Facultad de Ciencias Exactas, Físicas y Naturales

Ingeniería de Software - 2020

ARQUITECTURA Y DISEÑO DEL SISTEMA

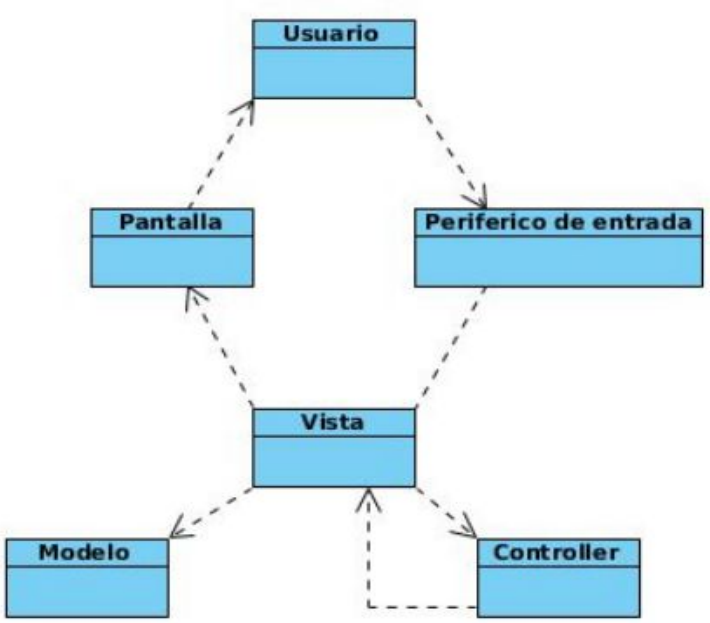
Grupo CodeRush

Integrantes:

- Molina Franco Elias.
- Nievas Boso Leonel Federico.
- Orionte Andrés.
- Palacios Lucero Matías Iván.

Arquitectura del Sistema

La arquitectura general, a nivel de aplicación, es la siguiente:



El patrón de arquitectura usado es MVC (Model-View-Controller) debido a que cuenta con algunos beneficios útiles para nuestro proyecto. Uno de los más notables es que divide el código en tres aspectos: el controlador, el modelo y la interfaz gráfica o vista:

- **Modelo**: Se encarga de presentarle la información de la aplicación al usuario, tiene todos los datos, estados y lógica de la aplicación. Define donde se encuentran los objetos, y el estado de los mismos.
- **Controlador**: Es el cerebro de ésta arquitectura. Actúa como vínculo entre el usuario y la aplicación, ya que recibe las entradas del usuario a través de la vista y las procesa en función del Modelo.
- **Vista**: Define y gestiona como se presentan los datos al usuario, a través de la interfaz gráfica.

Esta modularidad es la que nos permite modificar partes del software sin grandes consecuencias en el resto del proyecto, ya que al tener los códigos por separado, podemos trabajar sobre uno sin necesidad de modificar el resto, lo que nos beneficia a la hora de mantener el código, arreglar bugs o modificar funcionalidades.

Diagrama de Despliegue

El diagrama de despliegue, se encuentra dentro de la familia de diagramas estructurales y describe un aspecto del sistema en sí. En este caso, el diagrama de despliegue describe el despliegue físico de información generada por el programa de software en los componentes de hardware y también del software. La información que se ha generado por el software se llama artefacto. Las cajas tridimensionales, conocidas como nodos, representan los elementos básicos de software o hardware en el sistema.

