Curso Sistemas Físicos Interactivos 1

Versión rc

Juan Franco

15 de abril de 2023

SECCIONES

Docente diseñador del curso: Juan Fernando Franco Higuita

Nombre del docente moderador: Juan Fernando Franco Higuita

Programa: Ingeniería en Diseño de Entretenimiento Digital

Créditos y horas totales del curso: 2 créditos - 96 horas totales

CAPÍTULO 1

INTRODUCCIÓN AL CURSO

1.1 Descripción del curso

El objeto de este curso es aprender a integrar dispositivos periféricos a sistemas de cómputo para la construcción de aplicaciones interactivas.

¿Por ejemplo?

- Interactive Momento Factory
- Lumia de Moment Factory
- ARcade de Moment Factory
- TDAxis trabajo de grado entretenimiento digital
- The VOID
- AR + IoT
- Demo UPB
- Mario Kart
- Proyecto Galia
- AR Snapdragon Spaces
- Haptic VR Gloves

1.2 Propósito del curso

El entendimiento y uso de los fundamentos, herramientas y procesos mediante los cuales es posible integrar dispositivos periféricos a sistemas de cómputo con el fin de lograr que una aplicación interactiva entienda y modifique su entorno físico en base a unos requerimientos y metas de diseño.

1.3 Competencias

1. Construye aplicaciones que posibilitan la interacción entre personas, mediada por tecnologías digitales, utilizando lenguajes y metodologías apropiadas según el contexto (Ingeniería de software).
2. Materializa sistemas intermediados por el entretenimiento digital para resolver problemas de acuerdo con requerimientos condicionados por el contexto (Materialización).

1.4 Resultados de aprendizaje

1. RA1: aplico los conceptos necesarios para el correcto diseño, implementación, funcionamiento y diagnóstico del software en la producción de sistemas de entretenimiento digital utilizando los procedimientos y herramientas adecuadas según el contexto (para la competencia 1).
2. RA2: integro dispositivos de entrada, salida e interfaces mecánicas con sistemas de cómputo para la creación de sistemas intermediados por el entretenimiento digital (para la competencia 2).

1.5 Carta descriptiva del curso

En este enlace puedes descargar la carta descriptiva del curso.

1.6 Proyecto docente

El proyecto docente lo puedes descargar de este enlace.

1.7 Estructura y metodología del curso

El curso está divido en 4 unidades más una introducción corta para aprender a entregar las evaluaciones con control de versión:

1. Unidad 1: Software para sistemas embebidos
2. Unidad 2: Protocolos ASCII
3. Unidad 3: Protocolos binarios
4. Unidad 4: Plataformas de software interactivas de tiempo real

La metodología del curso es de aula invertida. En este sitio web está todo el material necesario para que puedas preparar la evaluación de cada unidad. Debes utilizar el tiempo autónomo para preparar el material y resolver los ejercicios y retos en clase con ayuda de tus compañeros de equipo y la asesoría del profesor.

1.8 Cronograma

- Introducción y control de versión: semana 1
- Unidad 1: semanas 2 a 6
- Unidad 2: semanas 7 a 10
- Unidad 3: semanas 11 a 13
- Unidad 4: semanas 14 a 16

1.9 Evaluación

- Control de versión. Semana 1. 5 %
- Unidad 1: 25 %. Semana 5.
- Unidad 2: 25 %. Semana 9.
- Unidad 3: 25 %. Semana 13.
- Unidad 4: 20 %. Semana 16.

1.10 ¿Cuándo se consideran entregada una unidad?

Cuando cumplas TODAS las condiciones siguientes:

- Incluye toda la documentación solicitada.
- Incluye los enlaces a videos públicos en youtube donde se muestre la evaluación funcionado (unidades 1 a 4).
- El aval de unidad entregada por parte del profesor.

Advertencia: IMPORTANTE

No hay entregas parciales. Repito. No hay entregas parciales. Si tu trabajo está incompleto el docente simplemente no te dará el aval de entregado. Recuerda que una vez tengas el aval la nota en el sistema de la unidad será 5.

1.11 Dedicación

Este curso es de 2 créditos con encuentros presenciales semanales de 3 hora 20 minutos y 2 hora 40 minutos de trabajo autónomo.

1.12 Evidencias de evaluación

En cada unidad te indicaré las consideraciones para entregar la evaluación; sin embargo, ten presente SIEMPRE este código de honor:

1.12.1 Código de honor

Para realizar el trabajo de cada unidad se espera que hagas lo siguiente:

- Colabora con tus compañeros cuando así se indique.
- Trabaja de manera individual cuando la actividad así te lo proponga.
- No busques la solución a los ejercicios y proyectos porque DAÑARÍAS tu proceso de aprendizaje. Recuerda, se trata de seguir un camino y aprender en el recorrido.
- ¿Entonces qué hacer si no me funciona algo? Te propongo que experimentes, crea hipótesis, experimenta de nuevo, observa y concluye.
- NO OLVIDES, este curso se trata de pensar y experimentar NO de BUSCAR soluciones en Internet.

1.13 Bitácora

- GitHub
- Crea una cuenta en GitHub a menos que ya la tengas.

CAPÍTULO 2

¿Qué herramientas necesito para el curso?

Para este curso vas a necesitar algunos programas y dispositivos electrónicos que conectarás por USB al computador.

2.1 ¿Qué software requieres?

Vamos a utilizar los computadores de la Universidad con sistema operativo windows. En la documentación del curso te iré sugiriendo que instales también los programas en tu equipo personal para que puedas practicar en casa.

2.2 ¿Hardware para el curso?

Requerimos un sistema de desarrollo microcontrolado. Te daré más detalles en clase para contarte qué necesitarás.

CAPÍTULO 3

Introducción al control de versión con git y GitHub

Con esta guía aprenderás los herramientas básicas para realizar todos los proyectos y ejercicios del curso bajo control de versión.

3.1 Evaluación

Antes de comenzar:

- Abre el browser e ingresa a Github.
- Si estás en una cuenta que no es la tuya realiza un sign out (esquina superior derecha desplegando el menú).
- Ingresa a tu cuenta.
- Ingresa a este este sitio para unirte al classroom del curso. Busca tu nombre y ID. En este paso asociarás tu cuenta de Github al classroom del curso.
- Acepta el assigment.
- Ya estamos listo para comenzar la evaluación.

3.1.1 Enunciado

En el repositorio remoto se debe evidenciar:

1. Un commit con la creación de un archivo.
2. Un commit con la creación de dos archivos.
3. Un commit con la edición de un archivo ya añadido.
4. Un commit regresando a una versión anterior del repositorio.
5. El archivo README.md con:
    - Un título de tipo H1 que diga: EVALUACIÓN CONTROL DE VERSIÓN.

- Coloca tu nombre y ID.
- Indica qué comandos se requieren para este proceso:
  - Clonar el repositorio.
  - Autenticar la terminal en Github.
  - Configurar el nombre de usuario y correo electrónico para este repositorio.
  - Hacer los commits solicitados.
  - Enviar los cambios del repositorio local al remoto.
  - Hacer un logout del cliente Github de la terminal si tienes gh, si no tienes gh entonces qué debes hacer con el credential manager de windows.
  - Salirse de la cuenta de Github en el browser (en este caso describe qué debes hacer).

Nota: MUY IMPORTANTE

Recuerda la condiciones para considerar que la evaluación está entregada.

3.2 Trayecto de actividades

3.2.1 Lectura 1: sistemas de control de versión

En este curso vas a realizar todos los ejercicios y evaluaciones usando un sistema de control de versión: Git y Github.

3.2.2 ¿Qué es un sistema de control versión?

Cuando estás desarrollando software, alguna vez te ha pasado que terminas nombrando tus archivos así:

```
* Versión buena con un error
* Versión casi lista con un bug
* Versión para compartir con Camila
* Versión para enviar al profesor
* Esta versión si está bien
* Versión 1, versión 1-1, versión 2, versión 3
* versión enero 11 de 2022.
```

¿No sería ideal que el nombre de una archivo siempre fuera el mismo y existiera una forma de acceder a todo el historial de cambios del archivo?

Lo anterior lo puedes lograr con un sistema de control de versión. Un sistema de control de versión te ayuda a evitar la necesidad de guardar tus archivos con nombres diferentes a medida que realizas cambios, incluyes nuevas características o tienes alguna nueva receta de archivos para producir tu programa. El sistema de control de versión te ayudará a gestionar la versión de los archivos de manera automática evitando procesos manuales tediosos y susceptibles al error.

El sistema de control de versión ES UN PROGRAMA (que instalas en tu computador) que te permitirá trazar y guardar información de los cambios que haces a tus archivos en el tiempo. Podrás recuperar incluso una versión pasada de un archivo si descubres que cometiste un error.

¿Te va sonando?

Quiero contarte además que hoy en día prácticamente es impensable una empresa que desarrolle cualquier producto de software que NO TENGA control de versión.

3.2.3 ¿Qué es Git y GitHub?

Git es un sistema de control de versión libre y de código abierto que instalas en tu computador para realizar el control de versión de tus proyectos. Por su parte GitHub te permite guardar tus proyectos de software en un servidor en Internet con la información del control de versión que tienes en tu computador. ¿Para qué quieres esto? Para compartir tu código, para hacer copias de seguridad, para mostrar tus habilidades y portafolio y SOBRE TODO para trabajar en EQUIPO.

Por medio de GitHub, los aportes de cada miembro del equipo se pueden sincronizar y compartir. De esta manera, es posible construir productos de software muy complejos en los cuales participen MUCHOS desarrolladores.

3.2.4 Ejercicio 1: introducción a la terminal

Para realizar el control de versión de tus programas vas a usar inicialmente la terminal. Una vez estés familiarizado con esta puedes explorar otras herramientas; sin embargo, la ventaja de la terminal es su rapidez y que definitivamente te obliga a entender qué estás haciendo. Esto es importante porque luego este conocimiento lo podrás extrapolar a cualquier herramienta gráfica.

Es posible que esta sea tu primera experiencia con la terminal. La terminal es un programa que te permite interactuar con el sistema operativo y los programas que tienes instalados por medio de comandos. Es por ello que a la terminal también la conocemos como interfaz de línea de comandos.

Abre la terminal y escribe en el buscador de aplicaciones la palabra `terminal`. Escribe el siguiente comando:

```
pwd
```

En mi caso (en tu caso será distinto) el resultado es:

```
/home/jfupb
```

Acabas de escribir tu primer comando en la terminal. `pwd` te permite conocer la ruta en la cual estás posicionado en el sistema de archivos. Por el momento, piensa en el sistema de archivos como una forma de organizar la información en el computador usando DIRECTORIOS.

Ahora vas a crear un nuevo DIRECTORIO:

```
mkdir demo1
```

Nota: RECUERDA

¿Qué comando debes ejecutar para saber en qué directorio estás posicionado en este momento?

¿Y si quieres posicionarte en el nuevo directorio que acabas de crear? Ejecutas el comando `cd` que quiere decir `change directory`:

```
cd demo1
```

Para listar el contenido del nuevo directorio deberás escribir el comando:

```
ls -al
```

Verás algo como esto:

```
total 8
drwxrwxr-x  2 jfupb jfupb 4096 Jan 11 15:40 .
drwxr-x--- 37 jfupb jfupb 4096 Jan 11 15:43 ..
```

Te estarás preguntando, qué es `.` y `..`. Se trata de referencias a dos directorios. `.` se refiere al directorio actual y `..` se refiere al directorio padre. Entonces, si escribes este comando:

```
cd ..
```

Nota: RETO

¿Cuál crees que sea el resultado?

¿Perdido? No te preocupes. Repitamos el proceso juntos. Supón que la posición actual es:

```
pwd
/home/jfupb/demo1
```

Luego de ejecutar el comando:

```
cd ..
```

El resultado será:

```
pwd
/home/jfupb
```

Nota: RECUERDA

En este momento debes estar en el directorio padre del directorio demo1. ¿Te cambias de nuevo al directorio demo1 por fa?

Debiste hacer algo como esto:

```
cd demo1
```

Ahora regresa de nuevo al directorio padre de demo1 y una vez estés allí ejecuta los comandos:

```
cd ./demo1
pwd
```

El resultado será:

```
/home/jfupb/demo1
```

¿Te diste cuenta?

Nota: RECUERDA

La entrada . se refiere al directorio actual y `..` se refiere al directorio padre del directorio actual.

Al cambiarte al padre de demo1, `.` se refiere al directorio padre de `demo1`. Por tanto, `./demo1` será la ruta RELATIVA de demo1 con respecto a su padre.

3.2.5 Ejercicio 2: Vas a practicar

Ahora te voy a pedir que hagas varias cosas y preguntes si tienes dudas:

- Crea el directorio demo2 en demo1. ¿Recuerdas cómo listar el contenido de un directorio?
- Cámbiate al directorio padre de demo1 y desde allí crea el directorio demo3 en el directorio demo2.
- ¿Cuál será la ruta relativa de demo3 con respecto al padre de demo1?

Advertencia: ALERTA DE SPOILER

Crea el directorio demo2 en demo1. ¿Recuerdas cómo listar el contenido de un directorio?:

```
mkdir demo2
ls -al
```

Cámbiate al directorio padre de demo1 y desde allí crea el directorio demo3 en el directorio demo2. Asumiendo que estás posicionado en demo1:

```
cd ..
mkdir ./demo1/demo2/demo3
```

¿Cuál será la ruta relativa de demo3 con respecto a al padre de demo1?:

```
../demo1/demo2/demo3
```

3.2.6 Ejercicio 3: experimenta

¿Qué comandos has visto hasta ahora?:

```
pwd
ls -al
cd
mkdir
```

Ahora tómate unos minutos para experimentar. ¿Cómo?

- Inventa tus propios ejemplo o retos.
- Antes de ejecutar un comando PIENSA cuál sería el resultado. Si el resultado es como te lo imaginaste, en hora buena, vas bien. Si no es así, MUCHO mejor, tienes una oportunidad de oro para aprender. Entonces trata de explicar qué está mal, discute con otros compañeros y si quieres habla con el profe.

3.2.7 Ejercicio 4: recuerda (evaluación formativa)

De nuevo tómate unos minutos para:

1. Listar cada uno de los comandos que has aprendido hasta ahora y escribe al frete de cada uno qué hace.
2. ¿Qué es una ruta absoluta?
3. ¿Qué es una ruta relativa?

3.2.8 Ejercicio 5: tu primer proyecto bajo control de versión

- Crea un directorio llamado project1 (mkdir)
- Cámbiate a ese directorio (cd)

En `project1` vas a simular la creación de un proyecto de software.

Ahora crea un archivo en el directorio:

```
touch main.c
```

Abre el directorio:

```
code .
```

Advertencia: MUY IMPORTANTE

Siempre que trabajes en visual studio code abre DIRECTORIOS completos, no ARCHIVOS individuales.

`code` es el comando que escribes en la terminal para abrir el programa visual studio code. ¿Qué significa el `.` luego del comando?

Nota: ALERTA DE SPOILER

No olvides que la entrada de directorio `.` se refiere al directorio actual en el que estás posicionado.

Trata de recordar de nuevo ¿Qué era `..`?

