**PROYECTO INTEGRADOR 2**

**ENTREGA 3 – DESARROLLO DE PRODUCTO**

**SPRINT 2**

**NOMBRE DEL EQUIPO:**

**WIMCAT**

**ESTUDIANTES:**

**CATALINA PATIÑO FORERO**

**WIILFER MANUEL SALAS GONZÁLES**

**FECHA:**

**29 DE ARBIL DE 2018**

**UNIVERSIDAD EAFIT**

**MEDELLÍN**

**2018**

TABLA DE CONTENIDO

[1. INFORME EJECUTIVO DE AVANCES 2](#_Toc512412046)

[1.1. RESUMEN DE HISTORIAS DE USUARIO 2](#_Toc512412047)

[1.2. RETROSPECTIVA Y GESTIÓN DE RIESGOS DEL PROYECTO 2](#_Toc512412048)

[2. URL DE LOS AMBIENTES DEVOPS 2](#_Toc512412049)

[2.1. URL DEL REPOSITORIO EN GIT (DEV) 2](#_Toc512412050)

[2.2. URL DE APLICACIÓN EN PRUEBAS (TESTING) 2](#_Toc512412051)

[2.3. URL DE APLICACIÓN EN PRODUCCIÓN (RELEASE) 2](#_Toc512412052)

[3. DOCUMENTO DE ARQUITECTURA BASE 2](#_Toc512412053)

[3.1. GESTIÓN DE LA CONFIGURACIÓN Y AMBIENTE V2 2](#_Toc512412054)

[3.2. ARQUITECTURA EVOLUTIVA V2 3](#_Toc512412055)

[3.3. MARCO DE REFERENCIA O TEÓRICO DE LA TECNOLOGÍA EMERGENTE V2 3](#_Toc512412056)

[4. PLATAFORMA VISUAL STUDIO 3](#_Toc512412057)

[5. INTEGRACIÓN CONTINUA 3](#_Toc512412058)

[6. ENTREGABLES ESPECIFICO POR TIPO PROYECTO 4](#_Toc512412059)

# INFORME EJECUTIVO DE AVANCES

## RESUMEN DE HISTORIAS DE USUARIO

* HU1 – Nivel 1: Se desarrolla el primer nivel del juego donde se preguntan por los tipos de intervalos escuchados. Se crean diferentes escenas cada uno por grupo de intervalos.



Imagen 1. Nivel 1

* HU3 – Nivel 2: Se desarrolla el segundo nivel del juego donde se pregunta por la segunda nota de un intervalo, dada la primera nota. No todas las tareas se completaron todas las tareas, solo las necesarias para dar una idea base del nivel.



Imagen 2. Nivel 2

* HU15 – Asignar puntos: Se asignan puntos a los usuarios, debido a acciones correctas en ambos niveles.

