75.07 Algoritmos y Programación III
Segundo Cuatrimestre 2020
4-Mar-2021
Trabajo Práctico 2
Grupo 9
Docente corrector: Eugenio Yolis

Sofía Gualdieri - 102995
Mariano Medela - 101769
Miguel Lederkremer - 61719

Contenido

Detalles de implementación

Diseñamos una interfaz basada en Buttons, con tres paneles:

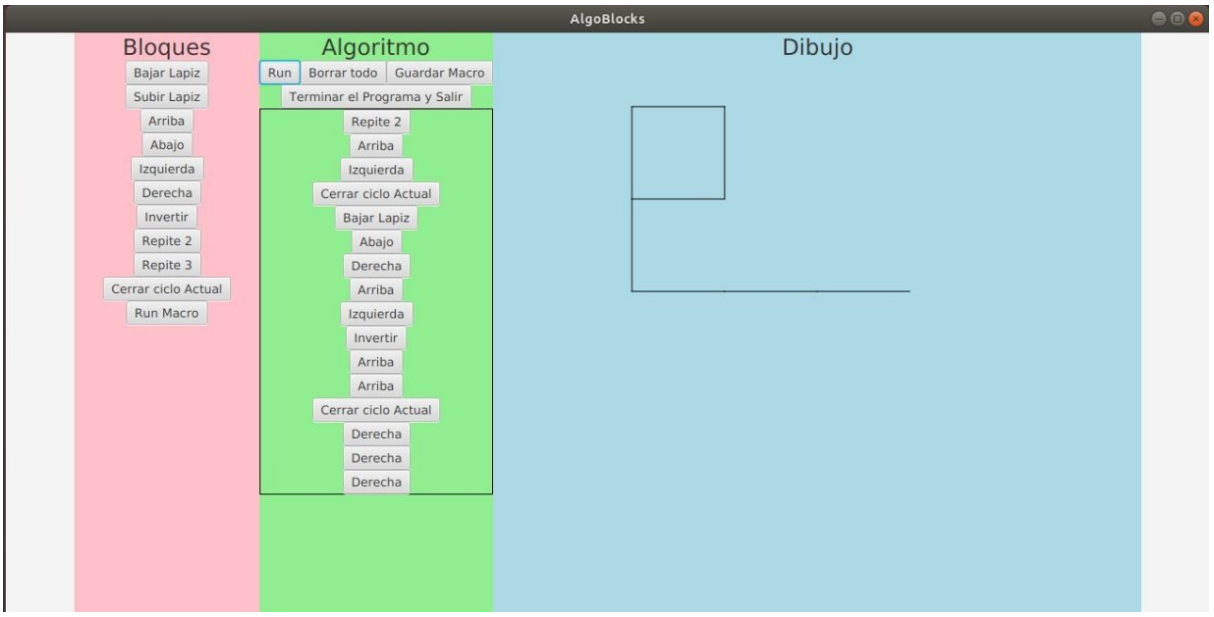

Fig. 1: Ejemplo de cómo se vería la ejecución de un determinado conjunto de bloques.

Utilizamos el modelo MVC, Modelo Vista Controlador, en su variante sin observers.

- Modelo: están definidas las clases que hacen funcionar AlgoBlocks
- Vista: se ocupa de manejar la pantalla mediante JavaFX. Hay una clase por cada panel de la pantalla: PanelAlgoritmoView, PanelBloquesView y PanelDibujoView.
- Controller: en este paquete centralizamos todos los event handlers de los buttons

Resaltamos a continuación algunos detalles del proyecto que nos resultaron especialmente desafiantes y nos gusta la solución que encontramos.

Bloque y ConjuntoBloques

Implementamos dos interfaces principales:
Bloque: que implementan todos los bloques de la aplicación. Cada tipo de bloque implementa a su manera los métodos abstractos ejecutar y ejecutarInvertido
ConjuntoBloques: la implementan AlgoBlocks y todos los bloques de tipo "macro", que tienen una lista de bloques a ejecutar y comparten el método agregarBloque()

Así, bloques como Repetir2, Invertir y Personalizado, se comportan como Bloque pero también como ConjuntoBloques, por lo que implementan ambas interfaces.
Estos bloques "macro" tienen la particularidad de que implementan la interfaz Bloque, pero también **contienen** una lista con Bloques, por eso la flecha en ambos sentidos en el diagrama de clases.

