Trabajo Practico 2

Algoritmos y programación 3

2c-2020

Nombre	padrón	Mail
Tomas Ayala	105336	tayala@fi.uba.ar
Mateo Liberini	104867	mliberini@fi.uba.ar
Nazareno Taibo	104909	ngtaibo@fi.uba.ar

Introducción

El presente informe documenta la solución del segundo trabajo práctico de la materia Algoritmos y Programación III, que consiste en desarrollar un video juego el cual le permite al usuario crear dibujos mediante bloques dentro de una secuencia, el cual muestra en cierta forma como es el funcionamiento de algoritmos con bloques. Haciendo hincapié en la metodología TDD y patrones de Diseño

Supuestos

No hubieron demasiados supuestos ya que la gran mayoría de las cosas estaban predeterminadas. El único supuesto fue que el personaje no tendría coordenadas fuera de la pantalla ya que el usuario no tendría forma de ver lo que sucede.

Excepciones

Lo que nosotros decidimos fue que el personaje solo tendrá coordenadas dentro del tablero, por lo tanto, este no se podrá mover más allá del 2 y -2 sea en el eje X o Y, ya que no se podrá visualizar. También consideramos que era necesario vaciar la secuencia una vez ejecutada para evitar que ciertas instrucciones se ejecuten más de una y confunda al usuario.

Modelo

El modelo consiste en 8 clases principales, de las cuales bloque tiene muchas clases hijas y lápiz tiene clases de estado. Las clases son:

-Bloque: esta es la clase mas usada en todo el trabajo, ya que todo lo relacionado al movimiento del personaje usa esta. Sus clases hijas son, BloqueMovimiento, la cual mueve al

personaje a la posición deseada. BloqueSecuencial, la cual tiene dos clases hijas, BloqueRepeticion y BloqueInversion, estas guardan mas bloques de movimiento, los cuales se ejecutan repetidas veces o de manera inversa, otra clase hija similar a esta es la clase BloqueSecuenciaGuardada, la cual guarda un algoritmo como variable el cual se ejecuta siempre que la llamen. Y por último BloquesLapiz la cual se encargan de levantar y bajar el lápiz para que este dibuje o no.

-Coordenada: Esta clase guarda la posición del personaje dentro del mapa y es utilizada por el lápiz para dibujar líneas.

-Lapiz: Esta clase contiene al lápiz y tiene una referencia a sus clases estados. Los cuales aclaran si el lápiz debe dibujar o no.

-Linea: Esta clase indica las líneas que se dibujaron en el mapa, solo es usada por SectorDibujo.

-SectorDibujo: es la encargada de guardar las líneas que fueron dibujadas e indicarle al lápiz como dibujar.

-Personaje: se encarga de mover al lápiz por el mapa para que este dibuje. También se encarga de mover a su imagen por el mapa.

-dirección: Esta es utilizada por bloque la cual la use para indicarle al personaje en qué dirección debe moverse, o si se debe mover en dirección opuesta.

-TableroDeDibujo: esta es la encargada de comunicarse con la parte visual del juego. Recibe todas las instrucciones y se las comunica al resto del programa.

Vista

Para poder tener una interfaz grafica prolija y funcional, lo que nosotros hicimos fue implementar una clase Stage proporcionada por javafx a la cual le añadimos otras 3 clases

para tener una mejor distribución de los elementos visibles creando menús, una lista y un mapa.

Los menús fueron, una clase VBox para tener un menú de los botones estándar (bajar/subir lápiz, repetir, invertir y mover) la cual se encuentra a la derecha. Dos HBox una que se encuentra arriba la cual es usada como un margen y otra abajo la cual se puede modificar mientras avanza el juego y es otro menú. Esta, solo contiene el botón de Ejecutar Algoritmo y Guardar Algoritmo, pero a medida que se le guarden algoritmos se agregaran aquí. En el caso que se decida guardar el algoritmo, se mostrará una nueva ventana por pantalla, la cual permitirá ingresar un nombre para crear un nuevo botón.

