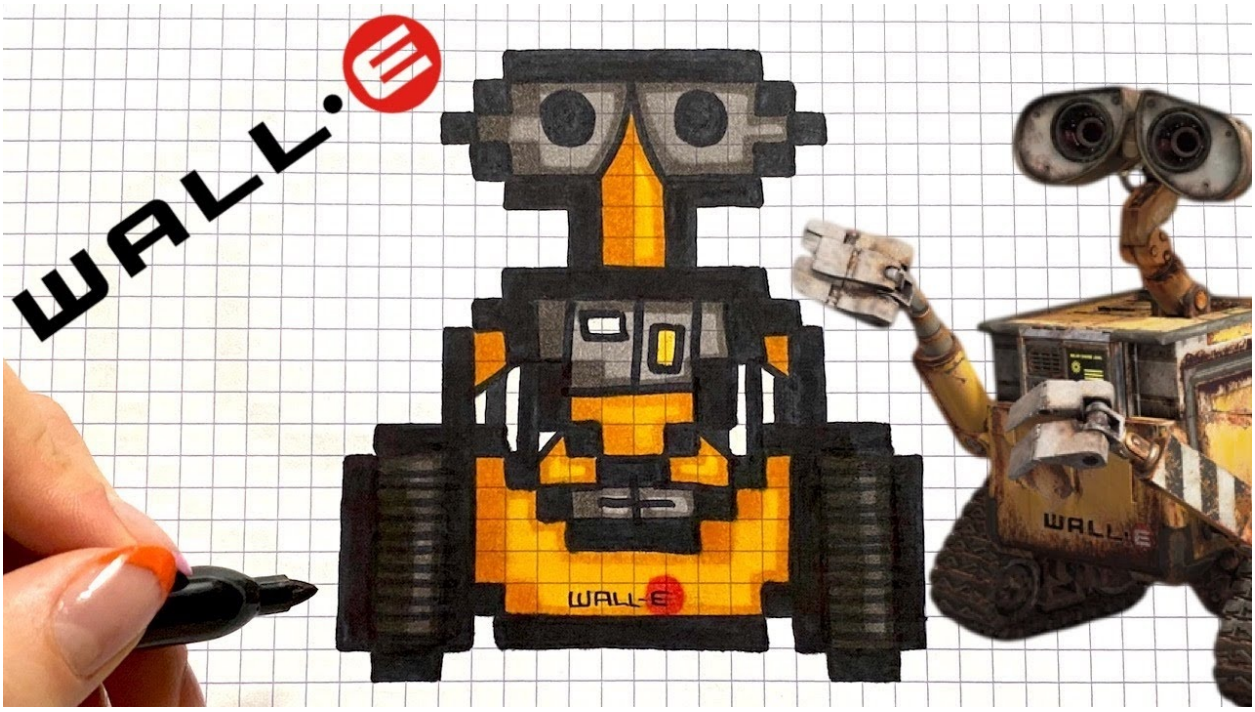
2do proyecto de Programación
Pixel Wall-E
MATCOM - 2024-2025



1. Introducción

Wall-E se aburrió de las figuras geométricas, en cambio, ahora quiere dedicarse al pixel-art y tú debes ayudarlo. Debe crear una aplicación capaz de leer e interpretar comandos (Los comandos forman un lenguaje de programación que se explicará pronto). Siguiendo los comandos, su robot debe pintar píxeles sobre un canvas cuadrado.

2. Código

2.1. Formato de entrada

Tu aplicación debe ser capaz de recibir entradas de dos formas:

- Editor de texto. La aplicación debe contar con un editor de texto sobre el que escribir. Los comandos se redactan en el editor y al pulsar algún botón, estos se ejecutan sobre el canvas.
- Leer archivos de extensión .pw. Debes ser capaz de importar archivos salvados y cargarlos en el editor de texto.

Adicionalmente, debe haber una opción para exportar el texto que esté actualmente en el editor.

2.2. El lenguaje

Nuestro lenguaje está formado por un conjunto de instrucciones (o comandos), asignaciones, funciones, etiquetas y saltos condicionales que deben ejecutarse secuencialmente y están separados por saltos de línea.

2.3. Instrucciones

Las instrucciones afectan a Wall-E y/o al canvas directamente. Reciben como entrada variables o literales. Después de una instrucción debe haber un salto de línea.