Ahora modifica el archivo main.c con el siguiente código:

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("La vida es bella\n");
    return(EXIT_SUCCESS);
}
```

Antes de continuar ejecuta el comando:

```
ls -al
```

Deberías tener solo tres entradas:

```
.
..
main.c
```

Ahora si vamos a crear el repositorio:

```
git init
```

Y solo con esto ya tienes un proyecto con control de versión. ¿Fácil, no?

Escribe en la terminal el comando:

```
ls -al
```

Notas que hay una nuevo directorio que no tenías antes:

```
.
..
main.c
.git
```

Ese directorio `.git` es lo que llamamos un `REPOSITORIO DE GIT`. En ese repositorio el sistema de control de versión que tenemos instalado realizará el control de versión de todo lo que le indiquemos. Ten presente que en este repositorio, Git guardará toda la información relacionada con los cambios e historia de los archivos de tu proyecto que estén bajo control de versión. Puedes pensar que el repositorio es una especie de base de datos donde Git almacena un diario de qué está pasando con cada uno de los archivos de tu proyecto.

3.2.9 Ejercicio 6: configura Git

Para hacer tus primeros experimentos con Git vas a realizar unas configuraciones mínimas para informarle a Git un nombre de usuario y un correo. Esta información permite que Git identifique a la persona responsable de realizar los cambios a un archivo. Recuerda que Git está diseñado para que puedas trabajar en equipo.

Escribe los siguientes comandos, pero cambia name y email por tus datos:

```
git config --local user.name "yo"
git config --local user.email "yo@yolandia.com"
```

3.2.10 Ejercicio 7: para pensar

¿Qué crees qué pase si borras el directorio `.git` en relación con el historial de cambios de tus archivos?

¿Qué crees que pase si creas un directorio vacío y mueves allí todo los archivos de tu proyecto incluyendo el directorio .git?

3.2.11 Ejercicio 8: reconocer el estado del repositorio

Ahora ejecuta el siguiente comando:

```
git status
```

Verás algo así:

```
On branch master

No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
    main.c

nothing added to commit but untracked files present (use "git add" to track)
```

El resultado por ahora es muy interesante. Verás que estás trabajando en la rama (branch) master. Las ramas son una característica MUY útil de Git. Como su nombre indica te puedes ir por las ramas. Te lo explico con una historia. Supón que estás trabajando en tu proyecto y se te ocurre una idea, algo nuevo para implementar; sin embargo, no quieres dañar tu proyecto principal. Entonces lo que haces es que te creas una rama que tomará como punto de partida el estado actual de tu proyecto. En esa nueva rama realizas los ensayos que quieras. Si al final no te gusta el resultado, simplemente destruyes la rama y tu proyecto seguirá como lo habías dejado antes de crear la rama. Pero si el resultado te gusta entonces podrás hacer un `MERGE` e incorporar las ideas de la nueva rama a la rama inicial. Ten presente que si no quieres trabajar en la nueva rama y deseas retomar el trabajo en la rama principal lo puedes hacer, te puedes cambiar de ramas. Incluso puedes crear muchas más y probar varias ideas en simultáneo.

Ahora observa el mensaje `No commits yet`. Este mensaje quiere decir que aún no has guardado nada en el repositorio. Luego te dice `Untracked files` y te muestra una lista de los archivos detectados en tu proyecto (main.c en este caso), pero que no están bajo control de versión. Tu debes decirle explícitamente a Git a qué archivos debe hacer `tracking`. Finalmente, `nothing added to commit but untracked files present (use "git add" to track)` quiere decir que si en este momento le pides a Git que guarde en el repositorio una `FOTO` (`commit`) del estado actual de los archivos que están bajo tracking, Git te dice que no hay nada para guardar. Nota que Git da sugerencias: `(use "git add" to track)`, es decir, te dice qué necesitas hacer para colocar el archivo main.c en tracking.

3.2.12 Ejercicio 9: adiciona tu primer archivo al repositorio

```
git add main.c
```

Y de nuevo observa el estado del repositorio:

```
git status
```

El resultado será:

```
On branch master

No commits yet

Changes to be committed:
(use "git rm --cached <file>..." to unstage)
    new file:   main.c
```

Te explico con una metáfora lo que está pasando. Imagina que Git le toma fotos al estado de tu proyecto cada que se lo solicitas; sin embargo, antes de tomar la foto tienes que decirle a Git (`con add`) a qué archivos le tomará la foto. Todos los archivos que serán tenidos en cuenta para la próxima foto se ubican en una zona lógica denominada el `STAGE`. Mira el mensaje `(use "git rm --cached <file>..." to unstage)`. Observa que Git te está diciendo que main.c ya está listo para la foto (`Changes to be committed`), pero si te arrepientes de incluir el archivo en la foto puedes ejecutar el comando sugerido. Prueba sacar de la foto a main.c:

```
git rm --cache main.c
```

Mira el estado del repositorio:

```
git status
```

Verás algo así:

```
On branch master
```

(continué en la próxima página)

(proviene de la página anterior)

```
No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
    main.c

nothing added to commit but untracked files present (use "git add" to track)
```

¿Te das cuenta? Acabas de sacar de la foto (DEL STAGE) a main.c. Ahora vuelve a invitar a main.c a la foto:

```
git add main.c
```

Ahora `TOMA LA FOTO` (realiza el commit):

```
git commit -m "Initial version of the project main file"
```

Consulta el estado del repositorio:

```
git status
```

El resultado será:

```
On branch master
nothing to commit, working tree clean
```

Puedes ver que Git está observando todo lo que pasa en el directorio de tu proyecto. Por ahora Git sabe que no has hecho nada más y por eso te dice `nothing to commit, working tree clean`.

Lo último que te voy a pedir que hagas con este ejercicio es que le preguntes a Git qué fotos (`COMMITS`) se han tomado en el repositorio:

```
git log
```

El resultado es:

```
commit 1f2009fabfc4895ee6b063c23c6f5c7ea7175209 (HEAD -> master)
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 10:52:46 2022 -0500

    Initial version of the project main file
```

Nota que el commit está identificado con el hash `1f2009fabfc4895ee6b063c23c6f5c7ea7175209`, el autor, correo, fecha, hora y la descripción del commit.

3.2.13 Ejercicio 10: recuerda

Para un momento. Repasa los ejercicios anteriores, actualiza tu lista de comandos con la explicación de qué hacen.

3.2.14 Ejercicio 11: modificar el contenido de un archivo

Modifica el contenido del archivo main.c añadiendo otro mensaje para imprimir (escribe lo que tu corazón te dicte). `Salva el archivo`. NO LO OLVIDES, salva el archivo.

Al verificar el estado del repositorio verás:

```
On branch master
Changes not staged for commit:
(use "git add <file>..." to update what will be committed)
(use "git restore <file>..." to discard changes in working directory)
    modified:   main.c

no changes added to commit (use "git add" and/or "git commit -a")
```

¿Ves la diferencia con respecto al momento en el que creaste el archivo? Déjame recordarte el mensaje:

```
On branch master

No commits yet

Untracked files:
(use "git add <file>..." to include in what will be committed)
    main.c

nothing added to commit but untracked files present (use "git add" to track)
```

Nota que al crear el archivo, Git te dice que no le está haciendo seguimiento (untracked); sin embargo, una vez está creado el archivo y lo modificas, Git te dice `Changes not staged for commit`.

En este caso, Git le hace tracking a tu archivo, pero tu no has decidido pasar el archivo a `STAGE` para poderle tomar la foto con los cambios que tiene ahora. ¿Cómo lo haces? Mira que en el mensaje Git te dice: `git add main.c`. Nota que Git también te dice que puedes descartar los cambios en el archivo con `git restore main.c`. ¿Por qué no haces la prueba?

Escribe:

```
git restore main.c
```

Vuelve a visual studio code y verifica qué paso con el archivo.

¿Ya no está la modificación anterior, cierto? Mira el estado del repositorio:

```
On branch master
nothing to commit, working tree clean
```

Vuelve a modificar main.c, pero esta vez si guardarás los cambios en el repositorio. Recuerda los pasos:

1. Cambias el archivo
2. Verifica el estado del repositorio (status)
3. Adiciona los cambios en el STAGE (add)
4. Toma la foto (commit)
5. Verifica de nuevo el estado del repositorio (status)
6. Verifica el historial del repositorio (log)

Te debe quedar algo así:

```
commit 2a0afbb7efa9c58a364143edf6c5cf76dccfab0b (HEAD -> master)
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 11:02:03 2022 -0500

    add a new print

commit 1f2009fabfc4895ee6b063c23c6f5c7ea7175209
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 10:52:46 2022 -0500

    Initial version of the project main file
```

Y ahora main.c está así:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("La vida es bella\n");
    printf("El feo es uno\n");
    return(EXIT_SUCCESS);
}
```

3.2.15 Ejercicio 12: volver a una versión anterior del proyecto

Ahora supón que quieres volver a una versión anterior del proyecto. Git ofrece varias alternativas que irás aprendiendo con el tiempo. Por ahora, piensa que lo que harás es pedirle a Git que traiga una versión del pasado y haga un nuevo commit de esa versión en el presente.

¿Cuál versión del proyecto quieres recuperar? Para saberlo puedes leer el historial de mensajes que adicionaste a cada COMMIT:

```
git log --oneline
```

En el ejemplo que estás trabajando:

```
2a0afbb (HEAD -> master) add a new print
1f2009f Initial version of the project main file
```

Ahora digamos que deseas ver cómo estaba el proyecto en el commit 1f2009f (estos son los primeros 7 números del identificador del commit o hash único que se calcula con el algoritmo sha-1):

```
git checkout 1f2009f
```

El resultado es:

```
Note: switching to '1f2009f'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.
```

(continué en la próxima página)

(proviene de la página anterior)

```
If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

git switch -c <new-branch-name>

Or undo this operation with:

git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 1f2009f Initial version of the project main file
```

Escribe el comando:

```
git status
```

El resultado es:

```
HEAD detached at 1f2009f
nothing to commit, working tree clean
```

Ahora revisa el archivo main.c. ¿Qué concluyes hasta ahora? En este momento estás en un estado especial llamado detached HEAD. En este estado puedes jugar con el código y hacer ensayos y luego puedes descartar todo lo que hagas sin dañar lo que ya tenías. Mira que Git te dice qué debes hacer para conservar los experimentos o para descartarlos.

En este caso, supon que solo quieres ver el estado del archivo main.c en el commit 1f2009f:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("La vida es bella\n");
    return(EXIT_SUCCESS);
}
```

¿Quieres volver main.c al último commit? Simplemente escribes:

```
git switch -
```

Ahora main.c se verá así:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("La vida es bella\n");
    printf("El feo es uno\n");
    return(EXIT_SUCCESS);
}
```

Luego de analizar las dos versiones de main.c decides que vas a conservar la versión del commit 1f2009f. Para que compares escribe:

```
git log --oneline
```

El resultado:

```
2a0afbb (HEAD -> master) add a new print
1f2009f Initial version of the project main file
```

Ahora:

```
git revert HEAD
```

El resultado:

```
[master 882d93e] Revert "add a new print"
1 file changed, 1 deletion(-)
```

Y si observas el historial:

```
git log --oneline
```

Verás:

```
882d93e (HEAD -> master) Revert "add a new print"
2a0afbb add a new print
1f2009f Initial version of the project main file
```

Si abres el archivo main.c:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("La vida es bella\n");
    return(EXIT_SUCCESS);
}
```

Entonces el comando:

```
git revert HEAD
```

Hace un `revert` del commit `2a0afbb` creando un nuevo commit, el `882d93e`, con el estado del proyecto en el commit `1f2009f`.

3.2.16 Ejercicio 13: configura GitHub

Advertencia: NECESITAS TENER INSTALADO UN PROGRAMA

Para realizar este ejercicio necesitas instalar un programa llamado Github CLI. Es posible que este programa no esté instalada en los computadores de la U. Lo puedes probar escribiendo en la terminal el comando gh auth logout. Si no funciona, no vas a poder realizar el ejercicio como te lo propongo y en ese caso salta al ejercicio 14a para probar una alternativa.

Ahora te pediré que compartas el repositorio local `project1` con el mundo. Para hacerlo necesitarás usar GitHub.

Abre tu browser y cierra la cuenta que esté activa en GitHub en este momento, claro, a menos que sea tu cuenta.

Abre una terminal y ejecuta el comando:

```
gh auth logout
```

Este comando termina la sesión del cliente de Git de tu computador con el servidor de Github. Pero el cliente de Git que corre en el browser sigue funcionando con el usuario actual. Ten presente que CONTROLAR quien está autenticado con el servidor lo haces cuando compartes computador con otros compañeros, pero si estás trabajando con tu computador personal no es necesario.

Ahora conecta el cliente local de git con tu cuenta de GitHub:

```
gh auth login
```

Acepta todas las opciones por defecto. Una vez hagas todo correctamente saldrá algo similar a esto:

```
✓ Authentication complete.
- gh config set -h github.com git_protocol https
✓ Configured git protocol
✓ Logged in as juanferfranco
```

El comando anterior te permitirá autorizar el acceso desde la termina de tu computador a tu cuenta en GitHub por medio de un proceso interactivo entre la terminal y el browser. Recuerda que en el browser ya tienes acceso a tu cuenta en el servidor.

En este punto tu computador tiene dos clientes autenticados con GitHub: la terminal y el browser.

3.2.17 Ejercicio 14: comparte tu trabajo usando GitHub

Ahora ejecuta el siguiente comando:

```
gh repo create project1 --public --source=. --push --remote=origin
```

Si todo sale bien verás esto:

```
✓ Created repository juanferfranco/project1 on GitHub
✓ Added remote https://github.com/juanferfranco/project1.git
✓ Pushed commits to https://github.com/juanferfranco/project1.git
  project1 git:(master)
```

¿Qué estás haciendo? `gh repo create project1` te permiten crear el repositorio remoto project1 en GitHub. `--public` hace que el repositorio sea público y lo puedas compartir con cualquier persona. `--source=.` especifica en dónde está el repositorio local que enviarás a Internet. `--push` permite enviar todos los commits locales al repositorio remoto. Finalmente, `--remote=origin` permite asignarle un nombre corto al servidor remoto, en este caso `origin`.

Ingresa al sitio: https://github.com/TU_USUARIO/project1 para observar tu repositorio en GitHub. NO OLVIDES modificar la cadena `TU_USUARIO` con tu nombre de usuario en GitHub.

3.2.18 Ejercicio 14a: creación manual del repositor en GitHub

Advertencia: TEN CUIDADO CON ESTE EJERCICIO

Este ejercicio solo tendrás que hacerlo si los ejercicios 13 y 14 no los pudiste hacer porque el programa gh no está instalado.

La idea de este ejercicio es que aprendas a publicar y sincronizar un repositorio local con un repositorio en GitHub.

- Ingresa a tu cuenta en Github.
- Selecciona la pestaña (tab) repositorios.
- Crea un nuevo repositorio con el botón NEW. Github te pedirá unos datos. Por ahora, solo coloca el nombre del repositorio, déjalo público y salta hasta el botón Create repository.
- Se debe crear un repositorio vacío. Busca la sección `or push an existing repository from the command line`.
- Escribe en la terminal el primer comando. El que comienza con `git remote`. Aquí lo que estás haciendo es decirle a tu Git local que guarde en tu repositorio local una referencia a un repositorio remoto, le dices donde está ese repositorio y además le colocas un nombre corto llamado `origin` para no tener que estar escribiendo siempre la URL larga.
- Escribe el segundo comando que comienza con `git branch`. Este comando cambia el nombre de la rama actual. Posiblemente tu rama actual se llame master. Luego del comando se llamará main.
- Finalmente, escribe el comando que comienza con `git push`. Ahora le dirás a tu sistema de control de versión local que sincronice el repositorio local con el remoto u origin en este caso.

Advertencia: ESTE ÚLTIMO PASO PUEDE PEDIRTE QUE TE AUTENTIQUES

Sigue los pasos que te proponen para autenticarte. Ten presente que en este paso Git te pedirá que te autentiques con el servidor Github para certificar que eres el dueño del repositorio.

Advertencia: ANTES DE TERMINAR UNA SESIÓN DE TRABAJO

Esto es muy importante y SOLO AL TERMINAR una sesión de trabajo. Si el equipo de cómputo en el que estás trabajando no es de tu propiedad, es muy importante que antes de apagarlo, elimines tus credenciales. Para hacerlo, escribe en el buscador de windows, `credential managener`. Selecciona las credenciales de windows. Busca la credencial de git con tus datos y elimina dicha credencial.

3.2.19 Ejercicio 15: actualiza tu repositorio remoto

Ahora modifica de nuevo el archivo main.c así:

```
#include <stdio.h>
#include <stdlib.h>

int main(){
    printf("La vida es bella!!!\n");
    return(EXIT_SUCCESS);
}
```

Realiza un commit en el respositorio local:

```
git commit -am "add exclamation marks"
```

¿Notaste algo? En un solo paso pasaste main.c a la zona de fotos (STAGE) y realizaste el commit.

Verifica el estado del repositorio:

```
On branch master
Your branch is ahead of 'origin/master' by 1 commit.
(use "git push" to publish your local commits)

nothing to commit, working tree clean
```

Observa el mensaje `Your branch is ahead of 'origin/master' by 1 commit.` Git detecta que tu repositorio local está adelantado un commit con respecto al repositorio remoto. Observa que el propio Git te dice cómo actualizar el repositorio remoto:

```
git push
```

Vuelve el verificar el estado:

```
git status
```

Y el resultado será:

```
On branch master
Your branch is up to date with 'origin/master'.

nothing to commit, working tree clean
```

Y finalmente vuelve a mirar el historial del proyecto:

```
git log
```

El resultado será:

```
commit 56cef2b7d4a8f6fd03dcf302890d4e110cccb861 (HEAD -> master, origin/master)
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 16:02:12 2022 -0500

    add exclamation marks

commit 882d93e233a7634ae03566c267f5cb9e55a42f45
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 15:22:00 2022 -0500

    Revert "add a new print"

    This reverts commit 2a0afbb7efa9c58a364143edf6c5cf76dccfab0b.

