Scrum Game

Introducción al diseño e implementación

1 de Septiembre de 2020

Versión 0.1

Versión 0.1

Índice

Versión 0.1

Introducción

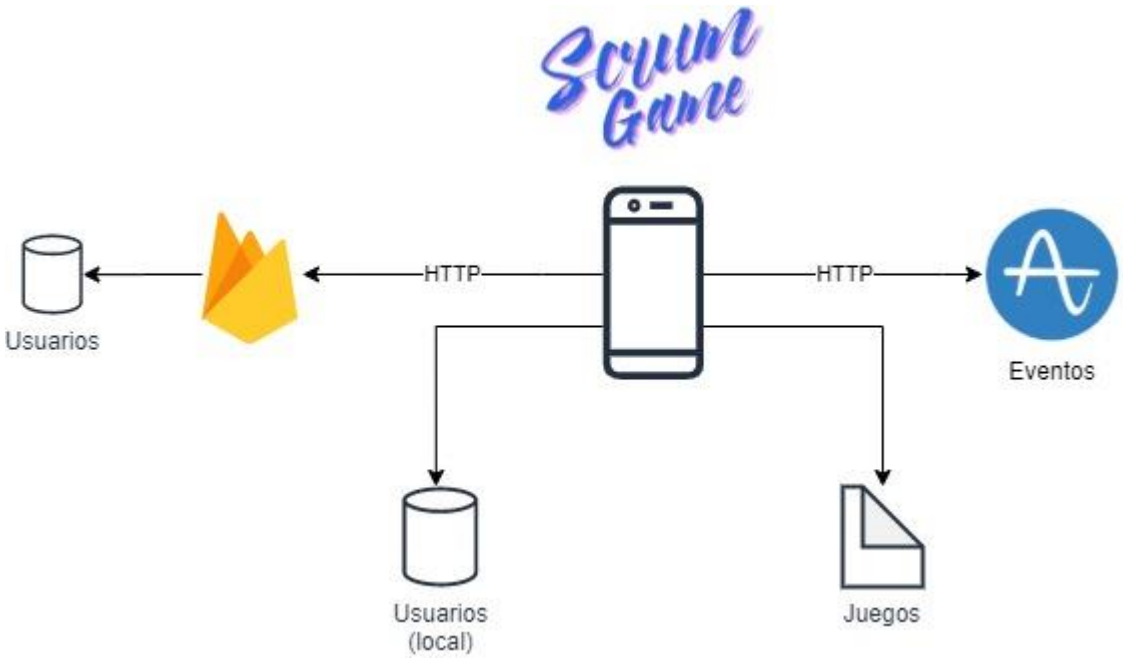
El presente documento busca explicar a modo general la arquitectura del juego Scrum Game, sus principales componentes y algunos flujos presentes en la aplicación.

Para ello, se describirán la arquitectura elegida, los patrones aplicados, y las tecnologías utilizadas.

Scrum Game se trata de un juego escrito en el lenguaje de programación Java. Incluye también a *Velocity*, un juego desarrollado en Unity. Cabe destacar que el desarrollo de ambos juegos se basó mayormente en el juego programado originalmente en Swift para iOS.

Arquitectura

Diagrama de arquitectura conceptual



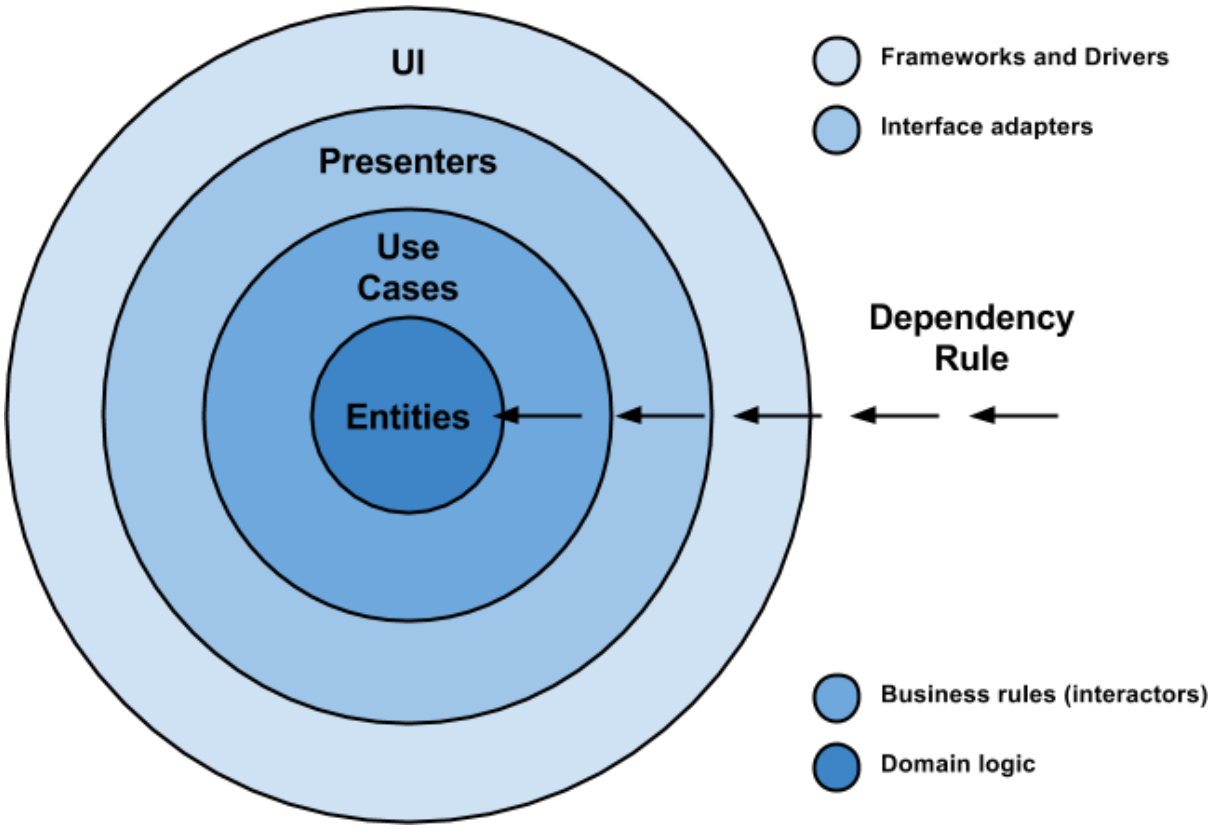
Scrum Game es una aplicación móvil concebida para ser utilizada en modo online, aunque puede adaptarse para poder jugar los distintos juegos de manera offline. Scrum Game almacena localmente el progreso del usuario en una base de datos local Realm y sincroniza el progreso con Firestone, donde también se almacenan los usuarios y su progreso.

También se envían eventos o logs a Amplitude, donde se puede consultar un dashboard con eventos que permiten analizar cómo los usuarios fueron progresando en el juego.

Por último, los juegos y su configuración están almacenados en un archivo de texto embebido en la aplicación.

Patrones arquitectónicos

La arquitectura elegida para el juego es Clean Architecture, basada en la arquitectura hexagonal.



Clean Android Architecture

Internamente la aplicación está dividida en tres capas: la capa de presentación, dominio y datos. Cada capa tiene una responsabilidad específica y es independiente de las demás.

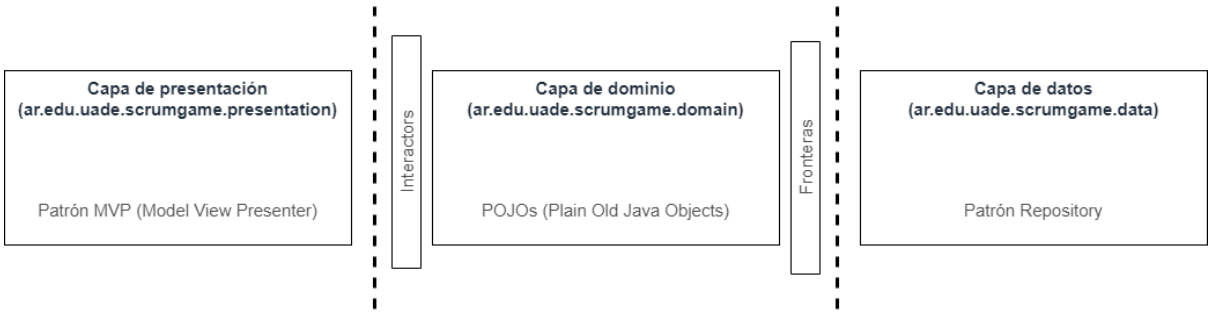
De forma análoga a la arquitectura *Clean*, la interfaz de usuario, los presentadores (que son un componente del patrón MVP), la inyección de dependencias y lógica de navegación entre pantallas se encuentran en la capa de presentación.

