> Nombre:
> DNI:
> compilar con: `g++ src/*.cpp -Iinclude -lraylib -Llib -o minijuego`

📜 Proyecto final MP: EL MINIGÜEJO

⚙️ Funcionamiento de El Minigüejo

- El minijuego tiene principalmente **tres** elementos: jugador, enemigos y disparos.
- El **jugador** se mueve horizontalmente en la parte inferior de la pantalla, su objetivo es evitar que los **enemigos** alcancen el borde inferior de la pantalla, esto lo hace **disparando** proyectiles que eliminan a los enemigos.
- El jugador comienza con cinco **vidas**. Cada vez que un **enemigo** alcanza el borde inferior de la pantalla, el jugador pierde una vida. Cuando el jugador tiene 0 vidas, termina el juego.
- Cada vez que el jugador **elimina** a un enemigo, obtiene un punto.
- Cuando el jugador tiene 20 puntos, aumenta su cadencia de disparo.
- Conforme avanza el juego, los **enemigos** aparecen con más frecuencia.

> **NOTA**
> Los enemigos nunca impactan con el jugador, lo sobrevuelan sin causarle daño.

- `main.cpp` se encarga de leer la entrada por teclado y de mostrar por pantalla la información del juego.
- La clase `Game` se encarga de gestionar la lógica del juego
  - Su método principal es `update()`, que se ejecuta una vez en cada frame. (en el guión se llama `step()`)

```
void Game::update(int inputDirection, bool inputFire) {
    updatePlayer(inputDirection * playerVelocity); // Mueve al jugador

    if (inputFire) {
        fireBullet();  // Intenta disparar un proyectil
    }

    updateBullets();   // Mueve las balas y elimina las que están fuera de límites
    updateEnemies();   // Mueve los enemigos y elimina los que han alcanzado su objetivo
    manageCollisions();// Gestiona colisiones entre balas y enemigos
    spawnEnemy();      // Intenta generar un nuevo enemigo
}
```

🏔️ Dificultades encontradas

⏰ El tiempo

- Para gestionar la frecuencia de aparición de enemigos y la cadencia de disparo, se hace necesario el uso del *tiempo*.
- Raylib tiene una función `GetTime()` para obtener el tiempo transcurrido, pero la clase `Game` no debería depender de raylib.
- Para hacer una función similar a `GetTime()` primero probé con `time()` y `clock()`, pero no tienen la precisión necesaria (`time()` solo mide segundos y el tiempo que mide `clock()` no coincide con el que mide la función de raylib).
- Terminé usando `std::chrono::high_resolution_clock`

```
// Tiempo transcurrido desde el inicio del juego
float Game::getElapsedTime() {
  std::chrono::time_point<std::chrono::high_resolution_clock> now;
  now = std::chrono::high_resolution_clock::now();
  unsigned long time =
        std::chrono::duration_cast<std::chrono::milliseconds>(now - startTime).count();
```

```
  return time / 1000.f;
}
```

- `GetElapsedTime()` se utiliza en el método de disparar, para comprobar si ha pasado el tiempo de "enfriamiento".

```
void Game::fireBullet() {
  if (getElapsedTime() - lastShot > shootCooldown) {
    Vector2D acel = Vector2D(0,0);
    Vector2D veloc = Vector2D(0, bulletVelocity);
    Vector2D pos = player.getPos();
    Particula bullet = Particula(player.getPos(), acel, veloc, bulletRadius, 1);
    bullets.agregar(bullet);
    lastShot = getElapsedTime();
  }
}
```

🏗️ Parámetros del juego y constructor de `Game`

Hace falta una cantidad moderadamente grande de parámetros para hacer funcionar el juego:

```
bool active = true;         // 'true' mientras el juego esté activo

ConjuntoParticulas enemies; // Conjunto de enemigos (naves)
ConjuntoParticulas bullets; // Conjunto de proyectiles (disparos)
Particula player;           // Partícula jugador (base)
Vector2D playerSpawnPoint;  // Posición de aparición del jugador (calculada en el
constructor según dimensiones de la pantalla en el constructor)

int screenWidth;            // Anchura de la pantalla
int screenHeight;           // Altura de la pantalla

int playerPoints = 0;       // Puntos del jugador
int playerLives;            // Número de vidas del jugador
float playerVelocity;       // Velocidad de movimiento del jugador

float bulletVelocity;       // Velocidad de movimiento de las balas
float bulletRadius;         // Radio de las balas

/* La velocidad de los enemigos se elige de forma aleatoria según los siguientes
parámetros */
float enemyMinVelocityX;    // Valor mínimo para la velocidad en X
float enemyMaxVelocityX;    // Valor máximo para la velocidad en X
float enemyMinVelocityY;    // Valor mínimo para la velocidad en Y
float enemyMaxVelocityY;    // Valor máximo para la velocidad en Y
float enemySpawnOffsetY;    // Ajuste de posición de aparición (para evitar que aparezcan
en el borde superior de la pantalla)
float enemyRadius;          // Radio de los enemigos

std::chrono::time_point<std::chrono::high_resolution_clock> startTime;
float shootCooldown;        // Intervalo entre disparos
float enemySpawnCooldown;   // Intervalo de aparición de enemigos

float lastShot = 0;         // Momento en el que se realizó el ultimo disparo
float lastSpawned = 0;      // Momento en el que apareció el último enemigo
```

🧭 Conclusión

Está muy bien, se aprende mucho haciendo jueguitos. Me ha parecido más o menos fácil, pero seguro que la he liado con algo.