INF-253 Lenguajes de Programación
Tarea 2 2025-2: Space Defender

Profesores: José Luis Martí Lara, Wladimir Ormazábal Orellana, Ricardo Salas Letelier
Ayudante coordinador: Bastián Ríos Lastarria
Ayudantes: Diego Duarte, Andrés Jablonca, Cristobal Tirado,
Martin Pardo, Carlos Bravo, Martín Palacios,
Blas Olivares, Sofía Ramirez, Benjamín Echeverria

25 de agosto de 2025

Índice

1. Contexto

Un familiar tuyo encuentra un antiguo arcade en el sótano de su casa, pero lastimosamente no prende... Tu familiar, lleno de nostalgia, te llama y solicita si eres capaz de recrear su juego favorito de la infancia, el *Space Invaders*, con tus grandes habilidades adquiridas como programador decides cumplir su deseo. Por lo que te encarga la siguiente misión: Debes crear un programa en C recreando el juego *Space Invaders*, el cual consiste en una sola nave controlada por el usuario que se desplaza únicamente de izquierda a derecha en una línea de defensa y dispara distintos tipos de láser para contener las olas de invasores que descienden desde el límite superior del tablero con una pizca de creatividad tuya para alegrar a tu familiar. La misión termina con la aniquilación total de las olas o con la caída de un enemigo sobre la base.

1.1. Objetivo principal

Implementar un minijuego de defensa espacial por turnos en terminal, modelando un **tablero bidimensional** con **triple puntero**, un **inventario de disparos** implementado mediante **punteros a función**, y un **sistema de olas** controlado. El estudiante deberá diseñar estructuras de datos dinámicas, respetar reglas deterministas de movimiento/colisión y producir una interfaz ASCII clara con HUD persistente.

2. Descripción de la tarea

A continuación se especifican los requerimientos funcionales y las reglas del juego. **Toda la interacción es por terminal**, renderizando el tablero como casillas de la forma `[ ]` y similares.

2.1. Especificaciones generales

- **Tablero y coordenadas**: dimensiones según dificultad (§2.6). Se utiliza un sistema de coordenadas con (izq→der), con $y = 0$ **en la base** (jugador) y $y = H - 1$ **arriba** (spawn).
- **Estructura de datos del tablero**: `void ***tablero` (arreglo de filas → arreglo de columnas → `Celda`). Cada `Celda` puede referenciar, como mínimo, `Alien`, celda vacía y una marca de disparo del turno.
- **Mostrar tablero en consola**: El programa debe mostrar en cada turno el estado completo del tablero en la terminal, usando un formato claro y fácilmente legible. Bajo el tablero se muestra el HUD (§2.7).
- **Un alien por celda**: no se permiten dos aliens en la misma `Celda`.
- **Derrota**: si cualquier alien entra a y=0 ('final' del tablero), la partida termina inmediatamente.

2.2. Flujo de turnos y entrada

T1. **Entrada del jugador (consume turno si procede)**: `A/D` para mover (con límites) o `1..3` para disparar. Intentos de salir del tablero o acciones inválidas **no consumen** turno.

T2. **Disparo instantáneo**: se resuelve inmediatamente. Debe haber un indicador de impacto en el HUD, ya sea una X en el tablero o un mensaje por pantalla.

T3. **Movimiento de aliens**: cada 2 turnos todos los aliens descienden 1 celda. Orden de movimiento: de arriba hacia abajo (desde y = H-1 hacia y = 0).

T4. **Spawn de aliens**: en cada turno se intenta spawnear 1 alien; con probabilidad 30 % se intenta un segundo, respetando el inventario de aliens y el tope de aliens simultáneos.