Para lista se usó la clase ListView a la cual se le mandan Strings y los muestra por pantalla para que el usuario sepa cuales son los próximos bloques por ejecutar. Una vez que se haya ejecutado el algoritmo, esta se limpiara para que se vean los nuevos bloques que se le asignen.

El mapa consiste en un GridPane, esta clase nos pareció la mas conveniente ya que la idea del juego es que se ejecute un bloque a la vez, por lo tanto, aplicar movimientos resulta más fácil ya que este tiene coordenadas para cada una de sus celdas. En nuestro caso estas coordenadas se nos hacen convenientes por que utilizamos sprites para mostrar el personaje y su dibujo.

Los sprites que utilizamos son un lápiz y 4 líneas las cuales ocupan media celda de cada coordenada. Estos nos parecieron mas convenientes, no solo porque fuesen más estéticos sino porque tienen la gran ventaja de que sus fondos son “invisibles” los cual permite que, aunque las líneas se crucen entre sí, estas no tapen a las otras. Lo mismo sucede con el lápiz.

Para acomodar todo de manera correcta se implementaron configuradores. Estos reciben la clase que deben configurar y en algunos casos también el tablero como otros

componentes de la Stage. La función de estos es muy simple, ya que se encargan de agregar los botones/grilla/lista en los componentes, además de asignarle el funcionamiento a los botones.

Controlador

En controlador se implementaron las funciones que se encargan del manejo de los botones. Estas son las encargadas de mandar la información a las distintas clases del modelo.

Todas estas tienen una referencia al tablero de algoritmos para que este agregue bloques a la secuencia, a la lista de bloques secuenciales, saque bloques de lista y los agregue a la secuencia, guarde bloques y que ejecute la secuencia establecida.

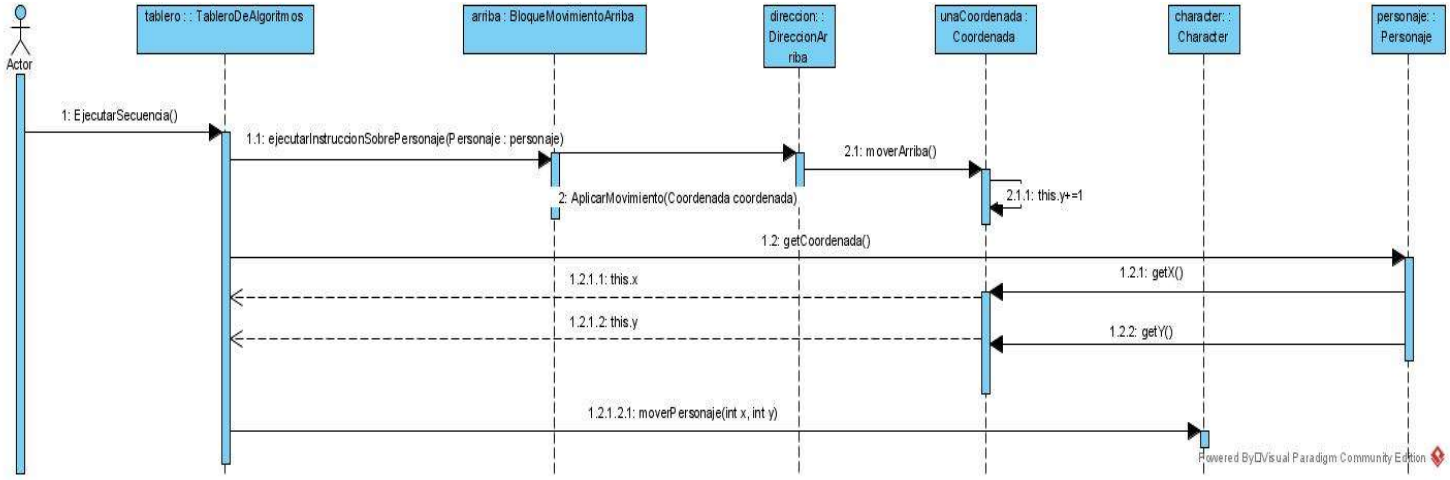
Estas también se encargan de modificar el estado del lápiz para que luego se pueda modificar de manera correcta el mapa del juego, dibujando o no las líneas correspondientes. Se comunicará con el estado del lápiz y sus coordenadas viejas y nuevas, con las cuales se fijará hacia donde fue el movimiento para poner ciertos ImageView en las celdas correspondientes.