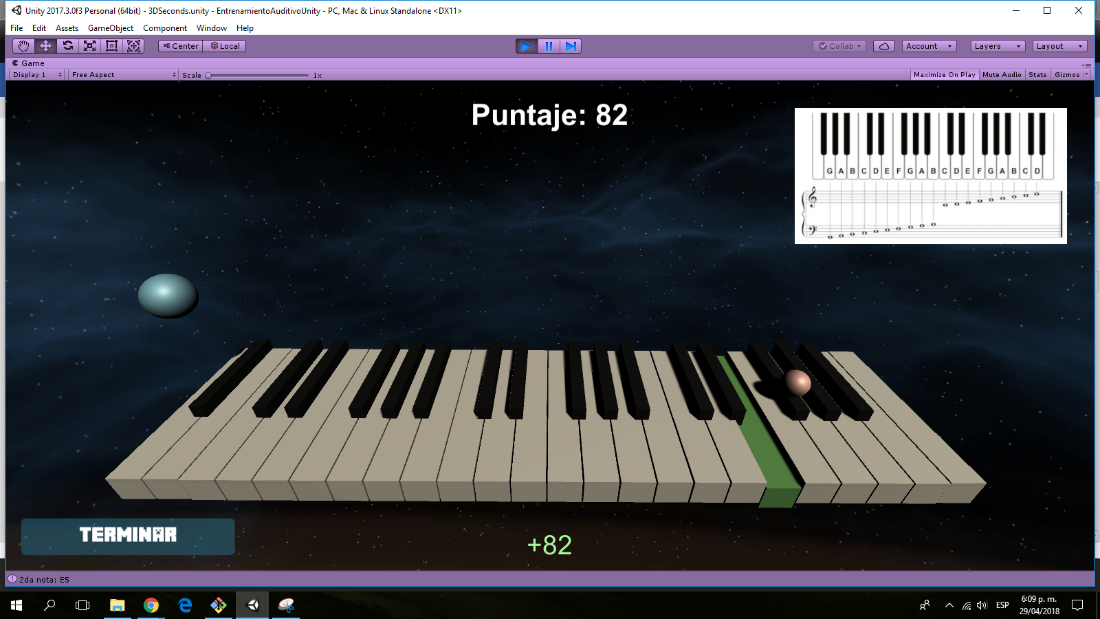


Imagen 3. Asignación de puntos

* HU11 – Retroalimentación: Se da retroalimentación al usuario para que conozca cuál era la respuesta correcta.

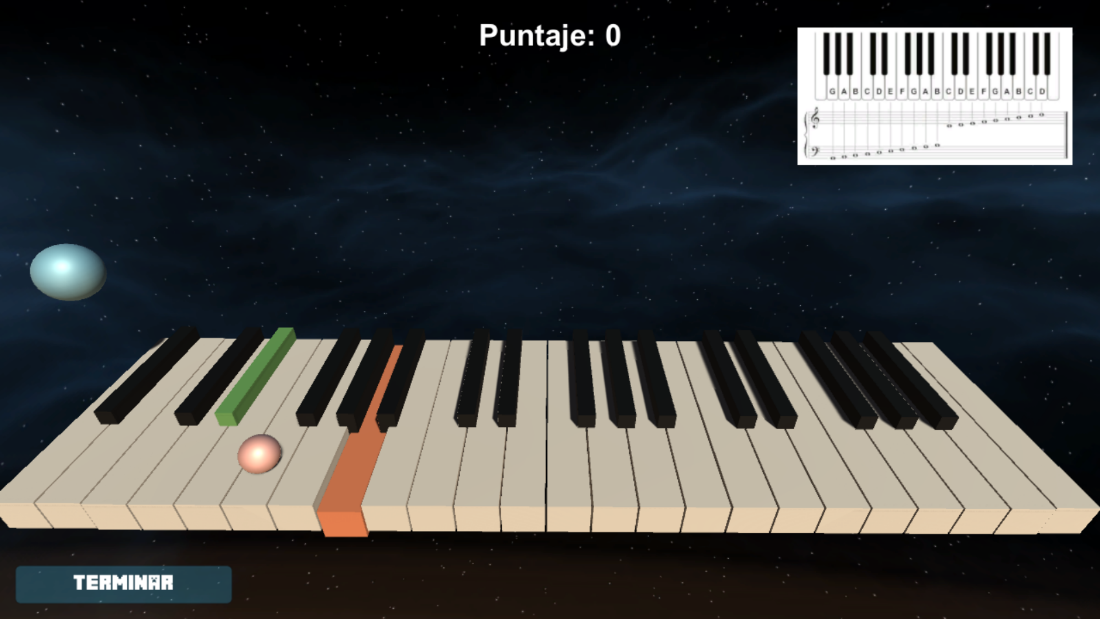


Imagen 4. Retroalimentación

* HU7 – Almacenar información: Se almacena la información sobre las respuestas dadas por el usuario durante la ejecución de la aplicación.

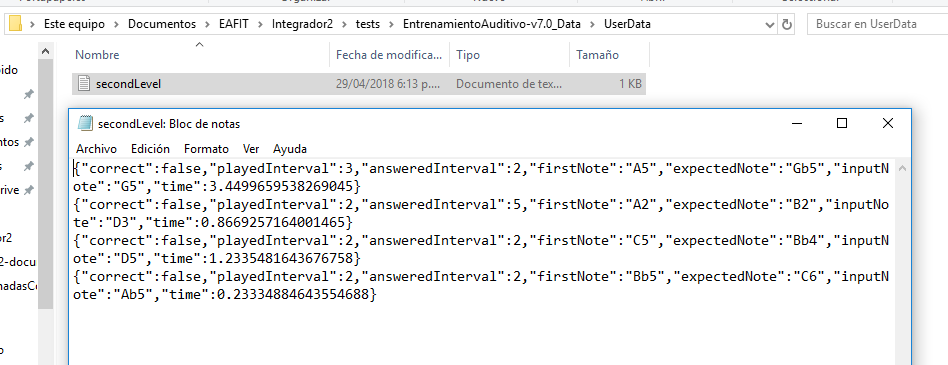


Imagen 5. Almacenamiento información

* HU-18 Recolección de datos: Se hace toma de datos con estudiantes de nivelatorio de la universidad EAFIT.

