



Universidad de San Andrés

Trabajo Práctico 03 **Rogue Game**

Pensamiento Computacional

Carrera: Ingeniería en IA

Profesores: Patricio Moreno, Álvaro Gaona

Fecha de entrega: 03/06/2022

Alumnos: Máximo Simian, Bruno Castagnino Rossi

Correo electrónico: msimian@udesa.edu.ar,

bcastagninorossi@udesa.edu.ar

1. Objetivos

El objetivo de este trabajo es implementar un juego tipo Roguelike Game a partir de templates provistos por los docentes, focalizandonos en desarrollar conceptos de programación orientada a objetos y de recursión.

2. Diseño del programa

El programa está compuesto por varios archivos, segmentados según que hacen las funciones dentro del archivo. Estos archivos son:

- main.py: archivo mediante el cual se ejecuta el programa.
- menu.py: archivo llamado por main.py. Imprime un menú a partir del cual se puede seleccionar las características del personaje y cuantos niveles tiene el calabozo.
- game.py: archivo en el cual se crean los ítems del calabozo, y a partir del cual se realiza el principal desarrollo del juego, así como las principales llamadas de otras funciones.
- mapping.py: archivo encargado de crear tanto los niveles, así como el calabozo entero. Tanto los niveles como el calabozo son objetos creados a partir de clases.
- player.py: archivo que contiene a la clase "Player", junto con sus atributos por omisión, a partir de la cual se crean el jugador y el gnomo.
- human.py: archivo que contiene la clase "Human", la cual hereda características de la clase "Player" y agrega la posibilidad de tener armas, herramientas y un tesoro, así como regenerar hit points al jugador.
- gnome.py: archivo que contiene la clase "Gnome", la cual también hereda los atributos de "Player" y agrega funciones que le permiten determinar una cantidad de daño por omisión y la posibilidad de revivir cuando el jugador baja de piso.
- actions.py: archivo que contiene todas las funciones relacionadas al movimiento, tanto del jugador como del gnomo, así como también las funciones encargadas de procesar el uso de escaleras, los ataques entre personajes y la regeneración de hit points al matar al gnomo.
- items.py: contiene la clase "Items", así como sus subordinadas "PickAxe", "Sword" y "Amulet", encargadas de definir y crear los objetos.
- functions.py: archivo que contiene la función encargada de correr la función recursiva el archivo mapping, asegurándose de que al terminar la función se devuelven dos coordenadas conectadas.

- magic.py: archivo que contiene la función magic, encargada de leer los inputs del usuario sin la necesidad de que este presione 'enter' cada vez que ingresa una tecla.

(Todas las funciones hechas por nosotros tienen los respectivos docstrings explicando su funcionalidad y parámetros.)

3. Problemas encontrados y soluciones

El principal problema surgido a la hora de desarrollar el programa consistió en desarrollar una función recursiva para poder evaluar si dos ubicaciones distintas elegidas al azar en el mapa se encontraban comunicadas entre sí por un trayecto transitable o si era imposible para el jugador al inicio del juego llegar de dicho punto A al punto B.

Esto era deseable ya que la primera ubicación del jugador y la ubicación del pico que era el ítem que le permitía al jugador atravesar las paredes de piedra se seteaban de manera randomizada al inicio de cada partida. Debido a la naturaleza aleatoria de la generación del mapa existía una posibilidad no despreciable de que el jugador apareciera en un sitio del mapa imposibilitado de acceder a los ítems, a los distintos niveles o simplemente imposibilitado de moverse hacia cualquier lugar.

El desarrollo de esta función permitía asegurar que cada vez que se crea un nuevo mapa el jugador y el pico que le permitirían de manera posterior a su adquisición avanzar a través de cualquier pared de piedra se encuentren comunicados a través de un camino transitable.

A la hora de desarrollar esta función tomamos dos ubicaciones aleatorias con la función provista en el Template del trabajo práctico "find_free_tile" Y a partir de una de estas dos ubicaciones designada como la inicial modificamos los parámetros de la tupla XY (que representa la ubicación del jugador) para designar las posibles movimientos futuros del jugador (arriba, abajo, izquierda y derecha).