Por su parte, la capa de dominio engloba los modelos y la lógica de negocio, ésta última encapsulada en casos de uso.

En la capa de datos consumimos las distintas fuentes de datos que tiene el juego. Entre ellas se encuentran las API de Firestone y Amplitude para enviar eventos y sincronizar el progreso de usuario.

Para acceder a Firestone basta con ingresar al sitio https://console.firebase.google.com/project/gestorscrum (es necesario ser invitado al proyecto para poder acceder) y seleccionar en el menú la opción “Cloud Firestone”.

Similarmente, para acceder a Amplitude hay que ser miembro del proyecto e ingresar a https://analytics.amplitude.com/scrumgameuade.



Patrones implementados en cada capa

La arquitectura del juego tiene un fuerte componente reactivo, pudiéndose apreciar el uso del patrón Observer para la comunicación entre capas.

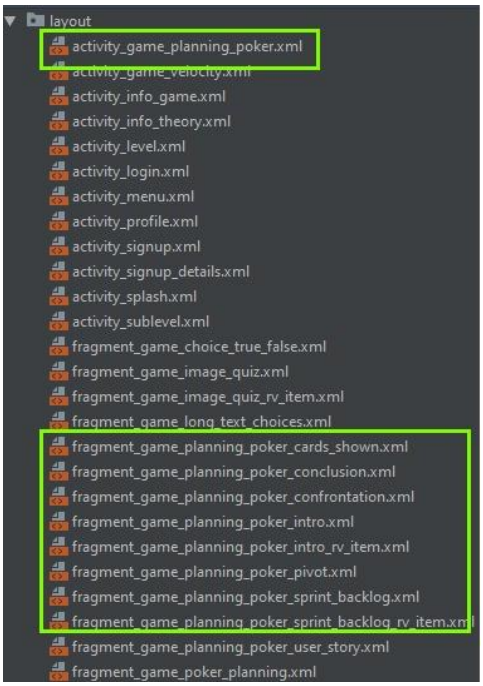
A continuación se detallan las responsabilidades clave y patrones elegidos en cada capa.

Versión 0.1

Capa de presentación

La capa de presentación sigue el patrón MVP (Model - View - Presenter). La responsabilidad de esta capa es presentar la interfaz de usuario y gestionar la interacción del usuario con la aplicación.

En general, por cada pantalla de la aplicación existe una actividad o *activity* asociada. Cada actividad tiene su archivo de diseño xml en la carpeta de recursos **layout**. Una actividad puede contener a su vez varios fragmentos o *fragments* que también tienen su archivo de diseño correspondiente.



Ejemplo de correspondencia entre actividades y fragmentos

Cada tipo de juego existente en el archivo de juegos está presentado en un fragmento que aplica la lógica correspondiente a la dinámica que éste posee. Como existen funcionalidades transversales a todos los juegos, existe un fragmento base (*GameFragment*) del cual heredan todos fragmentos de juego. Algunas de estas funcionalidades incluyen el envío de eventos de juego a Amplitude y la contabilización de tiempo transcurrido en un nivel.

Estructura de actividades y fragmentos en los juegos

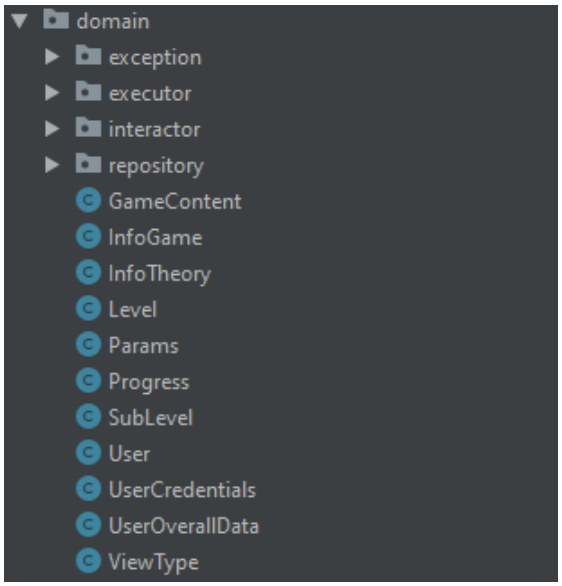
Versión 0.1

Capa de dominio

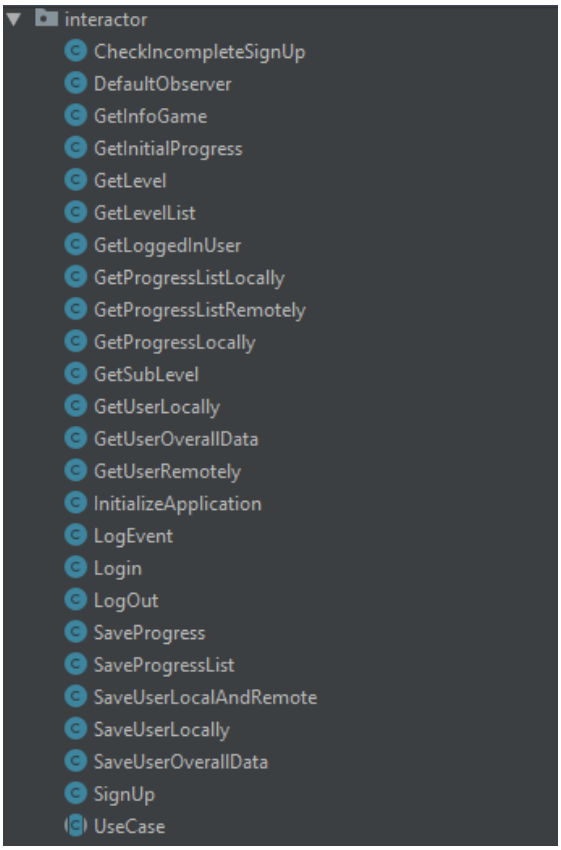
En esta capa se hacen las transformaciones necesarias de los objetos provenientes de la capa de presentación, se aplica la lógica relacionada a cada caso de uso, y en caso de corresponder se envían a la capa de datos los objetos procesados a un datasource.

Cada flujo principal de la aplicación (ejecutar un juego, iniciar sesión, guardar progreso, enviar datos a Amplitude) está encapsulado en un UseCase o caso de uso. Estos casos de uso pueden encontrarse en el paquete **interactors**.

Como su nombre lo indica, hay un archivo por caso de uso que responde a una funcionalidad específica. Por ejemplo, el caso de uso “LogEvent” se encarga de recibir la información de la capa de presentación que corresponde a un evento que se quiera informar, se procesa para enviarla a la capa de datos, y desde la capa de datos se envía a Amplitude.



Estructura de la capa de dominio en el proyecto



Casos de uso presentes en el proyecto

Capa de recursos

En esta capa se almacenan los datos de la aplicación tanto local como remotamente. Si bien es posible utilizar la aplicación de manera *offline*, actualmente es necesario que el dispositivo esté conectado a Internet para enviar el progreso de juego a Firestone y los eventos de usuario a Amplitude.

El patrón de diseño predominante en esta capa es Repository, siendo los *datasources* actuales Firestone, Amplitude, Realm como base de datos local y el archivo de niveles almacenado en los *assets* de la aplicación (db.json).

A continuación se detalla la estructura del archivo de juegos y la de los objetos que son almacenados localmente y enviados a los distintos *datasources*.

Archivo de niveles

A la fecha de redacción del presente documento, la información de cada juego es almacenada en un archivo con formato JSON correspondiente a un objeto con un listado de juegos (*"levels"*).

De esta forma, los juegos y tutoriales se pueden modificar y agregar a este archivo siguiendo la estructura correspondiente para que la aplicación pueda automáticamente leer la configuración y popular los datos en las ventanas.

La estructura de cada nivel está representada de la siguiente forma:

- *"name"*: El nombre del nivel.
- *"code":* Código del nivel.
- *"sublevels"*: Un listado con cada subnivel.

Cada subnivel está estructurado a su vez en:

- *"name":* El nombre del subnivel.
- *"code":* El código del subnivel. Está compuesto por el nivel y subnivel, siguiendo el formato [Código de nivel].[Código de subnivel].