Los datos recolectados hasta el momento se encuentran en el siguiente link de GitHub (carpeta que pertenece a un repositorio donde se está almacenando toda la documentación del proyecto):

<https://github.com/cpatin10/proyecto2-documentacion/tree/master/datasets>

* HU-8 Instrucciones: Se crean las instrucciones de cómo jugar, a las cuales accede el usuario por medio de la aplicación.

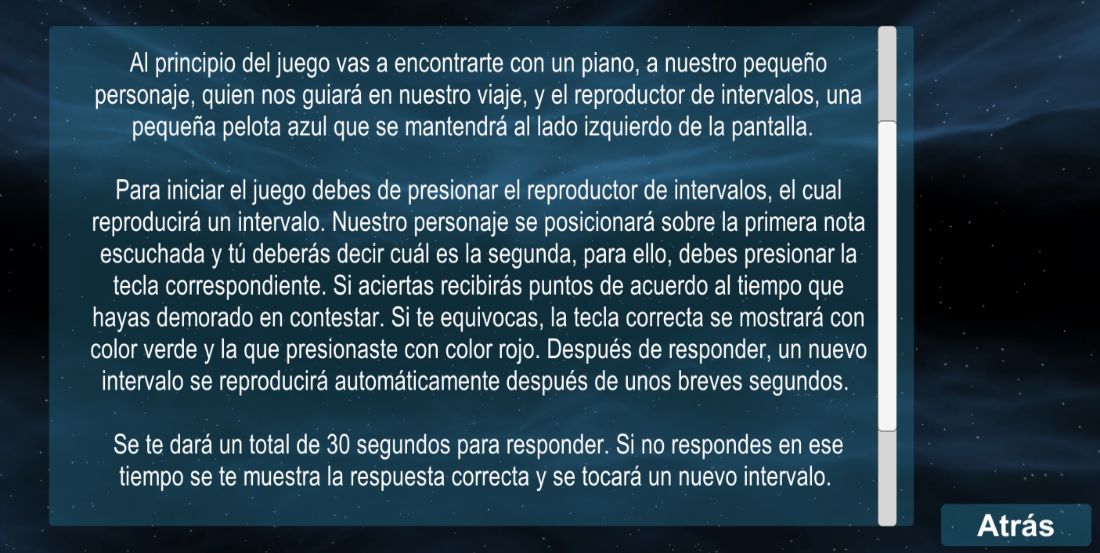


Imagen 6. Instrucciones de juego

* HU-9 Sincronización sonidos: Se ajustan los sonidos de las notas para que tengan una menor duración.
* HU-2 Cambio dinámico de acuerdo al jugador: Se implementa el modelo para realizar la analítica, predecir qué tipo de jugador es, y qué debe fortalecer dicho tipo de jugador. El juego debe cambiar dinámicamente de acuerdo a las tendencias del jugador presentadas en la analítica. Ésta historia de usuario lastimosamente no fue implementada en su totalidad. Se avanzó en el desarrollo del modelo, pero queda pendiente el ajuste dinámico de la aplicación. Por tanto, hay un retraso en el desarrollo.

Poner pantallazo del código o resultados del modelo (Sólo 1)

Imagen 7. Modelo

Por descuido de los integrantes del equipo no es posible calcular con precisión el porcentaje alcanzado hasta el momento, ya que no se asignaron puntos de esfuerzo a todas las historias, especialmente las primeras desarrolladas, y ya no deja asignar a éstas. Se calcula que se lleva aproximadamente un 70% del sistema desarrollado, sin embargo, esto es después de haber descartado historias de usuario, consideradas como secundarias y que no son el objetivo principal del proyecto.