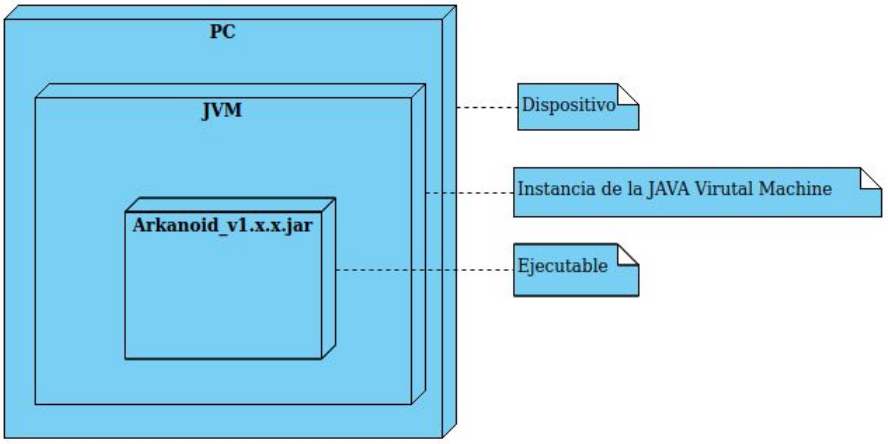
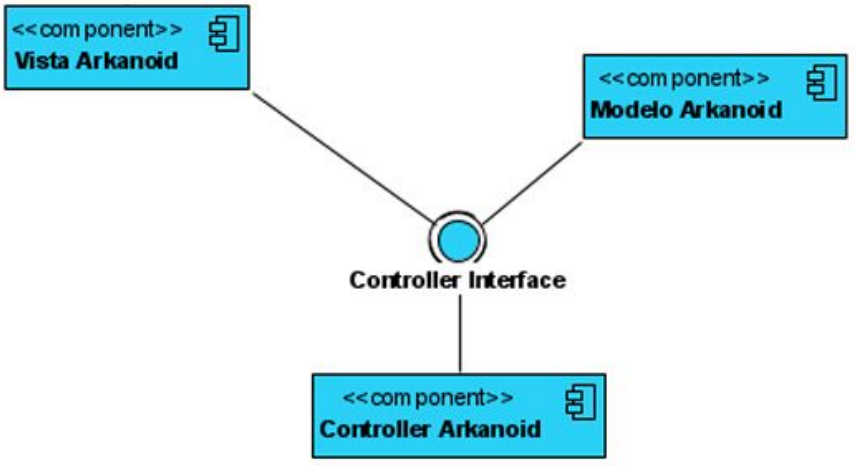


Diagrama de componentes

El diagrama de componentes representa cómo un sistema de software es dividido en sus componentes y muestra las dependencias que hay entre estos.
Los diagramas de Componentes prevalecen en el campo de la arquitectura de software pero pueden ser usados para modelar y documentar cualquier arquitectura de sistema. Este proyecto es representado de la siguiente manera:



Casos de prueba de integración

El objetivo de estas pruebas es verificar que los subsistemas funcionan correctamente y son capaces de interactuar unos con otros a través de las interfaces definidas.

PR-IN-01
Para corroborar el correcto funcionamiento entre los módulos del sistema en conjunto - Model, View y Controller -, se probó iniciar una partida ya que se requiere la intervención e interacción de dichos módulos y las respectivas clases que los representan - Arkanoid, View, y Modelos y Controladores de Ball, Paddle, Brick, ScoreBoard and CrunchSound. El resultado fue una partida generada correctamente, con sus respectivos componentes y sus correctas ejecuciones.

Diagrama de paquetes

Este diagrama representa las dependencias entre los paquetes que componen un modelo, muestra cómo un sistema está dividido en agrupaciones lógicas y las dependencias entre dichas agrupaciones.Como normalmente un paquete está pensado como un directorio, los diagramas de paquetes aportan una descomposición de la jerarquía lógica de un sistema. Los paquetes están organizados para maximizar la coherencia interna dentro de cada uno y minimizar el acoplamiento externo entre ellos. En nuestro proyecto, teniendo en cuenta lo desarrollado al momento de realizar este informe el diagrama de paquetes es el siguiente:

