Programación de Sistemas de Telecomunicación
Práctica 1: Universidades y Mazmorras

GSyC

Octubre de 2020

1. Introducción

En esta práctica debes realizar un programa en el que implementarás el juego Universidades y Mazmorras.

El programa simulará un juego en el que entre 1 y 4 jugadores deberán superar entre 1 y 10 niveles en los que se enfrentarán a diversos enemigos a los que derrotar. Los jugadores ganarán si al menos uno de los jugadores permanece con vida tras derrotar a los enemigos del último nivel y perderán si todos los jugadores pierden todos sus puntos de vida en alguno de los niveles.

2. Argumentos

Al ejecutar el programa se podrán pasar opcionalmente los siguientes arguments

`-p` Este argumento irá acompañado de un número entero (Por ejemplo -p 3) y permitirá establecer el número de jugadores de la partida. En el ejemplo propuesto, habrá un total de 3 jugadores en la partida. El programa también deberá permitir recibir como argumento `--players` (–players=3). En ningún caso el número de jugadores podrá ser inferior a 1 ni superior a 4. Tampoco se permiten valores no enteros para dar valor a este argumento.

`-s` Este argumento irá acompañado de un número entero (Por ejemplo -s 5) y permitirá establecer el número de niveles del juego. En el ejemplo propuesto,los jugadores tendrán que superar un total de 5 niveles. El programa también deberá permitir recibir como argumento `--stages` (–stages=5). En ningún caso el número de niveles puede ser inferior a 1 o superior a 10. Tampoco se permiten valores no enteros para dar valor a este argumento.

El programa por lo tanto debe poder iniciarse de esta forma.

- `python main.py -p 3 -s 5`
  Deberá iniciar una nueva partida para 3 jugadores con 5 niveles.
- `python main.py --players=3 -s 5`
  Deberá iniciar una nueva partida para 3 jugadores con 5 niveles.
- `python main.py --players=3 --stages=5`
  Deberá iniciar una partida para 3 jugadores con 5 niveles.
- `python main.py`
  Deberá iniciar una nueva partida para un jugador con un nivel.
- `python main.py -p 5 -s 3`
  Debe avisar de que el número de jugadores es incorrecto y finalizar la partida.
- `python main.py -p 5 -s 11`
  Debe avisar de que el número de jugadores y el número de niveles son incorrectos y finalizar la partida.

Si el argumento no recibiera un número o el número recibido fuera incorrecto según las especificaciones anteriones, el programa lanzara una excepción y terminará.

3. Elementos del juego

En esta sección se definen las diferentes opciones existentes para los jugadores así como los enemigos que pueden aparecer en cada nivel.

3.1. Personajes

- `Bookworm` tiene las siguientes carácteristicas:
  - Stats: 25 HP, 9 DM
  - Habilidad: Una vez cada 4 rondas puede resucitar a otro jugador que haya perdido todos los puntos de vida. Si se utiliza la habilidad y no hay ningún jugador sin puntos de vida no cuenta como usada.
- `Worker` tiene las siguientes carácteristicas:
  - Stats: 40 HP, 10 DM
  - Habilidad:(Daño base + tirada de daño) * 1.5 a un enemigo cada 3 rondas.
- `Procrastinator` tiene las siguientes características:
  - Stats: 30 HP, 6 DMG.
  - Pasiva: Cada ronda hace suma +1 al daño realizado (Ronda 1, +0 de daño; Ronda 2, +1 de daño; Ronda 3, +2 de daño). Al comenzar cada nivel el bonus vuelve a 0. Habilidad: Daño base + tirada de daño + nivel de la partida a todos los enemigos a partir de la tercera ronda de cada nivel y una vez por nivel.
- `Whatsapper` tiene las siguientes características:
  - Stats: 20 HP, 6 DMG.
  - Habilidad: Cura 2 * Daño base a cualquier jugador cada 3 rondas.