Bloques anidados

Aunque no era un requisito del enunciado, le dimos a AlgoBlocks la posibilidad de anidar una cantidad indefinida de bloques "macro", esto es: un bloque Invertir que tiene adentro un bloque Repite2 que tiene adentro un BloquePersonalizado, etc.
La solución que encontramos es análoga a un *function stack*: utilizamos una pila para guardar la referencia al último ciclo abierto. Al agregar un bloque desde la interfase visual, la aplicación se fija en la pila para saber dentro de qué ConjuntoBloques tiene que agregarlo. Cuando el algoritmo se encuentra con un bloque CerrarCicloActual, simplemente quita un elemento de la pila.
Esto se implementó en la vista / controlador porque está más relacionado con el manejo de la interfase visual que con el modelo.

Supuestos

Los principales supuestos se tomaron para los bloques "macro".
Decidimos que el método ejecutarInvertido() opere de la siguiente manera:

Bloque	**ejecutarInvertido()**
ArrAbaDerIzq	Se mueve en la direccion opuesta
LapizAbajo	Levanta el lapiz
LapizArriba	Baja el lapiz
BloqueRepite2/3	Invierte el comportamiento de todos los bloques q tiene guardados
BloquePersonalizado	Invierte el comportamiento todos los bloques que tiene guardados

También decidimos que estos bloques "macro" se puedan anidar, como está explicado en Detalles de Implementación.