## RETROSPECTIVA Y GESTIÓN DE RIESGOS DEL PROYECTO

* Al igual que en el sprint 1, dado que los integrantes del equipo de desarrollo no manejan la tecnología a plenitud, se pueden encontrar dificultades en el desarrollo. Hasta el momento las dificultades se han buscado resolver realizando búsquedas por internet, iteraciones, o con apoyo de los profesores.
* En lo que refiere al uso de la plataforma Jenkins, se logra configurar y hacer un build exitoso, sin embargo, se están teniendo dificultades que hasta el momento no han sido resueltas. Se piensa asistir a una asesoría con profesores para resolverlos.
* En lo que refiere a análisis de código estático, se encontró que SonarQube no es una herramienta apropiada para Unity, aun cuando si lo es para C#. Por tanto, fue necesario la búsqueda de una nueva herramienta gratuita. Lastimosamente, las herramientas encontradas son de uso manual y no permiten automatización, como lo es PVS-Studio. Por este motivo no se realizan pruebas de análisis de código estático. Más adelante se profundiza respecto al tema.
* Nuevamente, la gestión del tiempo es una de las más grandes dificultades. Se está implementando una distribución por horarios del tiempo que se le dedica al proyecto y a otras materias, sin embargo, suelen aparecer dificultades en cualquiera de éstas por lo que se consume más tiempo del presupuestado. Por esto, se está opta por identificar compañeros que hayan tenido dificultades similares y que nos puedan dar una guía. Sin embargo, esta solución no ha sido suficiente, y por este motivo no se han desarrollado pruebas unitarias para el código.
* En este sprint se percibieron bugs en el nivel 2D de la aplicación. Para evitar invertir mucho tiempo en esta actividad, se decidió dejar ésta en standby, para enfocarse más en cuestiones de desarrollo del modelo para machine learning y de integración continua. Esto es posible gracias a que el nivel 3D es funcional y no presenta ninguna falla mayor.
* La manipulación de los datos del DataSet para el entrenamiento y la evaluación del modelo tuvo algunos problemas debido a que el DataSet fue creado desde 0 lo cual requirió que antes de poder usarlos tuviesen que pasar por unos procesos para que estos funcionaran de forma correcta y no tuviese como resultado errores inesperados.

# URL DE LOS AMBIENTES DEVOPS

## URL DEL REPOSITORIO EN GIT (DEV)

Para el desarrollo en sí, se está usando el siguiente repositorio en git, donde se desarrolla sobre la rama 'global':

<https://github.com/cpatin10/EntrenamientoAuditivo>

Para la gestión de entregas y similares se creó el siguiente repositorio:

<https://github.com/cpatin10/proyecto2-documentacion>

Se crea un nuevo repositorio por lo grande que es del repositorio de desarrollo actualmente, y se quiere evitar agregar información no necesaria en el momento de compilación.

## URL DE APLICACIÓN EN PRUEBAS (TESTING)

Debido a que la aplicación es stand-alone, para el ambiente de pruebas se estaba manejando una carpeta diferente a donde se está desarrollando el proyecto. Para las pruebas se genera el ejecutable para Windows (sistema operativo usado por el grupo de desarrollo) dado por Unity, se almacena en dicha carpeta y se ejecuta. Los archivos generados, fueron subidos a la carpeta de Google Drive "tests" dentro de la carpeta asignada:

<https://drive.google.com/drive/folders/1si_uEWbYiDJ-lJvBOJhhe1mj5uAv-NhH?usp=sharing>

## URL DE APLICACIÓN EN PRODUCCIÓN (RELEASE)

*(Si aplica, link a dicha URL)*

*(Si aplica, link a dicha URL) (Alerta: el producto ya debe ser accedido mediante una URL, si es una app movil iOS o .APK deberá colocarlo en un link en el google drive. Si es una app para Windows o Mac, deberá colocarlo en el google drive)*

*Si su proyecto tiene unos productos especificos, por favor detallarlo aca y hacer las aclaraciones pertinentes.*

El ejecutable para producción se almacena en la carpeta "release" dentro de la carpeta asignada de Google Drive:

<https://drive.google.com/drive/folders/166L1Dg5YNEIBAkQ9IYPpH7RoT4WpqdmY?usp=sharing>

# DOCUMENTO DE ARQUITECTURA BASE

## GESTIÓN DE LA CONFIGURACIÓN Y AMBIENTE V2

Se puede encontrar las especificaciones de configuración y ambiente de desarrollo en la carpeta de Drive del Sprint 2, dentro de la carpeta "Plan de configuración de entorno y SCM"

* **Carpeta:**

<https://drive.google.com/drive/folders/1Mt25DBsKO3YweUStCS_KdUhf5mKdopfK?usp=sharing>

* **PDF**: <https://drive.google.com/file/d/1b1ohrPB2AkHUOEcPXcvw8pGMIrfoVpo6/view?usp=sharing>
* **Doc**: <https://docs.google.com/spreadsheets/d/14Ara6puP9CPMCPHRmJHJDMQXmft2UHuFVpamUxfvXZA/edit?usp=sharing>

## ARQUITECTURA EVOLUTIVA V2

Se puede encontrar el documento base de la arquitectura del sistema en la carpeta de Drive del Sprint 2, dentro de la carpeta "Documento Arquitectura"