2.3.1. `Spawn(int x, int y )`

Todo código válido debe, obligatoriamente, comenzar con un comando `Spawn(int x, int y)` y sólo puede utilizarse esta instrucción una vez. Este comando inicializa a Wall-E sobre el canvas. Las entradas **x, y** representan las coordenadas iniciales. Por ejemplo:

- → `Spawn(0, 0)`: Se posiciona a **Wall-E** en la esquina superior izquierda, posición (0, 0)
- → `Spawn(50, 50)`: Suponga que se tiene un canvas de 100 x 100 píxeles, entonces se posiciona a **Wall-E** en el centro, posición (50, 50)
- → `Spawn(255, 0)`: Suponga que se tiene un canvas de 256x256 píxeles, se posiciona a **Wall-E** en la esquina superior derecha, posición (255, 0)

Si la posición inicial de Wall-E cae fuera de los límites del canvas actual, se tiene un error en tiempo de ejecución.

2.3.2. `Color(string color)`

Este comando cambia el color del pincel. Deben aceptarse, en principio, los siguientes colores como entrada:

- ”Red”
- ”Blue”

- ”Green”
- ”Yellow”
- ”Orange”
- ”Purple”
- ”Black”
- ”White”
- ”Transparent”

Note que el color por defecto si no se ha utilizado ningún comando `Color` es el transparente (”Transparent”). Utilizar el color transparente implica no realizar ningún cambio sobre el canvas. Además, el canvas es inicialmente blanco, por lo que utilizar el color ”White”se puede interpretar como un borrador.

2.3.3. Size(int k)

Modifica el tamaño del pincel. La entrada `k` ¿0 representa el grosor, en píxeles, de la brocha. El grosor debe ser un número impar, por lo que si la entrada es par, debe utilizarse su número impar inmediatamente menor. El tamaño por defecto del pincel es de un pixel.

2.3.4. DrawLine(int dirX, int dirY, int distance)

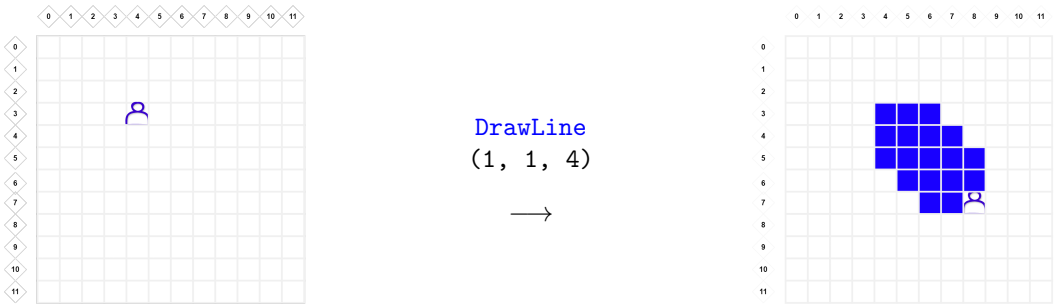
Dibuja una línea sobre el canvas que compienza en la posición actual de Wall-E y termina a distancia `distance` en píxeles en la dirección establecida. Las entradas válidas para `dirX` y `dirY` son: -1, 0, 1, dando como resultado 8 direcciones:

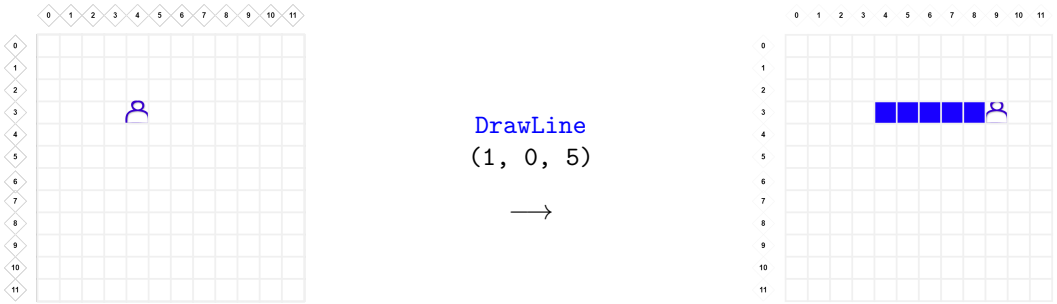
- (-1, -1) ← Diagonal arriba izquierda.
- (-1, 0) ← Izquierda
- (-1, 1) ← Diagonal abajo izquierda.
- (0, 1) ← Abajo
- (1, 1) ← Diagonal abajo a la derecha.
- (1, 0) ← Derecha.
- (1, -1) ← Diagonal arriba derecha.
- (0, -1) ← Arriba.