Diagrama de clases

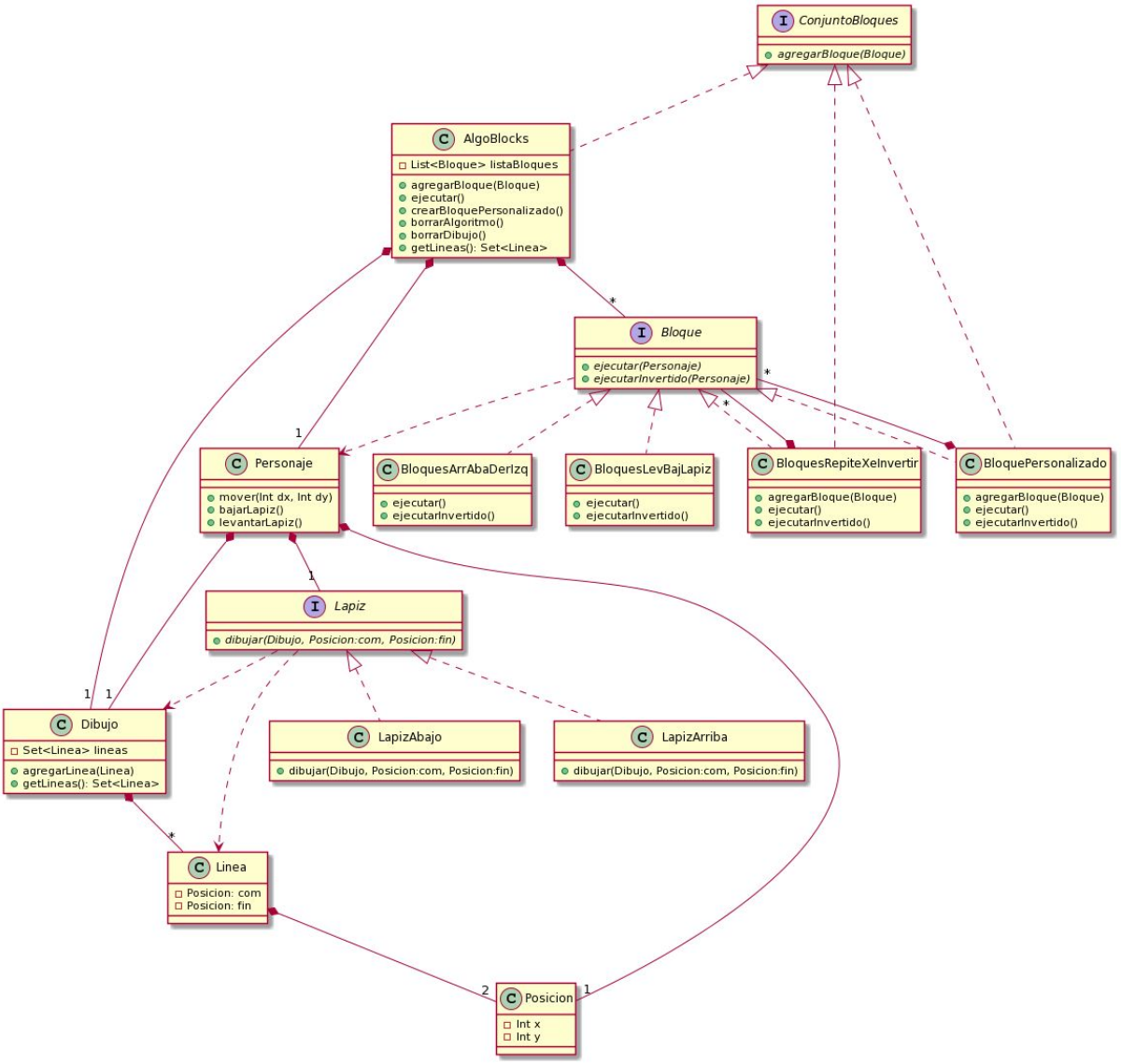

Fig. 2: Diagrama UML representando las clases principales de nuestro modelo.

Diagramas de secuencia

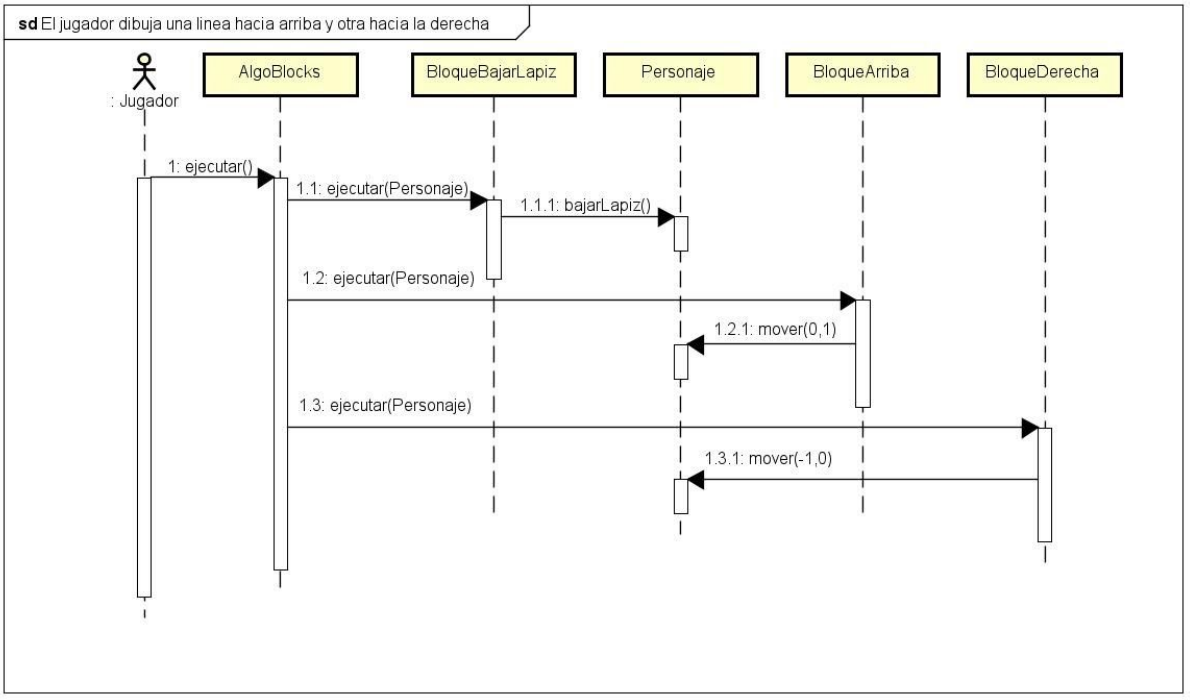

Fig. 3: Ejecución de una secuencia de comandos simples.

