HP restante de *Claude* haya sido 10, se anuncia que recibe un daño de 35.

Cálculo de daño

Tal como indica el enunciado general del proyecto, los cálculos de daño pueden generar números decimales. Cuando ello ocurre, hay que truncar el número a su entero más bajo. Esto se puede realizar en C# utilizando la función `Math.Floor(...)`. Luego el resultado puede ser convertido a entero con `Convert.ToInt32(...)`.

Input-Output

En tu proyecto **NO** debes usar `Console.WriteLine(...)` ni `Console.ReadLine()` para mostrar y pedir texto al usuario. Esto se debe a que nuestro código para comparar la salida de tu programa con los test cases ignora los mensajes mandados directamente a consola.

Para que el input-output de tu programa sea verificado por nuestros test cases debes usar el objeto `view` que te entregamos en el constructor de `Game.cs`. Ese objeto tiene los siguientes métodos:

- `ReadLine()`: Solicita un string al usuario y retorna el `string` correspondiente.
- `WriteLine(string message)`: Muestra `message` en consola.

El objeto `view` hace dos cosas. Por un lado, guarda los mensajes que se escriben mediante su método `WriteLine(...)`. Esos mensajes son comparados con el test case para verificar si tu programa está correcto. Por otro lado, cuando se llama a `ReadLine()` automáticamente retorna el `INPUT:` indicado en el test case.

En resumen, todo input pedido y mensaje mostrado mediante `Console` es ignorado al momento de evaluar los test cases. Si quieres que un input o texto sea considerado debes utilizar los métodos de `view`.

Rúbrica

Para evaluar tu entrega usaremos 2 grupos de test cases: `E1-InvalidTeams` y `E1-BasicCombat`.

Para calcular tu nota se le asignará a cada grupo de test un puntaje máximo, el cual será el limite superior del puntaje que obtendrás en dicho grupo de tests; el puntaje que obtengas variará proporcionalmente a la cantidad de tests del grupo que pases. Junto a esto, cada grupo de tests tendrá un puntaje asociado que solo se otorgará si es que tu código pasa todos los tests del grupo. Los puntajes se distribuyen de la siguiente manera:

- [**1.5 puntos**] Porcentaje de test cases pasados en `E1-InvalidTeams`.
- [**0.5 puntos**] Pasar todos los test cases en `E1-InvalidTeams`.
- [**3.5 puntos**] Porcentaje de test cases pasados en `E1-BasicCombat`.
- [**0.5 puntos**] Pasar todos los test cases en `E1-BasicCombat`.

Por ejemplo, digamos que tu entrega pasa todos los test cases `E1-InvalidTeams` y el 80 % de los test cases `E1-BasicCombat`. Entonces tu nota será: $1{,}5 + 0{,}5 + 3{,}5 \cdot 0{,}8 + 1 = 5{,}8$ (con punto base).

Importante: No está permitido modificar los test cases ni el proyecto `Fire-Emblem.Tests`. Hacerlo puede conllevar una penalización que dependerá de la gravedad de la situación