3.2. Enemigos

- `Partial Exam` tiene las siguientes carácteristicas:
  - Stats: 20 HP, 6 DM
- `Final Exam` tiene las siguientes carácteristicas:
  - Stats: 40 HP, 12 DM
  - Regla especial: Sólo puede aparecer a partir del 4º nivel.
- `Theoretical class` tiene las siguientes características:
  - Stats: 8 HP, 4 DMG.
  - Regla especial: Suma + 1 al daño según el nivel (nivel 1, +1; nivel 2, +2, nivel 3, +3)
- `Teacher` tiene las siguientes características:
  - Stats: 15 HP, 7 DMG.
  - Regla especial: Si su tirada de daño es 7, hace 14 de daño en su lugar.

3.3. Notas útiles

Los siguientes puntos son recomendables pero no obligatorios (su seguimiento o no no afecta a la nota de la práctica).

- Crear un diccionario dentro de la clase `Character` con las estadísticas actuales del personaje (dmg y hp) te puede ayudar a controlar los valores de dichos elementos en todo momento. Si añades además una clave para almacenar el enfriamiento de la habilidad y si el personaje está vivo o derrotado también podrás controlar el funcionamiento de las diversas habilidades existentes.
- Recuerda como funciona la herencia. Será muy probable que la mayoría de las funciones que necesites sólo tengan que implementarse en la clase padre `Character`.
- Puesto que los `Enemigos` (Enemies) no modifican sus valores a lo largo de un nivel (no se curan, no reviven, etc) no es necesario que apliques la misma mecánica que en el punto anterior. Simplemente recuerda aplicar sus características especiales según toque.
- Si necesitas saber el nombre de la clase de una instanciación de dicha clase para realizar ciertas acciones, utiliza el atributo `.__class__.__name__`, que devuelve el nombre de la clase.
- Tal y como se especificará en la siguiente sección, se generarán 4 enemigos aleatorios por ronda. Te ayudará guardar esos enemigos en una lista dentro de la clase `Game` que servirá como punto de gestión y control de todo lo que tiene que suceder en la partida.

4. Ejecución

A continuación se explican las mecánicas del juego:

- Según el número de jugadores se eligen los personajes. Pueden elegirse personajes repetidos.
- Una vez elegidos los personajes, empieza el primer nivel:
- Se generan aleatoreamente 4 enemigos, pudiendo repetirse (Pueden generarse 4 exámenes parciales en el mismo nivel).
- Se muestra en la terminal el nivel en el que se está actualmente y los enemigos que se han generado.
- Empiezan atacando los jugadores. Pueden elegir atacar (a) o usar una habilidad(s) si no estuviera en enfriamiento. Para atacar se hace una tirada de daño. Una tirada de daño consiste en un número aleatorio entre 1 y el daño del personaje. Por ejemplo, en el caso del personaje Worker la tirada de daño consistiría en un número aleatorio entre 1 y 10.
- Si tras el turno de los jugadores quedasen enemigos con vida, atacan los enemigos restantes.
- El objetivo del ataque, tanto para los jugadores como para los enemigos es aleatorio.
- Tras el turno de los enemigos acaba la ronda (A efectos de enfriamientos, reglas especiales, etc).
- Si tras el turno de los enemigos quedan jugadores con vida, se repite el proceso a partir del punto 4. Si no quedasen jugadores con vida el juego terminaría, con la consiguiente derrota de los jugadores.
- Si los jugadores vencen a todos los enemigos del nivel:
    - Si se ha superado el número de niveles establecidos, los jugadores ganan.
    - Si no se ha superado el número de niveles establecidos, se genera un nuevo nivel y se repite el proceso a partir del punto 3, aumentando en 1 el nivel. Cada jugador se cura 1/4 de la vida del personaje que haya elegido.