* **Carpeta:**

<https://drive.google.com/drive/folders/1Z_JvDU7mQlCr6KPF9m9pkxKjojf9YbFE?usp=sharing>

* **PDF**:

<https://drive.google.com/file/d/1S1b5NBLcHj5-8TeefpP23J4ViWcHeTPS/view?usp=sharing>

* **Doc**:

<https://drive.google.com/file/d/1sl3t0U2ANA-q-_wGMyUqGBb54hFIjvut/view?usp=sharing>

## MARCO DE REFERENCIA O TEÓRICO DE LA TECNOLOGÍA EMERGENTE V2

Se puede encontrar el marco teórico en la carpeta de Drive del Sprint 2, dentro de la carpeta "Marco Teórico"

* **Carpeta:**

<https://drive.google.com/drive/folders/1UdgP1e31tQk7nUnqNCuWRB0kezDfJkWH?usp=sharing>

* **PDF**:

<https://drive.google.com/file/d/1IxRrnryETy5Cq7R57Zle37Xt_N_-Peio/view?usp=sharing>

* **Doc**:

<https://drive.google.com/file/d/1SYYTibSUukXuebtgRaxhUzGDcFeAAkbq/view?usp=sharing>

# PLATAFORMA VISUAL STUDIO

*URL VS:* <https://p2wimcat.visualstudio.com/P2Wimcat/>

En la siguiente imagen se presentan las historias de usuario completamente implementadas en el Sprint 2.

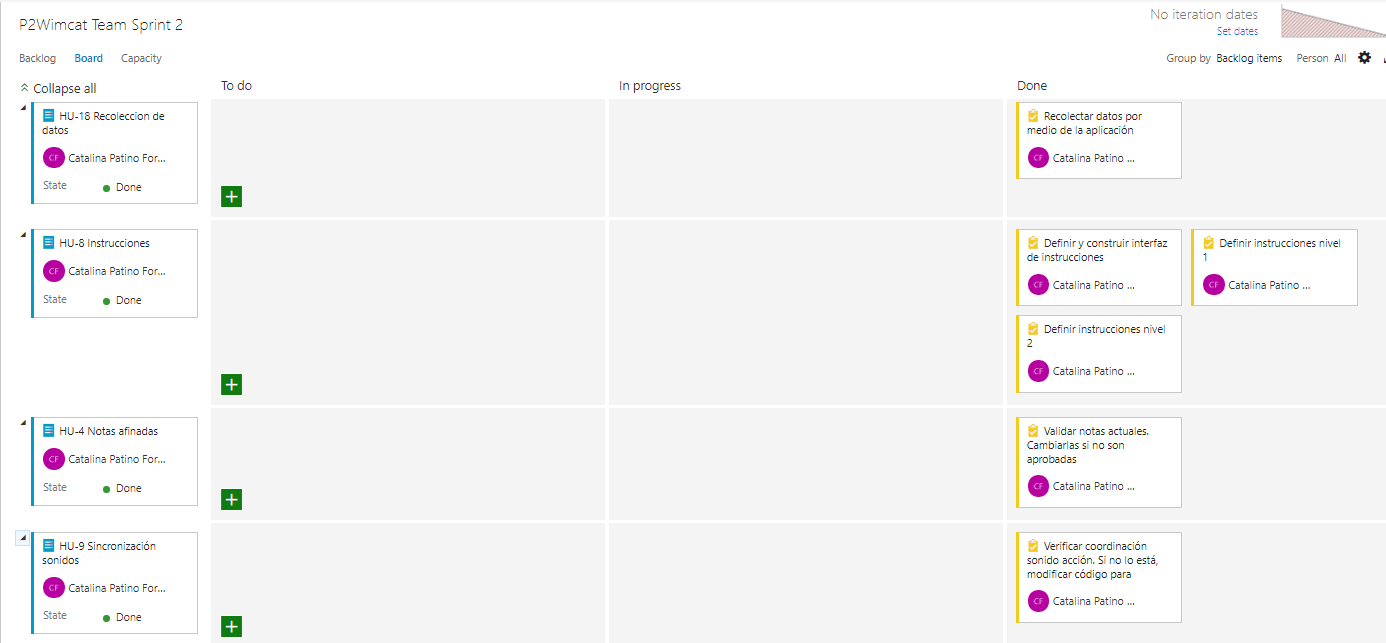


Imagen 8. Historias de usuario implementadas en Sprint 2

En la siguiente imagen se presentan las historias de usuario a implementar en el Sprint 3.