En el caso del botón de guardado, su función se encargará de tener dos handlers más, en el caso de que se quiera agregar un nuevo botón o cancelar la acción.

También la gran mayoría tiene una referencia a la ListView para agregarle los bloques a ejecutar por pantalla.

Diagramas de Secuencia

Este primer diagrama solo muestra el Código que se ejecuta una vez que el usuario decidió mover al usuario una vez hacia arriba y ejecuto el algoritmo



Este segundo diagrama muestra como seria el proceso cuando el usuario agrega un bloque bajar lápiz y un bloque subir lápiz. La segunda imagen es el mismo diagrama un poco mas cerca y centrado en la mitad derecha.

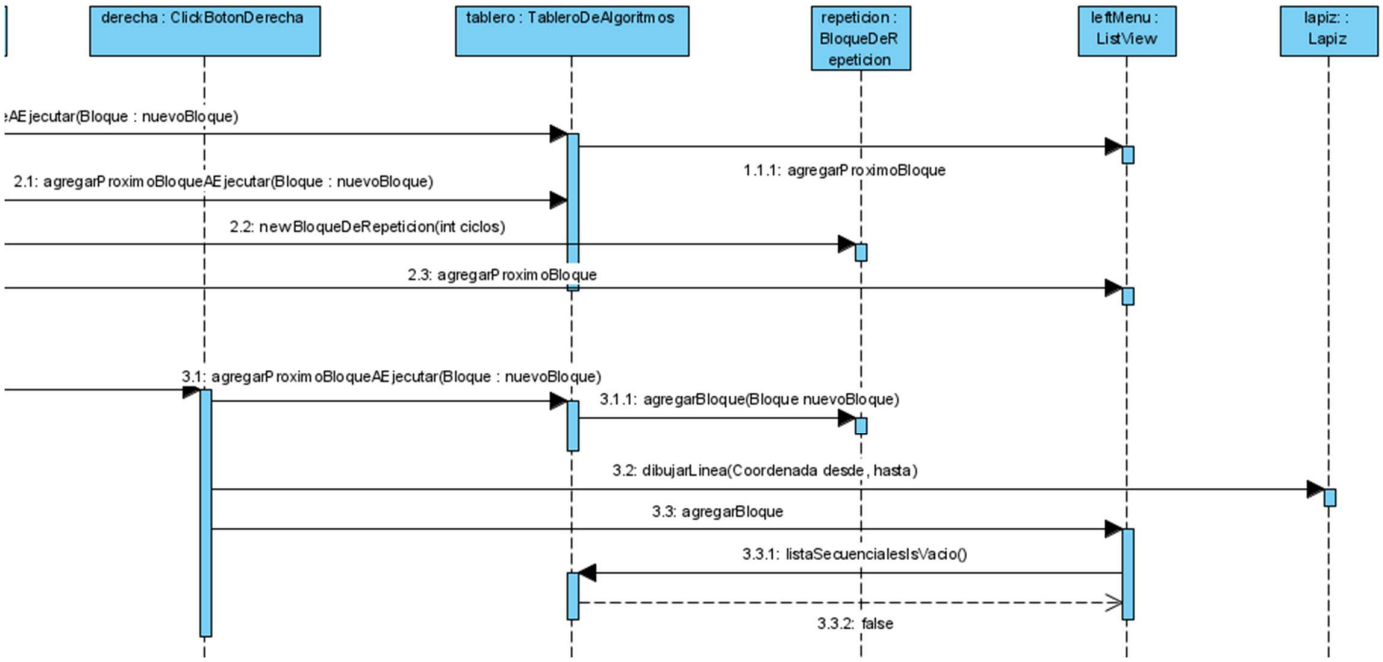
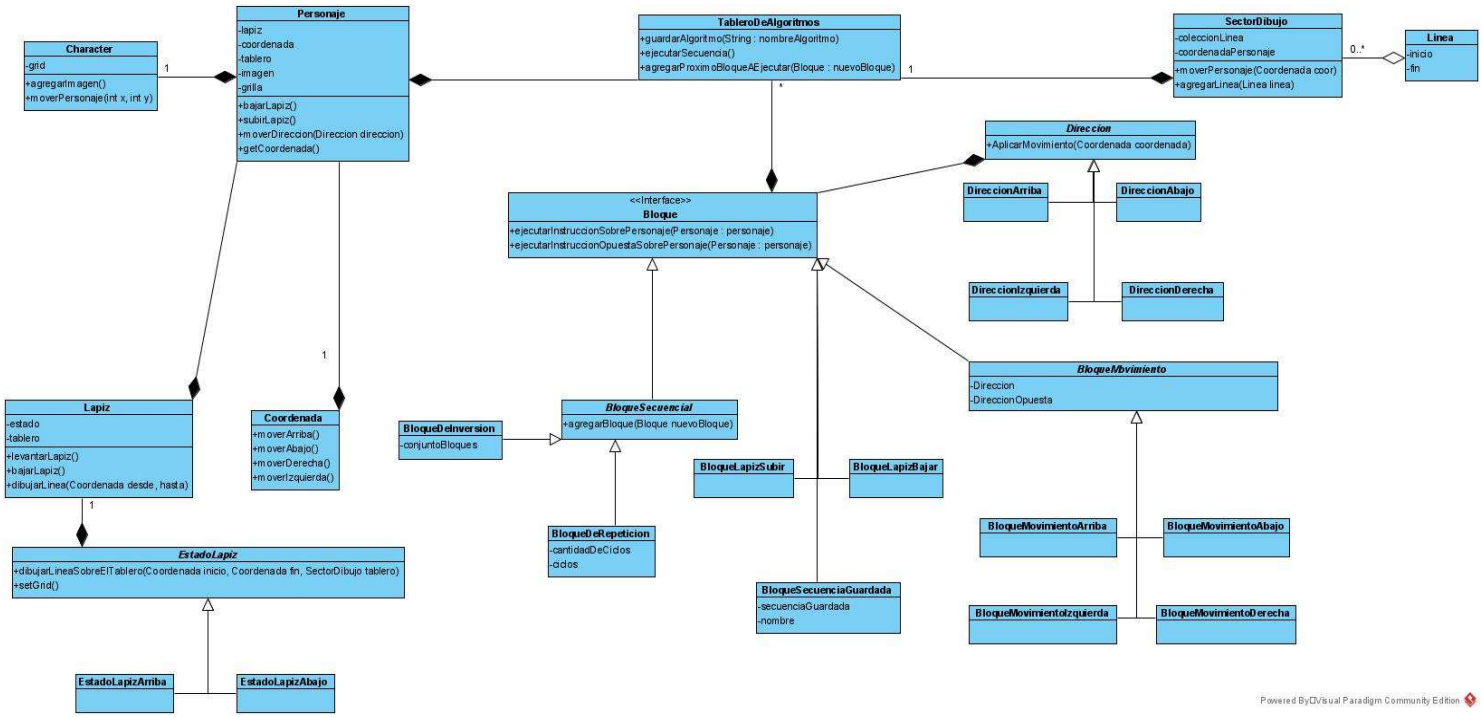
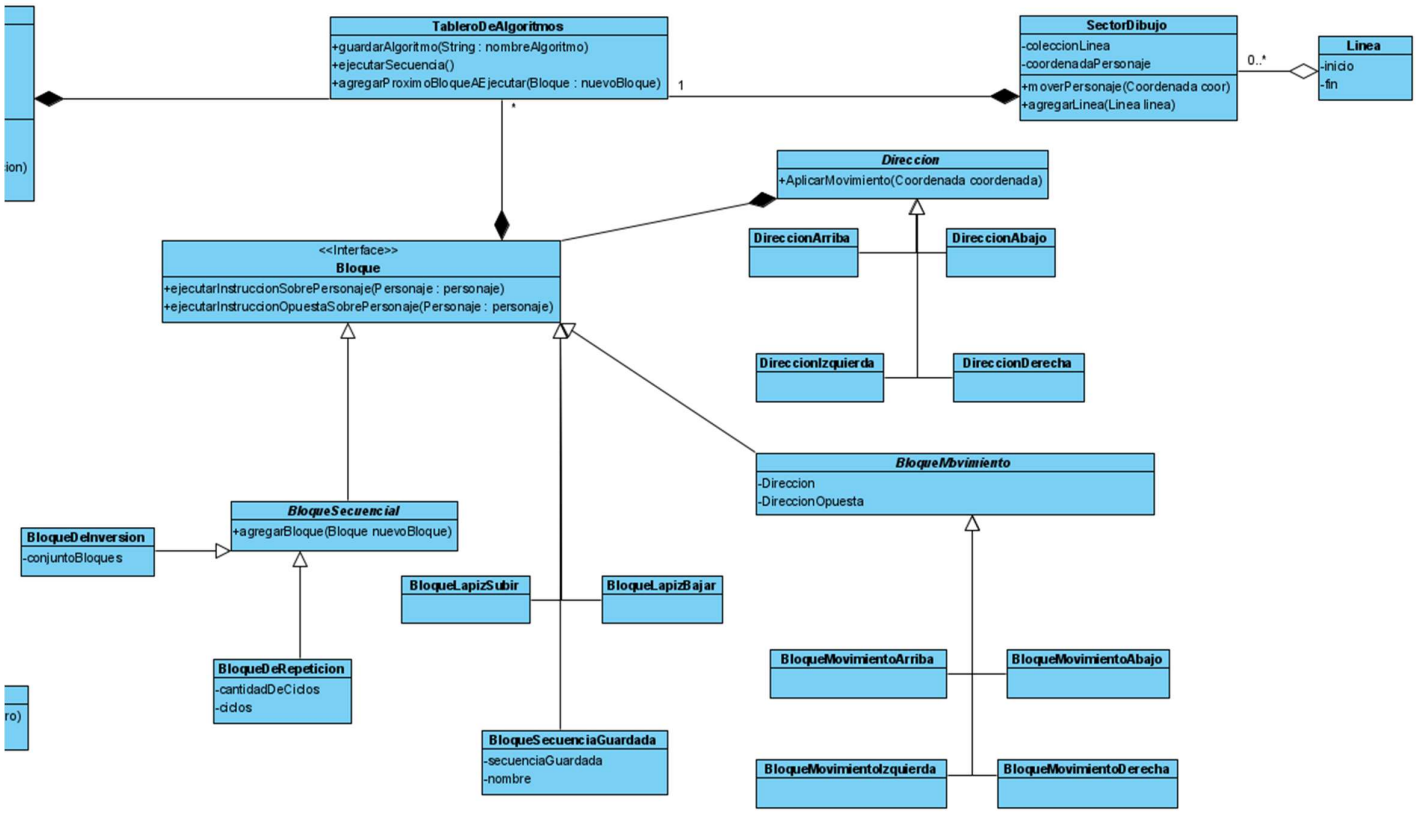


