



Universidad de  
**SanAndrés**

---

## **Trabajo práctico 3**

*Rogue Game*

**Profesores:** Patricio Moreno, Débora Copa

**Estudiantes:** Mateo López Vilaclara, Gonzalo Lewit,  
Mateo Costantini

[mlopezvilaclara@udesa.edu.ar](mailto:mlopezvilaclara@udesa.edu.ar) – [glewit@udesa.edu.ar](mailto:glewit@udesa.edu.ar) –  
[costantinim@udesa.edu.ar](mailto:costantinim@udesa.edu.ar)

## Objetivos propuestos

En este trabajo práctico, nuestro objetivo más básico y general era el de **crear un programa para jugar un estilo de juego similar al de Rogue Game** en el que se le ofrecía al usuario jugar siendo un personaje y con el objetivo final de encontrar un tesoro y llevarlo al exterior del calabozo. Hay algunos logros que se puede ganar el personaje si triunfa. **Unscathed**: cuando el jugador gana sin que haber perdido HP durante la partida. **Divine Intervention**: Cuando el jugador gana con 1/50 HP. **Genocidal**: Cuando el jugador consigue ganar matando a todos los monstruos en el mapa. **Pacifist**: Cuando el jugador gana sin haber matado ningún monstruo en el mapa. **Wrestler**: Cuando el jugador gana habiendo matado al menos un monstruo sin haber agarrado la espada. **Daredevil**: Cuando el jugador gana sin haber agarrado el escudo. Algunos logros dependen de la suerte a la hora de conseguirlos, ya que sin una buena ubicación de los objetos y/o identidades, es imposible conseguirlos.

El usuario no solo podrá ver su personaje y los movimientos que hace, sino que también le puede poner un nombre, ver cuanto HP tiene, que herramienta tiene, entre otras cosas. El jugador tendrá la posibilidad de recoger objetos del piso como picos y espadas para así poder llegar a los rincones más oscuros del calabozo y combatir con los personajes más malvados.

Ahora bien, más allá de aquellos objetivos que se nos fueron dados por la consigna, nosotros nos propusimos otros propios; más precisamente, buscamos que nuestro código sea **estéticamente agradable**, de **fácil comprensión**, **prolijo** y, por sobre todo, **funcional** sin importar que haya inputs inesperados por parte del usuario. Además, decidimos agregarle nuevos objetos para hacer mas interactivo el juego y que el usuario consiga una mejor experiencia jugándolo.

## Diseño del programa

Nuestro programa fue hecho en varios archivos distintos. Contiene nueve archivos en total: **game.py**, **actions.py**, **human.py**, **items.py**, **magic.py**, **mapping.py**, **player.py**, **gnome.py** y **phantom.py**. Dentro de cada archivo se presentan tanto funciones como clases. Con la utilización de clases fuimos capaces de generar tanto a los personajes, como a las herramientas entre otras cosas, de manera más organizada.

- **game.py** es el archivo que corre el juego. Dentro de este, se crean los objetos como el calabozo (dungeon), el jugador (player, '@'), el pico (pickaxe, '^'), el escudo (shield, 'O'), la espada (sword, '/'), el tesoro (amulet, 'X'). Además, en este archivo se asigna en qué nivel del calabozo se encuentran los ítems.

También cumple con la función de establecer cuantas filas y columnas tiene el territorio de juego.

El programa solo se puede correr en computadoras Mac o Linux. El usuario podrá mover su personaje sin la necesidad de presionar la tecla 'enter' después de oprimir las teclas 'w-a-s-d' o cualquier otra tecla.