commit 2a0afbb7efa9c58a364143edf6c5cf76dccfab0b
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 11:02:03 2022 -0500

```

(continué en la próxima página)

(proviene de la página anterior)

```
    add a new print

commit 1f2009fabfc4895ee6b063c23c6f5c7ea7175209
Author: yo <yo@yolandia.com>
Date:   Wed Jul 20 10:52:46 2022 -0500

    Initial version of the project main file
```

Mira el texto `(HEAD -> master, origin/master)`. Indica que tu repositorio local y remoto apuntan al mismo commit.

3.2.20 Ejercicio 16: repasa (evaluación formativa)

En este punto te pediré que descanses un momento. En este ejercicio vas a repasar el material que has trabajo. Te pediré que hagas lo siguiente:

1. Crea un directorio llamado project2. Ten presente cambiarte primero al directorio padre de project1. NO DEBES tener un repositorio en otro repositorio.
2. Inicia un repositorio allí.
3. Crea unos cuantos archivos de texto.
4. Dile a Git que haga tracking de esos archivos.
5. Realiza un primer commit.
6. Crea un repositorio remoto en GitHub que esté sincronizado con tu repositorio local. No olvides comprobar su creación.
7. Modifica los archivos creados.
8. Realiza un par de commits más.
9. Sincroniza los cambios con el repositorio remoto.

3.2.21 Ejercicio 17: clona un repositorio de GitHub

Ahora vas a descargar un repositorio de GitHub. Cámbiate al directorio padre de project2. Escribe el comando:

```
git clone https://github.com/juanferfrancoudea/demo4.git
```

Cámbiate al directorio demo4.

1. Verifica el estado del repositorio (status).
2. Verifica el historial (log).
3. Realiza un cambio a f1.txt.
4. Realiza un commit al repositorio local.

Ahora trata de actualizar el repositorio remoto con:

```
git push
```

Deberías obtener un mensaje similar a este:

```
remote: Permission to juanferfrancoudea/demo4.git denied to juanferfranco.
fatal: unable to access 'https://github.com/juanferfrancoudea/demo4.git/': The requested␣
↪URL returned error: 403
```

¿Qué está pasando? Lo que ocurre es que el repositorio que clonaste NO ES DE TU PROPIEDAD y por tanto NO TIENES permiso de actualizarlo. Para poderlo modificar, el dueño del repositorio te debe dar acceso.

Nota: Más de una persona puede trabajar en un repositorio siguiendo una serie de pasos y consideraciones. Para aprender más al respecto tendrías que leer sobre Git Workflows. De todas maneras no te preocupes, por ahora hay otras cosas que debes entender y practicar antes de abordar el TRABAJO EN EQUIPO usando Git. PERO OJO, TE RUEGO que más adelante lo aprendas porque será tu día a día cuando estés trabajando en la industria.

3.2.22 Ejercicio 18: documentación de las evaluaciones

Todas las entregas que realices deben estar acompañadas de una documentación que explique los aspectos técnicos (y otros que te pediré) de la solución que propongas a los problemas que te plantearé para las evaluaciones. Lo interesante de GitHub es que te permite almacenar repositorios no solo para el código, sino también para la documentación. Para documentar un repositorio lo único que debes tener es un archivo README.md en el repositorio.

Te voy a proponer que practique de nuevo lo que hemos trabajado juntos.

Crea un repositorio local y sincroniza ese repositorio con uno remoto. Añade la documentación adicionando el archivo README.

Para crear la documentación, debes escribir en un lenguaje de marcado conocido como markdown. Visita este sitio para que explores algunas etiquetas de marcado.

Escribe lo que quieras en el archivo README.md, experimenta. No olvides sincronizar el repositorio local con el remoto para que puedas ver los resultados.

CAPÍTULO 4

Unidad 1. Software para sistemas embebidos

4.1 Introducción

En esta unidad vas a familiarizarte con el funcionamiento del computador (microcontrolador) que permite interactuar directamente con los actuadores y sensores de una aplicaciones interactiva que utiliza dispositivos externos.

4.1.1 Propósito de aprendizaje

Aprenderás algunas técnicas de programación muy útiles para resolver problemas con sistemas de cómputo reactivos. Usarás estados, eventos y acciones.

4.2 Evaluación

Para realizar la evaluación vas a utilizar este repositorio.

Recuerda entregar la documentación solicitada en el archivo README.md

En un escape room se requiere construir una aplicación para controlar una bomba temporizada. La siguiente figura ilustra la interfaz de la bomba. El circuito de control de la bomba está compuesto por tres sensores digitales o en este caso botones que serán simulados usando el puerto serial. Los botones los llamaremos UP, DOWN y ARM. La bomba tiene un display que simularemos mediante el puerto serial.

Los dispositivos simulados funcionan así:

- Para presionar UP envías por el serial la letra u.
- Para presionar DOWN envías por el serial la letra d.
- Para presionar ARM envías por el serial la letra a.

El controlador funciona así:

- Inicia en modo de `configuración`, es decir, sin hacer cuenta regresiva aún, la bomba está `desarmada`. El valor inicial del conteo regresivo es de 20 segundos.
- El controlador indica SOLO UNA VEZ que está en modo configuración enviando el mensaje `CONFIG`.
- En el modo de configuración, los botones UP y DOWN permiten aumentar o disminuir el tiempo inicial de la bomba.
- El tiempo se puede programar entre 10 y 30 segundos con cambios de 1 segundo.
- Cada que modifiques el tiempo debes enviar por el puerto serial el valor del conteo programado. `SOLO` debes enviar el mensaje cuando el tiempo se ajuste.
- El pulsador ARM arma la bomba.
- Una vez armada la bomba, envía el mensaje `ARMED` y comienza la cuenta regresiva que será enviado por el puerto serial.
- La bomba explotará cuando el tiempo llegue a cero. En ese momento debes enviar el mensaje BOOM y 5 segundos después volver al modo de configuración.
- Una vez la bomba esté armada es posible desactivarla ingresando un código de seguridad. El código será la siguiente secuencia de pulsadores presionados uno después de otro: DOWN, DOWN, UP, UP, DOWN, UP, ARM. Ten presente que el controlador solo debe verificar si la secuencia es correcta una vez la reciba completa.
- Si la secuencia se ingresa correctamente la bomba pasará de nuevo al modo de configuración de lo contrario continuará la fatal cuenta regresiva.
- Debes almacenar la clave de desarmado de la bomba en un arreglo.
- Debes definir una función a la cual le pasarás la dirección en memoria de dos arreglos: uno con la clave recibida y otro con la clave correcta. La función deberá devolver un bool así: true si la clave recibida es igual a la clave almacenada o false si las claves no coinciden.

La aplicación debe ser construida usando estados, eventos y acciones.

Para la documentación:

- Define y explica los estados que usaste para resolver el problema.
- Define y explica los eventos que usaste.
- Define y explica las acciones.
- Explica cómo probaste el funcionamiento correcto de la aplicación.
- Explica cómo resolviste el problema de la clave.

4.3 Trayecto de actividades

4.3.1 Ejercicios

Ejercicio 1: introducción

Advertencia: RECUERDA LO QUE APRENDERÁS EN ESTE CURSO

En este curso aprenderás a construir aplicaciones interactivas que integren y envíen información desde y hacia el mundo exterior.

¿Recuerdas que te mostré al iniciar el curso un trabajo de grado realizado por estudiantes del programa? Te voy a pedir que veas algunos segundos del video del DEMO de este trabajo.

Déjame te hablo de nuevo de este sistema porque es un excelente resumen de lo que busco que aprendas con este curso.

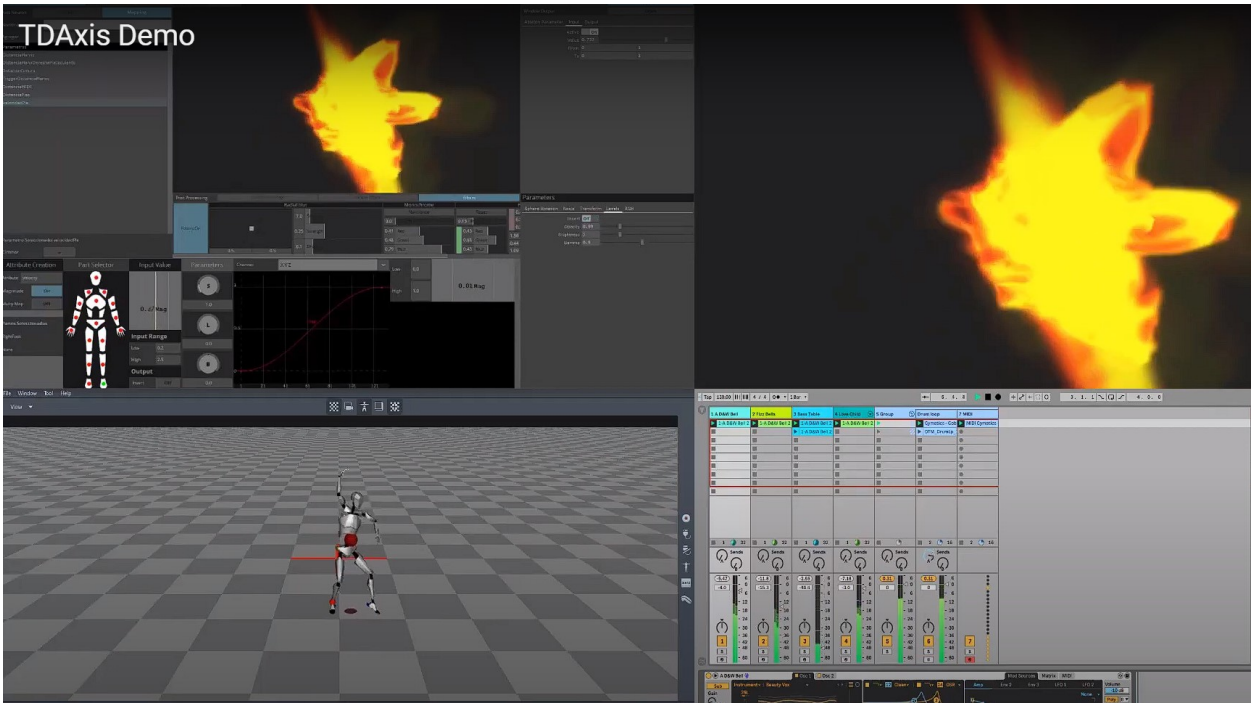


Figura 1: Demo del sistema TDAxis

La idea de la aplicación es VARIAR las visuales y el audio con la información del movimiento que se captura en tiempo real de una bailarina.

La imagen está dividida en 4 partes. En la esquina superior izquierda observarás `LA APLICACIÓN INTERACTIVA` que está corriendo en un computador. Esta aplicación se encargará de proyectar las visuales que están en la esquina superior derecha y controlador el software de audio que está en la esquina inferior derecha. Observa la esquina inferior izquierda. Allí verás una captura en tiempo real de los movimientos de una bailarina.

¿Cómo se captura este movimiento? Se hace por medio de unos dispositivos que te mostraré en estos videos:

- Perception Neuron Trailer.
- Bailarina controlando un metahumano.

Los dispositivos que llevan puestos las personas en los videos están computados por:

- Un sensor para medir el movimiento.
- Un computador embebido o microcontrolador que lee la información del sensor.
- Un radio de comunicación inalámbrica para transmitir la información leída.

La información se le entrega al computador que ejecuta la aplicación interactiva usando un `PROTOCOLO DE COMUNICACIÓN`. El protocolo es un acuerdo que se establece entre las partes involucradas en la comunicación de tal manera que ambas puedan entenderse.

¿Por qué te muestro todo esto?

Porque en este curso vamos a realizar un recorrido por los elementos que componente este tipo de aplicaciones.

En esta unidad vas a programar un microcontrolador similar al que tienen los dispositivos de captura de movimiento. En las unidades 2 y 3 vas experimentar con dos tipos de protocolos de comunicación. Finalmente, en la unidad 4 construirás una aplicación simple que integre todos los elementos y lo que aprendiste en las unidades previas.

Advertencia: ESTO ES MUY IMPORTANTE

Las aplicaciones que realizarás serán simples, PERO si lo analizas te darás cuenta que contienen todos los elementos necesarios para que entiendas cómo funcionan las aplicaciones que te mostré en los videos.

Nota: Hay otro curso en el programa para seguir profundizando

En el plan de estudios de la carrera encontrarás otro curso llamado sistemas físicos interactivos 2. Es una curso de la línea de experiencias interactivas que puedes tomar como optativa del ciclo profesional si no estás en la línea de experiencias. En este curso vas a construir una aplicación usando todo lo que aprenderás en sistemas físicos interactivos 1. Mira por ejemplo el proyecto que realizaron unos de tus compañeros.

Ejercicio 2: ¿Cómo funciona un microcontrolador?

Un microcontrolador es un computador dedicado a ejecutar una aplicación específica para resolver un problema muy concreto. Por ejemplo, leer la información de un sensor y transmitir esa información a un computador.

En este curso vas a utilizar un sistema de desarrollo llamado raspberry pi pico que cuanta con un microcontrolador que podrás programar.

Ejercicio 3: ¿Cómo puedes programar el microcontrolador?

Para programar el microcontrolador vas a necesitar:

- Un editor de código de C++.
- Varios programas que permitan transformar el código de C++ a instrucciones de máquina.
- Almacenar las instrucciones de máquina en la memoria flash del microcontrolador.

Sigue estos pasos:

- Descarga la versión .ZIP del IDE de arduino versión 1.8.19
- Descomprime el archivo .ZIP
- Busca la carpeta donde está el archivo arduino.exe y crea allí el carpeta portable.
- Abre el programa arduino.exe.
- Sigue las instrucciones de la sección Installing via Arduino Boards Manager que encontrarás en este repositorio.

Ahora vas a probar que puedes programar el raspberry pi pico:

- Conecta al computador el raspberry pi pico.
- En el menú Herramientas/Placa/Raspberry PI selecciona la tarjeta Raspberry Pi Pico.
- En el menú Herramientas/Puerto selecciona el puerto asignado por el sistema operativo al raspberry pi pico. Toma nota del nombre porque este mismo nombre lo usarás con otras aplicaciones (en mi caso es el COM5).

Ingresa el siguiente programa:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop() {
  static uint32_t previousTime = 0;
  static bool ledState = true;

  uint32_t currentTime = millis();

  if( (currentTime - previousTime) > 100){
    previousTime = currentTime;
    ledState = !ledState;
    digitalWrite(LED_BUILTIN, ledState);
  }
}
```

Por último presiona el ícono Subir, el segundo ubicado en la esquina superior izquierda. Al hacer esto ocurrirán varias cosas:

- Se transformará el programa de código C++ a lenguaje de máquina.
- Se enviará el código de máquina del computador a la memoria flash del raspberry pi a través del puerto serial que el sistema operativo le asignó a la tarjeta.

Deberás ver el LED ubicado al lado del conectar USB enciendo y apagando muy rápido.

Ejercicio 3a: retrieval practice (evaluación formativa)

En este punto te voy a pedir que coloques bajo control de versión un programa y practiques lo que estudiantes en la introducción de control de versión.

- Conforma tu equipo de trabajo (recuerda que el equipo es de dos personas, NO UNA).
- Acepta la evaluación que está aquí. El sistema primero te pedirá que crees el equipo de trabajo. Esto lo hace solo uno de los miembros del equipo, mientras que el otro solo tendrá que unirse al equipo. Luego aceptas la evaluación. Esto lo hace cada miembro.

Advertencia: MUY IMPORTANTE

DOS PERSONAS no pueden trabajar al mismo tiempo sobre el mismo archivo cuando el proyecto está bajo control de versión porque se crean conflictos, es decir, el sistema no sabe cuál de las dos versiones del archivo es la correcta.

- Ve a la terminal y clona el repositorio.
- En la carpeta donde está el repositorio vas a crear un proyecto para el raspberry pi pico usando el IDE de Arduino.
- Usa el mismo código del ejercicio, pero esta vez cambia el 100 por un 500. observa el resultado de esta cambio.
- Una vez termines el programa, lo veas funcionando y documentes el efecto del cambio, `DEBES cerrar el IDE de Arduino`.
- Regresa a la terminal. Realiza un commit con esta nueva versión del programa.
- Ahora añade el archivo README.md. Coloca el nombre del equipo, el nombre de los integrantes y el ID. Coloca en el archivo el resultado del cambio de 100 a 500. Describe lo que viste.
- Realiza un segundo commit para incluir el archivo README.md con la documentación.
- Envía los cambios del repositorio local al remoto.

Ejercicio 4: retrieval practice (evaluación formativa)

Acepta la evaluación aquí.