- *"info_theory":* Es un listado de objetos que representan las páginas de los tutoriales que son presentados al usuario antes de comenzar un juego. Cada página se representa de la siguiente manera:
    - *"code"*: El código del tutorial. Cada página de un tutorial posee un código único, siguiendo el formato [Código de nivel].[Código de subnivel].[Código de tutorial].
    - *"content":* Es el contenido propio de una página del tutorial. La aplicación va a recorrer este listado y generar los componentes correspondientes (siempre que sean de un tipo reconocido. Actualmente se reconocen títulos, imágenes, vídeos de YouTube y texto, aunque es extensible a nuevos tipos).
        - *"type":* Puede ser del tipo "title","image","video" o "text".
        - *"data":* Es un texto, en caso de los tipos "image" y "video" este campo corresponde al link de la imagen o video.
- *"info_games":* Es un listado con los juegos correspondientes al subnivel. Cada juego posee los siguientes atributos:
    - *"code":* El código del juego. El formato es [Código de nivel].[Código de subnivel].[Código de juego].
    - *"title":* El nombre del juego.
    - *"type":* El tipo de juego. Actualmente la aplicación soporta los tipos de juego "textQuiz","textChoiceQuiz","trueFalseQuiz","imageQuiz", "selection" y "shortTextQuiz". Existen juegos especiales como "scrumFlowDraggable","velocity" y "pokerPlanning" que son únicos en la aplicación.
    - *"content":* Es un listado de respuestas que el usuario puede seleccionar.
        - *"type":* Tipo de respuesta.
        - *"isCorrect":* Booleano que indica si esta respuesta es correcta.
        - *"data":* Generalmente corresponde a la opción en sí, es un label que se muestra en pantalla para que el usuario pueda responder.

Firestone

La colección que es consultada y actualizada por la aplicación es **users**. Cada documento de la colección corresponde a un usuario que es identificado mediante el mail registrado.

Al momento de registrar un usuario se envían los siguientes datos:

- age (edad).
- city (ciudad).
- country (país).
- gameTasteLevel (interés por los juegos, que puede ser Nada, Poco, Le da igual, Mucho o Fanático).
- gameTimeLevel (tiempo dedicado a jugar, pudiendo ser Nada, Poco, Le da igual, Mucho o Fanático).
- gender (género, puede ser Masculino o Femenino).
- mail (correo electrónico. Al día de la redacción de este documento **no se valida** su existencia ni se envía una confirmación para registrarse).
- name (nombre).
- profession (profesión u ocupación).
- state (provincia).
- uid (id generado por Firestone, va a ser el principal identificador del usuario a lo largo de la aplicación).

Una vez que el usuario se registra, el progreso se almacena dentro de la colección **levels** del usuario.

Cada documento dentro de *levels* corresponde al progreso que tiene el usuario en un nivel dado. Las colecciones dentro de level se nombran bajo la estructura *“level_[Código de nivel]”.* Cada colección posee la siguiente estructura

- actualGame (el código del juego dentro del subnivel).

- blocked (booleano que indica si dicho nivel está desbloqueado, esto ocurre al completar los todos los juegos del subnivel inmediatamente anterior).
- levelId (código de nivel).
- status (estado del nivel, puede ser “BLOQUEADO”,”INICIAR”, "EN CURSO" o “COMPLETO”).
- sublevelID (código de subnivel).
- totalGames (la cantidad de juegos dentro del subnivel).
- tutorialCompleted: booleano que indica si se completaron los tutoriales o el usuario los salteó.

Amplitude

En determinadas ocasiones a lo largo del juego (ver tabla “Tipos de Evento”) se emiten eventos a Amplitude para su posterior análisis. El SDK de Amplitude almacena localmente los eventos, y cada 30 segundos se envían conjuntamente.

Los datos que se envían a Amplitude en cada evento son:

Datos enviados	**Descripción**
uuid	Corresponde al uid de usuario de Firestone.
type	Nombre del evento. Puede ser uno de los detallados en la tabla “Eventos”.
value	Es el tiempo transcurrido en completar el nivel o completar un tutorial, medido en segundos.
level	Indica el nivel donde se produjo el evento. Sigue el formato [Código de nivel].[Código de subnivel].[Número de juego].

Los tipos de eventos que se envían corresponden a:

Nombre del evento	Descripción
game_correct_answer_in_first_try	El usuario respondió correctamente en el primer intento.
game_correct_answer	El usuario llegó a la respuesta correcta, aunque no en el primer intento.
game_wrong_answer	El usuario respondió incorrectamente.
tutorial_time_spent	El usuario completó el tutorial de un nivel.
tutorial_skipped	El usuario salteó el tutorial de un nivel.