Diagrama De Clases



Los dos siguientes imágenes don de este mismo diagrama pero más acercado, ya que sino no seria comprensible.



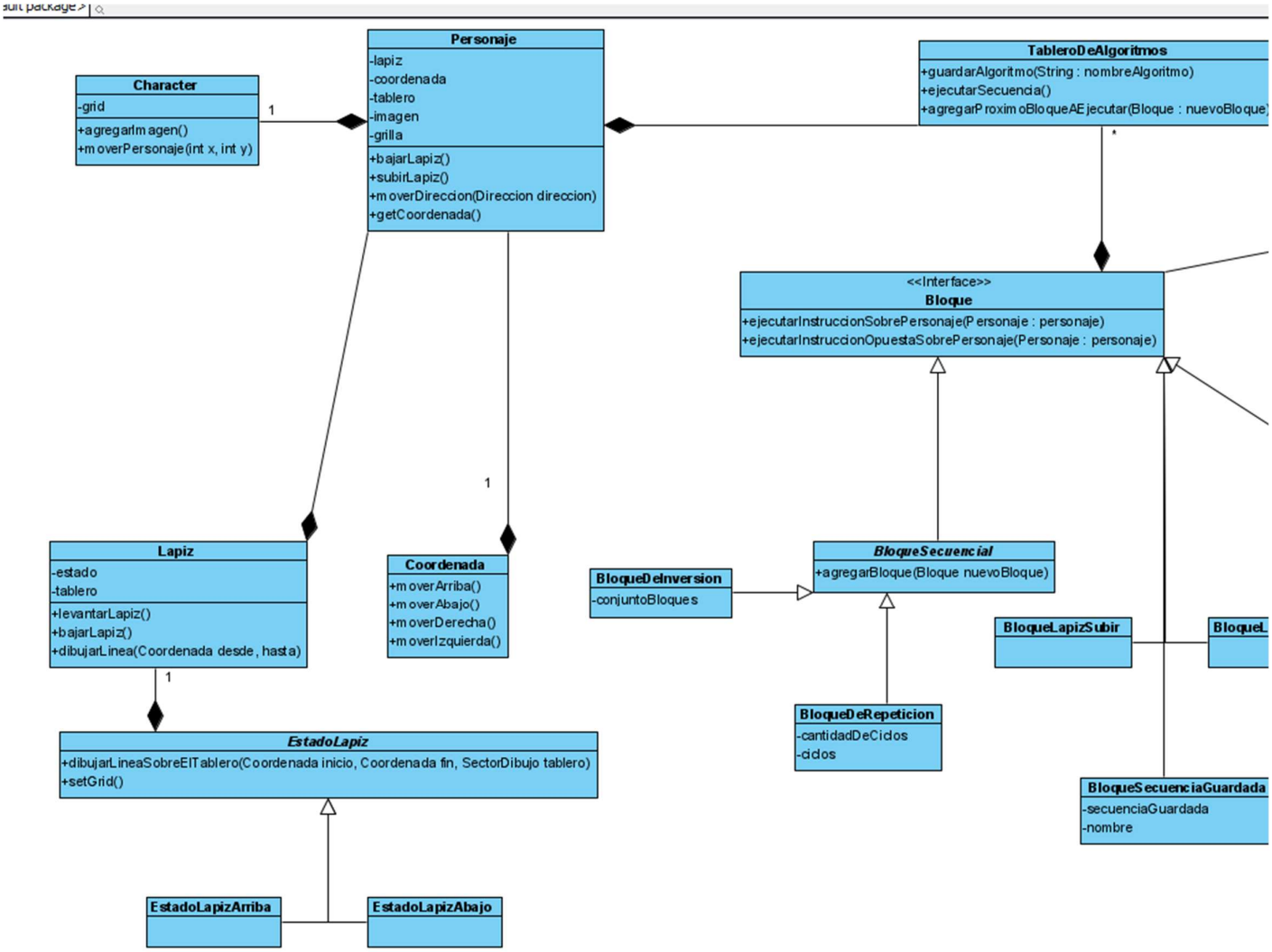

Character
-grid
+agregarImagen()
+moverPersonaje(int x, int y)
1
Personaje
-lapiz
-coordenada
-tablero
-imagen
-grilla
+bajarLapiz()
+subirLapiz()
+moverDireccion(Direccion direccion)
+getCoordenada()
TableroDeAlgoritmos
+guardarAlgoritmo(String : nombreAlgoritmo)
+ejecutarSecuencia()
+agregarProximoBloqueAEjecutar(Bloque : nuevoBloque)
*
<<Interface>>
Bloque
+ejecutarInstruccionSobrePersonaje(Personaje : personaje)
+ejecutarInstruccionOpuestaSobrePersonaje(Personaje : personaje)
1
Lapiz
-estado
-tablero
+levantarLapiz()
+bajarLapiz()
+dibujarLinea(Coordenada desde, hasta)
1
Coordenada
+moverArriba()
+moverAbajo()
+moverDerecha()
+moverIzquierda()
BloqueDeInversion
-conjuntoBloques
BloqueSecuencial
+agregarBloque(Bloque nuevoBloque)
BloqueLapizSubir
BloqueL
BloqueDeRepeticion
-cantidadDeCiclos
-ciclos
BloqueSecuenciaGuardada
-secuenciaGuardada
-nombre
EstadoLapiz
+dibujarLineaSobreElTablero(Coordenada inicio, Coordenada fin, SectorDibujo tablero)
+setGrid()
EstadoLapizArriba
EstadoLapizAbajo