- **actions.py** en este archivo están contenidas las distintas funciones donde se detallan todas las acciones que el personaje puede realizar a lo largo de la partida. Entre ellas están las funciones:
  - **clip** que cumple con el objetivo de que el personaje no pueda salir por fuera del mapa de juego. Esta función retorna la posición del personaje con la condición de que, si trata de sobrepasar la posición de una pared inquebrantable, la función retorna la posición anterior del límite.
  - **Attack** A lo largo del juego, el personaje se va a encontrar con monstruos que podrán hacerle daño y matarlo, aunque también él va a poder hacerle daño a las criaturas y matarlas. Esta función lleva a cabo la acción de atacar a los protectores del tesoro. Los gnomos tienen distinta vida que el fantasma. Es por eso que las peleas con ellos no siempre son fáciles. Cuando el personaje les pega a los monstruos, estos pierden vida y si se quedan sin, mueren.
  - **Gnome\_attack** esta función se accede si algún Gnomo le pega al jugador. Este hace un daño de entre 5 y 10 HP si el jugador no posee escudo y entre 2 y 6 HP si posee. Si la vida del personaje es 0 o menor, el jugador muere.
  - **Phantom\_attack** esta función se accede si el Fantasma le pega al jugador. Este hace un daño de entre 10 y 15 HP si el jugador no posee escudo y entre 5 y 10 HP si posee. Si la vida del personaje es 0 o menor, el jugador muere.
  - **Move\_to** marca los límites en donde el jugador puede moverse. El mapa es de 25x80, esto quiere decir que el personaje no va a poder salir de estos límites. Hay paredes inquebrantables que marcan el fin del terreno, el jugador es incapaz en todo momento de atravesarlas.
  - **Climb\_stair** El calabozo tiene tres niveles en tres distintos pisos. Es por eso que en el juego hay escaleras que le permiten al jugador subir un piso más arriba. En esta función, se encuentra la condición de que si el personaje se encuentra encima de una escalera pueda subirla. Cada vez que el personaje sube una escalera, se acerca más a la superficie y desciende un nivel. Si el personaje sube al nivel 0 con el tesoro, el jugador triunfa. Si el personaje sale sin el tesoro, entonces pierde.
  - **Lose\_message** esta función es la encargada de imprimir el mensaje de victoria o de derrota dependiendo si el jugador triunfo o no. Si el personaje sale del calabozo sin el tesoro, se imprime una de las posibles frases de derrota.
  - **Descend\_stair** en el juego hay escaleras que le permiten al jugador bajar un piso más abajo. Al igual que la función de climb\_stair, esta función tiene la condición de que, si la ubicación del personaje es la misma que la de la escalera, entonces puede descenderla. Cada vez que el personaje desciende una escalera, se acerca a lo más profundo del calabozo y asciende niveles.
  - **Pickup** Lo que realiza esta función es permitirle al jugador agarrar ítems del suelo. Esta función tiene en cuenta si el personaje está ubicado en la misma

posición que el ítem, si es así, se le otorga al personaje ese ítem (puede ser ‘tool’, ‘weapon’ o ‘treasure’).

- **Human.py** En este archivo se define la clase del humano, en esta se presentan los atributos con los que el jugador comienza la partida. Se establece que empieza sin ningún arma en el inventario, sin el escudo, sin el tesoro y sin el pico. Además, ahí establecemos si esta vivo y cual es la cara del personaje (‘@’). Por otro lado, `__str__` devuelve muchos de los atributos que tiene el personaje. Dentro de esa clase también se establece el daño que el personaje le hace a los monstruos dependiendo si tiene o no la espada.
- **items.py** dentro del archivo se establecen cuatro distintas clases:
  - **Item:** Esta clase se encarga de definir el nombre, la apariencia y el tipo de cada ítem en el juego. Esta es la clase madre de otras cuatro subclases: **Sword**, **Amulet**, **PickAxe** y **Shield**.
  - **Sword:** La clase Sword adquiere los atributos de la clase ítems, esto quiere decir que a Sword se le establece name, face y type.
  - **Amulet:** La clase Amulet adquiere los atributos de la clase ítems, esto quiere decir que a Sword se le establece name, face y type.
  - **PickAxe:** La clase PickAxe adquiere los atributos de la clase ítems, esto quiere decir que a Sword se le establece name, face y type.
  - **Shield:** La clase Shield adquiere los atributos de la clase ítems, esto quiere decir que al escudo se le establece name, face y type.
- **Magic.py** cumple un rol fundamental para tener una mejor experiencia jugando al Rogue Game. Magic.py brinda la posibilidad de que no se tenga que oprimir ‘enter’ luego de la tecla que presionas para realizar la acción que deseas. Esto quiere decir que una vez que presiones alguna tecla, automáticamente la computadora lo toma como input.
- **Mapping.py** En términos generales, este archivo se encarga de crear el mapa de juego. Dentro de este archivo se encuentran tres clases: **Tile**, **Level**, **Dungeon**.
  - **Tile:** Esta clase principalmente define los elementos que pertenecen al mapa como las paredes, el aire, y las escaleras. Esta clase les establece una apariencia (face) y no menos importante, define también si esos espacios son caminables por el personaje o no.