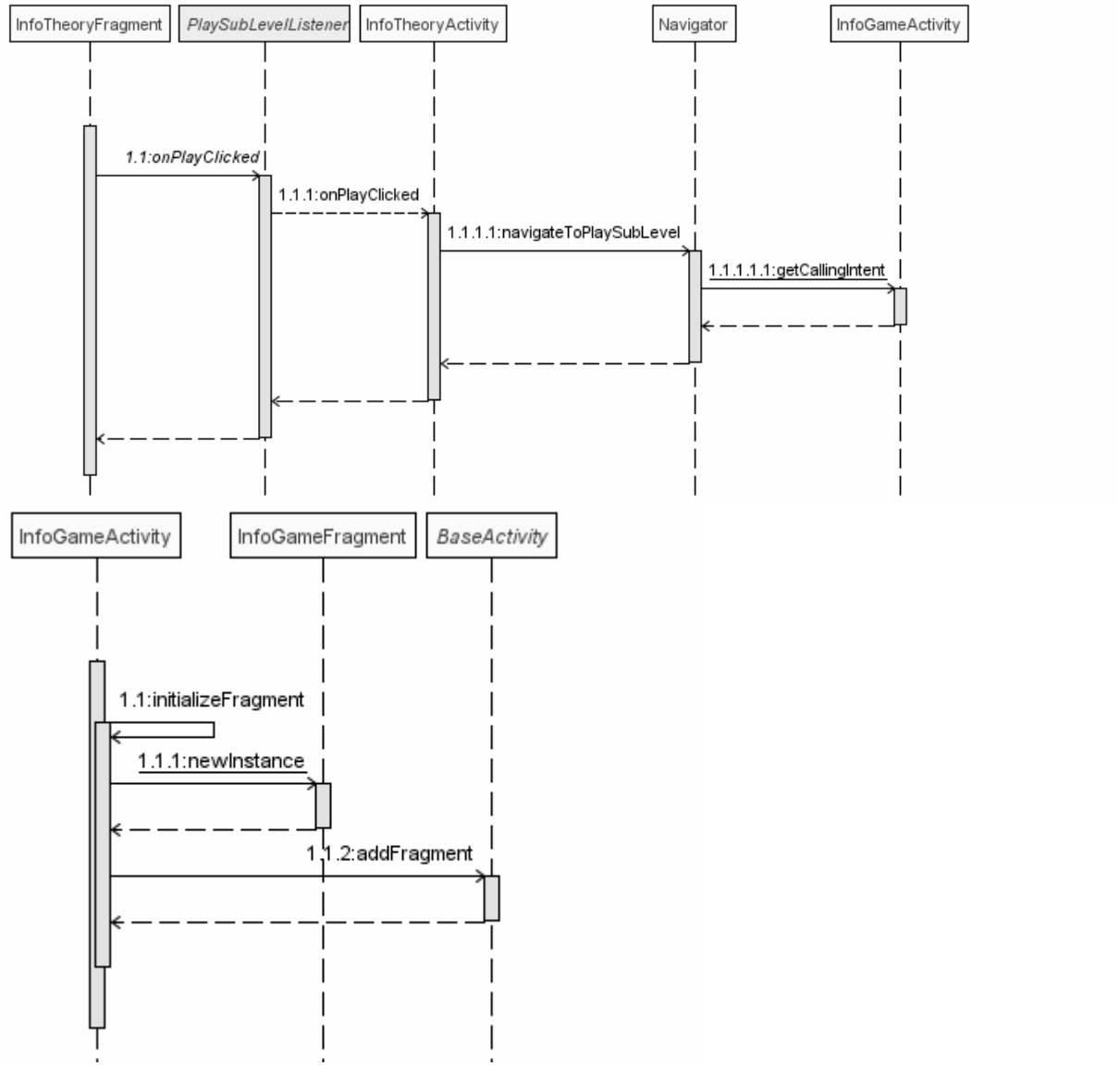
Versión 0.1

Flujos del juego

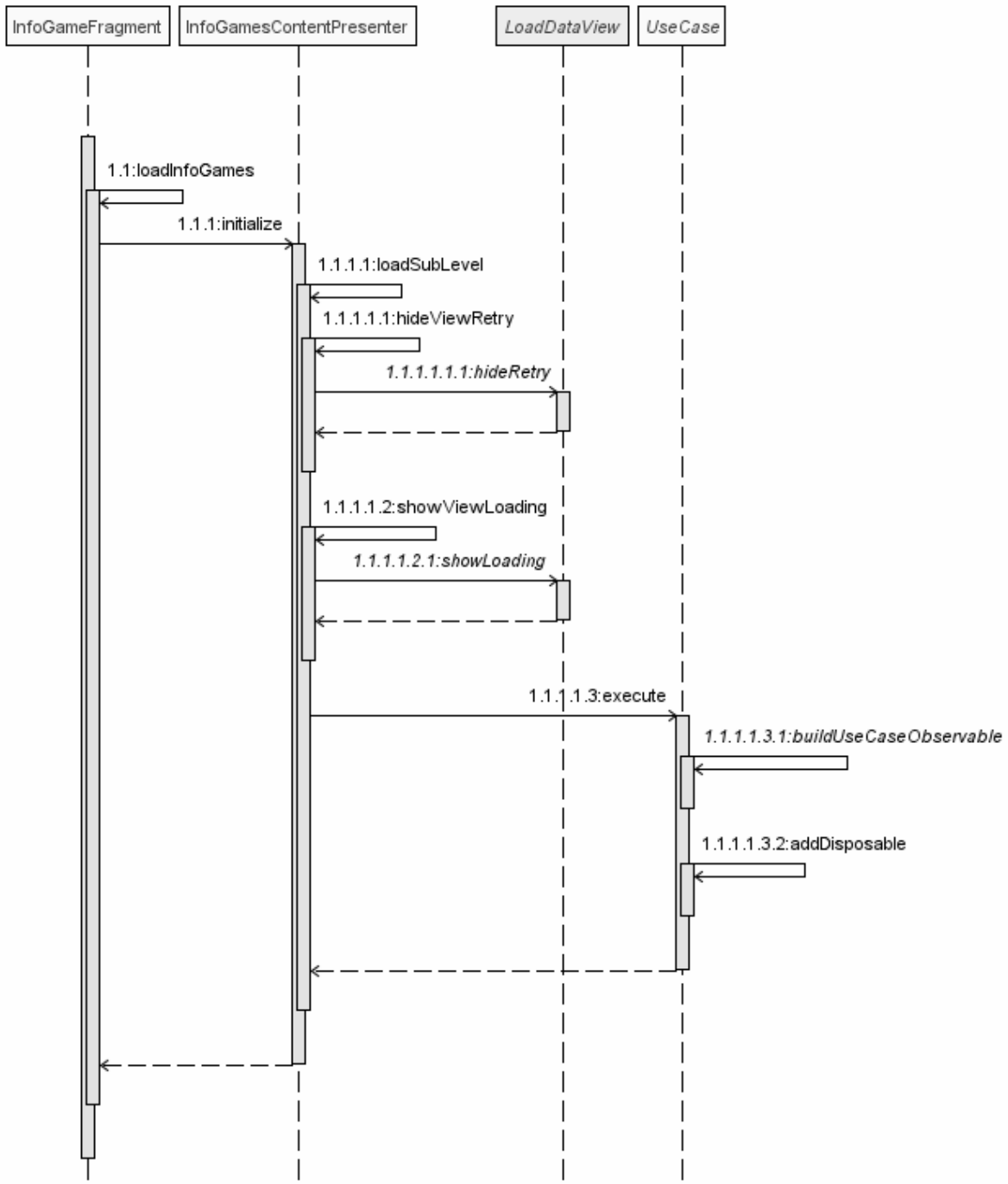
A continuación se presentan algunos diagramas de secuencia de los flujos “Jugar juego” y “Enviar evento” para visualizar cómo fluye la información a lo largo de las distintas capas.

Jugar juego

(Capa de presentación - Cargar juego)

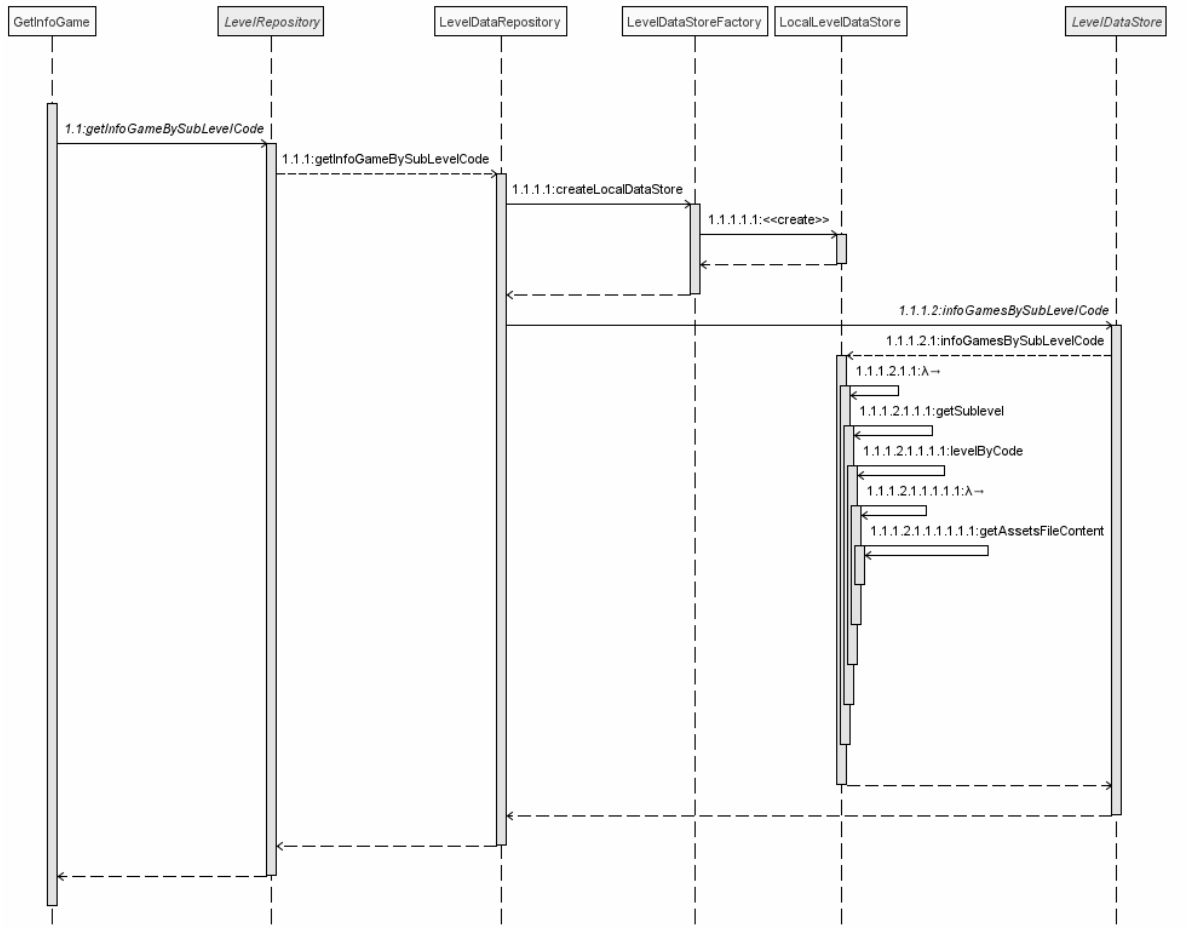


Versión 0.1

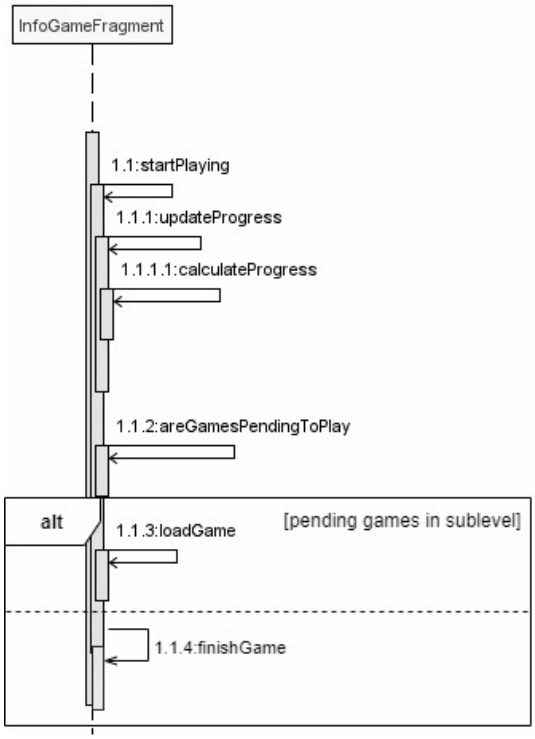

InfoGameFragment
InfoGamesContentPresenter
LoadDataView
UseCase
1.1:loadInfoGames
1.1.1:initialize
1.1.1.1:loadSubLevel
1.1.1.1.1:hideViewRetry
1.1.1.1.1.1:hideRetry
1.1.1.1.2:showViewLoading
1.1.1.1.2.1:showLoading
1.1.1.1.3:execute
1.1.1.1.3.1:buildUseCaseObservable
1.1.1.1.3.2:addDisposable