2.3. Armas del jugador (inventario de acciones)

Se disponen 3 armas, invocadas mediante punteros a función y activadas con `1..3`. El disparo **consume un turno** si procede. Parámetros y efectos:

- **(1) Normal**: daño 1, recto, munición ilimitada, impacta al primer alien en la columna.
- **(2) Perforador**: un rayo recto perforante, daño 1 a *todos* los aliens de la columna, 7 disparos.
- **(3) Especial**: Un disparo que *deberán implementar ustedes utilizando su creatividad*. Las únicas reglas son:
  - No romper el flujo del juego (que no acabe la partida inmediatamente)
  - Incluya una mecánica nueva y creativa (daño en área, que rebote, etc.)
  - Sea explicada en el README su funcionamiento

  Este disparo especial debe contar con al menos 1 disparo al comenzar la partida, pero debe ser un recurso limitado.

2.4. Tipos de alien y comportamiento

- **Drone**: HP=2; baja cada 2 turnos (recto).
- **Skater**: HP=1; baja cada 2 turnos y se desplaza diagonalmente (zig-zag), rebotando en bordes.
- **Especial**: baja cada 2 turnos, al igual que el disparo especial, *deberán implementarlo ustedes con una mecánica novedosa y creativa*, y explicar en el README su funcionamiento. El HP y la manera de descender la deciden ustedes, pero debe respetar la regla de descender una celda cada 2 turnos.

Restricción: máximo un alien por celda.

2.5. Municion extra

Al destruir un alien, se ejecuta una tirada aleatoria que puede otorgar de manera **instantánea** alguno de los siguientes efectos (se aplica directamente al jugador y se imprime un mensaje):

- **Munición Perforador**: +1 de munición al arma perforadora. (25 % probabilidad)
- **Munición Especial**: Definen ustedes cuánta munición y con cuánta probabilidad debe dar

2.6. Dificultades e *inventario* de aliens

Dimensiones del tablero. **Fácil:** $W = 5$, $H = 15$. **Difícil:** $W = 7$, $H = 15$.

Generación y cantidad de aliens. Cada partida usa un **inventario fijo de aliens** que debe agotarse para ganar.
Fácil: Total: 15 aliens, **mínimo** 6 **drones** y 4 **skaters**, el resto lo deciden ustedes.
Difícil: Total: 20 aliens, **mínimo** 8 **drones** y 6 **skaters**, el resto lo deciden ustedes.

Límites de simultáneos. A lo largo de la partida puede haber como máximo **6 aliens vivos** en pantalla (Fácil) y **8** (Difícil). Si se alcanza el tope, no aparecen nuevos hasta que haya espacio.

Aparición (simple y aleatoria).

a) **Sembrado inicial:** antes del primer turno, se instancian automáticamente **2 aliens** en Fácil y **3 aliens** en Difícil, tomados del inventario (probabilidades definidas por el usuario, debe ser **mayor a 15 %** por cada tipo).

b) **Por turno:** en cada turno, si queda inventario y no se alcanza el tope de simultáneos, **aparece de 1 a 2 aliens (con probabilidad de un 30 % el segundo)** tomado al azar del inventario restante.

c) **Columna de aparición:** el alien nace en la fila superior ($y = H - 1$) en una **columna libre** elegida uniformemente al azar. Si todas las columnas en $y = H - 1$ están ocupadas, el juego **pospone** ese spawn hasta el siguiente turno.

Movimiento y término. Los aliens **descienden cada 2 turnos**. La partida termina en **victoria** cuando el inventario se agota y no quedan aliens vivos; termina en **derrota** si cualquier alien llega a $y = 0$ (base).

2.7. HUD y mensajería

El HUD muestra, sobre o bajo el tablero: turno actual, aliens restantes del inventario, aliens vivos, municiones ((1)NOR, (2)PER, (3)ESP), mensajes de evento (que se entrega munición, por ejemplo) y avisos por entradas inválidas.

3. Archivos y módulos

Esta sección fija la organización mínima en archivos, las **funciones obligatorias** (con firmas exactas en C) y los **typedefs esenciales**. Pueden crear funciones auxiliares adicionales, pero **no** se pueden cambiar las firmas aquí definidas (nombres, tipos, orden de parámetros y tipo de retorno).