En el repositorio de la evaluación solo debes adicionar un archivo README.md con el nombre del equipo, integrantes y ID y un enlace a un repositorio `PÚBLICO` donde harás lo siguiente:

- Crea un proyecto para el raspberry pi.
- En el código modifica el 500 por 1000.
- Coloca el proyecto bajo control de versión.
- Sincroniza tu repositorio local con un repositorio público en Github. El enlace a este repositorio será el que coloques en el archivo README.md de la evaluación.

Ejercicio 5: documentación

Para programar el raspberry pi pico tienes mucha documentación con información. Algunos sitios que pueden serte de utilidad son:

- API de arduino.
- Port para raspberry pi pico del API de arduino.
- Sitio oficial del raspberry pi pico.

Ejercicio 6: caso de estudio

Programa la siguiente aplicación en el raspberry y analiza su funcionamiento. Para descubrir lo que hace debes dar click en el ícono que queda en la esquina superior derecha (Monitor Serie). Los números que vez allí son enviados desde el microcontrolador al computador por medio del puerto USB.

```
void task1()
{
    // Definición de estados y variable de estado
    enum class Task1States
    {
        INIT,
        WAIT_TIMEOUT
    };
    static Task1States task1State = Task1States::INIT;

    // Definición de variables static (conservan
    // su valor entre llamadas a task1)
    static uint32_t lastTime = 0;

    // Constantes
    constexpr uint32_t INTERVAL = 1000;

    // MÁQUINA de ESTADOS

    switch (task1State)
    {
    case Task1States::INIT:
    {
        // Acciones:

        Serial.begin(115200);

        // Garantiza los valores iniciales
        // para el siguiente estado.
        lastTime = millis();
        task1State = Task1States::WAIT_TIMEOUT;

        Serial.print("Task1States::WAIT_TIMEOUT\n");

        break;
    }
    case Task1States::WAIT_TIMEOUT:
```

(continué en la próxima página)

(proviene de la página anterior)

```
    {
        uint32_t currentTime = millis();

        // Evento
        if ((currentTime - lastTime) >= INTERVAL)
        {
            // Acciones:
            lastTime = currentTime;
            Serial.print(currentTime);
            Serial.print('\n');
        }
        break;
    }
    default:
    {
        Serial.println("Error");
    }
    }
}

void setup()
{
    task1();
}

void loop()
{
    task1();
}
```

- ¿Cómo se ejecuta este programa?
- Pudiste ver este mensaje: `Serial.print("Task1States::WAIT_TIMEOUT\n");`. ¿Por qué crees que ocurre esto?
- ¿Cuántas veces se ejecuta el código en el case Task1States::INIT?

Ejercicio 7: análisis del programa de prueba

Miremos algunos aspectos del programa:

- Los programas los dividiremos en tareas. En este caso solo tenemos una. Los programas más complejos tendrán muchas más.
- Este programa tiene un pseudo estado y un estado, pero desde ahora diremos que tiene 2 estados:

```
enum class Task1States
{
    INIT,
    WAIT_TIMEOUT
};
```

- ¿Qué son los estados? Son condiciones de espera. Son momentos en los cuales tu programa está esperando a que algo ocurra. En este caso en `Task1States::INIT` realmente no `ESPERAMOS` nada, por eso decimos que es un

pseudo estado. Este estado SIEMPRE lo usaremos para configurar las condiciones INICIALES de tu programa.

- Nota cómo se pasa de un estado a otro:

```
task1State = Task1States::WAIT_TIMEOUT;
```

- En el estado *Task1States::WAIT_TIMEOUT* estamos esperando a que ocurran un `EVENTO`. En este caso el evento lo identificamos mediante el `IF`. Por tanto, en un estado tu programa estará siempre preguntando por la ocurrencia de algunos eventos.
- Cuando la condición de un evento se produce entonces tu programa ejecuta ACCIONES. Por ejemplo aquí:

```
lastTime = currentTime;
Serial.print(currentTime);
Serial.print('\n');
```

  Si el evento `if ((currentTime - lastTime) >= INTERVAL)` ocurre, el programa ejecutará las acciones.
- La línea `Serial.print(currentTime);` te permite enviar mensaje por USB. Estos mensajes los puedes ver usando un programa como el Monitor Serie.
- Observa la función `millis();` ¿Para qué sirve? Recuerda que puedes buscar en Internet.

Ejercicio 8: retrieval practice (evaluación formativa)

Lo primero que debes hacer es aceptar esta evaluación e ingresar a tu equipo de trabajo.

- Entra al repositorio y copia la url para clonarlo en tu computador local.
- Realiza un programa que envíe un mensaje al pasar un segundo, dos segundos y tres segundos. Luego de esto debe volver a comenzar.

En el README.md del repositorio responde:

- ¿Cuáles son los estados del programa?
- ¿Cuáles son los eventos?
- ¿Cuáles son las acciones?

Ejercicio 9: tareas concurrentes (evaluación formativa)

Para sacar el máximo provecho a la CPU de tu microcontrolador lo ideal es dividir el problema en varias tareas que se puedan ejecutar de manera concurrente. La arquitectura de software que te voy a proponer es esta:

```
void task1(){

}

void task2(){

}

void task3(){

}
```

(continué en la próxima página)

(proviene de la página anterior)

```
void setup()
{
    task1();
    task2();
    task3();
}

void loop()
{
    task1();
    task2();
    task3();
}
```

Nota entonces que tu programa está dividido en tres tareas. La función setup se ejecuta una sola vez y ahí se llama por primera vez cada tarea. La función loop se ejecuta cada que las tareas terminan, es como un ciclo infinito.

Te voy a mostrar el código para la task1 y luego con tu equipo vas a construir las demás tareas. La frecuencia del mensaje será de 1 Hz

Acepta esta evaluación.

El objetivo es que hagas un programa donde tengas 3 tareas. La tarea 1 enviará un mensaje a 1 Hz. La tarea 2 a 0.5 Hz. La tarea 3 a 0.25 Hz

Te voy a dejar como ejemplo el programa de una de las tareas. Te queda entonces el reto de realizar las otras tareas. No olvides sincronizar tu repositorio local con el remoto donde está la evaluación.

```
void task1(){
    enum class Task1States{
        INIT,
        WAIT_FOR_TIMEOUT
    };

    static Task1States task1State = Task1States::INIT;
    static uint32_t lastTime;
    static constexpr uint32_t INTERVAL = 1000;

    switch(task1State){
        case Task1States::INIT:{
            Serial.begin(115200);
            lastTime = millis();
            task1State = Task1States::WAIT_FOR_TIMEOUT;
            break;
        }

        case Task1States::WAIT_FOR_TIMEOUT:{
            // evento 1:
            uint32_t currentTime = millis();
            if( (currentTime - lastTime) >= INTERVAL ){
                lastTime = currentTime;
                Serial.print("mensaje a 1Hz\n");
            }
            break;
```

(continué en la próxima página)

(proviene de la página anterior)

```
        }

        default:{
            break;
        }
    }

}

void setup()
{
    task1();
}

void loop()
{
    task1();
}
```

Ejercicio 10: monitor serial

Para profundizar un poco más en el funcionamiento de los programas vas a usar una herramienta muy interesante llamada monitor o terminal serial. En este curso vas a utilizar ScriptCommunicator. La aplicación la puedes descargar de este sitio. Al instalarla en los computadores de la Universidad usa un directorio del usuario y deshabilita la creación de accesos directos en el escritorio y no asocies los archivos .js con ScriptCommunicator.

Para lanzar la aplicación abre el directorio donde la instalaste y lanza el programa ScriptCommunicator.exe

Ingresa al menu Settings, selecciona la pestaña serial port y elige el puerto (el puerto asignado por el sistema operativo a tu sistema de desarrollo) y el BaudRate a 115200. Los demás parámetros los puedes dejar igual.

Selecciona la pestaña console options y allí marca ÚNICAMENTE las opciones: utf8, receive, hex, mixed. En new line at byte coloca None y en Send on enter key coloca None.

En la pestaña serial port ve a la sección general, selecciona como current interface `serial port`. Cierra la ventana de configuración.

> **Advertencia:** IMPORTANTE
>
> No olvides que para DEBES TENER conectado el sistema de desarrollo al computador para poder seleccionar el Port correcto.

Para conectar ScriptCommunicator al microcontrolador, solo tienes que dar click en Connect y para desconectar Disconnect.

> **Advertencia:** ESTO ES CRÍTICO
>
> SOLO UNA APLICACIÓN puede comunicarse a la vez con el microcontrolador. Por tanto, SOLO una aplicación puede abrir o conectarse al puerto serial que el sistema operativo le asigna al sistema de desarrollo.
>
> Esto quiere decir que no puedes programar el raspberry mientras tienes abierto ScriptCommunicator conectado al puerto serial.

Ejercicio 11: realiza algunas pruebas

Ahora vas a probar ScriptCommunicator con el sistema de desarrollo.

Utiliza el siguiente programa:

```
void task1()
{
    enum class Task1States
    {
        INIT,
        WAIT_DATA
    };
    static Task1States task1State = Task1States::INIT;

    switch (task1State)
    {
    case Task1States::INIT:
    {
        Serial.begin(115200);
        task1State = Task1States::WAIT_DATA;
        break;
    }

    case Task1States::WAIT_DATA:
    {
        // evento 1:
        // Ha llegado al menos un dato por el puerto serial?
        if (Serial.available() > 0)
        {
            Serial.read();
            Serial.print("Hola computador\n");
        }
        break;
    }

    default:
    {
        break;
    }
    }
}

void setup()
{
    task1();
}

void loop()
{
    task1();
}
```

Ahora abre ScriptCommunicator:

- Presiona el botón Connect.
- Selecciona la pestaña Mixed.
- Luego escribe una letra en la caja de texto que está debajo del botón `send`. Si quieres coloca la letra *s*.
- Al lado del botón send selecciona la opción utf8.
- Dale click a send.
- Deberías recibir el mensaje `Hola computador`.
- Nota que al final del mensaje hay un 0a ¿A qué letra corresponde?

Ahora PIENSA:

1. Analiza el programa. ¿Por qué enviaste la letra con el botón send? ¿Qué evento verifica si ha llegado algo por el puerto serial?
2. Abre esta tabla.
3. Analiza los números que se ven debajo de las letras. Nota que luego de la r, abajo, hay un número. ¿Qué es ese número?
4. ¿Qué relación encuentras entre las letras y los números?
5. ¿Qué es el 0a al final del mensaje y para qué crees que sirva?
6. Nota que luego de verificar si hay datos en el puerto serial se DEBE HACER UNA LECTURA del puerto. Esto se hace para retirar del puerto el dato que llegó. Si esto no se hace entonces parecerá que siempre tiene un datos disponible en el serial para leer. ¿Tiene sentido esto? Si no es así habla con el profe.

Ejercicio 12: punteros

Vas a explorar un concepto fundamental de los lenguajes de programación C y C++. Se trata de los punteros. Para ello, te voy a proponer que escribas el siguiente programa. Para probarlo usa ScriptCommunicator.

```
void task1()
{
    enum class Task1States
    {
        INIT,
        WAIT_DATA
    };
    static Task1States task1State = Task1States::INIT;

    switch (task1State)
    {
    case Task1States::INIT:
    {
        Serial.begin(115200);
        task1State = Task1States::WAIT_DATA;
        break;
    }

    case Task1States::WAIT_DATA:
    {
        // evento 1:
        // Ha llegado al menos un dato por el puerto serial?
```

(continué en la próxima página)

(proviene de la página anterior)

```
        if (Serial.available() > 0)
        {
            // DEBES leer ese dato, sino se acumula y el buffer de recepción
            // del serial se llenará.
            Serial.read();
            uint32_t var = 0;
            // Almacena en pvar la dirección de var.
            uint32_t *pvar = &var;
            // Envía por el serial el contenido de var usando
            // el apuntador pvar.
            Serial.print("var content: ");
            Serial.print(*pvar);
            Serial.print('\n');
            // ESCRIBE el valor de var usando pvar
            *pvar = 10;
            Serial.print("var content: ");
            Serial.print(*pvar);
            Serial.print('\n');
        }
        break;
    }

    default:
    {
        break;
    }
    }
}

void setup()
{
    task1();
}

void loop()
{
    task1();
}
```

¿No te está funcionando? No olvides que en un ESTADO siempre esperas eventos. Si estás enviando el evento?

La variable `pvar` se conoce como puntero. Simplemente es una variable en la cual se almacenan direcciones de otras variables. En este caso, en pvar se almacena la dirección de `var`. Nota que debes decirle al compilador el tipo de la variable (uint32_t en este caso) cuya dirección será almacenada en pvar.

Ejecuta el programa. Observa lo que hace. Ahora responde las siguientes preguntas mediante un ejercicio de ingeniería inversa:

- ¿Cómo se declara un puntero?
- ¿Cómo se define un puntero? (cómo se inicializa)
- ¿Cómo se obtiene la dirección de una variable?
- ¿Cómo se puede leer el contenido de una variable por medio de un puntero?

- ¿Cómo se puede escribir el contenido de una variable por medio de un puntero?

Advertencia: IMPORTANTE

No avances hasta que este ejercicio no lo tengas claro.

Ejercicio 13: punteros y funciones

Vas a escribir el siguiente programa, pero antes de ejecutarlo vas a tratar de lanzar una HIPÓTESIS de qué hace. Luego lo vas a ejecutar y compararás el resultado con lo que creías. Si el resultado no es el esperado, no deberías seguir al siguiente ejercicio hasta que no experimentes y salgas de la duda.

```
static void changeVar(uint32_t *pdata)
{
    *pdata = 10;
}

static void printVar(uint32_t value)
{
    Serial.print("var content: ");
    Serial.print(value);
    Serial.print('\n');
}

void task1()
{
    enum class Task1States
    {
        INIT,
        WAIT_DATA
    };
    static Task1States task1State = Task1States::INIT;

    switch (task1State)
    {
    case Task1States::INIT:
    {
        Serial.begin(115200);
        task1State = Task1States::WAIT_DATA;
        break;
    }

    case Task1States::WAIT_DATA:
    {
        // evento 1:
        // Ha llegado al menos un dato por el puerto serial?
        if (Serial.available() > 0)
        {
            Serial.read();
            uint32_t var = 0;
            uint32_t *pvar = &var;
            printVar(*pvar);
```

(continué en la próxima página)

(proviene de la página anterior)

```
            changeVar(pvar);
            printVar(var);
        }
        break;
    }

    default:
    {
        break;
    }
    }
}

void setup()
{
    task1();
}

void loop()
{
    task1();
}
```

Ejercicio 14: retrieval practice (evaluación formativa)

Realiza un programa que intercambie mediante una función el valor de dos variables.

Aquí está el enlace de la evaluación.

Ejercicio 15: punteros y arreglos

Escribe el siguiente programa. `ANALIZA` qué hace, cómo funciona y qué necesitas para probarlo. No olvides revisar de nuevo una tabla ASCII. Para hacer las pruebas usa ScriptCommunicator y abre la pestaña Utf8.

```
static void processData(uint8_t *pData, uint8_t size, uint8_t *res)
{

    uint8_t sum = 0;
    for (int i = 0; i < size; i++)
    {
        sum = sum + (pData[i] - 0x30);
    }
    *res = sum;
}

void task1()
{
    enum class Task1States
    {
        INIT,
        WAIT_DATA
```

(continué en la próxima página)

(proviene de la página anterior)

```
    };
    static Task1States task1State = Task1States::INIT;
    static uint8_t rxData[5];
    static uint8_t dataCounter = 0;

    switch (task1State)
    {
    case Task1States::INIT:
    {
        Serial.begin(115200);
        task1State = Task1States::WAIT_DATA;
        break;
    }

    case Task1States::WAIT_DATA:
    {
        // evento 1:

        if (Serial.available() > 0)
        {
            rxData[dataCounter] = Serial.read();
            dataCounter++;
            if (dataCounter == 5)
            {
                uint8_t result = 0;
                processData(rxData, dataCounter, &result);
                dataCounter = 0;
                Serial.print("result: ");
                Serial.print(result);
                Serial.print('\n');
            }
        }
        break;
    }

    default:
    {
        break;
    }
    }
}


void setup()
{
    task1();
}

void loop()
{
    task1();
}
```

Piensa en las siguientes cuestiones:

- ¿Por qué es necesario declarar `rxData` static? y si no es static ¿Qué pasa? ESTO ES IMPORTANTE, MUCHO.
- dataCounter se define static y se inicializa en 0. Cada vez que se ingrese a la función loop dataCounter se inicializa a 0? ¿Por qué es necesario declararlo static?
- Observa que el nombre del arreglo corresponde a la dirección del primer elemento del arreglo. Por tanto, usar en una expresión el nombre rxData (sin el operador []) equivale a &rxData[0].
- En la expresión `sum = sum + (pData[i] - 0x30);` observa que puedes usar el puntero pData para indexar cada elemento del arreglo mediante el operador [].
- Finalmente, la constante `0x30` en `(pData[i] - 0x30)` ¿Por qué es necesaria?