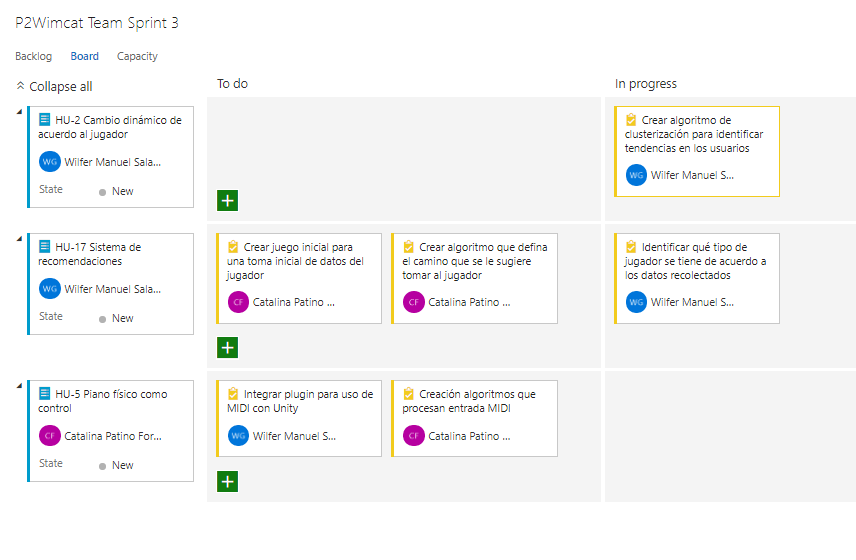


Imagen 9. Planeación Sprint 3.