Versión 0.1

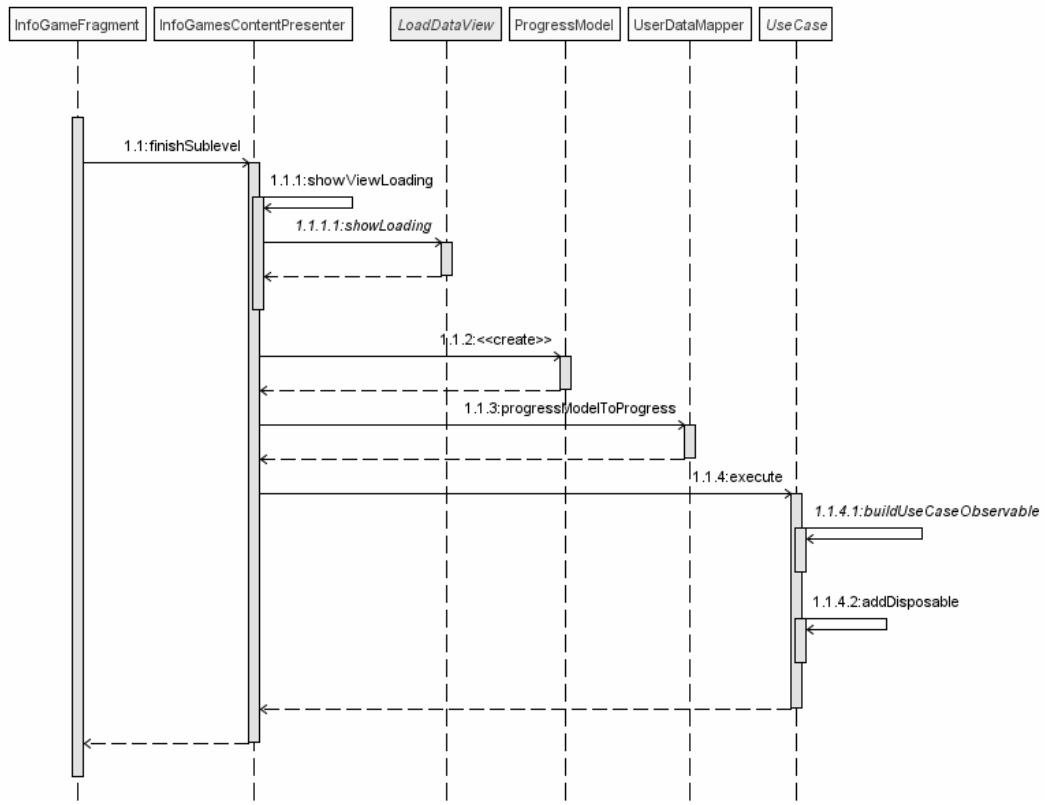
(Capa de dominio y datos - Cargar juego)



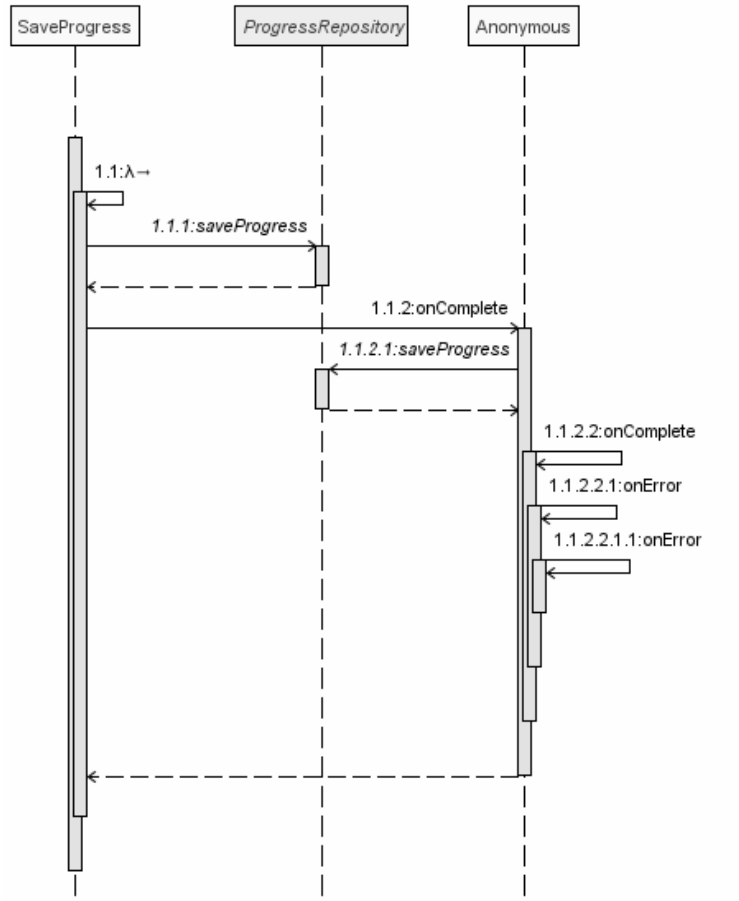
(Capa de presentación - Cargar juego y completar juego)



Versión 0.1

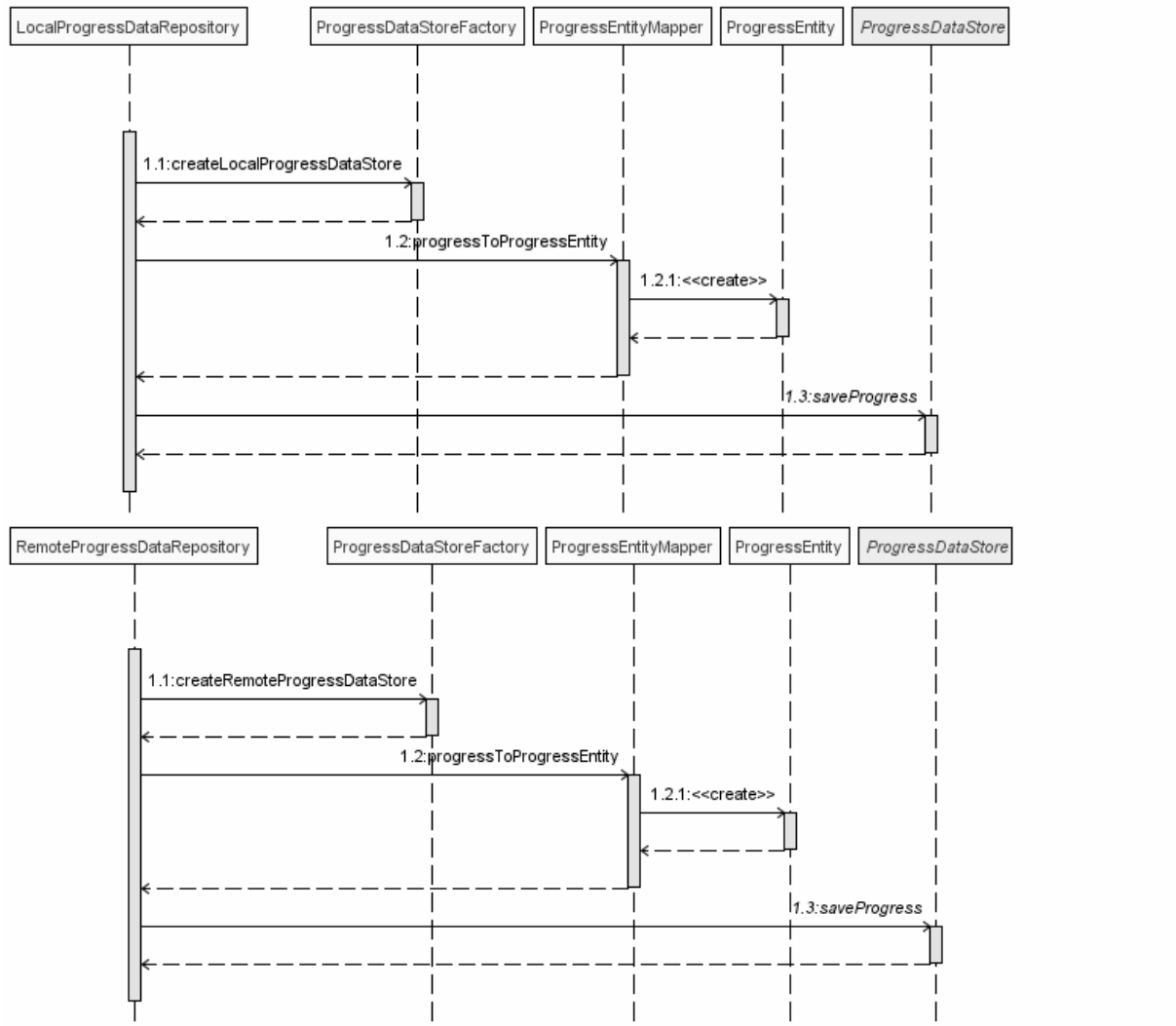


(Capa de negocio - Completar juego)