Truco: ALERTA DE SPOILER

Con respecto a la pregunta anterior. Al enviar un carácter numérico desde ScriptCommunicator este se envía codificado, es decir, se envía un byte codificado en ASCII que representa al número. Por tanto, es necesario decodificar dicho valor. El código ASCII que representa los valores del 0 al 9 es respectivamente: 0x30, 0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39. De esta manera, si envías el 1 recibirás el valor 0x31. Si restas de 0x31 el 0x30 obtendrás el número 1.

Repite el ejercicio anterior pero esta vez usa la pestaña Mixed.

Ejercicio 16: análisis del api serial (investigación: hipótesis-pruebas)

Qué crees que ocurre cuando:

- ¿Qué pasa cuando hago un Serial.available()?
- ¿Qué pasa cuando hago un Serial.read()?
- ¿Qué pasa cuando hago un Serial.read() y no hay nada en el buffer de recepción?
- Un patrón común al trabajar con el puerto serial es este:

```
if(Serial.available() > 0){
    int dataRx = Serial.read()
}
```

- ¿Cuántos datos lee Serial.read()?
- ¿Y si quiero leer más de un dato? No olvides que no se pueden leer más datos de los disponibles en el buffer de recepción porque no hay más datos que los que tenga allí.
- ¿Qué pasa si te envían datos por serial y se te olvida llamar Serial.read()?

Advertencia: NO AVANCES SIN ACLARAR LAS PREGUNTAS ANTERIORES

Te pido que resuelvas las preguntas anteriores antes de avanzar. ES MUY IMPORTANTE.

Ejercicio 17: buffer de recepción

Así se pueden leer 3 datos que han llegado al puerto serial:

```
if(Serial.available() >= 3){
    int dataRx1 = Serial.read()
    int dataRx2 = Serial.read()
    int dataRx3 = Serial.read()
}
```

¿Qué escenarios podría tener en este caso?

```
if(Serial.available() >= 2){
    int dataRx1 = Serial.read()
    int dataRx2 = Serial.read()
    int dataRx3 = Serial.read()
}
```

Para responder, es necesario que experimentes. ESTOS son los ejercicios que realmente te ayudarán a aprender.

Ejercicio 18: retrieval practice (evaluación formativa)

Acepta la evaluación que está aquí.

Piense cómo podrías hacer lo siguiente:

- Crea una aplicación con una tarea.
- La tarea debe tener su propio buffer de recepción y una capacidad para 32 bytes.
- La tarea almacena los datos del serial en su propio buffer de recepción (el buffer será un arreglo).
- El buffer debe estar encapsulado en la tarea.
- Los datos almacenados en el buffer no se pueden perder entre llamados a la tarea.
- La tarea debe tener algún mecanismo para ir contando la cantidad de datos que han llegado. ¿Cómo lo harías?

Inventa un programa que ilustre todo lo anterior y en el archivo README.md escribe cómo solucionaste el problema.

CAPÍTULO 5

Unidad 2. Protocolos ASCII

5.1 Introducción

En la unidad anterior te has concentrado en la construcción de software para sistemas embebidos. En esta unidad aprenderás como integrar a una plataforma de cómputo interactiva dichos sistemas embebidos mediante el uso de protocolos de comunicación ASCII.

5.1.1 Propósitos de aprendizaje

Construir aplicaciones interactivas que integren información del mundo exterior mediante el intercambio de información codificada en ASCII.

5.2 Evaluación

Advertencia: SUSTENTACIÓN

No olvides que la evaluación solo se considera entregada cuando la sustentes. Antes de sustentarla DEBES entregarla, es decir, toda la información solicitada debe estar en el repositorio. Verifica con tus compañeros de equipo que todos los requisitos se cumplen antes de sustentar.

5.2.1 Enunciado

Vas a realizar un sistema de aplicaciones interactivas que se comunicarán entre ellas. Una aplicación correrá en el PC y la otra en un controlador.

Los requisitos son los siguientes:

- La aplicación en el PC debe interactuar con el usuario por medio de elementos de interfaz de usuario tales como botones, cajas de texto, textos en pantalla, etc. Solo debes usar la consola para depuración, no para interacción.

- La aplicación en el PC debe leer el estado de tres variables de la aplicación en el controlador. La aplicación en el PC debe solicitar las variables y el controlador las debe reportar TODAS en el mismo mensaje. `OJO, no por separado, en el mismo mensaje`.
- Desde el PC se debe configurar la velocidad a la cual cambiará la variable y si debe cambiar o no. También se debe poder definir el valor inicial de la variable.
- El controlador verificará si debe cambiar la variable y la modificará en tiempo real siempre y cuando esté habilitada para cambiar. La función de cambio será simplemente aumentar en uno el valor previo a la velocidad expecificada. Por ejemplo, si la variable 1 se configura con una velocidad de cambio de 10 conteos por segundo, el controlador deberá incrementar la variable cada 100 ms. Ten presente que el PC puede detener (deshabilitar) el conteo en cualquier momento. Así mismo, podrá cambiar el valor de la variable.
- El controlador deberá mantener un LED funcionando a una frecuencia de 1 Hz. El objetivo de este LED es que verifiques de manera visual que la aplicación en el controlador NUNCA se bloquea.
- Define de MANERA CREATIVA cómo visualizarás el estado de las tres variables en la aplicación del PC. Dado que el requisito aquí es ser creativo, se espera que la representación de las variables que haga cada equipo sea ÚNICA. `NO USES TEXTO en la interfaz de usuario para visualizar los valores`.

5.2.2 Entrega

> **Advertencia:** LEE PRIMERO
>
> Lee primero todos los pasos antes de comenzar el proceso.

Vas a realizar tu entrega en este repositorio.

Sigue los siguientes pasos:

1. Verifica en el administrador de credenciales de Windows que el computador de la U en el que trabajas no tenga credenciales de Github de otra persona guardadas. Si es el caso elimina las credenciales.
2. Ingresa en el navegador web a tu cuenta en Github.
3. Abre git bash y clona el repositorio en tu computador local. Te pedirá las credenciales y las guardará en el administrador de credenciales. No olvides borrarlas antes de irte y también cerrar la sesión en el navegador.
4. Descarga el archivo .gitignore que está aquí.
5. Copia la carpeta de tu proyecto terminado en Unity en la carpeta con el repositorio.
6. En el repositorio, abre la carpeta con el proyecto de Unity y copia el archivo .gitignore allí. Asegúrate que el archivo esté en la misma carpeta donde puedes ver los directorios Library, Temp, Obj, Logs, UserSettings.
7. Copia la carpeta con el proyecto de Arduino en el repositorio, pero verifica que lo hagas fuera de la carpeta con el proyecto de Unity.
8. Adiciona los archivos al repositorio.
9. Realiza el commit.
10. Realiza el push con la entrega.

Ahora crea el archivo README.md en el repositorio y escribe allí para cada aplicación (Arduino y Unity):

1. ¿Cuáles son los estados y por qué definiste esos estados?
2. ¿Cuáles son los eventos y por qué definiste esos eventos?

3. Incluye un enlace a un video de Youtube donde muestres la aplicación funcionando. RECUERDA que es un enlace, NO el video.

5.3 Trayecto de actividades

5.3.1 Ejercicios

Ejercicio 1: comunicación computador-controlador

La idea de este ejercicio es comunicar a través del puerto serial un computador con un controlador. La aplicación del computador la construirás usando una plataforma de creación de contenido digital interactivo llamada Unity 2021 LTS.

Estudia con detenimiento el código para el controlador y para el computador. Busca la definición de todas las funciones usadas en la documentación de Arduino y de Microsoft.

- ¿Quién debe comenzar primero, el computador o el controlador? ¿Por qué?

Programa el controlador con este código:

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  if(Serial.available()){
    if(Serial.read() == '1'){
      Serial.print("Hello from ESP32");
    }
  }
}
```

Prueba la aplicación con ScriptCommunicator. ¿Cómo funciona?

Ahora crea un proyecto en Unity 2021 LTS. Antes de continuar con la escritura del código configura:

- La herramienta que usarás para editar tus programas. En este caso usarás Visual Studio. Recuerda que este paso lo puedes hacer en el menú Edit, Preferences, External Tools y seleccionar Visual Studio en la opción External Script Editor.
- Configura un scripting backend que permita soportar las comunicaciones seriales con el controlador. Ve al menú Edit, Project Settings, Player, Other Settings, busca la opción Scripting backend y selecciona Mono, luego busca API Compatibility Level y seleccionar .NET Framework.

> **Advertencia:** MUY IMPORTANTE
>
> Siempre que trabajes con comunicaciones seriales y Unity es necesario seleccionar .NET Framework como el Scripting backend de lo contrario tendrás un error de compilación relacionado con el puerto Serial.

Crea un nuevo C# Script y un Game Object. Añade el Script al GameObject. Ve al menu Assets y luego selecciona Open C# Project.

```
using UnityEngine;
using System.IO.Ports;
public class Serial : MonoBehaviour
```

(continué en la próxima página)

(proviene de la página anterior)

```
{
    private SerialPort _serialPort = new SerialPort();
    private byte[] buffer = new byte[32];

    void Start()
    {
        _serialPort.PortName = "/dev/ttyUSB0";
        _serialPort.BaudRate = 115200;
        _serialPort.DtrEnable = true;
        _serialPort.Open();
        Debug.Log("Open Serial Port");
    }

    void Update()
    {

        if (Input.GetKeyDown(KeyCode.A))
        {
            byte[] data = {0x31}; // or byte[] data = {'1'};
            _serialPort.Write(data,0,1);
            Debug.Log("Send Data");
        }

        if (Input.GetKeyDown(KeyCode.B))
        {
            if (_serialPort.BytesToRead >= 16)
            {
                _serialPort.Read(buffer, 0, 20);
                Debug.Log("Receive Data");
                Debug.Log(System.Text.Encoding.ASCII.GetString(buffer));
            }
        }

    }
}
```

Analiza:

- ¿Por qué es importante considerar las propiedades PortName y BaudRate?
- ¿Qué relación tienen las propiedades anteriores con el controlador?

Ejercicio 2: experimento

Advertencia: ESTE SEMESTRE CAMBIAMOS DE controlador

Los videos que te mostraré utilizan un controlador y editor diferente para escribir los programas de este. NO HAY PROBLEMA. Puedes usar el mismo código para experimentar con el controlador que tienes ahora y con el IDE de Arduino.

(Si quires ver antes unos videos cortos donde te explico un poco más el ejercicio te dejo este link).

Ahora realiza este experimento. Modifica la aplicación del PC así:

```
using UnityEngine;
using System.IO.Ports;
using TMPro;

public class Serial : MonoBehaviour
{
    private SerialPort _serialPort = new SerialPort();
    private byte[] buffer = new byte[32];

    public TextMeshProUGUI myText;

    private static int counter = 0;

    void Start()
    {
        _serialPort.PortName = "/dev/ttyUSB0";
        _serialPort.BaudRate = 115200;
        _serialPort.DtrEnable = true;
        _serialPort.Open();
        Debug.Log("Open Serial Port");
    }

    void Update()
    {
        myText.text = counter.ToString();
        counter++;

        if (Input.GetKeyDown(KeyCode.A))
        {
            byte[] data = {0x31}; // or byte[] data = {'1'};
            _serialPort.Write(data,0,1);
            int numData = _serialPort.Read(buffer, 0, 20);
            Debug.Log(System.Text.Encoding.ASCII.GetString(buffer));
            Debug.Log("Bytes received: " + numData.ToString());
        }
    }
}
```

Advertencia: Ojo con el puerto serial

Ten cuidado con el programa anterior. Nota esta línea:

_serialPort.PortName = «/dev/ttyUSB0»;

En tu sistema operativo debes averiguar en qué puerto está el controlador y cómo se llama. En Windows se usa COMx donde x es el número del puerto serial asignado por el sistema operartivo a tu controlador.

Debe adicionar a la aplicación un elemento de GUI tipo Text - TextMeshPro y y luego arrastrar una referencia a este elemento a myText (si no sabes cómo hacerlo llama al profe).

Y la aplicación del controlador:

```
void setup() {
  Serial.begin(115200);
}

void loop() {
  if(Serial.available()){
    if(Serial.read() == '1'){
      delay(3000);
      Serial.print("Hello from Raspi");
    }
  }
}
```

Ejecuta la aplicación en Unity. Verás un número cambiar rápidamente en pantalla. Ahora presiona la tecla A (no olvides dar click en la pantalla Game). ¿Qué pasa? ¿Por qué crees que ocurra esto?

Truco: MUY IMPORTANTE

¿Viste entonces que la aplicación se bloquea? Este comportamiento es inaceptable para una aplicación interactiva de tiempo real.

¿Cómo podemos corregir el comportamiento anterior?

Prueba con el siguiente código, luego ANALIZA CON DETENIMIENTO (no olvides) cambiar el puerto serial.

```
using UnityEngine;
using System.IO.Ports;
using TMPro;

public class Serial : MonoBehaviour
{
    private SerialPort _serialPort = new SerialPort();
    private byte[] buffer = new byte[32];

    public TextMeshProUGUI myText;

    private static int counter = 0;

    void Start()
    {
        _serialPort.PortName = "/dev/ttyUSB0";
        _serialPort.BaudRate = 115200;
        _serialPort.DtrEnable = true;
        _serialPort.Open();
        Debug.Log("Open Serial Port");
    }

    void Update()
    {
        myText.text = counter.ToString();
        counter++;

        if (Input.GetKeyDown(KeyCode.A))
```

(continué en la próxima página)

(proviene de la página anterior)

```
        {
            byte[] data = {0x31}; // or byte[] data = {'1'};
            _serialPort.Write(data,0,1);
        }

        if (_serialPort.BytesToRead > 0)
        {
            int numData = _serialPort.Read(buffer, 0, 20);
            Debug.Log(System.Text.Encoding.ASCII.GetString(buffer));
            Debug.Log("Bytes received: " + numData.ToString());
        }
    }
}
```

¿Funciona? ¿Qué pasaría si al momento de ejecutar la instrucción `int numData = _serialPort.Read(buffer, 0, 20);` solo han llegado 10 de los 16 bytes del mensaje? ¿Cómo puede hacer tu programa para saber que ya tiene el mensaje completo?

¿Cómo podrías garantizar que antes de hacer la operación Read tengas los 16 bytes listos para ser leídos?

Y si los mensajes que envía el controlador tienen tamaños diferentes ¿Cómo haces para saber que el mensaje enviado está completo o faltan bytes por recibir?

Truco: Piensa antes de continuar

Por favor piensa antes de continuar; sin embargo, no te preocupes porque te voy a contar en un momento qué puedes hacer para responder las preguntas anteriores.

Ejercicio 3: eventos externos

Nota que en los experimentos anteriores el PC primero le pregunta al controlador (le manda un 1) por datos. ¿Y si el PC no pregunta? Realiza el siguiente experimento. Programa ambos códigos y analiza su funcionamiento.