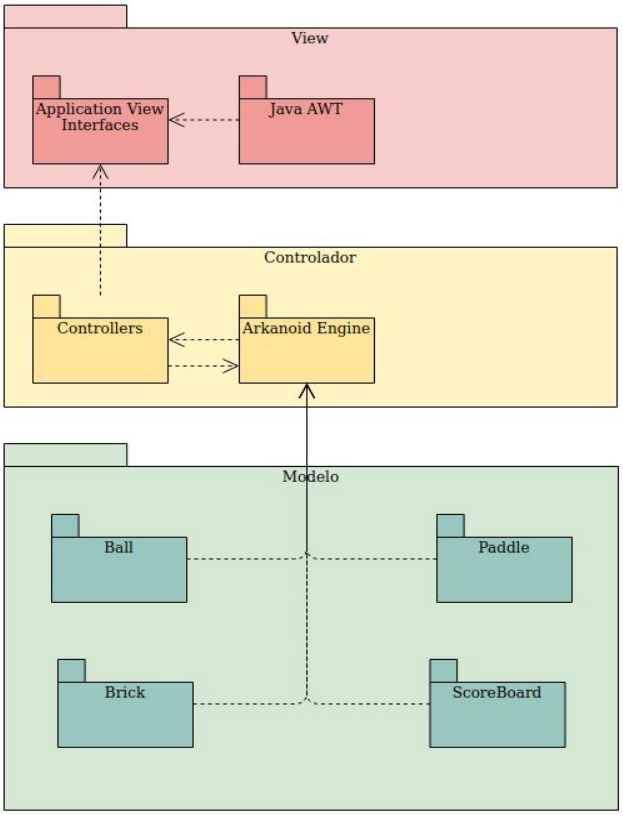
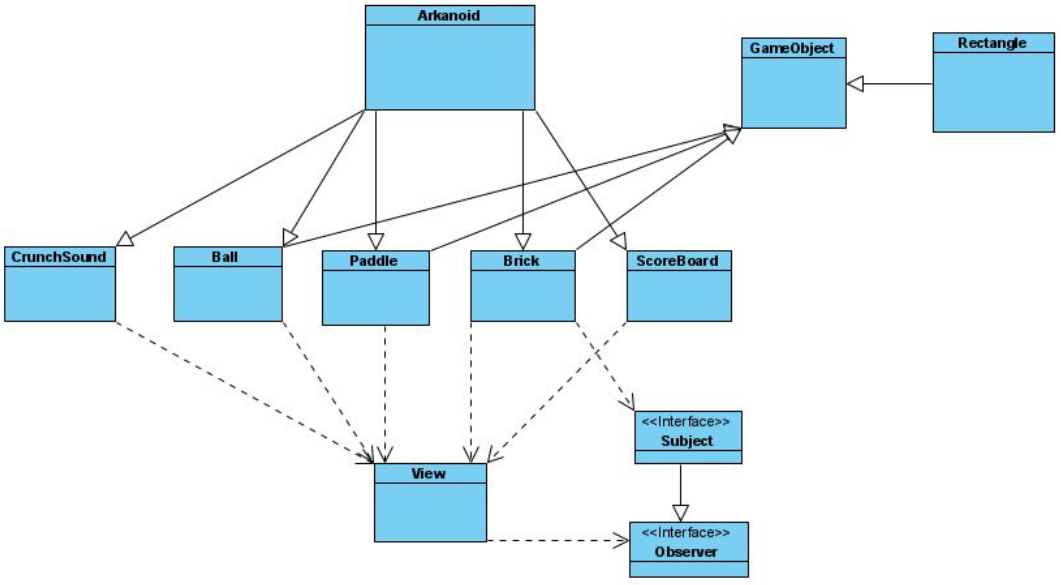


Diagrama de clases

Diagrama de clases de los recursos usados en el modelo:

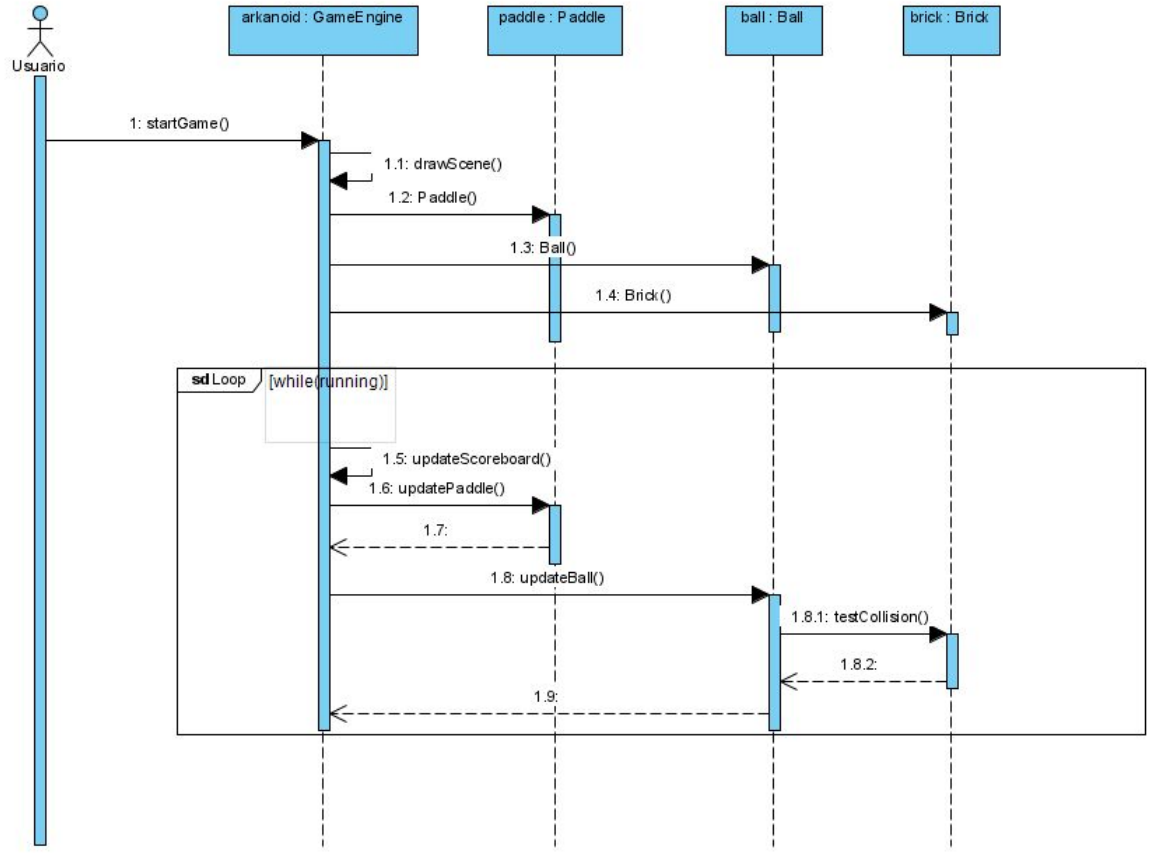


Al usar el patrón de arquitectura Model-View-Controller, se utilizaron los patrones de diseño Observer y Strategy. El patrón Observer, se utiliza para cuando el modelo cambia alguno de sus valores observados, éste notifique a sus observadores para que actualicen sus

valores. En nuestro caso usamos como Subject a Brick que notificará a los observadores ScoreBoard y CrunchSound el cambio de estado de un Brick mediante su ruptura.

Diagrama de secuencia

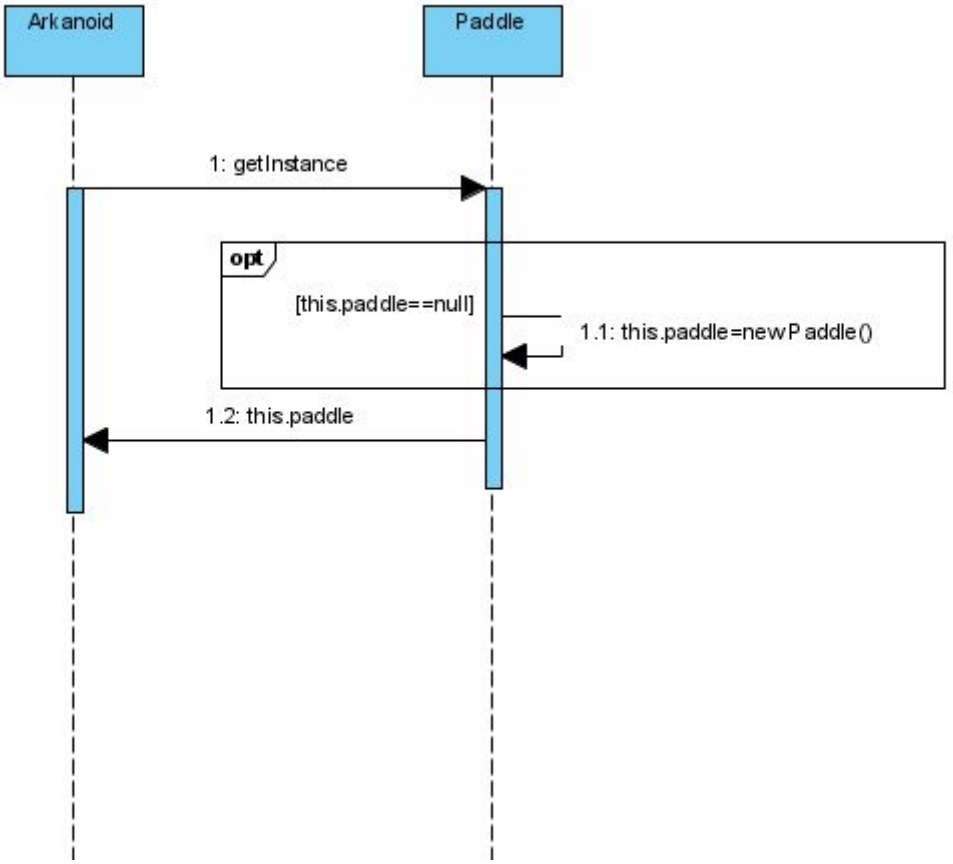
A continuación se detalla mediante un diagrama de secuencia el inicio y desarrollo de una partida.