Versión 0.1

(Capa de datos - Completar juego)

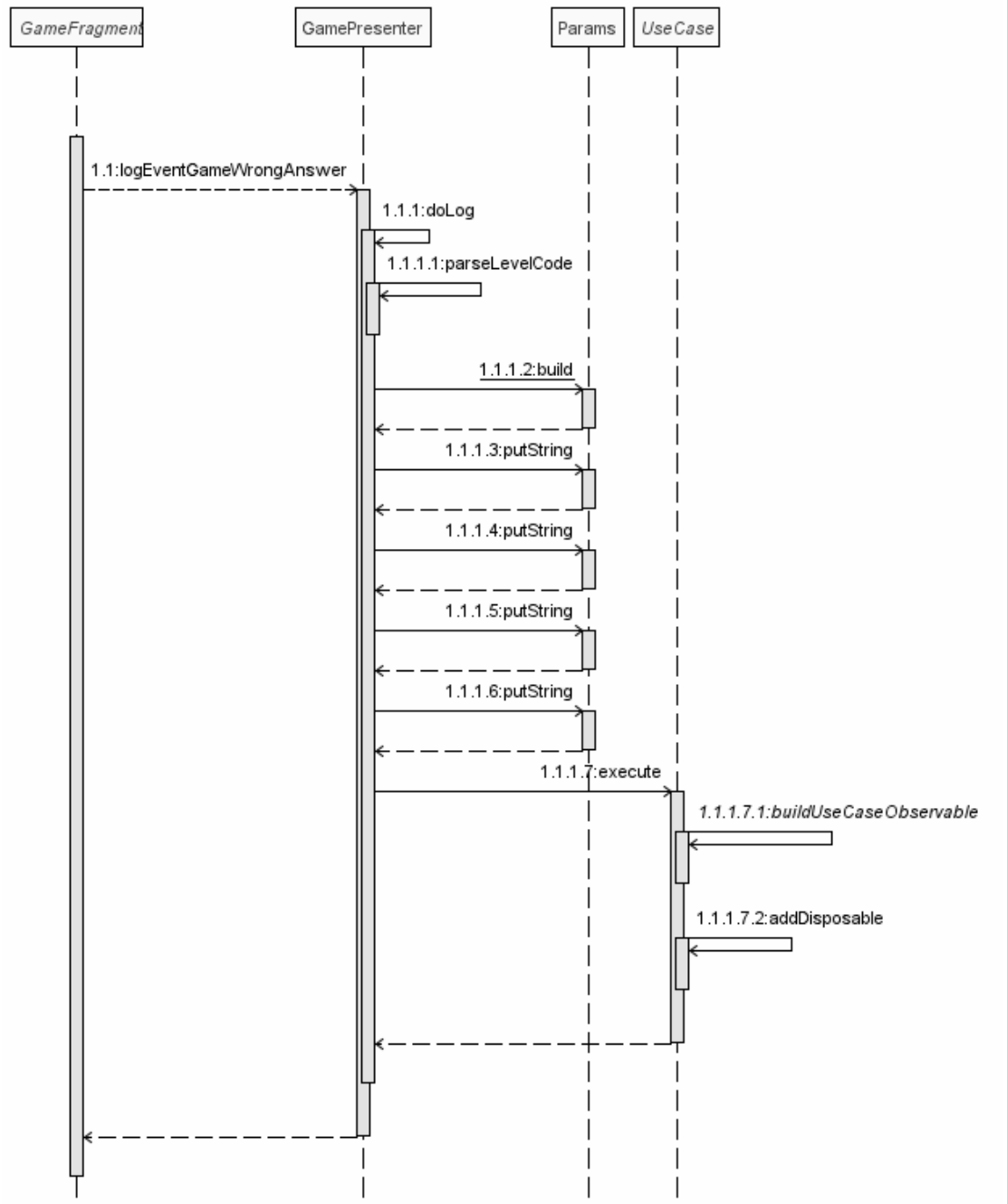


Versión 0.1

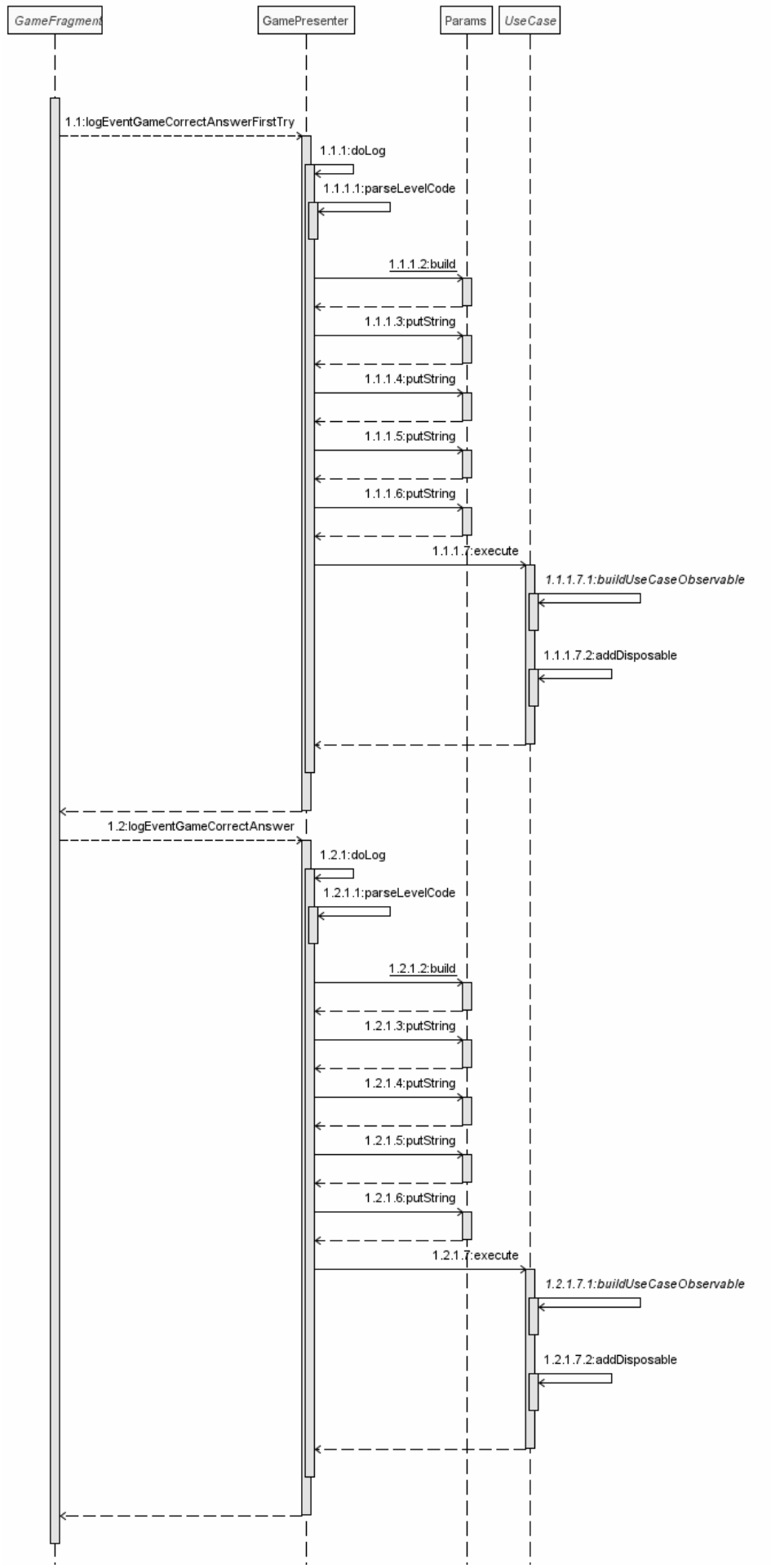
Enviar eventos

Nota: GameFragment es una clase abstracta de la cual cada tipo de juego hereda. Por lo tanto, éste flujo aplica a todos los juegos de la aplicación

(Capa de presentación)