```
void task()
{
  enum class TaskStates
  {
    INIT,
    WAIT_INIT,
    SEND_EVENT
  };
  static TaskStates taskState = TaskStates::INIT;
  static uint32_t previous = 0;
  static u_int32_t counter = 0;

  switch (taskState)
  {
  case TaskStates::INIT:
  {
    Serial.begin(115200);
    taskState = TaskStates::WAIT_INIT;
```

(continué en la próxima página)

(proviene de la página anterior)

```
    break;
  }
  case TaskStates::WAIT_INIT:
  {
    if (Serial.available() > 0)
    {
      if (Serial.read() == '1')
      {
        previous = 0; // Force to send the first value immediately
        taskState = TaskStates::SEND_EVENT;
      }
    }
    break;
  }
  case TaskStates::SEND_EVENT:
  {
    uint32_t current = millis();
    if ((current - previous) > 2000)
    {
      previous = current;
      Serial.print(counter);
      counter++;
    }

    if (Serial.available() > 0)
    {
      if (Serial.read() == '2')
      {
        taskState = TaskStates::WAIT_INIT;
      }
    }

    break;
  }
  default:
  {
    break;
  }
  }
}

void setup()
{
  task();
}

void loop()
{
  task();
}
```

```
using UnityEngine;
using System.IO.Ports;
using TMPro;

enum TaskState
{
    INIT,
    WAIT_START,
    WAIT_EVENTS
}

public class Serial : MonoBehaviour
{
    private static TaskState taskState = TaskState.INIT;
    private SerialPort _serialPort;
    private byte[] buffer;
    public TextMeshProUGUI myText;
    private int counter = 0;

    void Start()
    {
        _serialPort = new SerialPort();
        _serialPort.PortName = "/dev/ttyUSB0";
        _serialPort.BaudRate = 115200;
        _serialPort.DtrEnable = true;
        _serialPort.Open();
        Debug.Log("Open Serial Port");
        buffer = new byte[128];
    }

    void Update()
    {
        myText.text = counter.ToString();
        counter++;

        switch (taskState)
        {
            case TaskState.INIT:
                taskState = TaskState.WAIT_START;
                Debug.Log("WAIT START");
                break;
            case TaskState.WAIT_START:
                if (Input.GetKeyDown(KeyCode.A))
                {
                    byte[] data = {0x31}; // start
                    _serialPort.Write(data,0,1);
                    Debug.Log("WAIT EVENTS");
                    taskState = TaskState.WAIT_EVENTS;
                }

                break;
            case TaskState.WAIT_EVENTS:
                if (Input.GetKeyDown(KeyCode.B))
```

(continué en la próxima página)

(proviene de la página anterior)

```
                {
                    byte[] data = {0x32}; // stop
                    _serialPort.Write(data,0,1);
                    Debug.Log("WAIT START");
                    taskState = TaskState.WAIT_START;
                }

                if (_serialPort.BytesToRead > 0)
                {
                    int numData = _serialPort.Read(buffer, 0, 128);
                    Debug.Log(System.Text.Encoding.ASCII.GetString(buffer));
                }
                break;
            default:
                Debug.Log("State Error");
                break;
        }
    }
}
```

¿Recuerdas las preguntas del otro experimento? Aquí nos pasa lo mismo. Analicemos el asunto. Cuando preguntas `_serialPort.BytesToRead > 0` lo que puedes asegurar es que al MENOS tienes un byte del mensaje, pero no puedes saber si tienes todos los bytes que lo componen. Una idea para resolver esto sería hacer que todos los mensajes tengan el mismo tamaño. De esta manera solo tendrías que preguntar `_serialPort.BytesToRead > SIZE`, donde SIZE sería el tamaño fijo; sin embargo, esto le resta flexibilidad al protocolo de comunicación. Nota que esto mismo ocurre en el caso del programa del controlador con `Serial.available() > 0`.

¿Cómo podrías solucionar este problema?

Truco: PIENSA primero

El siguiente ejercicio te servirá para responder esta pregunta.

Ejercicio 4: carácter de fin de mensaje

Ahora vas a analizar cómo puedes resolver el problema anterior.

Analiza el siguiente programa del controlador:

```
String btnState(uint8_t btnState){
  if(btnState == HIGH){
    return "OFF";
  }
  else return "ON";
}

void task()
{
  enum class TaskStates
  {
    INIT,
```

(continué en la próxima página)

(proviene de la página anterior)

```
  WAIT_COMMANDS
};
static TaskStates taskState = TaskStates::INIT;
constexpr uint8_t led = 25;
constexpr uint8_t button1Pin = 12;
constexpr uint8_t button2Pin = 13;
constexpr uint8_t button3Pin = 32;
constexpr uint8_t button4Pin = 33;

switch (taskState)
{
case TaskStates::INIT:
{
  Serial.begin(115200);
  pinMode(led, OUTPUT);
  digitalWrite(led, LOW);
  pinMode(button1Pin, INPUT_PULLUP);
  pinMode(button2Pin, INPUT_PULLUP);
  pinMode(button3Pin, INPUT_PULLUP);
  pinMode(button4Pin, INPUT_PULLUP);
  taskState = TaskStates::WAIT_COMMANDS;
  break;
}
case TaskStates::WAIT_COMMANDS:
{
  if (Serial.available() > 0)
  {
    String command = Serial.readStringUntil('\n');
    if (command == "ledON")
    {
      digitalWrite(led, HIGH);
    }
    else if (command == "ledOFF")
    {
      digitalWrite(led, LOW);
    }
    else if (command == "readBUTTONS")
    {

      Serial.print("btn1: ");
      Serial.print(btnState(digitalRead(button1Pin)).c_str());
      Serial.print(" btn2: ");
      Serial.print(btnState(digitalRead(button2Pin)).c_str());
      Serial.print(" btn3: ");
      Serial.print(btnState(digitalRead(button3Pin)).c_str());
      Serial.print(" btn4: ");
      Serial.print(btnState(digitalRead(button4Pin)).c_str());
      Serial.print('\n');
    }
  }
  break;
}
```

(continué en la próxima página)

(proviene de la página anterior)

```
  default:
  {
    break;
  }
  }
}

void setup()
{
  task();
}

void loop()
{
  task();
}
```

Analiza el siguiente programa del PC:

```
using UnityEngine;
using System.IO.Ports;
using TMPro;

enum TaskState
{
    INIT,
    WAIT_COMMANDS
}

public class Serial : MonoBehaviour
{
    private static TaskState taskState = TaskState.INIT;
    private SerialPort _serialPort;
    private byte[] buffer;
    public TextMeshProUGUI myText;
    private int counter = 0;

    void Start()
    {
        _serialPort = new SerialPort();
        _serialPort.PortName = "/dev/ttyUSB0";
        _serialPort.BaudRate = 115200;
        _serialPort.DtrEnable = true;
        _serialPort.NewLine = "\n";
        _serialPort.Open();
        Debug.Log("Open Serial Port");
        buffer = new byte[128];
    }

    void Update()
    {
        myText.text = counter.ToString();
```

(continué en la próxima página)

(proviene de la página anterior)

```
        counter++;

        switch (taskState)
        {
            case TaskState.INIT:
                taskState = TaskState.WAIT_COMMANDS;
                Debug.Log("WAIT COMMANDS");
                break;
            case TaskState.WAIT_COMMANDS:
                if (Input.GetKeyDown(KeyCode.A))
                {
                    _serialPort.Write("ledON\n");
                    Debug.Log("Send ledON");
                }
                if (Input.GetKeyDown(KeyCode.S))
                {
                    _serialPort.Write("ledOFF\n");
                    Debug.Log("Send ledOFF");
                }

                if (Input.GetKeyDown(KeyCode.R))
                {
                    _serialPort.Write("readBUTTONS\n");
                    Debug.Log("Send readBUTTONS");

                }
                if (_serialPort.BytesToRead > 0)
                {
                    string response = _serialPort.ReadLine();
                    Debug.Log(response);
                }

                break;
            default:
                Debug.Log("State Error");
                break;
        }
    }
}
```

Ejercicio 5: retrieval practice

Con todo lo que has aprendido hasta ahora vas a volver a darle una mirada al material desde el ejercicio 1. Una iteración más. Pero la idea de este ejercicio es que le expliques a un compañero cada ejercicio. Y la misión de tu compañero será hacerte preguntas.

RETO: protocolo ASCII

El reto consiste en implementar un sistema que permita, mediante una interfaz gráfica en Unity interactuar con el controlador. La idea será que puedas leer el estado de una variable que estará cambiando en el controlador y cambiar el estado del LED verde del controlador. Ten presente que aunque este ejercicio usa un controlador simple, los conceptos asociados a su manejo pueden fácilmente extrapolarse a dispositivos y sistemas más complejos.

Este reto está compuesto por dos partes: aplicación para el PC y aplicación para el controlador.

Aplicación para el PC:

- Debes gestionar las comunicaciones seriales y al mismo tiempo mostrar un contenido digital dinámica que permita observar fácilmente caídas en el framerate. Si quieres puedes usar la estrategia del contador que se incremente en cada frame o cambiar por algo que te guste más.
- Implementa una interfaz de usuario compuesta por botones y cajas de texto para controlar y visualizar.

Aplicación para el controlador:

- Programa un tarea que espere solicitudes de datos por parte de la aplicación interactiva. Por favor, recuerda de los ejercicios del trayecto de actividades cómo se hace esto.
- La tarea debe incrementar cada segundo un contador.
- La tarea debe poder modificar el estado del LED por solicitud de la aplicación interactiva.

Protocolo de comunicación:

- El PC SIEMPRE inicia la comunicación solicitando información al controlador. Es decir, desde la aplicación del PC siempre se solicita información y el controlador responde.
- Desde el PC se enviarán tres solicitudes: `read`, `outON`, `outOFF`.
- Para enviar los comandos anteriores usarás los botones de la interfaz de usuario.
- El controlador enviará los siguientes mensajes de respuesta a cada solicitud:
  - Respuesta a `read`: `estadoContador,estadoLED`. Por ejemplo, una posible respuesta será: `235,OFF`. Quiere decir que el contador está en 235 y el LED está apagado.
  - Respuesta a `outON` y `outOFF`: `estadoLED`. Es decir, el controlador recibe el comando, realiza la orden solicitada y devuelve el estado en el cual quedó el LED luego de la orden.
- No olvides que DEBES terminar TODOS los mensajes con el carácter NEWLINE (`\n`) para que ambas partes sepan que el mensaje está completo.

CAPÍTULO 6

Unidad 3. Protocolos binarios

6.1 Introducción

En esta unidad seguiremos profundizando en la integración de dispositivos periféricos a aplicaciones interactivas, pero esta vez usando protocolos binarios.

6.1.1 Propósitos de aprendizaje

Integrar aplicaciones interactivas y periféricos utilizando protocolos seriales binarios.

6.2 Evaluación de la unidad

Advertencia: SUSTENTACIÓN

No olvides que la evaluación solo se considera entregada cuando la sustentes. Antes de sustentarla, toda la información solicitada debe estar en el repositorio. Verifica con tus compañeros de equipo que todos los requisitos se cumplen antes de sustentar.

6.2.1 Enunciado

Vas a modificar el proyecto de la unidad anterior para cumplir los siguientes requisitos:

- Vas a utilizar un protocolo binario para comunicar la aplicación del PC y el microcontrolador.
- Para leer las variables del microcontrolador, el PC enviará un BYTE (tu decides cuál). El microcontrolador responderá la solicitud con la información de `TODAS` las variables. ¿Qué información tendrá la respuesta? Para cada variable dirá el valor actual, si está habilitada, el intervalo de cambio y el delta del cambio. Adicionalmente, deberás incluir un byte extra al final que cambiará en función de la información que envíes. La idea con este byte es que el receptor pueda verificar que la información recibida no se dañó en el camino. A esta idea se le conoce

como checksum. NO PIERDAS DE VISTA que el mensaje reporta TODAS las variables, ya no puedes tener mensajes para cada variable. Profe ¿Cómo calculo el checksum? (Internet, Chatgpt, Bingchat).

- Ten presente que ahora las variables son números en punto flotante y el delta de cambio ya no es UNO, sino un número en punto flotante.
- Para modificar las variables del microcontrolador, el PC deberá enviar una trama igual a la anterior. Es decir, en un solo mensaje ENVIAR TODA la información de las variables.
- Vas a simular un CASO de error, es decir, un caso en el cual el checksum calculado por el PC y el microcontrolador no es el mismo. Visualiza el resultado de esta simulación de manera creativa. Puedes simular este caso usando un botón en la GUI de la aplicación o como tu quieras.
- El microcontrolador deberá mantener un LED funcionando a una frecuencia de 0.5 Hz. El objetivo de este LED es que verifiques de manera visual que la aplicación en el microcontrolador NUNCA se bloquea.

6.2.2 Entrega

Advertencia: LEE PRIMERO

Lee primero todos los pasos antes de comenzar el proceso.

- Vas a realizar tu entrega en este repositorio.
- Sigue los pasos de la unidad anterior para entregar tu proyecto de Unity.
- Ahora crea el archivo README.md en el repositorio y escribe allí para cada aplicación (Arduino y Unity):
  1. ¿Cuáles son los estados y por qué definiste esos estados?
  2. ¿Cuáles son los eventos y por qué definiste esos eventos?
  3. Incluye un enlace a un video de Youtube donde muestres la aplicación funcionando. RECUERDA que es un enlace, NO el video.

6.3 Trayecto de Actividades

6.3.1 Ejercicios

Ejercicio 1: introducción a los protocolos binarios - caso de estudio

¿Cómo se ve un protocolo binario? Para explorar este concepto te voy a mostrar una hoja de datos de un sensor comercial que usa un protocolo de comunicación binario. La idea es que explores tanto como quieras, pero te quiero invitar a que mires con detenimiento hasta la página 5.

Para responder la pregunta vas a utilizar como ejemplo este sensor. Cuyo manual del fabricante se encuentra aquí

Te recuerdo la pregunta:

- ¿Cómo se ve un protocolo binario?
- ¿Puedes describir las partes de un mensaje?
- ¿Para qué sirve cada parte del mensaje?

Ejercicio 2: API de arduino para implementar comunicaciones binarias

En este enlace vas a mirar los siguientes métodos. Te pediré que los tengas a mano porque te servirán para resolver problemas.

```
Serial.available()
Serial.read()
Serial.readBytes(buffer, length)
Serial.write()
```

Nota que la siguiente función no está en el repaso:

```
Serial.readBytesUntil()
```

La razón es que en un protocolo binario usualmente no tenemos un carácter de FIN DE MENSAJE, como si ocurre con los protocolos ASCII, donde usualmente el último carácter es el \n.

Ejercicio 3: ¿Qué es el endian?

Analicemos el siguiente asunto:

Cuando trabajamos con protocolos binarios es necesario transmitir variables que tienen una longitud mayor a un byte. Por ejemplo, los números en punto flotante cumplen con el estándar IEEE754 y se representan con 4 bytes.

Algo que debemos decidir al trabajar con número como los anteriormente descritos es el orden en cual serán transmitidos sus bytes. En principio tenemos dos posibilidades: transmitir primero el byte de menor peso (little endian) o transmitir primero el byte de mayor peso (big endian). Al diseñar un protocolo binario debes escoger una de las dos posibilidades.

Ejercicio 4: transmitir números en punto flotante

Nota: Desempolva ScriptCommunicator

Para este ejercicio vas a necesitar una herramienta que te permita ver los bytes que estás transmitiendo sin interpretarlos como caracteres ASCII. Usa ScriptCommunicator en Windows y Linux y CoolTerm en MacOS (te soporta la arquitectura Mx).

¿Cómo transmitir un número en punto flotante?

Veamos dos maneras:

```
void setup() {
    Serial.begin(115200);
}

void loop() {
    // 45 60 55 d5
    // https://www.h-schmidt.net/FloatConverter/IEEE754.html
    static float num = 3589.3645;

    if(Serial.available()){
        if(Serial.read() == 's'){
            Serial.write ( (uint8_t *) &num,4);
```

(continué en la próxima página)

(proviene de la página anterior)

```
        }
    }
}
```

Y esta otra forma. Aquí primero se copia la información que se desea transmitir a un buffer o arreglo:

```
void setup() {
    Serial.begin(115200);
}

void loop() {
    // 45 60 55 d5
    // https://www.h-schmidt.net/FloatConverter/IEEE754.html
    static float num = 3589.3645;
    static uint8_t arr[4] = {0};

    if(Serial.available()){
        if(Serial.read() == 's'){
            memcpy(arr,(uint8_t *)&num,4);
            Serial.write(arr,4);
        }
    }
}
```

- ¿En qué endian estamos transmitiendo el número?
- Y si queremos transmitir en el endian contrario?

Nota: ALERTA DE SPOILER

Te dejo una posible solución a la pregunta anterior.

```
void setup() {
    Serial.begin(115200);
}

void loop() {
    // 45 60 55 d5
    // https://www.h-schmidt.net/FloatConverter/IEEE754.html
    static float num = 3589.3645;
    static uint8_t arr[4] = {0};

    if(Serial.available()){
        if(Serial.read() == 's'){
            memcpy(arr,(uint8_t *)&num,4);
            for(int8_t i = 3; i >= 0; i--){
              Serial.write(arr[i]);
            }
        }
    }
}
```

Ejercicio 5: envía tres números en punto flotante

Ahora te voy a pedir que practiques. La idea es que transmitas dos números en puntos flotante en ambos endian.

Ejercicio 6: aplicación interactiva

Te voy a pedir dos cosas en este punto:

- Que repases (de la unidad anterior o en la documentación de C# de Microsoft) para qué sirven los siguientes fragmentos de código y qué están haciendo:

```
SerialPort _serialPort = new SerialPort();
_serialPort.PortName = "/dev/ttyUSB0";
_serialPort.BaudRate = 115200;
_serialPort.DtrEnable = true;
_serialPort.Open();
```

```
byte[] data = { 0x01, 0x3F, 0x45};
_serialPort.Write(data,0,1);
```

```
byte[] buffer = new byte[4];
.
.
.

if(_serialPort.BytesToRead >= 4){

    _serialPort.Read(buffer,0,4);
    for(int i = 0;i < 4;i++){
        Console.Write(buffer[i].ToString("X2") + " ");
    }
}
```

Nota: A PRACTICAR

Inventa una aplicación en Unity que utilice TODOS los métodos anteriores. Ten presente que necesitarás inventar también la aplicación del microcontrolador.

Ejercicio 7: RETO

Vas a enviar 2 números en punto flotante desde un microcontrolador a una aplicación en Unity usando comunicaciones binarias. Inventa una aplicación en Unity que modifique dos dimensiones de una game object usando los valores recibidos.

Truco: Te voy a dejar una ayuda

¿Para qué puede servir el siguiente código?