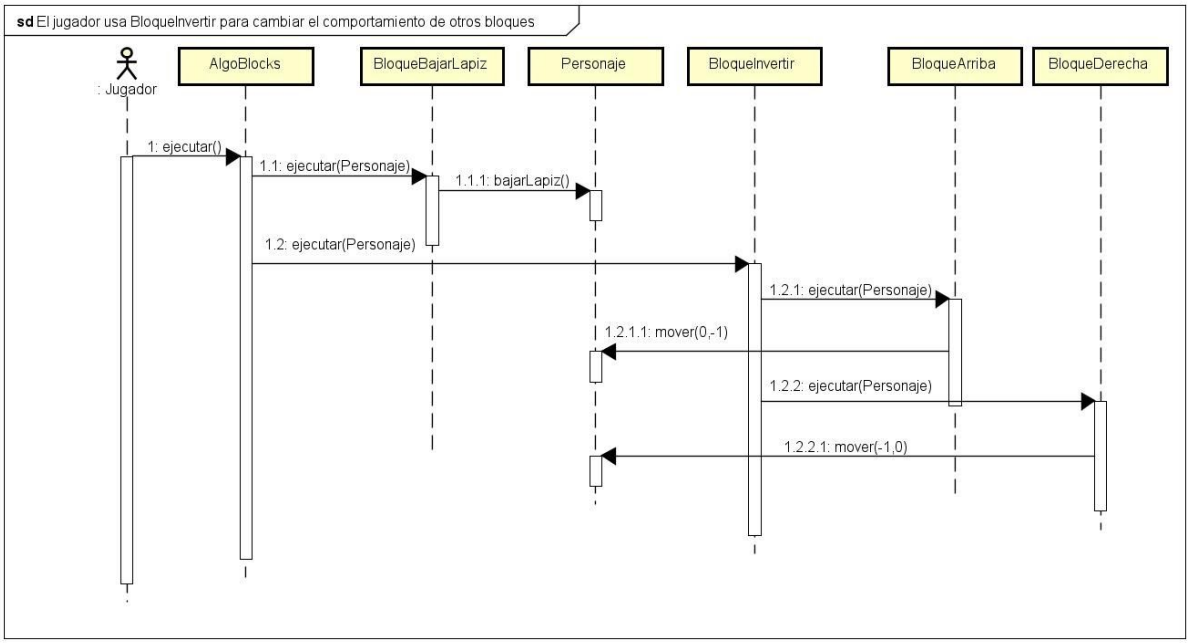

Fig. 4: Ejecución de una secuencia de comandos simples dentro de un ciclo.