Diagrama de Paquetes

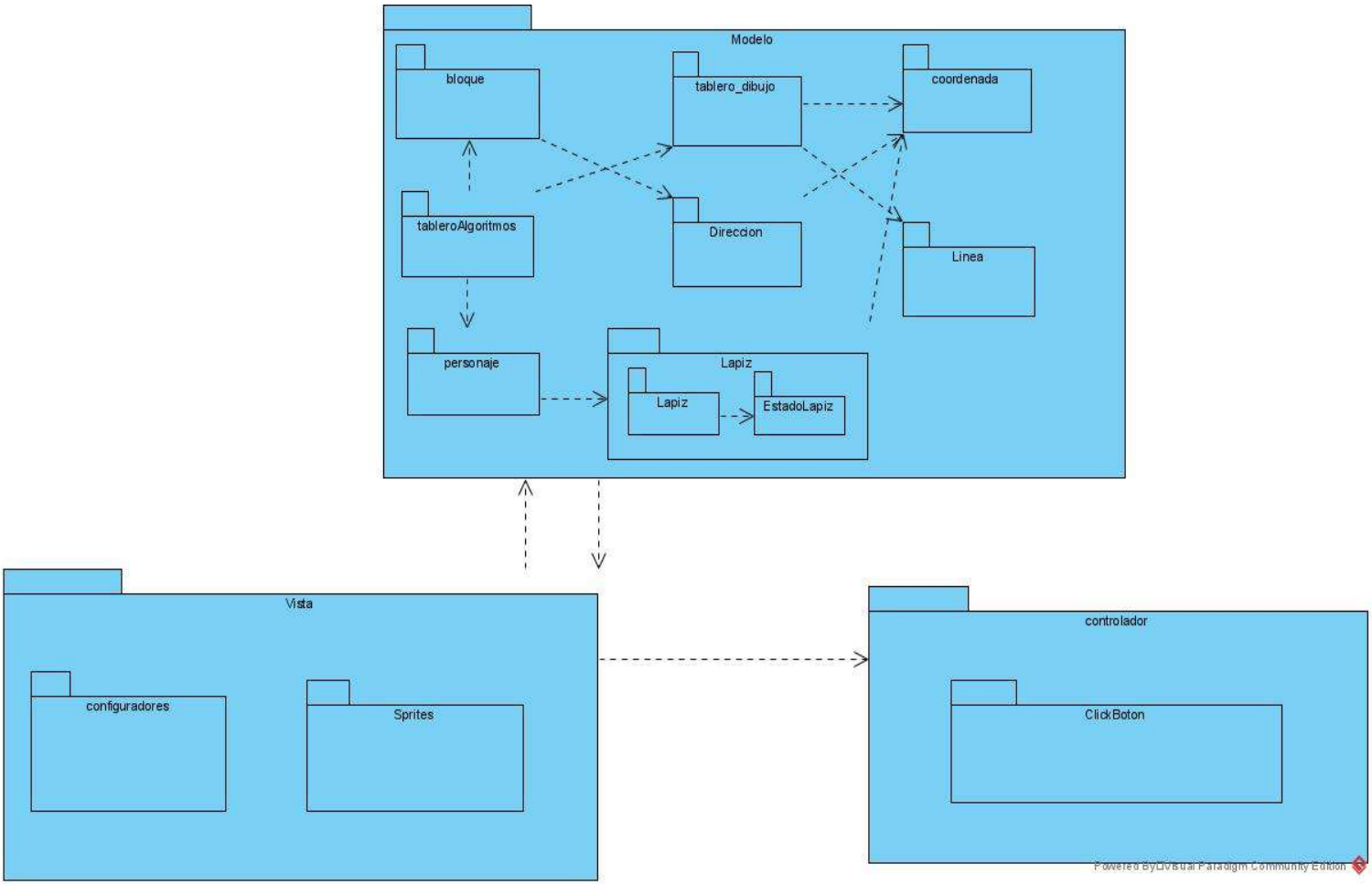
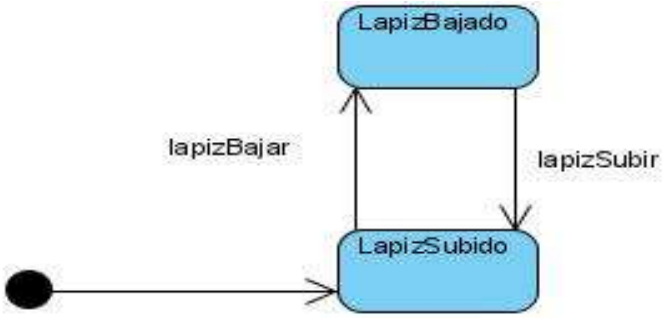
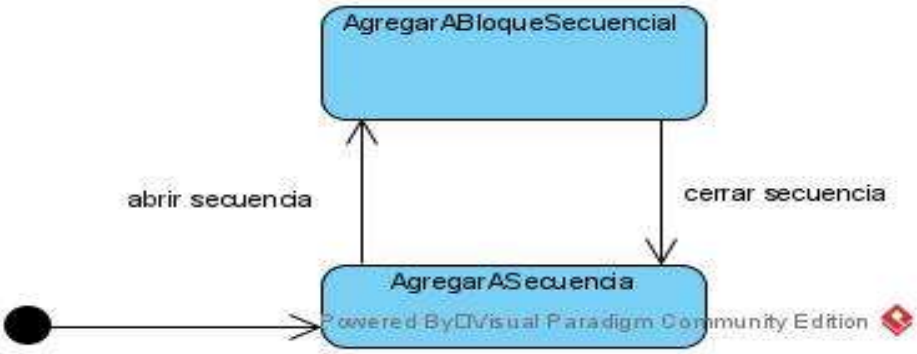


Diagrama de estados





Patrones

Los patrones que implementamos para este código fueron el patrón singleton y MVC, este primero se utilizó ya que muchas de las clases eran únicas y para que sea mas simple acceder a ellas se utilizo este patrón. El patrón MVC lo usamos para separar mejor la parte visual de la parte lógica, así evitar que el usuario interactúa demasiado con la parte lógica.

También se utilizo el patrón de fachada, que fue implementado por la misma razón que el MVC, para esto se utilizó el tablero de algoritmos el cual recibía ordenes y se la comunicaba al resto del código.

Y por ultimo se utilizo el patrón state para aclarar al lápiz si este debe dibujar o no, dándole dos estados, arriba o abajo.