```
byte[] buffer = new byte[4];
.
.
.
if(_serialPort.BytesToRead >= 4){
  _serialPort.Read(buffer,0,4);
  Console.WriteLine(System.BitConverter.ToSingle(buffer,0));
```

Nota: PRESTA ESPECIAL ATENCIÓN

Presta especial atención System.BitConverter.ToSingle. Te pediré que busques en la documentación de Microsoft de C# qué más te ofrece System.BitConverter

CAPÍTULO 7

Unidad 4. Plataformas de software interactivas de tiempo real

7.1 Introducción

Llegaste al final del curso. En esta unidad aplicarás todos los conceptos que has aprendido para la construcción de aplicaciones interactivas que integren sistemas embebidos con plataformas de cómputo interactivas.

7.1.1 Propósitos de aprendizaje

Aplicar los conceptos y procedimientos aprendidos en el curso para resolver un problema que requiera la integración de una plataforma de tiempo real con un sistema microcontrolado.

7.2 Evaluación de la Unidad 4

Advertencia: FECHA MÁXIMA DE ENTREGA

Recuerda que el curso termina en la última sesión de la semana 16.

7.2.1 Enunciado

Acabas de llegar a un nuevo estudio que desarrolla EXPERIENCIAS INTERACTIVAS y te encargan que DISEÑES e IMPLEMENTES una EXPERIENCIA INTERACTIVA que combine un desarrollo anterior (unidad 3) con algunos de los conceptos implementados en un plugin llamado Ardity. Mira, el equipo de desarrollo del estudio simplemente quiere mejorar el proyecto anterior pero sin depender de un plugin de un tercero. Tu misión será entonces entender los conceptos del plugin y transferirlos al desarrollo de la unidad 3. En particular deberás realizar un refactoring del proyecto de la unidad 3 así:

- El código que maneja el puerto serial debe estar en su propio Thread o hilo.

- El thread del serial debe comunicarse con el thread del motor utilizando una queue. Debes garantizar acceso único a la queue a cada thread para evitar posibles `condiciones de carrera`.
- Debes gestionar excepciones: abrir un puerto serial que no esta disponible, el cable serial se desconectó.
- La aplicación debe recuperarse de una excepción y volver a funcionar una vez se restablezcan las condiciones. Esto debe ocurrir sin necesidad de reiniciarla.
- La mayor parte de código de manejo del puerto serial debe estar en una clase abstracta. Utiliza una clase para implementar los métodos para enviar y recibir el mensaje serial tal como lo muestra Ardity con los métodos SendToWire y ReadFromWire.

7.2.2 ¿Qué debes entregar?

- Vas a realizar tu entrega en este repositorio.
- Sigue los pasos de la unidad anterior para entregar tu proyecto de Unity.
- Ahora crea el archivo README.md en el repositorio y escribe allí para la aplicación de Arduino:
  1. ¿Cuáles son los estados y por qué definiste esos estados?
  2. ¿Cuáles son los eventos y por qué definiste esos eventos?

  Para la aplicación en Unity:
  1. Muestra y explica cómo usaste el concepto de Thread
  2. Muestra y explica cómo usaste el concepto de Queue o cola.
  3. Muestra y explica cómo usaste la gestión de excepciones.
- Incluye un enlace a un video de Youtube donde muestres la aplicación funcionando. RECUERDA que es un enlace, NO el video.

7.3 Trayecto de actividades

7.3.1 Ejercicios

Ejercicio 1: caso de estudio-plugin Ardity

Lo primero que te propondré es que pongas a funcionar el demo del plugin. Todo lo relacionado con el plugin estará en este repositorio. Comienza leyendo el archivo README.md.

Cuando te sientas listo para comenzar a experimentar ten presente que la versión de Unity con la cual se realizó el proyecto fue la 2018.2. Tu estás usando la versión 2021 LTS. Abre el proyecto con tu versión de Unity.

¿Por dónde comienzo? Ve a la carpeta Scenes y comienza a analizar cada una de las escenas DemoScene. Son en total 5. En uno de los ejercicios que vienen vamos a analizar juntos la scene DemoScene_UserPoll_ReadWrite. Tu debes decidir luego cuál scene se podría ajustar mejor a tu encargo y tratar de analizarla a fondo tal como lo haremos a continuación.

Advertencia: TE DIGO ALGO PARA QUE NO TE FRUSTRES

Mira, esta unidad toma tiempo. Entre más frescos e interiorizados tengas los conceptos de este curso y del curso de programación orientada a objetos más rápido podrás realizar el análisis. Piensa en esta unidad como la oportunidad

de realizar un ejercicio de retrieval practice y aclarar con la ayuda de tu profesor algunos vacíos conceptuales que aún tengas. Quiero insistir con algo. Esta unidad no es trivial, tendrás que analizar a fondo y ser autocrítico.

Ejercicio 2: concepto de hilo

Observa y analiza este video donde te explicarán rápidamente el concepto de hilo.

Te voy a pedir que veas este video corto de 15 minutos donde verás aplicado el concepto de hilo y por qué es necesario.

Ahora, no te voy a pedir que hagas lo siguiente ya, claro, a menos que seas una persona muy curiosa y además tengas tiempo. En este sitio puedes profundizar sobre el concepto de hilo.

Ejercicio 3: análisis del plugin

Ahora, vamos a analizar más detalladamente una de las escenas demo de Ardity: DemoScene_UserPoll_ReadWrite

Primero, vamos a analizar rápidamente el código de arduino (para un protocolo ASCII):

```
uint32_t last_time = 0;

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    // Print a heartbeat
    if ( (millis() - last_time) >  2000)
    {
        Serial.print("Arduino is alive!!");
        Serial.print('\n');
        last_time = millis();
    }

    // Send some message when I receive an 'A' or a 'Z'.
    switch (Serial.read())
    {
        case 'A':
            Serial.print("That's the first letter of the abecedarium.");
            Serial.print('\n');
            break;
        case 'Z':
            Serial.print("That's the last letter of the abecedarium.");
            Serial.print('\n');
            break;
    }
}
```

Consideraciones a tener presentes con este código:

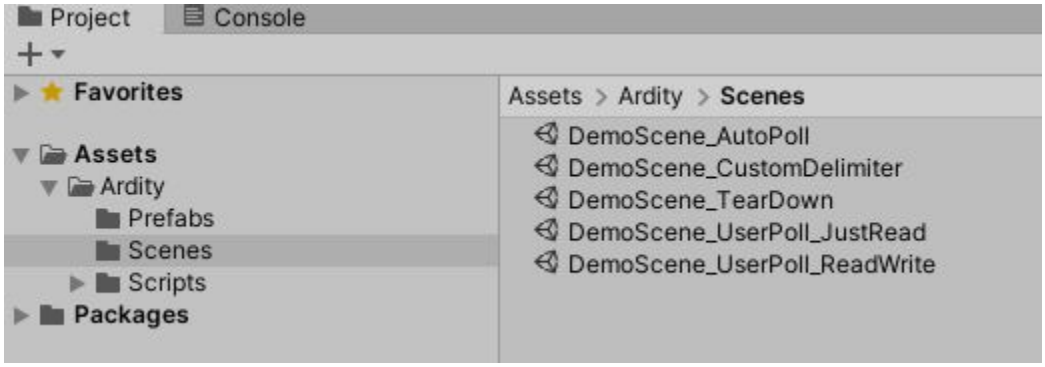
- La velocidad de comunicación es de 9600. Esa misma velocidad se tendrá que configurar del lado de Unity para que ambas partes se puedan entender.

- Nota que no estamos usando la función delay(). Estamos usando millis para medir tiempos relativos. Nota que cada dos segundos estamos enviando un mensaje indicando que el arduino está activo: `Arduino is alive!!`
- Observa que el buffer del serial se lee constantemente. NO estamos usando el método available() que usualmente utilizamos. ¿Recuerdas lo anterior? Con available() nos aseguramos que el buffer de recepción tiene al menos un byte para leer; sin embargo, cuando usamos Serial.read() sin verificar antes que tengamos datos en el buffer, es muy posible que el método devuelva un -1 indicando que no había nada en el buffer de recepción. NO OLVIDES ESTO POR FAVOR.
- Por último nota que todos los mensajes enviados por arduino finalizan con:

```
Serial.print('\n');
```

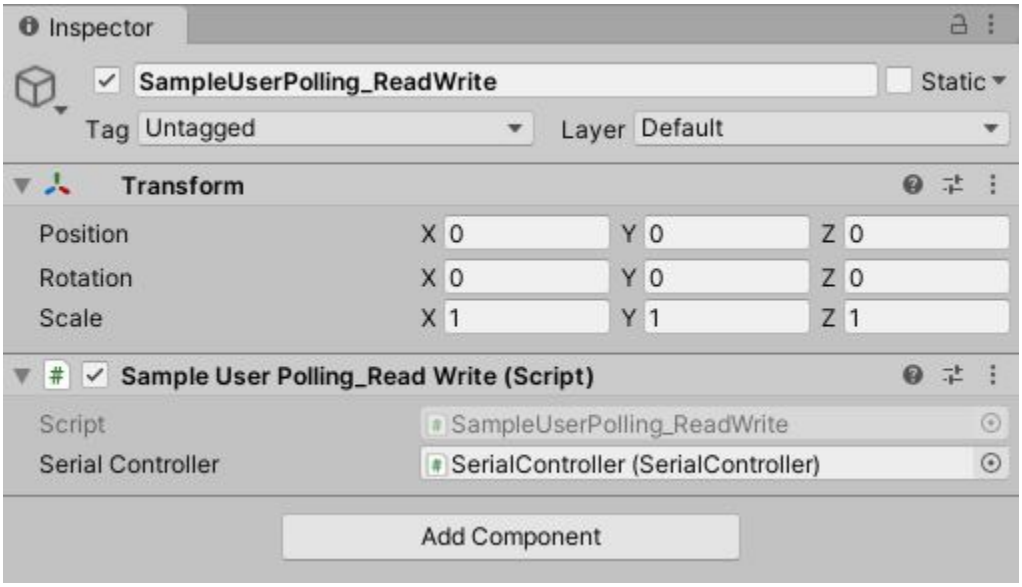
  ¿Recuerdas en la unidad 2 para qué hacemos esto? ¿Podrías desde ahora predecir que tipo de protocolo utilizará este demo (ASCII o binario)? Si tienes dudas llama a tu profe.

Ahora analicemos la parte de Unity/Ardity. Para ello, carguemos una de las escenas ejemplo: DemoScene_UserPoll_ReadWrite



Nota que la escena tiene 3 gameObjects: Main Camera, SerialController y SampleUserPolling_ReadWrite.

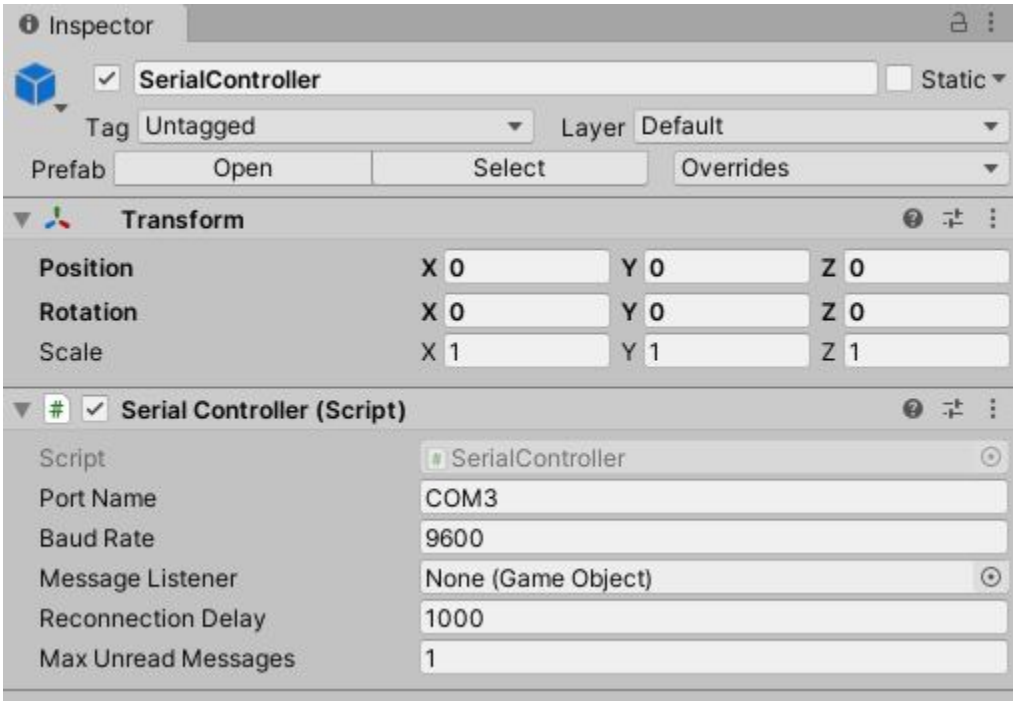
Veamos el gameObject SampleUserPolling_ReadWrite. Este gameObject tiene dos components, un transform y un script. El script tiene el código como tal de la aplicación del usuario.



Nota que el script expone una variable pública: serialController. Esta variable es del tipo SerialController.

```
12  /**
13   * Sample for reading using polling by yourself, and writing too.
14   */
15  public class SampleUserPolling_ReadWrite : MonoBehaviour
16  {
17      public SerialController serialController;
```

Esa variable nos permite almacenar la referencia a un objeto tipo SerialController. ¿Donde estaría ese objeto? Pues cuando el gameObject SerialController es creado nota que uno de sus componentes es un objeto de tipo SerialController:



Entonces desde el editor de Unity podemos arrastrar el gameObject SerialController al campo SerialController del gameObject SampleUserPolling_ReadWrite y cuando se despliegue la escena, automáticamente se inicializará la variable serialController con la referencia en memoria al objeto SerialController:

De esta manera logramos que el objeto SampleUserPolling_ReadWrite tenga acceso a la información del objeto SerialController.

Observemos ahora qué datos y qué comportamientos tendría un objeto de tipo SampleUserPolling_ReadWrite:

```
/**
 * Ardity (Serial Communication for Arduino + Unity)
 * Author: Daniel Wilches <dwilches@gmail.com>
 *
 * This work is released under the Creative Commons Attributions license.
 * https://creativecommons.org/licenses/by/2.0/
 */
```

(continué en la próxima página)

(proviene de la página anterior)

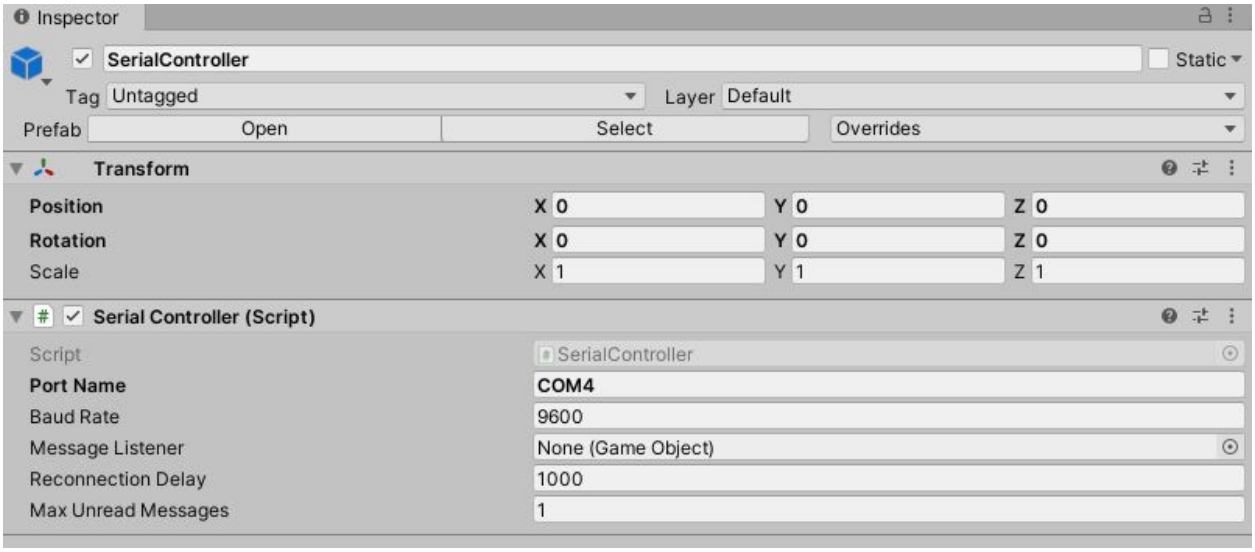
```
using UnityEngine;
using System.Collections;