- **Level:** Esta segunda clase tiene dentro todas las funciones importantes para la creación del terreno de juego en los niveles. Esto quiere decir que asigna los espacios de pared y los espacios de aire. Dentro de este se encuentra una función que busca espacios donde hay aire (libre) que luego puede ser útil para meter en esos espacios las herramientas o incluso los personajes. Luego se encuentran las funciones que agregan las escaleras y los ítems como el tesoro y demás. Por otro lado, dentro de la clase Level hay una función muy importante **render** que sirve para imprimir en la terminal todos los ítems, personajes que se encuentran en el mapa. Es por eso que es muy importante que todos los objetos tengan una face (la apariencia que tienen) porque de esa forma es como el usuario los va a ver una vez que corra el programa. La función **is\_walkable** sirve para ver si el personaje puede caminar por una posición específica (depende si hay pared, aire o algún ítem). **Index** es utilizada para obtener la ubicación de los elementos en el mapa. **Loc** devuelve el tipo de elemento que se encuentra en una ubicación específica. **Dig** esta función se accede cuando el jugador rompe con el pico una pared. Lo que hace es cambiar la pared por aire para que luego el jugador pueda pasar caminando. **Is\_free** sirve para ver si en una posición específica se encuentran entidades o no. **Are\_connected** y **get\_path** se relacionan mutuamente. Sirven para determinar si hay un camino posible entre dos posiciones en el mapa. La función **get\_path** es recursiva. Si esa función encuentra un camino posible entre las dos ubicaciones, **are\_connected** devuelve True, de otro modo devuelve False.
- **Dungeon** es la tercera clase del archivo. Si bien es similar a la clase **level**, estas dos se conectan ya que las funciones de Dungeon están en constante contacto con las funciones de Level. Esta clase hace que se pueda aplicar la clase level a todos los niveles del juego.
- **Player.py** tiene dentro una clase **Player** que es la que crea al jugador, pero no solo eso, sino que determina la ubicación y la vida que tiene en el momento que esta entre 0 y 50, y máxima vida posible que puede tener y con la que arranca, la cual es 50.
- **Gnome.py** La función de este archivo es crear a los gnomos. Es por eso que dentro de este se encuentra la clase **Gnome**, como atributos se establecen la apariencia del gnomo, la ubicación previa en la que estuvo antes de hacer el último movimiento ya que como se mueve de forma aleatoria, decidimos que no se pueda mover a la ubicación previa en la que estuvo.
- **Phantom.py** esta función fue diseñada para crear el fantasma del tercer nivel. Tiene los métodos de atacar al jugador, moverse por el mapa, ser eliminado. Por otro lado, el fantasma tiene la posibilidad de atravesar paredes quebrantables cada vez que camina por sobre ellas.

## Problemas y soluciones

Error: Había un mismo gnomio en los tres niveles del calabozo, por lo que, si se pasaba al próximo nivel y en el lugar que estaba el gnomio había pared, surgía un error.

Solucionado:

```
while dungeon.level >= 0 and player.hp > 0:
    turns += 1

    if dungeon.level == 0:
        enemy = gnome1
    elif dungeon.level == 1:
        enemy = gnome2
    elif dungeon.level == 2:
```

Para optimizar el código, hicimos una variable general para los tres gnomos (enemy), esta variable va cambiando valor (gnomo) dependiendo del nivel en el que se encuentre el personaje.

```
key = magic.read_single_keypress()

if key[0] == 'q':
    break
if key[0] == 'w':
    actions.move_to(dungeon, player, (player.x, player.y - 1), enemy)
elif key[0] == 'a':
    actions.move_to(dungeon, player, (player.x - 1, player.y), enemy)
elif key[0] == 's':
    actions.move_to(dungeon, player, (player.x, player.y + 1), enemy)
elif key[0] == 'd':
    actions.move_to(dungeon, player, (player.x + 1, player.y), enemy)
elif key[0] == 'p':
    actions.pickup(dungeon, player)
elif key[0] == 'u':
    actions.climb_stair(dungeon, player)
elif key[0] == 'v':
    actions.descend_stair(dungeon, player)

if dungeon.enemy_alive(enemy) == True:
    enemy.move(dungeon, player)
```

Un problema que se nos presentó fue a la hora de hacer la función recursiva **get\_path**. Esta tenía como primera opción fijarse si era posible moverse para arriba. Al tener en el resto de las posibilidades de movimiento ‘elif’, se cancelaban las posibilidades de continuar un camino por el costado una vez la recursiva devolvía valores. Con esto quiero decir que descartaba como camino posible los caminos que estaban a los costados. Debido a esto, el programa en casi su totalidad de casos devolvía False refiriéndose a que el personaje no tenía vía posible para alcanzar el pico.