Posteriormente analizamos si el casillero que se encontraba en la dirección arriba era posible de ser transitado, si no se encontraba dentro de una lista que contenía los casilleros previamente transitados, si no se encontraba dentro de una lista que contenía los casilleros previamente descartados, Y en caso de cumplirse estas condiciones y de no haberse excedido de los límites del mapa llamamos de manera recursivo nuevamente a la función esta vez designando como un casillero inicial el casillero al cual nos movimos y agregando a la lista de casilleros transitados el casillero previo.

De no cumplirse ninguna de estas condiciones se prueba lo mismo para cada una de las direcciones posibles.

En caso de no cumplirse ninguna de las condiciones que verificaban si el jugador podía moverse en alguna de las cuatro direcciones entonces y la ubicación que se estaba analizando en ese momento pasaba a ser agregada a la lista de casilleros no transitables, y se repetía el ciclo con el casillero que se había transitado inmediatamente anterior a éste.

Tuvimos dos problemas principales a la hora de implementar esta función el primero de los cuales será que al trabajar con listas que tenían que ser pasadas como parámetros a la función recursivo, la función presentaba errores persistentemente ya que a la hora de descartar un casillero no teníamos en cuenta que la lista se modifica de manera global y que teníamos que manejarla para vaciarla de cualquier tipo de valor.

El otro problema que nos encontramos fue que a la hora de implementar la función necesitábamos que la misma se repitiese en caso de dar un resultado negativo (es decir que no hubiese un camino comunicando los puntos A y B) y debido a la naturaleza de las listas nuevamente, esto acarrea errores que producían el retorno de falsos positivos.

Por último pero no menos importante luego de terminar de desarrollar la función al tratar de ejecutarla la enorme mayoría de las veces saltaba un error indicando que se había excedido en la máxima profundidad de recursión.

Analizando nuestra función con mucho detenimiento llegamos a la conclusión de que probablemente la lógica funcionara por lo cual tomamos la decisión de probar aumentar el límite de recurso comprometido por python. Esto tenía sentido también al tener en cuenta que el límite máximo de reclusión permitido era de 1000 y el mapa provisto por el juego tenía una dimensión de 25 × 80 casilleros con lo cual el total de los mismos era de 2000.

Éste último problema se soluciona importando el módulo sys y expandiendo el límite de recursión a 10000.

4. Indicaciones para ejecutar el programa y pruebas

Para ejecutar correctamente el programa se debe de ejecutar únicamente en archivo main.py. En las instancias que se le pida ingresar algo al usuario en algún menú, la única respuesta esperada es alguno de los números que represente una opción, de ser algo distinto, se le pedirá ingresar otro valor. Cualquier otro carácter que se ingrese va a ser tomado como no válido y se le pedirá al jugador que ingrese otro hasta que el mismo sea válido.

Puede elegir visualizar las instrucciones del juego desde el menú principal, eligiendo la opción 1.

Con la opción 2 se le envía a otro menú, en el cual puede elegir jugar una partida por omisión o customizar el juego eligiendo un nombre para su personaje, un carácter especial para visualizarlo en pantalla y la cantidad de niveles que tenga el calabozo(entre 3 y 6).

En el caso del juego, los inputs válidos son solo las teclas 'w', 'a', 's' y 'd' que se utilizan para mover al jugador en el mapa en los sentidos 'arriba', 'abajo', 'izquierda' y 'derecha' respectivamente.

La única otra tecla que deberá usarse es 'v' cuando se posiciona sobre una escalera para subir o bajar de nivel en el calabozo. Para esto es necesario matar al gnomo. De lo contrario la escalera no se desbloqueara para el jugador.

Al salir del calabozo, ya sea ganando (saliendo con el tesoro) o perdiendo (saliendo sin el tesoro o muriendo por quedarse sin HP) se retorna al menú principal desde donde se puede suspender la ejecución del programa con la opción 3.