/**
 * Sample for reading using polling by yourself, and writing too.
 */
public class SampleUserPolling_ReadWrite : MonoBehaviour
{
    public SerialController serialController;

    // Initialization
    void Start()
    {
        serialController = GameObject.Find("SerialController").GetComponent
↪<SerialController>();

        Debug.Log("Press A or Z to execute some actions");
    }

    // Executed each frame
    void Update()
    {
        //---------------------------------------------------------------------
        // Send data
        //---------------------------------------------------------------------

        // If you press one of these keys send it to the serial device. A
        // sample serial device that accepts this input is given in the README.
        if (Input.GetKeyDown(KeyCode.A))
        {
            Debug.Log("Sending A");
            serialController.SendSerialMessage("A");
        }

        if (Input.GetKeyDown(KeyCode.Z))
        {
            Debug.Log("Sending Z");
            serialController.SendSerialMessage("Z");
        }


        //---------------------------------------------------------------------
        // Receive data
        //---------------------------------------------------------------------

        string message = serialController.ReadSerialMessage();

        if (message == null)
            return;

        // Check if the message is plain data or a connect/disconnect event.
        if (ReferenceEquals(message, SerialController.SERIAL_DEVICE_CONNECTED))
```

(continué en la próxima página)

(proviene de la página anterior)

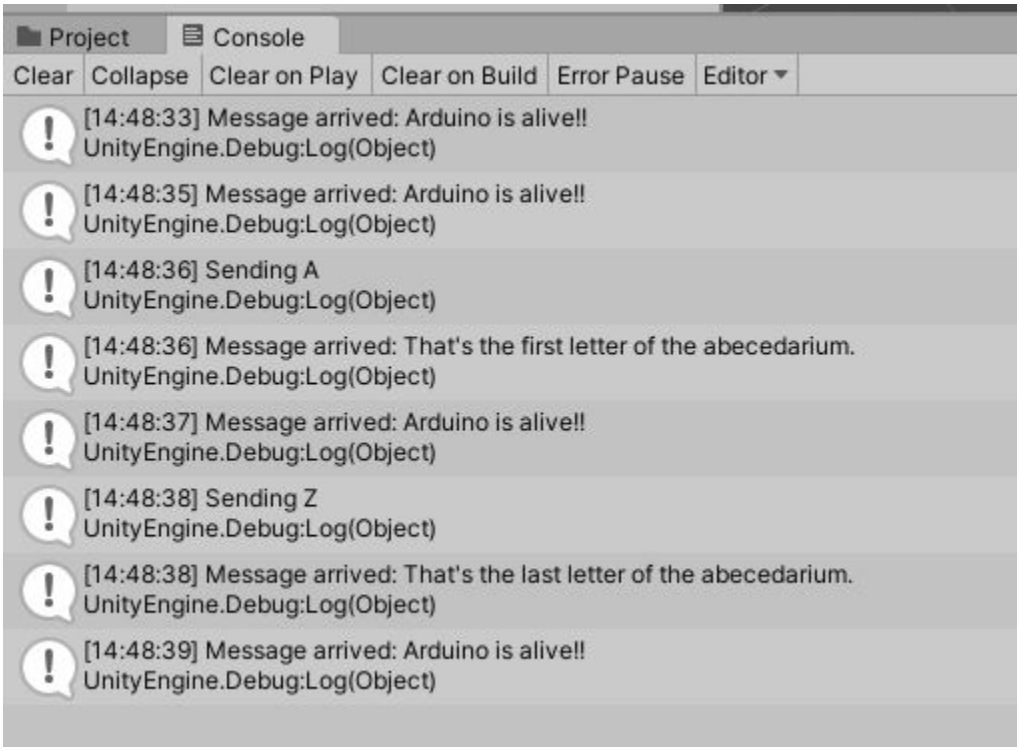
```
            Debug.Log("Connection established");
        else if (ReferenceEquals(message, SerialController.SERIAL_DEVICE_DISCONNECTED))
            Debug.Log("Connection attempt failed or disconnection detected");
        else
            Debug.Log("Message arrived: " + message);
    }
}
```

Vamos a realizar una prueba. Pero antes configuremos el puerto serial en el cual está conectado el arduino. El arduino ya debe estar corriendo el código que te mostré al comienzo.



En este caso el puerto es COM4.

Corre el programa, abre la consola y selecciona la ventana Game del Editor de Unity. Con la ventana seleccionada (click izquierdo del mouse), escribe las letras A y Z. Notarás los mensajes que aparecen en la consola:

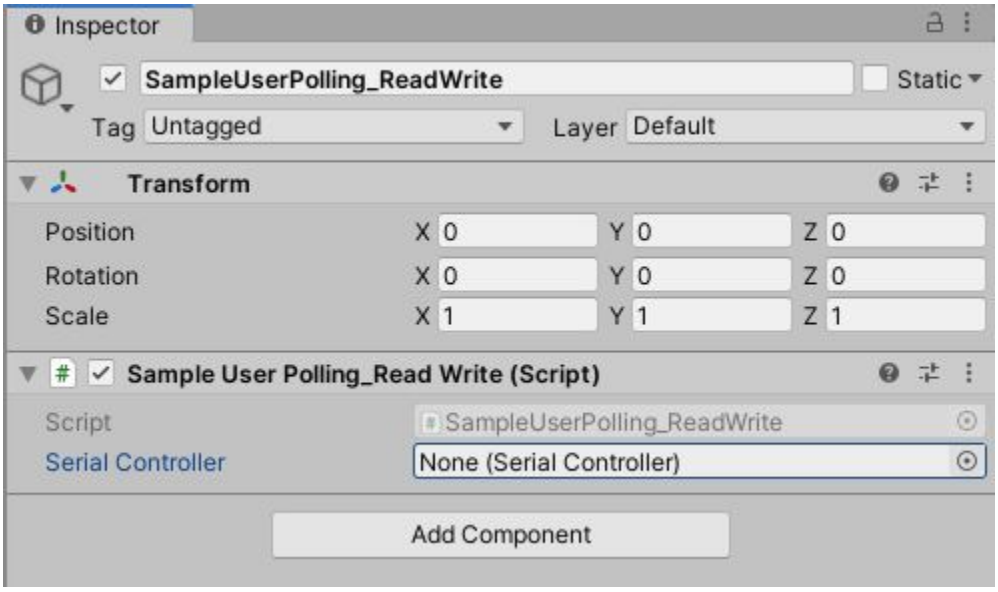


Una vez la aplicación funcione nota algo en el código de SampleUserPolling_ReadWrite:

```
serialController = GameObject.Find("SerialController").GetComponent<SerialController>();
```

Comenta esta línea y corre la aplicación de nuevo. Funciona?

Ahora, elimina el comentario de la línea y luego borra la referencia al SerialController en el editor de Unity:



Corre de nuevo la aplicación.

- ¿Qué puedes concluir?
- ¿Para qué incluyó esta línea el autor del plugin?

Ahora analicemos el código del método Update de SampleUserPolling_ReadWrite:

```
// Executed each frame
void Update()
{
  .
  .
  .
  serialController.SendSerialMessage("A");
  .
  .
  .
  string message = serialController.ReadSerialMessage();
  .
  .
  .
}
```

¿Recuerdas cada cuánto se llama el método Update?

Update se llama en cada frame. Lo llama automáticamente el motor de Unity

Nota los dos métodos que se resaltan:

```
serialController.SendSerialMessage("A");
string message = serialController.ReadSerialMessage();
```

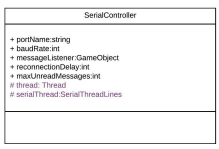
Ambos métodos se llaman sobre el objeto cuya dirección en memoria está guardada en la variable serialController.

El primer método permite enviar la letra A y el segundo permite recibir una cadena de caracteres.

- ¿Cada cuánto se envía la letra A o la Z?
- ¿Cada cuánto leemos si nos llegaron mensajes desde el arduino?

Ahora vamos a analizar cómo transita la letra A desde el SampleUserPolling_ReadWrite hasta el arduino.

Para enviar la letra usamos el método SendSerialMessage de la clase SerialController. Observa que la clase tiene dos variables protegidas importantes:



```
protected Thread thread;
protected SerialThreadLines serialThread;
```

Con esas variables vamos a administrar un nuevo hilo y vamos a almacenar una referencia a un objeto de tipo SerialThreadLines.

En el método onEnable de SerialController tenemos:

```
serialThread = new SerialThreadLines(portName, baudRate, reconnectionDelay,␣
↪maxUnreadMessages);
thread = new Thread(new ThreadStart(serialThread.RunForever));
thread.Start();
```

Aquí vemos algo muy interesante, el código del nuevo hilo que estamos creando será RunForever y ese código actuará sobre los datos del objeto cuya referencia está almacenada en serialThread.

Vamos a concentrarnos ahora en serialThread que es un objeto de la clase SerialThreadLines:

```
public class SerialThreadLines : AbstractSerialThread
{
    public SerialThreadLines(string portName,
                             int baudRate,
                             int delayBeforeReconnecting,
                             int maxUnreadMessages)
        : base(portName, baudRate, delayBeforeReconnecting, maxUnreadMessages, true)
    {
    }

    protected override void SendToWire(object message, SerialPort serialPort)
    {
        serialPort.WriteLine((string) message);
    }

    protected override object ReadFromWire(SerialPort serialPort)
    {
        return serialPort.ReadLine();
    }
}
```

Al ver este código no se observa por ningún lado el método RunForever, que es el código que ejecutará nuestro hilo. ¿Dónde está? Observa que SerialThreadLines también es un AbstractSerialThread. Entonces es de esperar que el método RunForever esté en la clase AbstractSerialThread.

Por otro lado nota que para enviar la letra A usamos el método SendSerialMessage también sobre los datos del objeto referenciado por serialThread del cual ya sabemos que es un SerialThreadLines y un AbstractSerialThread

```
public void SendSerialMessage(string message)
{
    serialThread.SendMessage(message);
}
```

Al igual que RunForever, el método SendMessage también está definido en AbstractSerialThread.

Veamos entonces ahora qué hacemos con la letra A:

```
public void SendMessage(object message)
{
    outputQueue.Enqueue(message);
}
```

Este código nos da la clave. Lo que estamos haciendo es guardar la letra A que queremos transmitir en una COLA. Esta estructura de datos permite PASAR información de un HILO a otro HILO.

¿Cuáles hilos?

Pues tenemos en este momento dos hilos: el hilo del motor y el nuevo hilo que creamos antes. El hilo que ejecutará el código RunForever sobre los datos del objeto de tipo SerialThreadLines:AbstractSerialThread. Por tanto, observa que la letra A la estamos guardando en la COLA del SerialThreadLines:AbstractSerialThread

Si observas con detenimiento el código de RunForever:

```
public void RunForever()
{
    try
    {
        while (!IsStopRequested())
        {
            ...
            try
            {
                AttemptConnection();
                while (!IsStopRequested())
                    RunOnce();
            }
            catch (Exception ioe)
            {
            ...
            }
        }
    }
    catch (Exception e)
    {
    ...
    }
}
```

Los detalles están en RunOnce():

```
private void RunOnce()
{
    try
    {
        // Send a message.
        if (outputQueue.Count != 0)
        {
            SendToWire(outputQueue.Dequeue(), serialPort);
        }
        object inputMessage = ReadFromWire(serialPort);
        if (inputMessage != null)
```

(continué en la próxima página)

(proviene de la página anterior)

```
        {
            if (inputQueue.Count < maxUnreadMessages)
            {
                inputQueue.Enqueue(inputMessage);
            }
        }
    }
    catch (TimeoutException)
    {
    }
}
```

Y en este punto vemos finalmente qué es lo que pasa. Para enviar la letra A, el código del hilo pregunta si hay mensajes en la cola. Si los hay, nota que el mensaje se saca de la cola y se envía:

```
SendToWire(outputQueue.Dequeue(), serialPort);
```

Si buscamos el método SendToWire en AbstractSerialThread vemos:

```
protected abstract void SendToWire(object message, SerialPort serialPort);
```

Y aquí es donde se conectan las clases SerialThreadLines con AbstractSerialThread, ya que el método SendToWire es abstracto, SerialThreadLines tendrá que implementarlo

```
public class SerialThreadLines : AbstractSerialThread
{
    ...
    protected override void SendToWire(object message, SerialPort serialPort)
    {
        serialPort.WriteLine((string) message);
    }
    ...
}
```

Aquí vemos finalmente el uso de la clase SerialPort de C# con el método WriteLine

Finalmente, para recibir datos desde el serial, ocurre el proceso contrario:

```
public class SerialThreadLines : AbstractSerialThread
{
    ...
    protected override object ReadFromWire(SerialPort serialPort)
    {
        return serialPort.ReadLine();
    }
}
```

ReadLine también es la clase SerialPort. Si leemos cómo funciona ReadLine queda completamente claro la razón de usar otro hilo:

> **Advertencia:** By default, the ReadLine method will block until a line is received. If this behavior is undesirable, set the ReadTimeout property to any non-zero value to force the ReadLine method to throw a TimeoutException if a line is not available on the port.

> Note that while this method does not return the NewLine value, the NewLine value is removed from the input buffer.

Por tanto, volviendo a RunOnce:

```
private void RunOnce()
{
    try
    {
        if (outputQueue.Count != 0)
        {
            SendToWire(outputQueue.Dequeue(), serialPort);
        }

       object inputMessage = ReadFromWire(serialPort);
        if (inputMessage != null)
        {
            if (inputQueue.Count < maxUnreadMessages)
            {
                inputQueue.Enqueue(inputMessage);
            }
            else
            {
                Debug.LogWarning("Queue is full. Dropping message: " + inputMessage);
            }
        }
    }
    catch (TimeoutException)
    {
        // This is normal, not everytime we have a report from the serial device
    }
}
```

Vemos que se envía el mensaje:

```
SendToWire(outputQueue.Dequeue(), serialPort);
```

Y luego el hilo se bloquea esperando por una respuesta:

```
object inputMessage = ReadFromWire(serialPort);
```

Nota que primero se envía y luego el hilo se bloquea. NO SE DESBLOQUEARÁ HASTA que no envíe una respuesta desde Arduino o pasen 100 ms que es el tiempo que dura bloqueada la función antes de generar una excepción de timeout de lectura.

¿Cómo sabemos que son 100 ms?

Mira con detenimiento el código. La siguiente línea te dará una pista.

```
// Amount of milliseconds alloted to a single read or connect. An
 // exception is thrown when such operations take more than this time
 // to complete.
 private const int readTimeout = 100;
```

Ejercicio 4: excepciones

¿Cómo puedes identificar la gestión de excepciones en el código? El código que considera las posibles excepciones está arropado con la estructura de control try catch.

Regresa al método `RunForever`. Observa AttemptConnection. ¿Qué pasa al ejecutar serialPort.Open(); si no tienes un microcontrolador conectado o el puerto serial configurado no el correcto?

En este punto quiero que veas un poco de documentación. ¿Cómo hago para saber si un método puede generar excepciones? Eso te lo dice la documentación. Te dejo un ejemplo aquí para el método Open.

Preguntas para que pienses:

- Luego de ver la documentación podrías decir si ¿Es posible determinar que la excepción está ocurriendo porque el nombre del puerto no comienza por la palabra COM?
- ¿Serial posible indicarle al usario que el puerto no puede abrirse porque otro programa ya lo tiene abierto?
- En la scene DemoScene_UserPoll_ReadWrite ¿Qué pasa si desconectas el microcontrolador? y ¿Qué pasa si lo vuelves a conectar?
- Explica tan detallado como puedas qué ocurre con la aplicación al desconectar y luego volver a conectar el microcontrolador.

Ejercicio 5: evaluación formativa

- Crea un proyecto nuevo en Unity.
- Configura el soporte para el puerto serial.
- OJO, no instales el paquete Ardity. SI YA LO HICISTE, vuelve a comenzar.
- Ahora toma únicamente LOS SCRIPTS de Ardity necesarios (SOLO LOS NECESARIOS) para hacer que la aplicación DEMO del ejercicio anterior funcione.