

# Pong - Proyecto para "UADE::Introducción a la programación"

---

- Autor: Graziani Luciano [lgraziani@uade.edu.ar](mailto:lgraziani@uade.edu.ar). LU: 1222435.
- Materia: Introducción a la programación.
- Cursada: 2025.
- Fecha entrega: 25/06/2025.
- Profesor: Pepe Jonathan [jpepe@uadeeducar.onmicrosoft.com](mailto:jpepe@uadeeducar.onmicrosoft.com).
- Ayudante: Narducci Adrián [adnarducci@uade.edu.ar](mailto:adnarducci@uade.edu.ar).
- Repositorio en github: <https://github.com/lgraziani2712/uade-iprog-pong>.

## Herramientas utilizadas en el proyecto:

---

































- **Adobe Audition**: para recortar y comprimir sonidos.
- **Adobe Photoshop**: para editar las imágenes.
- **VSCode**: editor de código, compilador y debugger.
- **PowerShell**: terminal de línea de comandos.
- **Conan**: para administrar las dependencias del proyecto.
- **CMake**: para compilar la versión debugger y producción.
- **Clang**: para formatear el código.
- **Git**: para llevar historial de cambios y documentar el proceso.

## Recursos de terceros utilizados:

---

- Imágenes originales obtenidas a través de [duckduckgo](https://duckduckgo.com).
- Sonido original del fuego: <https://pixabay.com/sound-effects/short-fire-whoosh-1-317280/>.
- Stackoverflow y la web en general para las guías de uso de la librería SDL.
- Sonido original del punto: <https://pixabay.com/sound-effects/success-340660/>.
- Sonido original de navegación de menú: <https://pixabay.com/sound-effects/radio-338296/>.
- Sonido original de selección de opción del menú: <https://pixabay.com/sound-effects/8-bit-victory-sound-101319/>.
- Música: <https://pixabay.com/music/video-games-best-game-console-301284/>.
- Ícono original: [https://www.freepik.com/icon/ping-pong\\_1687641](https://www.freepik.com/icon/ping-pong_1687641).

## Proceso de desarrollo

Graph	Description	Date	Author	Commit
	<b>Uncommitted Changes (1)</b>	25 Jun 2025 1...	*	*
  <b>main</b>  <b>origin</b>	feat: v1.0.4 agrega versión en menú	25 Jun 2025 1...	Luciano Grazi...	aee6dbe1
	feat: v1.0.3 agrega título, sonidos extras y autoría	24 Jun 2025 1...	Luciano Grazi...	0fbf1eb7
	feat: v1.0.2 agrega música de fondo	24 Jun 2025 1...	Luciano Grazi...	b5716bda
	fix: v1.0.1 corrige bug de colisión	23 Jun 2025 2...	Luciano Grazi...	40a602b3
	feat: juego v1.0 termina modo pvp y pve	20 Jun 2025 2...	Luciano Grazi...	c8d56362
	feat: agrega estado de SAQUE al pong	20 Jun 2025 1...	Luciano Grazi...	95bf197d
	feat: agrega logo	20 Jun 2025 1...	Luciano Grazi...	46adfc66
	feat: agrega sonidos minecrafteros	20 Jun 2025 1...	Luciano Grazi...	3a7083cb
	feat: agrega skins a juego e integra sdl2/image	20 Jun 2025 1...	Luciano Grazi...	2e6470ef
	feat: agrega renderizado de tiempo en hud	19 Jun 2025 1...	Luciano Grazi...	11d72fdc
	feat: completa finalizado y agrega estado Ranking	19 Jun 2025 1...	Luciano Grazi...	1e22bbd4
	feat: agrega pausa/fin al pong y estado finalizado	19 Jun 2025 0...	Luciano Grazi...	11ab269b
	feat: maneja estado de pausa	18 Jun 2025 2...	Luciano Grazi...	f427fb36
	feat: agrega menú principal y máquina de estado	18 Jun 2025 1...	Luciano Grazi...	561e6a9a
	feat: agrega audios a la pelota	17 Jun 2025 1...	Luciano Grazi...	24009584
	feat: agrega contador y aumenta con cada punto realizado	17 Jun 2025 1...	Luciano Grazi...	3b84ebc8
	feat: agrega colision de pelota con pared	17 Jun 2025 0...	Luciano Grazi...	28d2f4f4
	feat: mejora lógica de colisión	16 Jun 2025 2...	Luciano Grazi...	74e819df
	refactor: simplifica gameloop al eliminar timer personalizado	14 Jun 2025 1...	Luciano Grazi...	6bbbd8e
	feat: mejora gameloop	14 Jun 2025 1...	Luciano Grazi...	f4746085
	feat: agrega físicas para paleta y pelota	14 Jun 2025 1...	Luciano Grazi...	d05ded70
	chore: agrega inputs para paletas	11 Jun 2025 1...	Luciano Grazi...	16fbfee0
	chore: agrega puntaje_jugador	5 Jun 2025 12...	Luciano Grazi...	47d6d024
	chore: agrega implementacion de paleta	5 Jun 2025 10...	Luciano Grazi...	411dd97b
	refactor: reescribe pong_render como clase y agrega modelo pelota	5 Jun 2025 10...	Luciano Grazi...	403aa458
	chore: edita nombres al español	4 Jun 2025 18...	Luciano Grazi...	e05331ab
	chore: configura CMake para consumir todos los archivos de src	4 Jun 2025 17...	Luciano Grazi...	80839a98
	feat: agrega archivos para cada parte de la app y el gameloop	4 Jun 2025 17...	Luciano Grazi...	a39a9292
	Hello World	29 May 2025 ...	Luciano Grazi...	6e41a1cf