Solución:

Además de cambiar los ‘elifs’ por ‘ifs’ para solucionar el problema mencionado anteriormente, cambiamos también lo que la función retorna si no hay conexión entre los dos objetos. Pasamos de retornar "" a [].

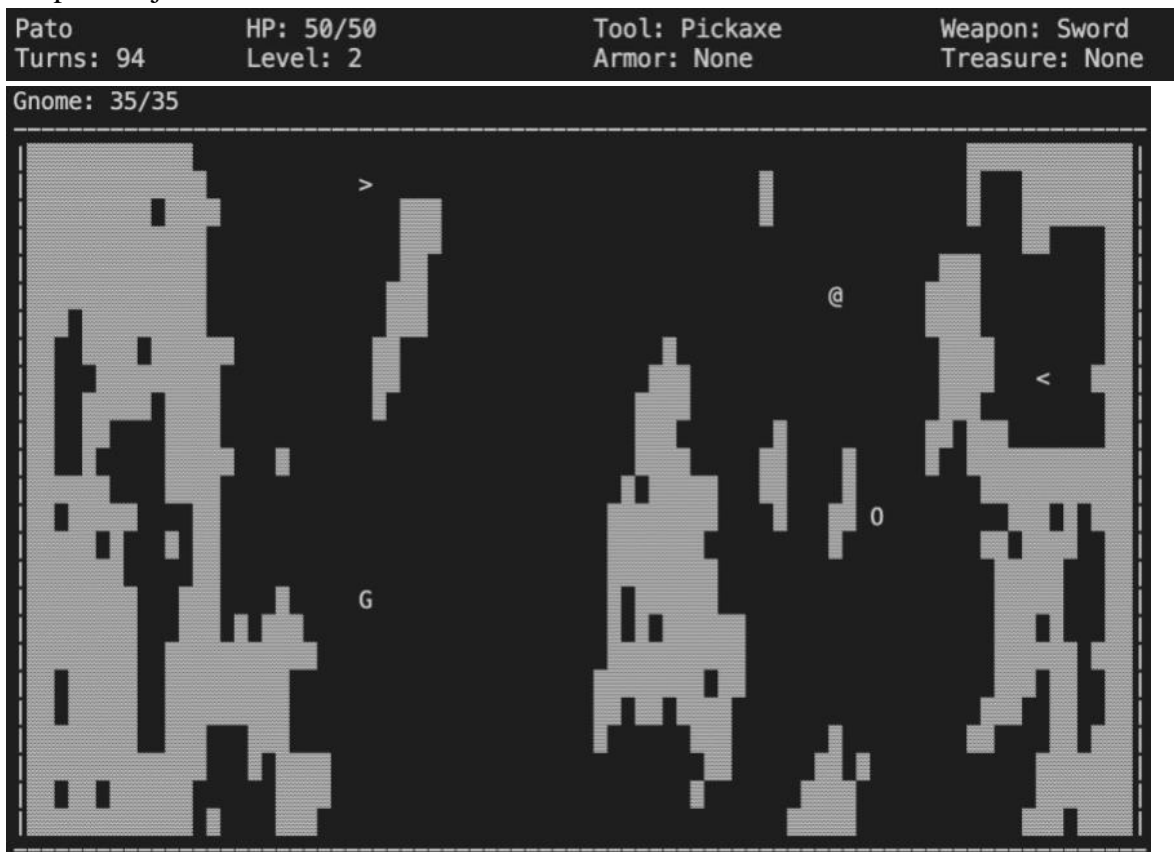
```
def get_path(self, initial: Location, end: Location, path = [], traversed = []):
    if (initial in path) or (initial in traversed):
        return []
    if initial == end:
        path.append(end)
        return path
    path.append(initial)
    if initial[0] + 1 < 80 and self.is_walkable((initial[0] + 1, initial[1])) == True:
        new_initial = (initial[0] + 1, initial[1])
        r = self.get_path(new_initial, end, path, traversed)
        if r != []:
            return r
    if initial[0] - 1 > 0 and self.is_walkable((initial[0] - 1, initial[1])) == True:
        new_initial = (initial[0] - 1, initial[1])
        r = self.get_path(new_initial, end, path, traversed)
        if r != []:
            return r
    if initial[1] + 1 < 25 and self.is_walkable((initial[0], initial[1] + 1)) == True:
        new_initial = (initial[0], initial[1] + 1)
        r = self.get_path(new_initial, end, path, traversed)
        if r != []:
            return r
    if initial[1] - 1 >= 0 and self.is_walkable((initial[0], initial[1] - 1)) == True:
        new_initial = (initial[0], initial[1] - 1)
        r = self.get_path(new_initial, end, path, traversed)
        if r != []:
            return r
    path.remove(initial)
    traversed.append(initial)
    return []
```