3.1. Estructura general y restricciones

- **Tablero con triple puntero:** el tablero debe representarse con `void ***` (tres niveles de indirección). El tercer puntero es el contenido de la celda (p. ej., `Celda *` casteado desde `void *`).
- **Compilación separada y headers:** declarar en `.h`, definir en `.c`, y compilar por módulos mediante `Makefile`.
- **Firmas inmutables:** no se permite cambiar los retornos ni parámetros de las funciones obligatorias descritas a continuación.

3.2. Módulos, structs y funciones obligatorias

3.2.1. `main.c` / `main.h`

```
typedef struct Juego {
    Tablero *t; /* encapsula W/H y la politica de memoria/render */
    Armas armas;
    PoolAliens pool;
    int dificultad;
    int turno, vivos, jugador_x;
} Juego;
```

Rol: inicializa el juego (tablero + armas + pool), ejecuta el ciclo de turnos y libera recursos.

3.2.2. `tablero.c` / `tablero.h` — Tablero (`void ***`) + UI/HUD

```
typedef struct {
    int W, H;
    void ***celdas; /* celdas[y][x] -> (void*) que apunta a Celda* */
} Tablero;

struct Tablero* tablero_crear(int ancho, int alto);
void tablero_imprimir(struct Juego *juego);
void tablero_cerrar(struct Tablero *tablero);
```

Rol: administrar memoria del `void ***` y toda la visualización (tablero + HUD).

3.2.3. `entidades.c` / `entidades.h` — Aliens, movimiento y resolución de daños

```
typedef struct {
    int tipo; /* tipo de alien */
    int hp; /* vida actual */
    int x, y; /* posicion en el tablero */
    int dx; /* direccion horizontal (zig-zag), p.ej., -1/+1 */
} Alien;

typedef struct {
    Alien *alien; /* NULL si la celda esta vacia */
    int dano_pend; /* danio marcado a consolidar */
} Celda;
struct Juego; /* forward-declaration para punteros a funcion (armas) */

void mover_aliens(struct Juego *juego);
void resolver_danos(struct Juego *juego);
```

Rol: mover según tipo, registrar daño de disparos del turno y consolidar bajas/drops al cierre.

3.2.4. `armas.c` / `armas.h` — Punteros a función

```
struct Juego; /* declaracion adelantada para usar la siguiente... */
typedef bool (*FuncArmas)(struct Juego* j);
typedef struct {
```

```
    int ammo_perforador; /* arma 2 */
    int ammo_especial; /* arma 3 */
    FuncArmas fn[3]; /* 0..2 mapean las 3 armas */
} Armas;

bool disparar_armas(struct Juego *juego, int arma_id);
/* Tres armas */
bool arma_normal (struct Juego *juego); /* id 0 */
bool arma_perforador (struct Juego *juego); /* id 1 */
bool arma_especial (struct Juego *juego); /* id 2 */
```

Rol: validar recursos y disparar vía punteros a función; las armas sólo marcan daño (la eliminación real ocurre en `resolver_danos`).

3.2.5. `spawn.c` / `spawn.h` — Inventario único por partida y aparición

```
typedef struct {
    int especial; int skater; int drone;
    int vivos_tope; /* tope de aliens simultaneos en pantalla */
} PoolAliens;

void spawn_inicio(struct Juego *juego);
```

Rol: definir el pool por dificultad, sembrado inicial y apariciones.

4. Sobre la entrega

- El código debe venir indentado y ordenado.
- Se deben entregar los siguientes archivos:
  - `main.c`, `main.h`
  - `tablero.c`, `tablero.h`
  - `entidades.c`, `entidades.h`
  - `armas.c`, `armas.h`
  - `spawn.c`, `spawn.h`
  - `Makefile`
  - README.txt
- En el README deben ir todas las instrucciones para una correcta ejecución, así como información respecto al arma especial y alien especial, consideraciones a tener y supuestos utilizados.

▶ **El Uso de IA está prohibido**. Si se detecta, la tarea recibirá nota 0 sin derecho a apelación. (aplica para la totalidad de la entrega)

- Se puede crear nuevas funciones, estructuras y/o definiciones que estime conveniente. Como también agregar nuevos atributos a las estructuras existentes.