Note que se deben pintar todos los píxeles en el camino recto de la línea y aquellos adyacentes según el ancho actual del pincel. Después de esta instrucción, la nueva posición de Wall-E será en el final de la línea (último pixel dibujado). Por ejemplo:

- Si Wall-E está posicionado en el pixel (10, 10) y recibe el comando DrawLine(1, 0, 5). Dibijará una línea horizontal desde esa posición inicial hasta la posición (15, 10). Después de ejecutado el comando, la posición de Wall-E será (15, 10).

Si se tiene el siguiente canvas con el pincel azul, de ancho 3 y posición inicial (4, 3):

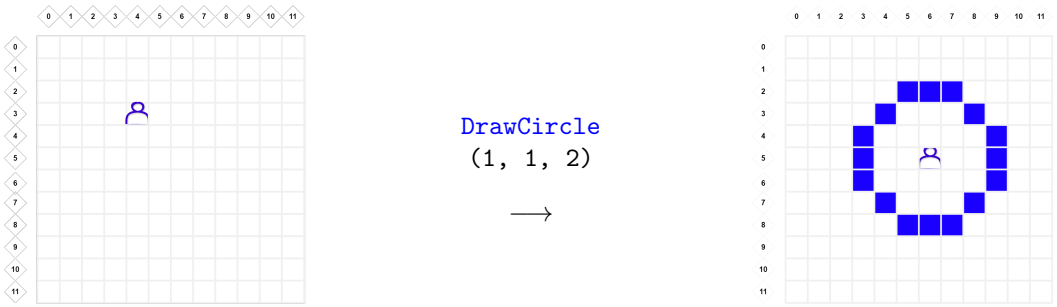




2.3.5. DrawCircle(int dirX, int dirY, int radius)

Dibuja un círculo de radio `radius` en la dirección establecida. La posición final de Wall-E será el centro del círculo. La línea del radio no se debe dibujar, sólo la circunferencia.

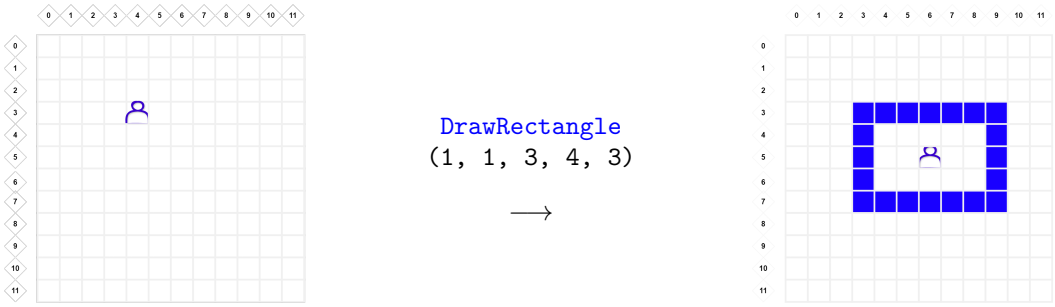
Por ejemplo, dada la posición inicial (4, 3), ancho de pincel 1 y color azul:



2.3.6. DrawRectangle(int dirX, int dirY, int distance, int width, int height)

Similar al círculo pero dibujando un rectángulo. Wall-e se moverá en la dirección `(dirX, dirY)` una cantidad de píxeles igual a `distance` y esta posición representará el centro del rectángulo de largo `width` y altura `height`

Por ejemplo, dada la posición inicial (4, 3), ancho de pincel 1 y color azul:



2.3.7. Fill()

Pinta con el color de brocha actual todos los píxeles del color de la posición actual que son alcanzables sin tener que caminar sobre algún otro color.