## Indicaciones y ejecuciones

1. Descomprimir el archivo todo en una carpeta (es importante que sea la misma).
2. Abrir el archivo mediante Spyder.
3. Ejecutar el programa desde el botón de "Run file", con atajo en F5 desde el archivo "game.py":
4. En este momento, la consola le va a pedir al usuario que ingrese un nombre para el personaje

**Player?: Pato**

5. Una vez ingresado el nombre del personaje, se va a imprimir en la consola el mapa de juego y el menú donde se encuentra el nombre, la vida, entre otras características del personaje.




6. Para pegarle a un enemigo, simplemente el jugador debe caminar hacia la misma posición en la que está el monstruo.
7. Para romper paredes con el pico, el jugador debe caminar hacia la pared que desee romper.
8. Con el escudo, el jugador recibirá menos daño que si los monstruos le pegan cuando no tiene el escudo.

## Juego

Al ser un juego por turnos, cada vez que el usuario presione una tecla, ya sea para moverse o no, se considerara como un turno y los monstruos realizaran un movimiento.

Elementos que posiblemente encuentres durante el juego:

- ‘@’: Es el personaje del usuario
- ‘G’: Son los Gnomos (enemigo). Hay uno por cada nivel en el juego. Tienen la característica de no ser muy inteligentes. Se mueven aleatoriamente por el mapa aunque si se encuentran al lado del personaje, el Gnomo le va a sacar vida.
- ‘P’: Es el Fantasma (enemigo). Se encuentra únicamente en el tercer nivel. Tiene la capacidad de traspasar las paredes sin romperlas. Además, son muy inteligentes ya que persiguen al jugador a toda costa. Si se encuentra al lado del personaje, el fantasma le va a hacer daño.
- ‘%’: Es el esqueleto de los Gnomos y el fantasma una vez muertos.
- ‘)’: Es un pico (herramienta) una vez agarrado sirve para romper las paredes quebrantables.
- ‘/’: Es la espada (arma), sirve para combatir a los enemigos y poder hacerles más daño.
- ‘O’: Es un escudo y sirve para recibir menos daño. El juego sin el escudo es difícil. Esperamos que el usuario aproveche y obtenga esta ventaja competitiva para poder triunfar.
- ‘ ’ ’’: Es el tesoro (amuleto), es necesario buscarlo y agarrarlo para luego poder ganar el juego.
- : Son las paredes quebrantables que forman parte del calabozo.
- ‘<’: Esta es una escalera que sube, es utilizada para moverse a través de los niveles dentro del calabozo.
- ‘>’: Esta es una escalera que baja, es utilizada para moverse a través de los niveles dentro del calabozo.
- ‘|’ y ‘-’: Son las paredes inquebrantables que marcan el limite del calabozo. No hay forma de atravesarlas.
- ‘ ’’: Es espacio vacío por el que se puede caminar libremente.

Por otra parte, las teclas que admite el juego son las siguientes:

- ‘w’: es utilizada para moverse una posición arriba en el mapa.
- ‘a’: es utilizada para moverse una posición a la izquierda en el mapa.
- ‘s’: es utilizada para moverse una posición abajo en el mapa.
- ‘d’: es utilizada para moverse una posición a la derecha en el mapa.
- ‘q’: es utilizada para salir del juego en cualquier punto de la partida.