Diagrama de paquetes

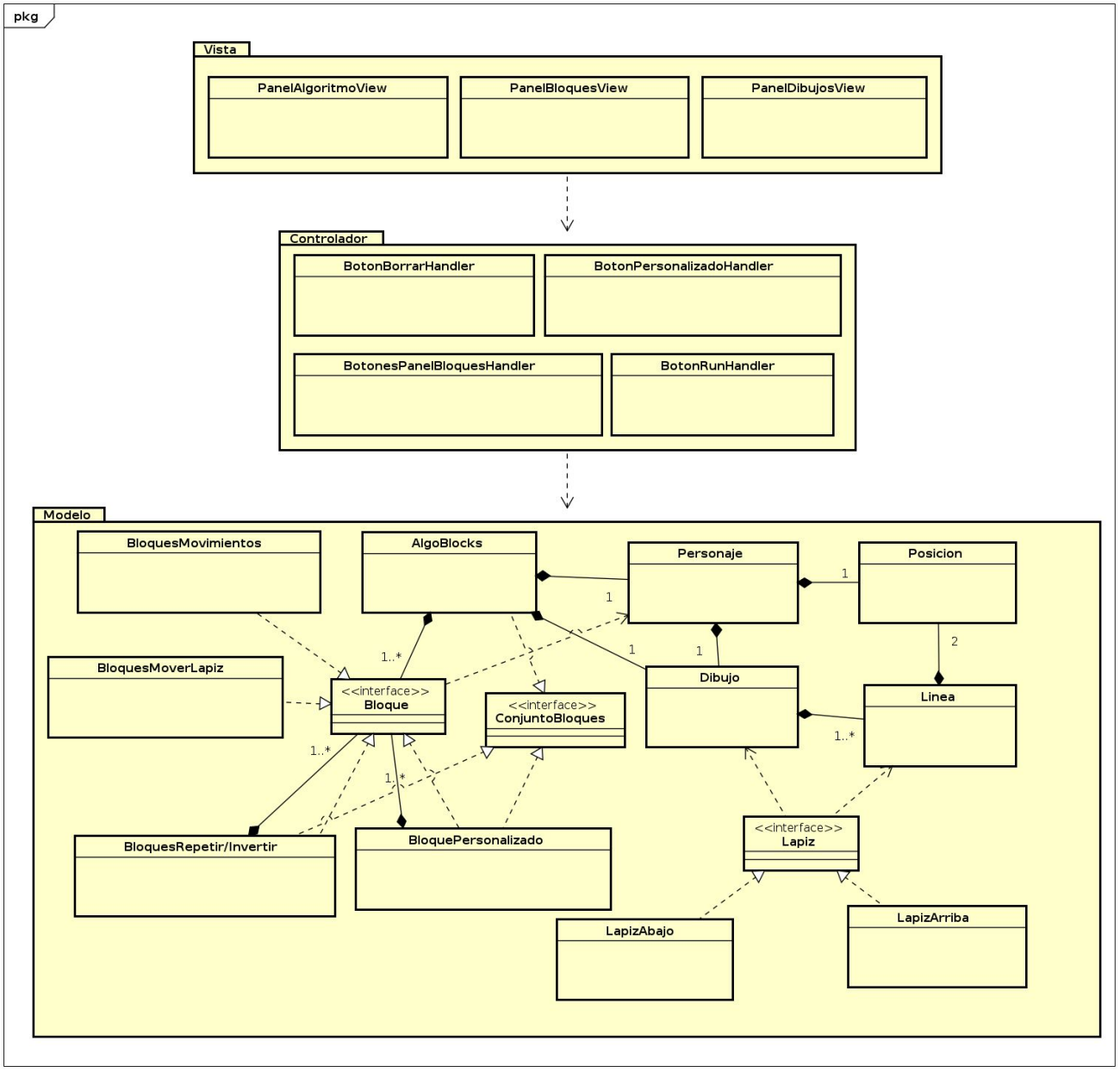

Fig. 5: Diagrama de paquetes del proyecto.

Diagramas de estado

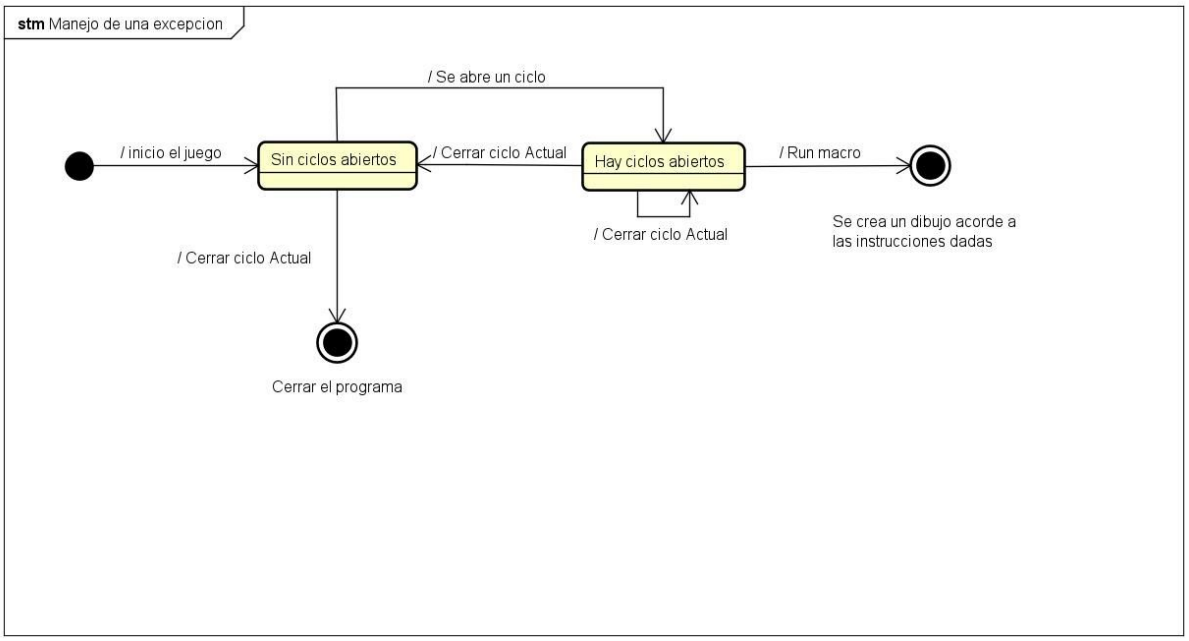

Fig. 6: Diagrama de estado del algoritmo a la hora de manejar una excepción.