Por ejemplo, dada la posición inicial siguiente sobre un píxel púrpura y color de brocha azul:

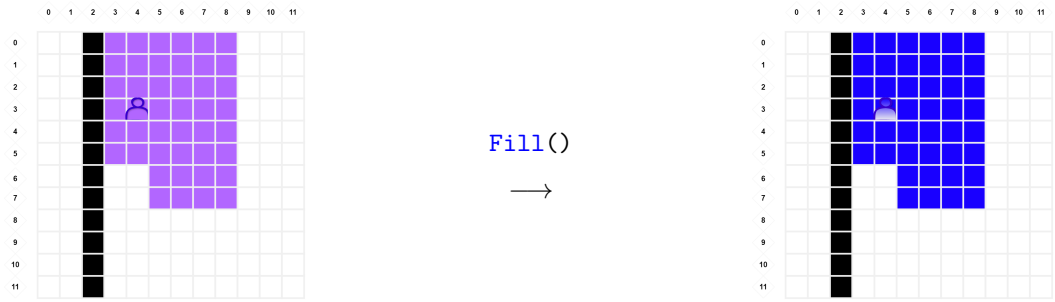


Figura 1: Comparación entre la imagen inicial y la imagen resultante tras `Fill()`

En cambio, si se tiene la siguiente posición inicial sobre un píxel blanco y color de brocha azul:

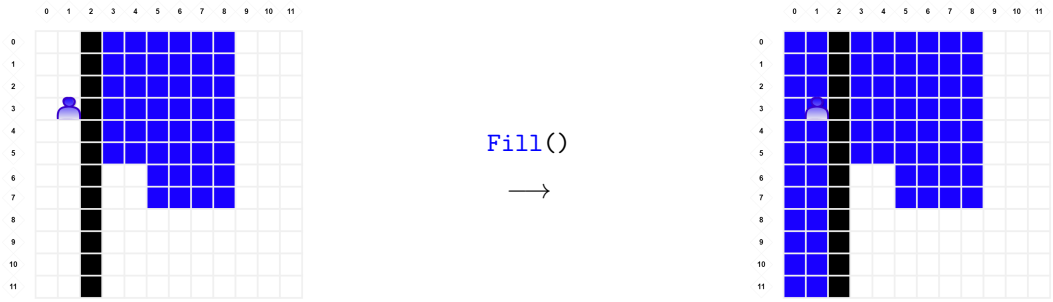


Figura 2: Comparación entre la imagen inicial y la imagen resultante tras `Fill()`

2.3.8. Asignación de variables

Nuestro código de entrada también soporta asignación de variables numéricas o booleanas. La sintaxis de asignación es la siguiente:

$$var \leftarrow Expression$$

Donde *var* tiene forma de cadena de texto formada por los 27 caracteres del alfabeto español, caracteres numéricos y el símbolo "_". Además, no pueden comenzar con caracteres numéricos ni el símbolo "_z *Expression* es cualquier expresión aritmética o booleana. Una variables es numérica si su *Expression* es aritmética, y booleana si su *Expression* es booleana. Después de una asignación de variable debe haber un salto de línea.

2.4. Expresiones

Las expresiones pueden ser aritméticas o booleanas.

2.4.1. Expresiones aritméticas

Una expresión aritmética (o numérica) está formada por:

- Un literal: Número entero.
- Una variable numérica.
- Una operación aritmética entre dos o más expresiones aritméticas.
- Una invocación de función.

Las operaciones aritméticas que debe soportar el lenguaje son:

- Suma (+)
- Resta (-)
- Multiplicación (*)
- División (/)
- Potencia (**)
- Módulo (%)

2.4.2. Expresiones booleanas

Una expresión booleana puede evaluar como **Verdadera** o **Falsa** y está formada por:

- Concatenación **and** (&&) entre dos comparaciones.
- Concatenación **or** (||) entre dos comparaciones.
- Comparación ($==, >=, <=, >, <$) entre dos variables numéricas o literales.

La operación **or** tendrá siempre más precedencia que la operación **and**.

2.5. Funciones

Las funciones retornan un valor numérico que se puede asignar a una variable. Reciben como entrada variables o literales (no otras funciones). Después de una función debe haber un salto de lína.

2.5.1. GetActualX()

Retorna el valor **X** de la posición actual de Wall-E.

2.5.2. GetActualY()

Retorna el valor **Y** de la posición actual de Wall-E.