# INTEGRACIÓN CONTINUA

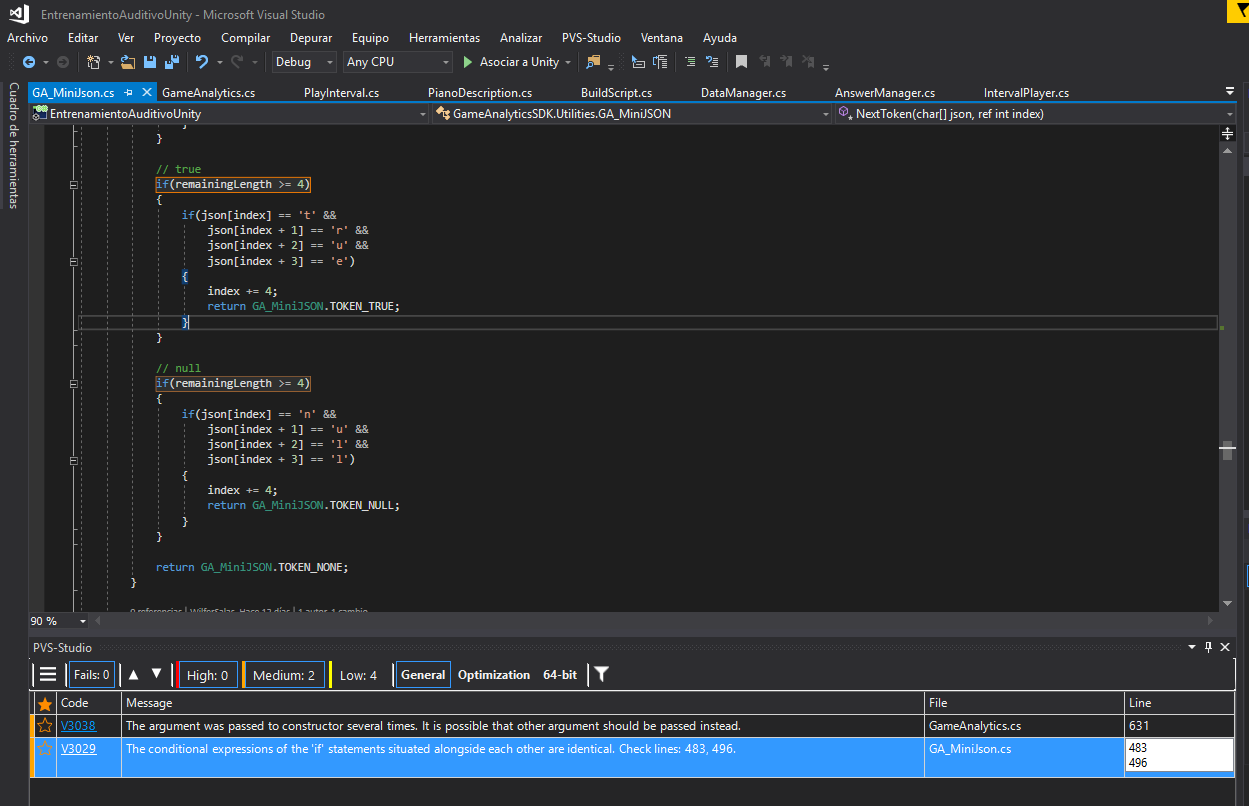
**Calidad de código fuente**

Lastimosamente no se encontraron herramienta que facilitaran el análisis de código estático que pudieran ser automatizadas. Esto se debe, a que no hay herramientas gratuitas apropiadas para el análisis de código en el contexto de Unity, este es el caso de SonarQube. La herramienta encontrada anteriormente, PVS-Studio, que sí hace este análisis, solo puede ser usada de forma manual.

En los siguientes links se pueden encontrar referencias que nos permitieron concluir lo anterior:

* <https://stackoverflow.com/questions/41015278/sonarqube-using-it-with-unity-3d-tons-of-problems>
* <https://community.gamedev.tv/t/static-code-analysis-options-for-unity-and-or-unreal/33462/9>
* <https://forum.unity.com/threads/sonarqube-and-unity-code-quality.444490/>

En la siguiente imagen se demuestra el uso de PVS-Studio, pero ejecutado de forma manuel. Las advertencias no se corrigen a propósito con motivo de ejemplificar el funcionamiento en la imagen.

Imagen 10. Análisis de código estático con PVS-Studio

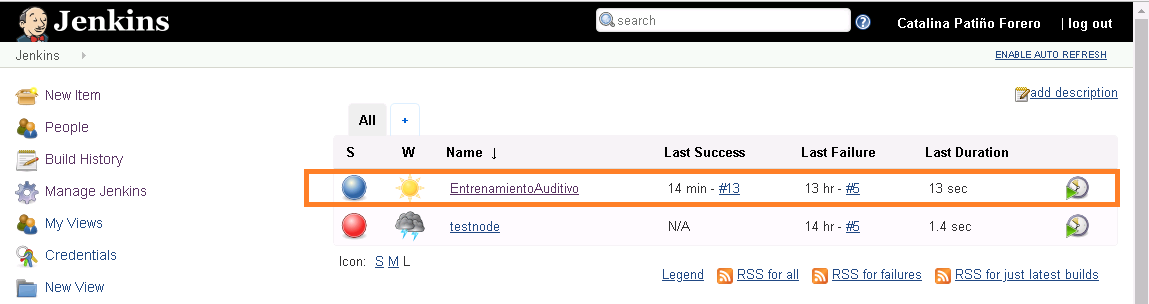
**Integración continua**

Para integración continua se utiliza Jenkins con el plugin de Unity3D, el cual corre sobre un servidor Windows, prestado por la universidad. Esto se debe a que no existe una versión nativa de Unity para Linux.

El comando utilizado dentro para el build con el plugin de Unity3D es:

"-quit -batchmode -logFile=C:\Users\cpatin10\AppData\Local\Unity\Editor\Editor.log -projectPath=C:\Program Files (x86)\Jenkins\Workspace\EntrenamientoAuditivo\EntrenamientoAuditivoUnity -executeMethod=BuildScript.PerformWindowsBuild -nographic"

En las siguientes imágenes se presentan pantallas de la página de Jenkins donde se muestra la creación del job y los resultados de los últimos builds.

Imagen 11. Job en Jenkins

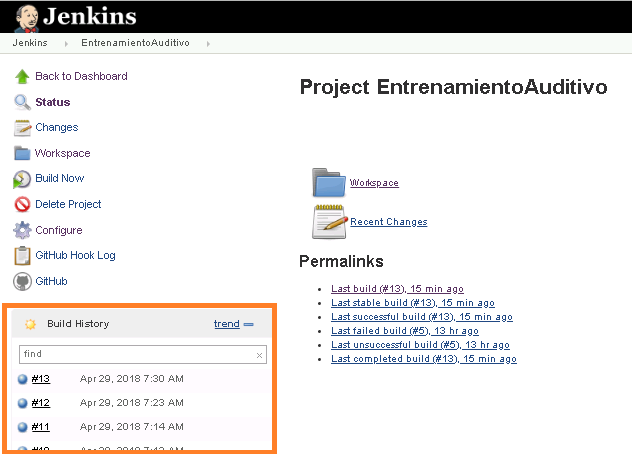


Imagen 12. Resultados de los últimos builds en Jenkins

Como se mencionó anteriormente se han encontrado dificultades con el manejo de Jenkins, como el encontrar el archivo ejecutable que debería ser generado después de la ejecución del Build.