4.1. Ejemplos de ejecución

```
$ ./python main.py --players=5
The number of players must be between 1 and 4. Finishing program.

$ ./python main.py --stages=12
The number of stages must be between 1 and 10. Finishing program.

$ ./python main.py --players=5 --stages=12
The number of players must be between 1 and 4. The number of stages must be between 1 and 10. F


$ ./python main.py
A game with one stage will be set up for one player.
***********       AVAILABLE CHARACTERS       ***********
1.- The bookworn -> Stats: 25HP and 9DMG
        Skill: Revives one player(4 rounds)
2.- The worker -> Stats: 40HP and 10DMG
        Skill: 1.5 * (DMG + DMG roll) damage to one enemy (3 rounds)
3.- The whatsapper -> Stats: 20HP and 6DMG
        Skill: Heals 2*DMG to one player (3 rounds)
4.- The procrastinator-> Stats: 30HP and 6DMG
        Passive: Adds +1 DMG each round. Resets at the beginning of each level.
        Skill: DMG + DMG roll + stage level to all the enemies
               after the third round of each stage and once per stage.
*****************************************************
Player 1. Please, choose a character (1-4): 2

            ************************
            *       STAGE 1        *
            ************************
            ---- CURRENT MONSTERS ----
      ++++++++++++++++++++++++++++++++++++
       Partial exam: Stats: 10HP and 12DMG
        Partial exam: Stats: 6HP and 3DMG
        Partial exam: Stats: 6HP and 3DMG
      ++++++++++++++++++++++++++++++++++++

               ------------------------
               -     PLAYERS TURN     -
               ------------------------

Worker (Player 1). What are you going to do?: a
The Worker (Player 1) did 2 damage to Partial Exam. Partial Exam has 8 hp left.

               ------------------------
               -     MONSTERS TURN    -
               ------------------------
The Partial Exam did 3 damage to Worker (Player 1). Worker has 37 hp left.
The Normal Teacher did 3 damage to Worker (Player 1). Worker has 34 hp left.
The Normal Teacher did 2 damage to Worker (Player 1). Worker has 32 hp left.

               ------------------------
               -     PLAYERS TURN     -
               ------------------------
Worker (Player 1). What are you going to do?: s
The Worker (Player 1) used his/her skill.
```

```
The Worker did 19.5 damage to Normal Teacher. Normal Teacher has 0 hp left.

                ------------------------
                -     MONSTERS TURN    -
                ------------------------
The Partial Exam did 4 damage to Worker (Player 1). Worker has 28 hp left.
The Normal Teacher did 1 damage to Worker (Player 1). Worker has 27 hp left.

                        .
                        .
                        .
                        .
                        .
                        .

# The player kills all the enemies:
All the stages have been cleared. You won the game!

#The enemies kill the player:
All characters have been defeated. Try again.


$ ./python main.py --players=2 -s 5

***********       AVAILABLE CHARACTERS       ***********
1.- The bookworn -> Stats: 25HP and 9DMG
        Skill: Revives one player(4 rounds)
2.- The worker -> Stats: 40HP and 10DMG
        Skill: 1.5 * (DMG + DMG roll) damage to one enemy (3 rounds)
3.- The whatsapper -> Stats: 20HP and 6DMG
        Skill: Heals 2*DMG to one player (3 rounds)
4.- The procrastinator-> Stats: 30HP and 6DMG
        Passive: Adds +1 DMG each round. Resets at the beginning of each level.
        Skill: DMG + DMG roll + stage level to all the enemies after the third round of each st
*****************************************************
Player 1. Please, choose a character (1-4): 1
Player 2. Please, choose a character (1-4): 3
*****************************************************
1.- The bookworn -> Stats: 25HP and 9DMG
        Skill: 1.5 * (DMG + DMG roll) damage to one enemy (3 rounds)
2.- The whatsapper -> Stats: 20HP and 6DMG
        Skill: Heals 2*DMG to one player (3 rounds)
*****************************************************
             ************************
             *       STAGE 1        *
             ************************
            ---- CURRENT MONSTERS ----
      ++++++++++++++++++++++++++++++++++++
        Partial exam: Stats: 6HP and 3DMG
        Partial exam: Stats: 6HP and 3DMG
     Theoretical class:  Stats: 8HP and 3DMG
      ++++++++++++++++++++++++++++++++++++
                        .
                        .
                        .
                        .
                        .
```

```
                          .
                          .
 #If player HP drops to zero or below.     Example made with the whatsapper.
 The whatsapper (Player 2) has been defeated.  It can not make any move until revived.
Bookworn. What are you going to do?: s

#if skill can be used
******************************************************
1.- The worker -> Stats: 40HP and 10DMG
        Skill: 1.5 * (DMG + DMG roll) damage to one enemy (3 rounds)
******************************************************
 Who do you want to revive?  2
******************************************************
1.- The worker -> Stats: 40HP and 10DMG
        Skill: 1.5 * (DMG + DMG roll) damage to one enemy (3 rounds)
******************************************************
Incorrect choice. Choice must be between 1 and 1. Who do you want to revive?: 1
The worker (Player 2) has been revived.

#if skill can be used but there is not anyone to revive
All players are alive, so the skill will not be used.
Bookworn. What are you going to do?: a

#if skill is in cooldown
The skill is currently in cooldown for 3 more rounds.
```