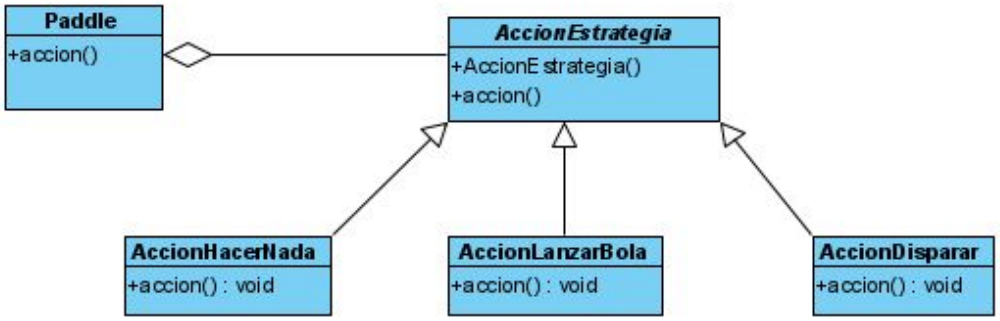
pruebas unitarias automáticas para el código (cómo correrlas y verificar su estado)

Aplicación Patrones de diseño

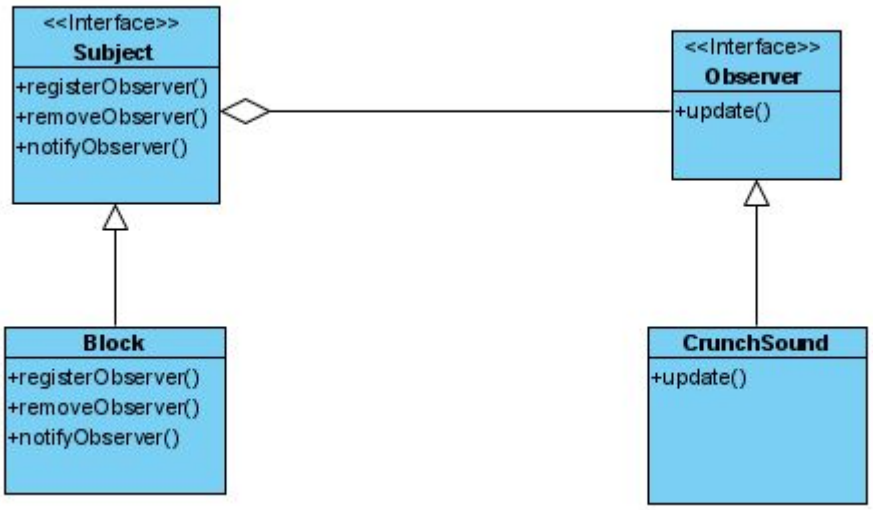
Singleton: en la siguiente imagen se muestra un diagrama de secuencia que muestra la creación de la plataforma (o paddle). Es de interés tener activa únicamente una instancia de esta clase, es por eso el uso de este patrón de diseño.



Strategy: Se utiliza este patrón porque se espera que la plataforma haga distintas cosas dependiendo de cómo esté el juego en el momento. Por ejemplo: al inicio del nivel vamos a querer elegir por dónde empezar a jugar, al pulsar el botón de acción se lanza la bola desde una posición elegida.



Observer: El patrón observer es implementado en nuestro código a través del sujeto, la clase Block, y el observador que es la clase CrunchSound. La finalidad de dicha implementación es el de notificar a CrunchSound cuando un bloque haya sido destruido, para que de esta manera reproduzca un efecto de sonido.



Unit Test

Clase Controller -- ControllerTest.java

@Test
public void testkeyPressed() {}
Descripción: Se prueba si una tecla ha sido presionada.
@Test
public void testkeyReleased() {}
Descripción: Se prueba el correcto movimiento dado las teclas LEFT and RIGHT.
@Test
public void testrun(){}
Descripción: Se prueba la correcta inicialización del nivel.

Clase Block -- BlockTest.java

@Test
public void left() {}
Descripción: Se comprueba el límite izquierdo del ladrillo.
@Test
public void right() {}
Descripción: Se comprueba el límite derecho del ladrillo.
@Test
public void top() {}
Descripción: Comprueba el límite superior del ladrillo.
@Test
public void bottom() {}
Descripción: Comprueba el límite inferior del ladrillo.

Clase Paddle -- BallTest.java

@Test
public void left() {}
Descripción: Comprueba el límite izquierdo de la bola.
@Test
public void right() {}
Descripción: Comprueba el límite derecho de la bola.
@Test
public void top() {}
Descripción: Comprueba el límite superior de la bola.
@Test
public void bottom() {}
Descripción: Comprueba el límite inferior de la bola.