- La falta de declaración de los structs y funciones descritos en este documento en sus respectivos archivos de encabezado (.h) resultará en un 0 como nota final de la tarea.
- Es de **uso obligatorio la estructura Tablero y sus atributos para la entrega mínima**, así como la implementación de *Armas* a través de punteros a función como se describe en 3.2.4. de no usarse significará un 0 en la nota final de la tarea.
- No se permite cambiar las firmas, parámetros y retorno, como también agregar nuevos parámetros en las funciones a menos que sea consultado y aprobado previamente por un ayudante/profesor a través del foro.
- Las funciones deben ir comentadas, explicando clara y brevemente lo que realiza, los parámetros que recibe y los que devuelve (en caso de que devuelva algo).
- Si no existe orden en el código habrá descuento.
- Se permite únicamente el uso de la librería estándar de C. No está permitido incluir librerías externas, no estándar ni específicas del sistema operativo.
- Debe estar presente el archivo **Makefile** para que se efectúe la revisión, este debe compilar TODOS los archivos mediante compilación separada.
- Se utilizará Valgrind para detectar las fugas de memoria.
- El trabajo es individual obligatoriamente.
- Las copias serán evaluadas con nota 0 y se informará a las respectivas autoridades.
- Todas las funciones deben ir comentadas con el siguiente formato:

```
/*
 * Nombre:
 * Parámetros:
 * Retorno:
 * Descripción:
 */
```

  Se harán descuentos por función no comentada

- La entrega debe realizarse en un archivo comprimido en tar.gz y debe llevar el nombre: `Tarea2LP_RolAlumno.tar.gz`. Ej.: `Tarea2LP_202473000-k.tar.gz`.
- El archivo README.txt debe contener el nombre y rol del alumno e instrucciones detalladas para la correcta utilización de su programa.
- La entrega será vía aula y el plazo máximo de entrega es hasta el día **Miércoles 10 de Septiembre, 23:59 horas, vía aula.**
- Por cada día de atraso se descontarán 20 puntos (10 puntos dentro de la primera hora)
- Las copias y Uso de código generado por IA serán evaluadas con nota 0 y se informarán a las respectivas autoridades.
- Solo se pueden realizar consultas respecto a la tarea hasta 2 días antes de la entrega.
- Las consultas del enunciado se debe realizarse mediante el foro de la tarea disponible en AULA o Discord.

5. Calificación

5.1. Entrega

Para la calificación de su tarea, debe realizar una entrega con requerimientos mínimos que otorgarán 30 pts base, luego se entregará puntaje dependiendo de los otros requerimientos que llegue a cumplir.

5.1.1. Entrega Mínima (total 30 pts)

- Inicializa el tablero según la dificultad fácil, asignando espacio en la memoria *heap* para la matriz **void*** celdas** dentro del struct Tablero. La inicialización debe configurar solo el tamaño mínimo pedido, es decir 5x15. **(10 pts)**
  Notas adicionales: El tablero se debe inicializarse y borrarse a través de **malloc** y **free**, no puede haber *leaks* de memoria.

- Movimiento del jugador dentro de los márgenes del tablero y descenso cada 2 turnos de los aliens implementados correctamente sin comportamientos inesperados ni caídas del programa. **(10 pts)**
  Notas adicionales: Asumir que el jugador se moverá siempre dentro de los márgenes permitidos.

- implementación y correcto funcionamiento del arma especial de acuerdo a como la describan en su README. **(10 pts)**
  Notas adicionales: Se espera que el arma *especial* sea capaz de ser disparada e inflingir daño/eliminar a un alien, además de **al menos una de** la(s) funcionalidad(es) extra que le agreguen al arma. Si ustedes conocen de errores o *bugs* que pueda tener el arma especial deben ser informados a través del README para su correcta evaluación. No se considerará casos del estilo 'está implementada pero no se puede disparar', debe funcionar para por lo menos el caso general de su uso.