5. Condiciones obligatorias de funcionamiento

1. El programa finalizará cuando:
    1.1. Los argumentos sean incorrectos.
    1.2. Se pulse Ctrl + C.
    1.3. Un jugador gane.
    1.4. Todos los jugadores sean derrotados.
2. Los programas deberán escribirse teniendo en cuenta las consideraciones sobre legibilidad y reutilización del código que hemos comentado en clase.
3. Debe crearse una clase `Game` dónde se incluirá toda la funcionalidad relacionada con el juego: gestión de rondas, de turnos, de niveles, de comandos realizados por los personajes, etc. También habrá una clase `Characters` de la cuál derivarán las clases de los personajes y una clase `Enemies` de la cual derivarán todas las clases de los enemigos. Éstas clases son las mínimas a crear, pudiéndose crear otras nuevas si se considera oportuno.
4. Los programas deberán ser robustos, comportándose de manera adecuada cuando no se arranquen con los parámetros adecuados en línea de comandos.
5. Es **obligatorio** controlar todas las excepciones que puedan surgir durante el transcurso del programa: argumentos inválidos, selección de opciones incorrectas, ctrl + c cuando el programa se está ejecutando, etc.
6. Debe seguirse el modelo de salida mostrado en la sección 4 de este enunciado.

6. Pautas de Implementación

- Es muy importante que aquella funcionalidad estrechamente relacionada con el juego en si se implemente dentro de game.py. Por ejemplo, siguiente turno o la impresión por pantalla de lo que va ocurriendo deben ser funciones dentro de dicho paquete. El programa prinipal se ejecutará mediante el fichero main.py y los personajes y los monstruos se implementarán en characters.py y enemies.py respectivamente.
- El programa principal (main.py) debería contener la funcionalidad necesaria para leer los argumentos, iniciar la partida, la selección de los personajes y un bucle en el que se vayan gestionando los turnos y los niveles (Utilizando las funciones de game.py). Intenta que sea lo más breve posible (<200 líneas).
- Se pueden crear excepciones personalizadas según se estime oportuno. Por ejemplo, una excepción para lanzar un error de argumentos es recomendable.
- Se pueden utilizar los módulos que se consideren oportunos. Es recomendable utilizar el módulo `getotp` para una gestión más sencilla de los argumentos.
- Recuerda evitar duplicar código. Si hay más de 5 líneas que se repiten a lo largo de tu programa, es recomendable que crees una función.
- Recuerda que cada vez que utilizas la función `input()` se está leyendo un string. Cada vez que se lee un argumento se está leyendo también un string.

7. Entrega

La práctica puede realizarse en grupos de hasta 3 personas. Sólo hará falta entregar la práctica una vez. Deben aparecer los nombres y apellidos de todos los autores en el fichero main.py en forma de comentario al principio del fichero. Se podrán subir los ficheros de uno en uno, por lo que no es necesario comprimirlos.

El límite para la entrega de esta práctica es el **Domingo, 15 de Noviembre a las 23:59** a través de la tarea correspondiente en el AulaVirtual.

La **Prueba de Laboratorio 1** correspondiente a esta práctica se realizará en el aula de prácticas habitual el **Jueves 12 de Noviembre a las 11:00**