5. Resultados de ejecuciones

Ejecución normal - Menú principal:

```
MENU
1. See Rules
2. Play
3. Quit
> █
```

Ejecución normal - See Rules:

```
MENU
1. See Rules
2. Play
3. Quit
> 1

The rules are simple:
-To move use w(up), a(left), s(down), d(right).
-To pick up items, just run over them.
-To attack enemies move onto them, and if you have a weapon('/'), you can deal extra damage.
-To break down walls, just run over them while holding the pickaxe('7').
-To use stairs(up'<' and down'>'), press "v" while standing on top, but remember to kill the gnome('G') first.
-To win, you must grab the trasure('8') and escape the dungeon.

MENU
1. See Rules
2. Play
3. Quit
> █
```

Ejecución normal - Play:

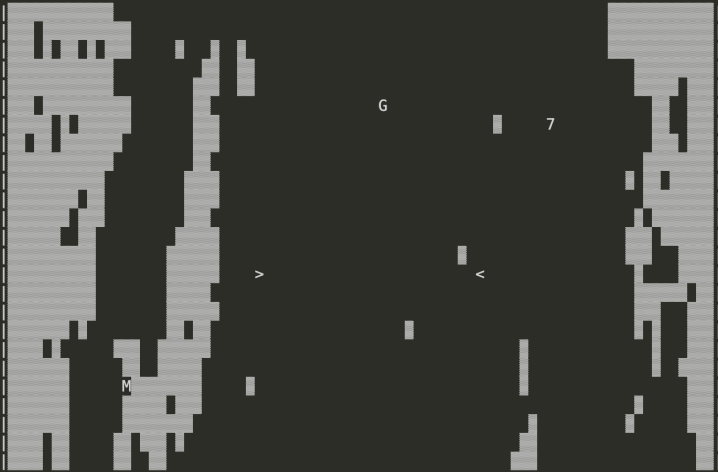
```
MENU
1. See Rules
2. Play
3. Quit
> 2
Please select a gamemode or go back to the main menu.
1. Play with custom character
2. Standard game
3. Go Back
> █
```

Ejecución normal - Play with custom character:

```

Please select a gamemode or go back to the main menu.
1. Play with custom character
2. Standard game
3. Go Back
> 1
Give your character a name: >Maxi
Choose a single charcter for your face: >M
Choose a number of levels from 3 to 6: 4
You were teleported into the dungeon. Try to escape!
-----

```



```

-----
Player:Maxi HP:(200/200) Gnome HP:(100/100) Location:(13/20) Level: 0
Time:0:00:01

```

Ejecución normal - Standard game:

[illegible]

Ejecución normal - Go Back:

```
MENU
1. See Rules
2. Play
3. Quit
> 2
Please select a gamemode or go back to the main menu.
1. Play with custom character
2. Standard game
3. Go Back
> 3
MENU
1. See Rules
2. Play
3. Quit
```

Ejecución normal - Quit:

```
MENU
1. See Rules
2. Play
3. Quit
> 3
(base) Maxi@Maximos-MacBook-Pro pensamiento_computacional %
```

INPUTS INESPERADOS

Diseñamos el código del menú de tal manera que pueda manejar cualquier input inesperado (una opción que no está en el menú). Usualmente en caso de recibir un input inesperado el programa da aviso al usuario y solicita que se intente realizar el input nuevamente.

Luego, durante el correr del juego, gracias a la función `magic`, la cual lee los inputs del teclado sin la necesidad de que el usuario presione 'enter', cualquier input no deseado simplemente es pasado por alto y no tiene efectos en el juego. Cualquier tecla que no sea las de movimiento, al ser ingresada es tomada por el programa como intención del jugador de permanecer en el lugar. Por lo tanto, al ingresar una tecla que no sea las implicadas para el movimiento, el jugador permanecerá en sus coordenadas, mientras que el gnomo se continúa moviendo por el mapa.

6. Bibliografía

[-https://stackoverflow.com/questions/45096347/increasing-recursion-depth-in-python-to-100000](https://stackoverflow.com/questions/45096347/increasing-recursion-depth-in-python-to-100000)