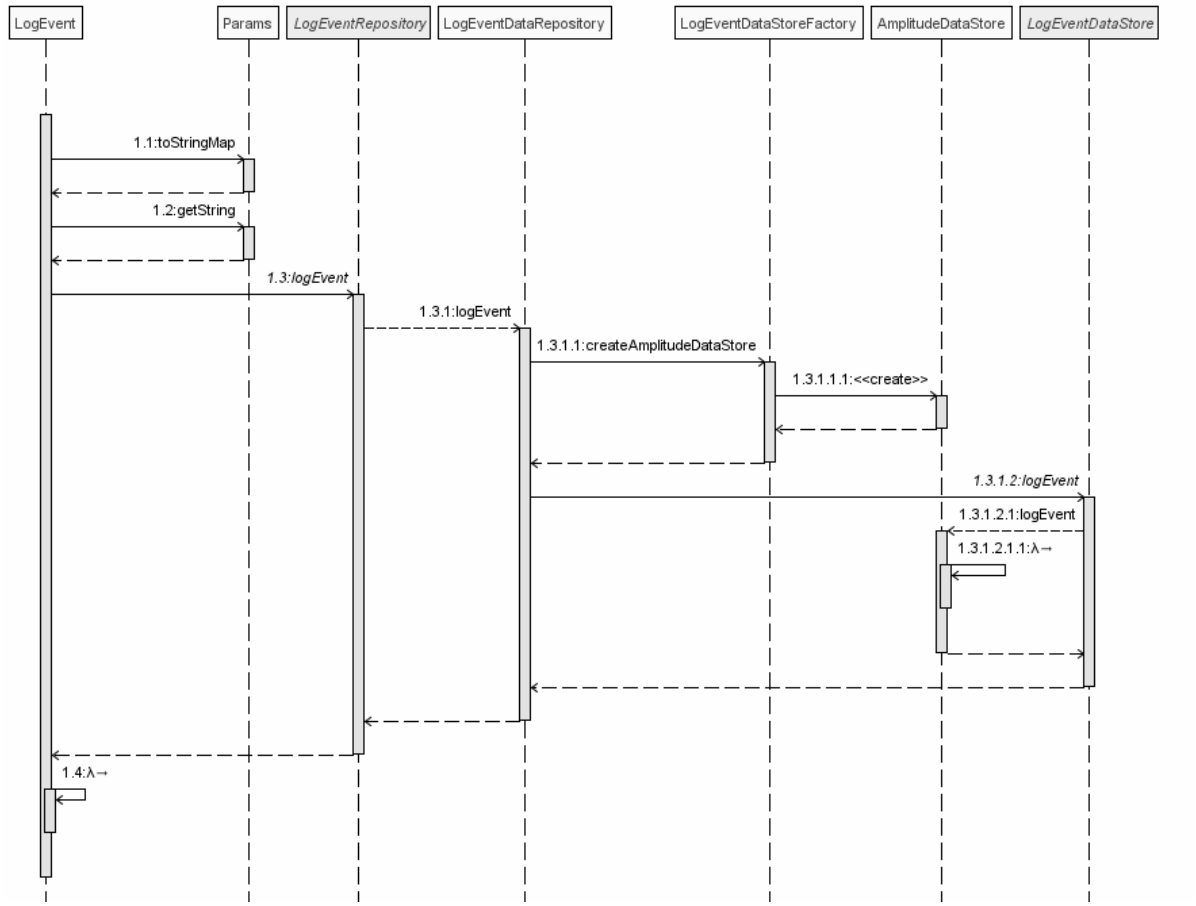


Versión 0.1


GameFragment
GamePresenter
Params
UseCase
1.1:logEventGameCorrectAnswerFirstTry
1.1.1:doLog
1.1.1.1:parseLevelCode
1.1.1.2:build
1.1.1.3:putString
1.1.1.4:putString
1.1.1.5:putString
1.1.1.6:putString
1.1.1.7:execute
1.1.1.7.1:buildUseCaseObservable
1.1.1.7.2:addDisposable
1.2:logEventGameCorrectAnswer
1.2.1:doLog
1.2.1.1:parseLevelCode
1.2.1.2:build
1.2.1.3:putString
1.2.1.4:putString
1.2.1.5:putString
1.2.1.6:putString
1.2.1.7:execute
1.2.1.7.1:buildUseCaseObservable
1.2.1.7.2:addDisposable

Versión 0.1

(Capa de dominio y capa de datos)

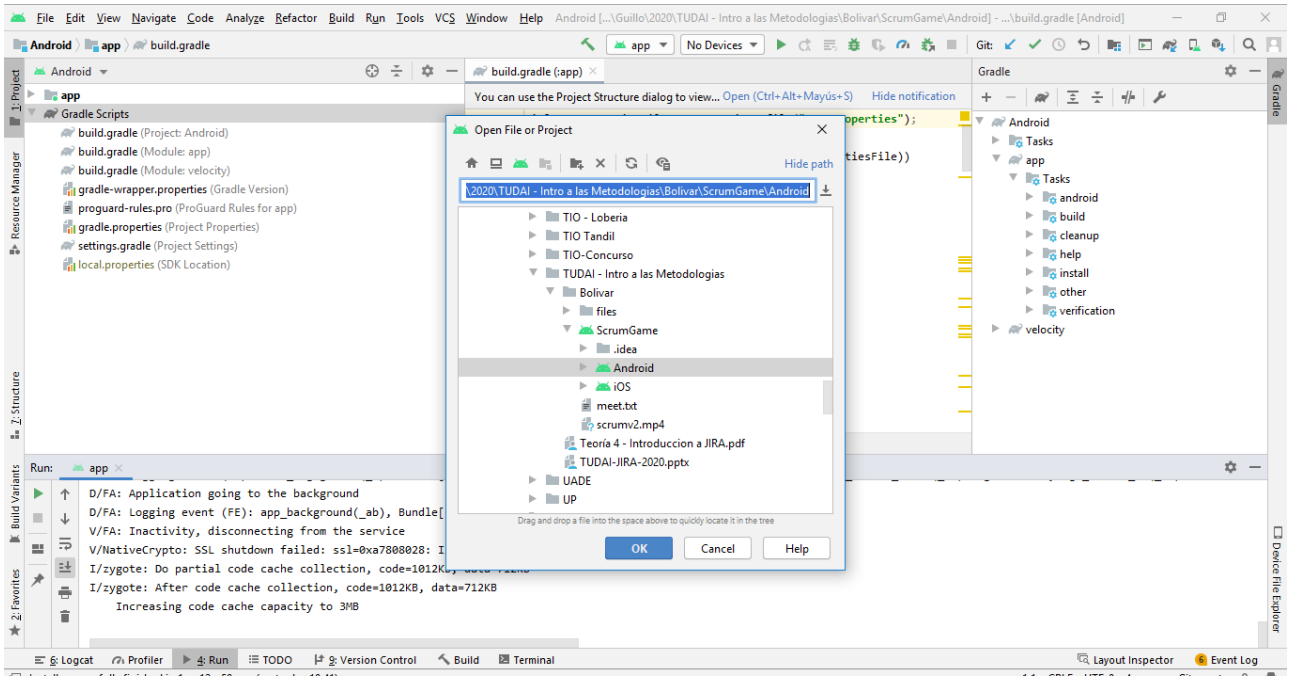


¿Cómo ejecutar la aplicación?