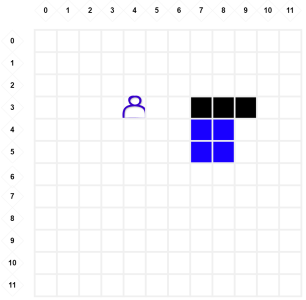
2.5.3. GetCanvasSize()

Retorna tamaño largo y ancho del canvas. Para un canvas de $n \times n$ se retorna n.

2.5.4. GetColorCount (string color, int x1, int y1, int x2, int y2)

Retorna la cantidad de casillas con color `color` que hay en el rectángulo formado por las posiciones x1,y1 (una esquina) y x2, y2(la otra esquina). Si cualquiera de las esquinas cae fuera de las dimensiones del canvas, debe retornar 0.

Por ejemplo, para el canvas:



GetColorCount ("Blue", 3, 7, 6, 9) retorna 4, mientras que GetColorCount ("Blue", 4, 8, 6, 9) retorna 2.

2.5.5. IsBrushColor(string color)

Retorna 1 si el color de la brocha actual es `string` color, 0 en caso contrario.

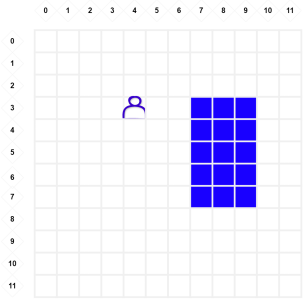
2.5.6. IsBrushSize(int size)

Retorna 1 si el tamaño de la brocha actual es `size`, 0 en caso contrario.

2.5.7. IsCanvasColor(string color, int vertical, int horizontal)

Retorna 1 si la casilla señalada está pintada del parámetro `color`, 0 en caso contrario. La casilla en cuestión se determina por la posición actual de Wall-E **(X, Y)** y se calcula como: (**X** + `horizontal`, **Y** + `vertical`). Note que si tanto `horizontal` como `vertical` son 0, se verifica entonces la casilla actual de Wall-E. Si la posición cae fuera del canvas debe retornar false.

Por ejemplo, para el canvas:



IsColor ("Blue", 5, 0) retorna 1, mientras que IsColor ("Blue", 6, 0) retorna 0.

2.6. Saltos Condicionales

Nuestro lenguaje acepta también saltos condicionales.

2.6.1. Etiquetas

Una etiqueta es una cadena de texto que marca un lugar del cóigo al cual se puede llegar a través de un `GoTo`. Las etiquetas no son asignaciones, instrucciones ni funciones, pues por sí mismas, no tienen efecto cuando llega su turno secuencial de ejecutarse. Una etiqueta, al igual que los nombres de variables, tiene forma de cadena de texto formada por los 27 caracteres del alfabeto, caracteres numéricos y el símbolo "_". Además, no pueden comenzar con caracteres numéricos ni el símbolo **-**. Después de una etiqueta debe haber un salto de línea. Ejemplos de etiquetas válidas:

- start-loop-1
- this-is-where-my-loop-saddly-ends
- my-programming-teachers-are-awesome

2.6.2. Saltos condicionales

Un salto condicional tiene la forma **GoTo** [**label**] (**condition**).

El **label** debe ser una etiqueta declarada en el código tanto antes como después de la declaración del salto. Si se trata de hacer un salto a una etiqueta inexistente, debe desencadenarse un error de compilación.

condition es una variable booleana o una comparación (==, >=, <=, >, <) entre dos variables numéricas o literales.

Los saltos condicionales tienen el efecto de que, si la condición tiene valor **Verdadero**, el código continúe su ejecución en la línea de la **etiqueta** correspondiente. Si la condición tiene valor **Falso**, entonces la esta línea se ignora y se continúa la ejecución con la línea siguiente. Después de un **GoTO** debe haber un salto de línea. Note que los paréntesis y corchetes son parte de la sintaxis del salto condicional.