5.1.2. Asignación de puntajes general (total 70 pts)

1. **Armas y punteros a función (25 pts)**
   - **Normal**: disparo funciona como debe, marcando daño y eliminando aliens cuando corresponde, colisiones correctas (no atraviesa). (**5 pts**)
   - **Perforador**: disparo atraviesa aliens, daño 1 a *todos* en la columna, **7** disparos. (**7 pts**)
   - **Especial**: Disparo funciona según fue especificado en el README, sin errores ni comportamiento inesperado, implementación completa. **(3 pts)**
   - Inventario de armas mediante *punteros a función* (arreglo `fn[3]`), y `disparar_armas(arma_id)` funcional. (**5 pts**)
   - Disparo consume turno solo si procede + validación de munición/recursos. (**3 pts**)
   - Aliens tienen probabilidad de dar munición al morir. **(2 pts)**
2. **Flujo de turnos y entrada (10 pts)**
   - Entradas inválidas/no permitidas *no consumen turno.* (**2 pt**)
   - Disparo instantáneo con indicador de impacto claro y visible. (**3 pts**)

- Chequeo de término: derrota si un alien llega a $y = 0$. (**5 pts**)

3. **Aliens, movimiento y resolución de daño (15 pts)**
   - **Drone**: HP=2; baja recto cada 2 turnos. (**2 pts**)
   - **Skater**: zig–zag con rebote; baja cada 2 turnos. (**4 pts**)
   - **Especial**: mecánica/HP definida y coherente con reglas. (**5 pts**)
   - Restricción *un alien por celda* respetada. (**1 pts**)
   - `resolver_danos`: consolida bajas correctamente (liberación, conteos, drops). (**3 pts**)
4. **Dificultades, inventario y spawns (15 pts)**
   - Dimensiones por dificultad: Fácil 5×15, Difícil 7×15. (**3 pts**)
   - Inventario fijo por partida (totales y mínimos por tipo) según dificultad. (**3 pts**)
   - Tope de simultáneos: 6 (Fácil), 8 (Difícil). (**4 pts**)
   - Sembrado inicial: 2 (Fácil) / 3 (Difícil); probabilidades por tipo $\geq 15\,\%$. (**2 pts**)
   - Spawn por turno: 1 alien y con 30 % intentar un segundo; posponer si fila superior llena. (**3 pts**)
5. **HUD y visualización (5 pts)**
   - HUD completo: turno, *inventario restante*, *aliens vivos*, municiones (1/2/3) y mensajes de evento/entradas inválidas. (**3 pts**)
   - Render del tablero en cada turno con HUD integrado. (**2 pts**)

5.2. Descuentos

- Falta de declaraciones en archivos .h (-100 pts)
- falta de uso de tablero y/o punteros a función (armas) (-100 pts)
- Representación gráfica del tablero inconsistente o confusa (-10 a 25 puntos según gravedad)
- Porcentaje de leak de memoria (bytes asignados) ((1 - 5) % -3 pts, (6 - 50) % -15 pts, (51 - 90) % -30 pts, (91 - 100) % - 40 pts))
- Falta de comentarios sobre cada función (–5 pts c/u, máx. –20 pts).
- Uso de librerías **no estándar** de C. (-100 pts)
- Falta de orden (Max -20 pts)
- Código no compila (-100 pts)
- Compilación Unificada (-5 pts)
- Warnings (-5 pts c/u)
- Falta de Makefile (-100 pts)
- Falta de README (–20 pts)

- Falta de información obligatoria en el README (nombre, rol, instrucciones) (–5 pts c/u)
- Falta de orden o mala indentación en el código (–5 a –20 pts según gravedad)
- Mal nombre en los archivos entregados (–5 pts c/u; –10 pts si es el `.tar.gz`)
- Uso de código y/o texto generado por IA implicará un 0 y se le informará a las respectivas autoridades.
- Entrega tardía (–10 pts si es dentro de la primera hora; –20 pts por cada día o fracción de retraso)
- En caso de existir nota negativa, esta será remplazada por un 0.