## Requisitos técnicos

Este proyecto funciona con **Conan** + **CMake** para simplificar el proceso de instalación de dependencias, *build*eo y *debug*geo.

Antes de correr nada, necesitamos asegurarnos que tenemos instaladas estas dependencias en nuestra computadora:

### Conan

De forma general y muy resumidamente, Conan es una herramienta para administrar dependencias de un proyecto escrito en C o C++. Para poder instalar Conan necesitamos tener instalado `python`. Python está fuera del alcance de este proyecto por lo que se recomienda buscar la forma oficial de instalarlo a parte. A continuación el comando para instalar Conan:

```
pip install conan
```

Tener `python` y con este comando recomendado de forma oficial como el proceso de instalación estándar nos aseguramos de tener autocompletado en los archivos `conanfile.py`.

### Configuración extra para Conan

- **Cambiar el path por defecto donde Conan instala dependencias:** por defecto Conan guarda toda la información dentro de `C:\{USER}\.conan`. Si se desea cambiar el path por defecto, se debe crear una **variable de entorno** con `CONAN_HOME` como valor de la clave y un path como valor, por ejemplo: `CONAN_HOME="D:\.conan"`.

### CMake

CMake, `cl.exe` y todas las herramientas del ecosistema de C++ son instaladas por MSVC de Visual Studio. En el installer de Visual Studio debería haberse seleccionado la opción "Desktop development with C++".

## Comandos comunes

Si ejecutamos comandos en la terminal, necesitamos abrir la terminal "Developer Command Prompt for VS 2022".

### (Primer comando) Configurar el perfil por defecto de conan:

```
conan profile detect
```

### Instalar dependencias para builds release y configurar conan:

```
conan install . --build=missing
```

Con este comando Conan instala las dependencias especificadas en el archivo de recetas `conanfile.py`. Si alguna de estas dependencias no existe en el caché local, los buscará en los repositorios remotos. Cuando el grafo de dependencias esté computado y todas las dependencias fueran encontradas, buscará los binarios que coincidan con las settings de la computadora actual (que fueron generadas durante la detección del perfil).

Si no hay un binario para alguna de las dependencias, fallará, a menos que se haya pasado el argumento `--build`, que indica a Conan que compile las dependencias de forma local.

Luego de la instalación de las dependencias, Conan llamará a los "generators" y "deployers".

Esta documentación fue traducida sin inteligencia artificial del texto que el comando `conan install --help` entrega.

### Instalar y construir dependencias con el perfil de Debug:

NOTA: no es necesario y lleva mucho tiempo, a menos que se quieran correr el entorno de debugeo.

```
conan install . -s build_type=Debug --build=missing
```

## Construir la aplicación con CMake

```
cmake --preset conan-default
```

Para build de debuggeo:

```
cmake --build --preset conan-debug
```

Para build release:

```
cmake --build --preset conan-release
```

Estos últimos dos comandos crean los ejecutables dentro de las carpetas `build/Debug` y `build/Release` respectivamente.

## Configuración básica para VSCode

---

- Las extensiones recomendadas están declaradas en `.vscode/extensions.json`.
- Ejecutar los comandos básicos para Conan y CMake para generar los presets de build al menos una vez antes de abrir VSCode.
- La tarea para regenerar el build de Debug está configurada en `.vscode/tasks.json`.
- La ejecución por defecto está configurada para que levante la versión de Debug y definida en `.vscode/launch.json`.

## Configuración del entorno

---

Conan detecta y declara las siguientes propiedades en el perfil por defecto:

```
[settings]
arch=x86_64
build_type=Release
compiler=msvc
compiler.cppstd=20
compiler.runtime=dynamic
compiler.version=194
os=Windows
```

RECOMENDACIÓN: cambiar el valor de la entrada `compiler.cppstd` a 20 mínimo.

Para saber dónde guarda el perfil por defecto podemos correr el siguiente comando:

```
conan profile path default
```