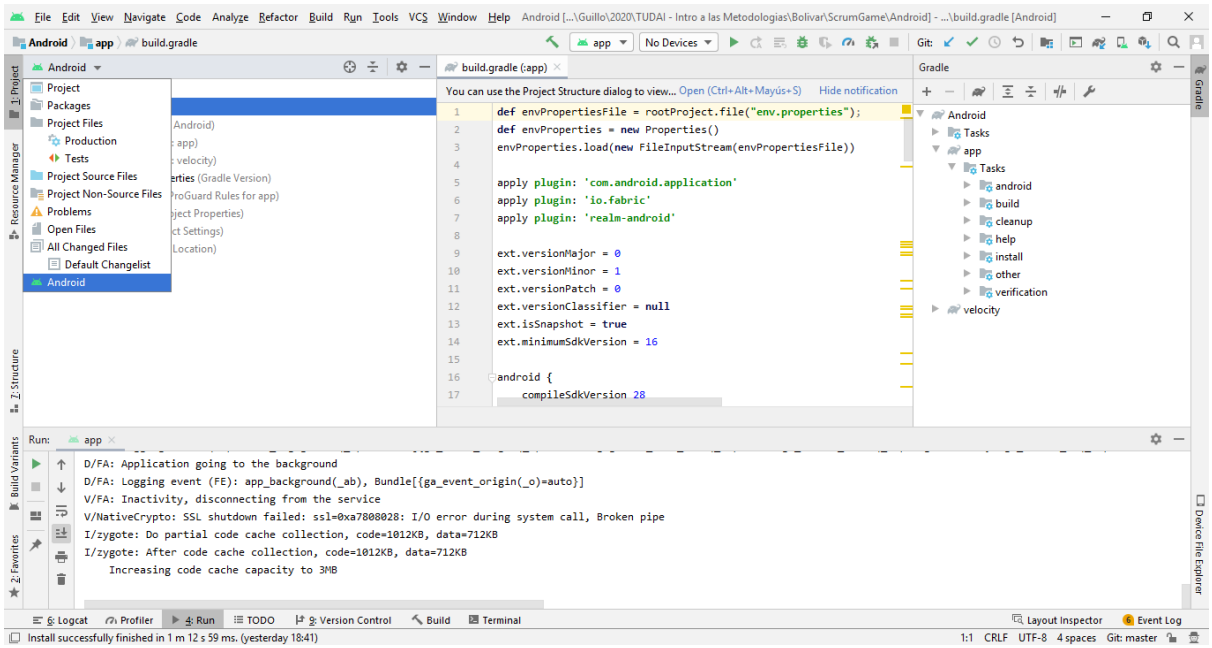
Para desarrollar el juego y ejecutar recomendamos usar el IDE Android Studio (https://developer.android.com/studio).
Se requerirá descargar el siguiente plugin de Android en la sección “Command line tools only” (https://developer.android.com/studio#downloads). Descomprimir el contenido del archivo comprimido dentro de la carpeta C:/Android.
Modificar la variable de entorno del sistema Path y adicionar: C:/Android/tools y C:/Android/platform-tools.

Primer paso: Abrir el proyecto “Android” que ya existe en el disco luego del fork.

Versión 0.1

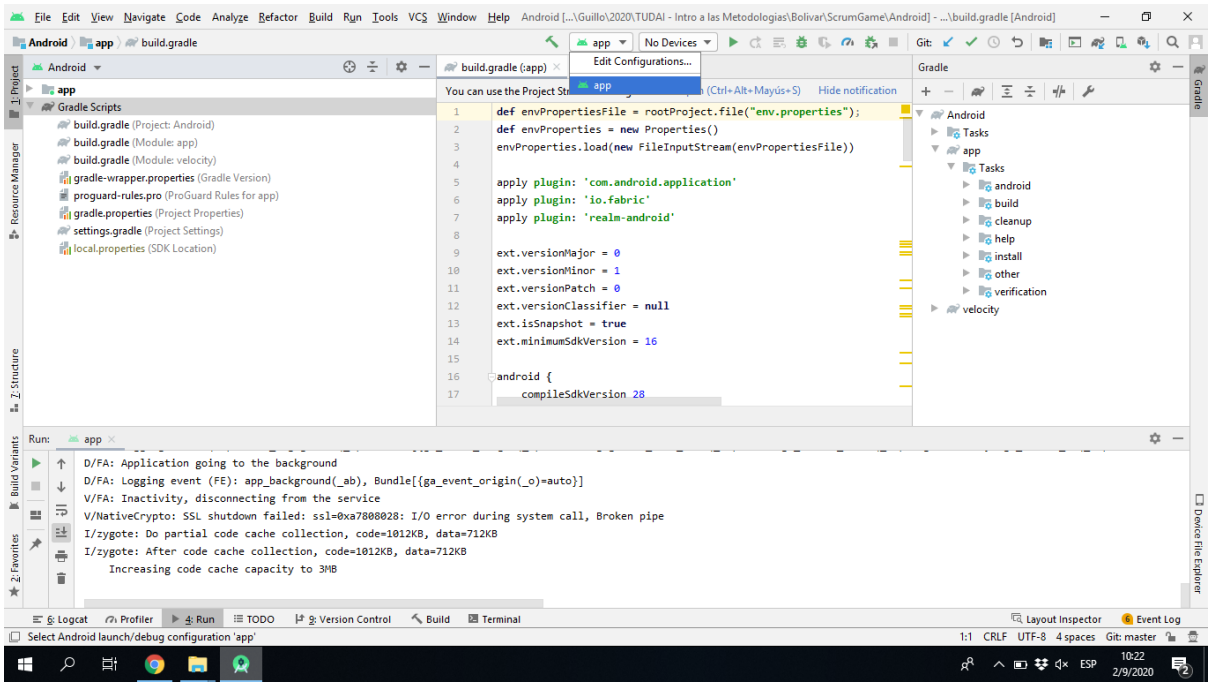


Segundo Paso: Esperar a que Gradle actúe y construya el proyecto, y luego cambiar a la perspectiva de Android en la parte izquierda superior.



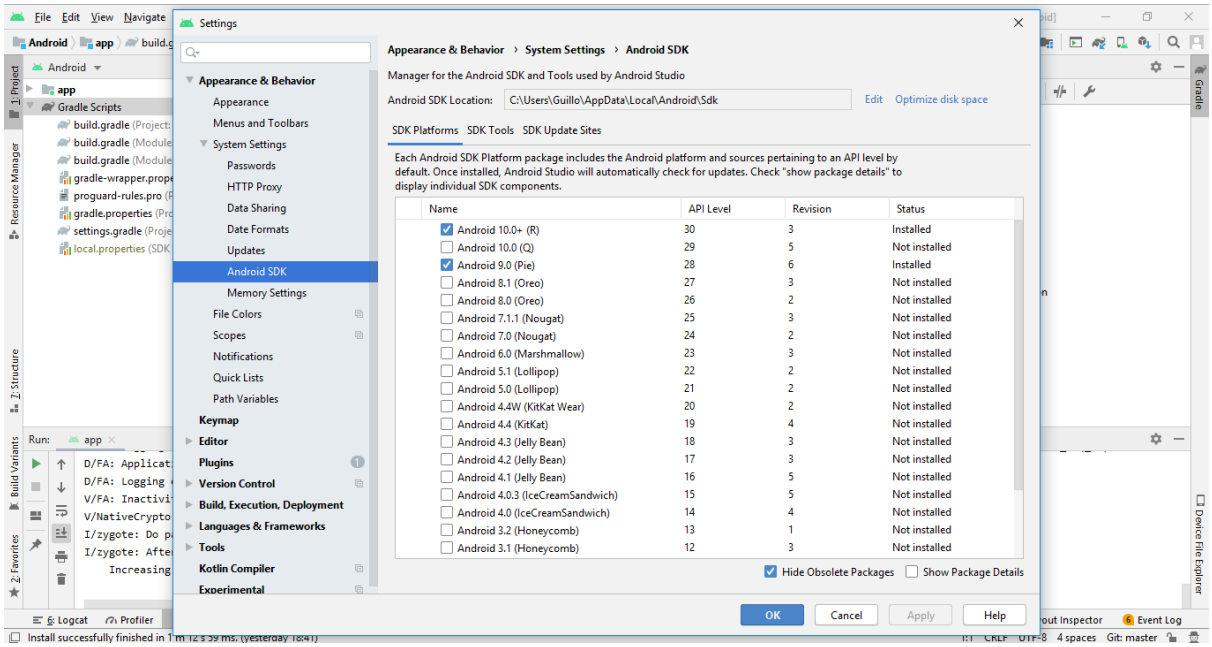
Tercer paso: Crear un nuevo template de Android como se muestra en la parte superior centro-derecha

Versión 0.1



Cuarto Paso: Conectar por USB el celular con el que van a desarrollar y probar el juego, y asegurarse que el celular tenga el modo de Depuración por USB activado. Seguir las instrucciones (https://elandroidelibre.elespanol.com/2017/06/depuracion-usb-android-como-utilizarla.html).

Quinto Paso: Ir a File → Settings en la parte de Android SDK y asegurar la instalación de las bibliotecas señaladas en la imagen.



Versión 0.1

Sexto Paso: Descomprimir el archivo comprimido “Scrum Config.zip” a la altura del proyecto “Android”. Leer atentamente el archivo README y seguir lo siguientes subpasos:

- Pegar los archivos en las siguientes ubicaciones
    - env.properties en el directorio raíz del proyecto, a la misma altura del archivo gradlew.
    - google-services.json en la carpeta /app
    - (En caso de querer usar la cuenta de Firebase configurada para producción en vez de la desarrollo, renombrar google-servicesPROD.json por google-services.json).

Séptimo Paso: Correr la aplicación presionando el botón “play” verde en la parte superior del IDE.

Octavo Paso: Aparecerá el juego en el celular y pueden usar los siguientes datos de registro para probar:
user: exarodriguez@gmail.com
pass: sistemas