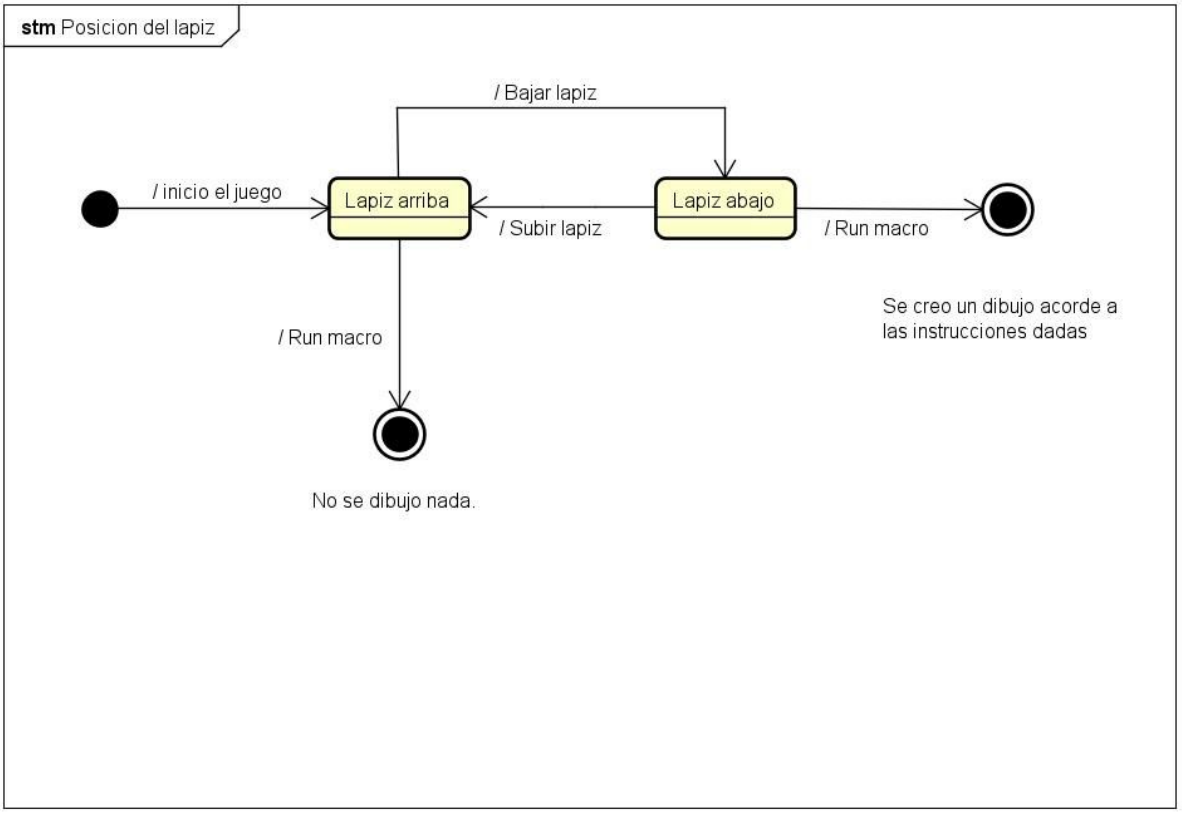

Fig. 7: Diagrama de estado del algoritmo a la hora de decidir si el dibujo se muestra en la pantalla o no.

Excepciones

Para implementar visualmente la posibilidad de anidar bloques, creamos el bloque "Cerrar Ciclo", que elimina el último elemento de la pila que utilizamos como function stack. Pero ahí nos enfrentamos a un problema: ¿qué pasa si el usuario agrega al algoritmo un bloque Cerrar Ciclo pero no había ninguno abierto?

Para resolver esto utilizamos la excepción EmptyStackException que ya está incluida dentro de las excepciones que vienen en el lenguaje Java, ya que no era necesario crear una nueva excepción que se comportara exactamente igual a la ya existente.

Esta excepción la lanza el método cerrarCiclo del paquete Vista cuando se quiere quitar un elemento de una pila que está vacía.

Se atrapa esta excepción en el paquete Controller dentro de la clase BotonesPanelBloquesHandler en el momento en el que se presiona el botón de cerrar el ciclo, mediante un bloque try-catch en el que se maneja la misma mediante el método manejar excepción.

Este método, que se encuentra en Panel Algoritmo View dentro del paquete de vista se encarga de mostrar una ventana con una advertencia de que se está intentando cerrar un ciclo que nunca se abrió y por lo tanto se cerrará el programa

Otra posible excepción que consideramos es intentar correr una secuencia vacía, guardar un algoritmo personalizado o correr dicho algoritmo personalizado sin haberlo inicializado. Pero optamos porque el programa no realice nada en vez de lanzar una excepción, ya que las excepciones se utilizan para elevar un posible error del programa que no es controlado por el flujo del mismo, sino debido a algún error o violacion del contrato por parte del usuario.