2.7. Ejemplo de código

2.7.1. Asignación

```
1   Spawn(0, 0)
2   Color(Black)
3   n <- 5
4   k <- 3 + 3 * 10
5   n <- k * 2
6   actual_x <- GetActualX()
7   i <- 0
8
9   loop1
10  DrawLine(1, 0, 1)
11  i <- i + 1
12  is_brush_color_blue <- IsBrushColor("Blue")
13  Goto [loop_ends_here] (is_brush_color_blue == 1)
14  GoTo [loop1] (i < 10)
15
16  Color("Blue")
17  GoTo [loop1] (1 == 1)
18
19  loop_ends_here
```

En la línea 1 se inicializa a Wall-E en la esquina superior izquierda del canvas. En la línea 2 la brocha se pone negra. Las líneas 3 a 7 son asignaciones de variables. En la línea 9 se delcara la etiqueta `loop1` que representa el inicio del ciclo. El ciclo se ejecuta 10 veces y en cada iteración se dibuja un caracter a la derecha de negro y se desplaza a Wall-E una posición a la derecha (línea 10). En la línea 12 se llama a la función que pregunta si la brocha es azul. Como en las primeras 10 iteraciones del ciclo la línea será negra, el `GoTo` de la línea 13 se ignora y el `GoTo` de la línea 14 salta hacia el inicio del ciclo en la línea 9. En la décima iteración, la variable *i* tendrá valor 10 y por tanto el `GoTo` de la lúnea 14 se ignora. Luego se ejecuta la línea 16, la brocha pasa a ser azul y como el `GoTo` de la línea 17 siempre es verdadero, entonces se regresa a la línea 9. En esta iteración, la brocha sí será azul y por tanto se saltará a la línea 19 terminando la ejecución del programa.

3. Interfaz

Debe crear una interfaz gráfica que tenga:

- Editor de texto: Entrada de texto con números de línea en la parte izquierda para escribir el código.
- Canvas: Una sección cuadriculada. Cada casilla representa un pixel. El canvas se modifica en función del código cuando este se ejecuta.
- Entrada de dimensiones del canvas. Se debe poder introducir un número entero que modificará el largo y ancho del canvas en píxeles. Puede tener un valor cualquiera por defecto cuando la aplicación inicia.

- Botón de redimensión de canvas: Cambia las dimensiones del canvas en dependencia del valor de entrada. Cuando se redimensiona el canvas, también se limpian todos los colores modificados por ejecuciones anteriores. Todas las casillas del canvas comienzan con color blanco.
- Botón de ejecución: Un botón para ejecutar el código y que este tenga efecto sobre el canvas. Note que si el canvas ya estaba modificado por una ejecución anterior, la segunda ejecución comenzará con el canvas modificado.
- Botón de carga: Debe poder cargar en el editor de texto el contenido de un archivo de extensión .gw.
- Botón de salva: Debe poder guardar en un archivo de extensión .gw el contenido actual del editor de texto.

4. Evaluación

A continuación se listan los requisitos mínimos para aprobar:

- Interfaz gráfica como se pide en la sección anterior.
- Debe poder leerse en el editor de texto el código de entrada.
- Si el código de entrada es válido sintáctica y semánticamente para nuestro lenguaje, debe poder ejecutarse.
- Si se ejecuta un código sintáctica y semánticamente válido que tiene un error en ejecución, debe capturar este error, reportarlo y parar la ejecución. El código ejecutado antes del error se debe quedar aplicado en el canvas.
- Si se ejecuta un código correcto sintáctica y semánticamente que no tiene errores en tiempo de ejecución, el canvas debe modificarse acorde a ese códgo.
- Si el código de entrada no es válido sintáctica o semánticamente, debe reportarse el tipo de error.

Aspectos que pueden mejorar su evaluación:

- Reporte de errores inteligente. Detalles sobre las líneas con errores sintácticos o semánticos. Reportar más de un error en caso de existir y no sólo el primero que se encuentre.
- Interfaz bien trabajada. Animaciones fluidas.
- Buenas prácticas de programación (como se ha enseñado en clase).
- Extensibilidad de código. Se debe poder declarar sin mucho esfuerzo una nueva instrucción, función o tipo de expresión, sin tener que afectar el código ya existente.
- Editor de texto inteligente. Autocompletado, recomendaciones, marcas de errores.
- Extensión del lenguaje. Además de los features del lenguaje mínimos (todos los de este documento), puede pensar en otros extra (e implementarlos) que permitan hacer mejores pixel arts.

5. Entrega y Git

La entrega se hará vía GitHub. Debe abrir un issue en el repositorio que su profesor le proveerá. El formato del issue es el siguiente: **proyecto-wall-e-[Nombre]-[Apellido1]-[Grupo]**. Tenga en cuenta que el historial de commits será revisado por su profesor. El issue debe tener, en la descripción, un enlace al repositorio de su proyecto.