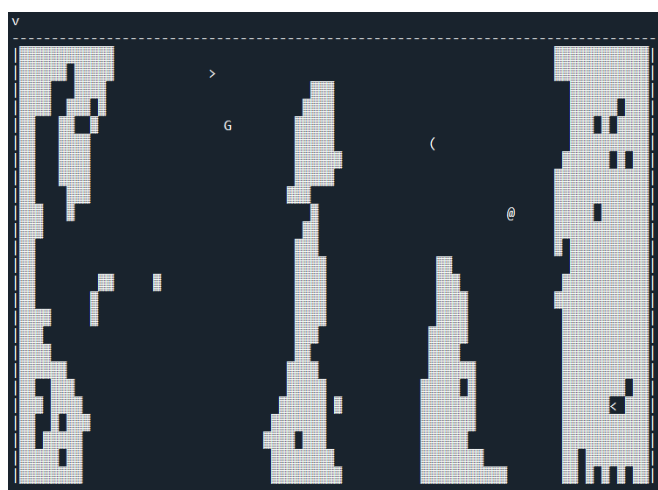
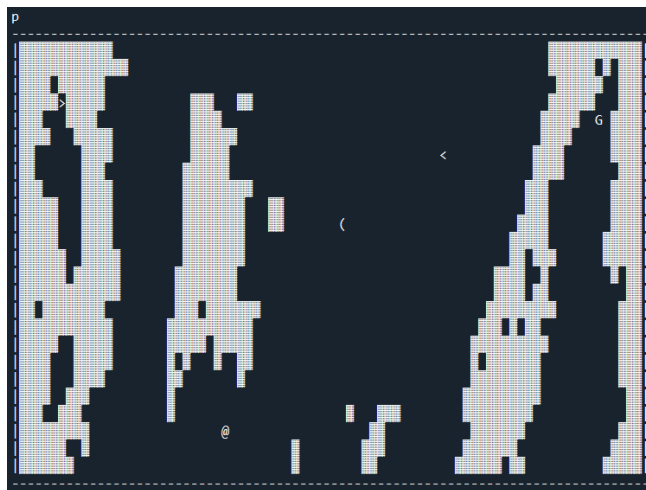


- ‘u’: Es utilizada para subir una escalera. Únicamente funciona si el personaje está parado encima de la escalera que sube (‘<’).
- ‘v’: Es utilizada para bajar una escalera. Únicamente funciona si el personaje está parado encima de la escalera que baja (‘>’).
- ‘p’: Es utilizada para agarrar herramientas, escudos, armas y amuletos del piso. Únicamente funciona si el personaje esta parado encima de alguno de estos elementos.

### Objetivo del usuario:

El usuario tiene como principal objetivo buscar el amuleto que se encuentra en el tercer nivel y sacarlo del calabozo. El juego le presenta algunas dificultades al personaje ya que en el calabozo existen los monstruos (Gnomos y Fantasma). Estos van a defender el amuleto a toda costa. Para esto están las herramientas y las armas que van a ayudar al personaje a cumplir su objetivo. Dentro de las herramientas esta el pico, este sirve para que el personaje pueda romper las paredes quebrantables. La utilización del pico puede ser útil para llegar a los lugares más recónditos del calabozo. Puede pasar que el tesoro este escondido y encerrado entre cuatro paredes y la única forma de acceder a este es con el pico. Para equilibrar las ventajas entre los monstruos y el personaje, si el jugador desea atravesar una pared y camina hacia ella, primero se rompe y luego el jugador deberá hacer otro movimiento para caminar por el bloque libre. Por otro lado, dentro de las armas esta la espada, que sirve para poder eliminar manera mas sencilla a los enemigos. El jugador tiene la capacidad de pegarles aun así sin la espada, pero realiza mucho menos daño.

### Algunas entradas inesperadas



Estas entradas inesperadas consisten en inputs validos con una utilidad en el juego como lo son ‘p’ para agarrar elementos del piso y ‘v’ para descender escaleras. Si el usuario presiona esas teclas sin estar encima de un elemento o unas escaleras, entonces no agarra nada y no cambia de nivel respectivamente.

*Trabajo hecho por Mateo Costantini, Gonzalo Lewit y Mateo López Vilaclara*

*Junio 2022.*

---

### **Bibliografía**

Apuntes de pizarrón.

Presentaciones de Patricio y de Débora.

---