# ENTREGABLES ESPECÍFICO POR TIPO PROYECTO

Para los proyectos de analítica, relacionar todo lo referente a:

1. Datos de entrenamiento

Se adjunta el link en git del DataSet utilizado para entrenar y evaluar el modelo:

<https://github.com/cpatin10/proyecto2-documentacion/tree/master/datasets>

1. Modelos evaluados y el seleccionado.

Se descarto el modelo de regresión debido a que a que no se encontró una relación entre distintas variables capaz de predecir en todo momento el resultado del jugador que varía según va jugando.

El modelo seleccionado ha sido el del Arboles de Clasificación y Regresión (CART) debido a que el DataSet se adapta a la estructura de este, en donde se tiene un nodo llamado raíz que corresponde al atributo de si la respuesta del jugador ha sido o no correcta, luego se descomponen el resto de atributos de entradas en ramas que corresponden a las notas que sonaron y se plantea la condición que lleva a realizar recomendaciones sobre cuáles son las que representan una dificultad para el usuario.

* 1. Modelos matemáticos / estadísticos

El modelo matemático de CART es el siguiente:

El proceso puede esquematizarse en 4 fases: construcción *(building)* del árbol, parada *(stopping)* del proceso de crecimiento del árbol (se constituye un árbol máximo que sobreajuste la información contenida en nuestra base de datos), podado *(pruning)* del árbol haciéndolo más sencillo y dejando sólo los nodos más importantes y, por último, selección (*selection*) del árbol óptimo con capacidad de generalización.

La construcción del árbol comienza en el nodo raíz, que incluye todos los registros de la base de datos. A partir de este nodo el programa debe buscar la variable más adecuada para partirlo en 2 nodos hijos. Para elegir la mejor variable debe utilizarse una medida de pureza *(purity)* en la valoración de los 2 nodos hijos posibles (la variable que consigue una mayor pureza se convierte en la utilizada en primer lugar, y así sucesivamente). Debe buscarse una función de partición *(splitting function)* que asegure que la pureza en los nodos hijos sea la máxima. Una de las funciones más utilizada es la denominada Gini (se alcanza un índice de pureza que se considera como máximo).

El índice de Gini en el nodo *t*, *g(t)*, se puede formular del modo siguiente:

Insertando imagen...

donde *i* y *j* son las categorías de la variable predictora y *p* es proporción.

La función de criterio Φ *(s,t)* para la división *s* en el nodo *t* se define como:

Insertando imagen...

donde *pL* es la proporción de casos de t enviados al nodo hijo de la izquierda, y *pR* al nodo hijo de la derecha.

Cuando se comienza en el nodo raíz *t = 1* (y también en las particiones sucesivas), se busca la división *s\**, de entre todas las posibles de *S*, que de un valor con mayor reducción de la impureza:

Insertando imagen...

Luego se divide el nodo 1 en 2 nodos hijos *(t = 2 y t = 3)* utilizando la división *s\**. Este valor de la función de impureza, ponderado por la proporción de todos los casos del nodo *t*, es el valor del que se informa en el árbol como «mejora».

En el proceso sucesivo de construcción y crecimiento del árbol se debe asignar una clase (etiqueta) a cada nodo (desde la raíz hasta los nodos hoja). El procedimiento de asignación de clase debe hacerse por medio de una función de asignación, en la que se tiene en cuenta la probabilidad a priori asignada a cada clase (según la base de datos empleada), la pureza de la partición y la proporción final de casos que aparecen en los nodos hojas. Al igual que puede determinarse la pureza para un nodo concreto, puede evaluarse de forma conjunta para todo el árbol.

El crecimiento de un árbol continúa hasta que se produce cualquiera de estas 3 posibilidades: sólo hay una observación (caso) en cada nodo hoja, todas las observaciones tienen la misma probabilidad asignada en los nodos hoja (es imposible determinar el criterio de máxima pureza), o se ha fijado un límite externo de la profundidad (número de niveles máximo) del crecimiento del árbol. El árbol que se ha generado de esta forma clasifica correctamente los registros utilizados en su proceso de aprendizaje (se dice que este «sobre aprendizaje» se obtiene porque el modelo ha «sobre ajustado» los datos empleados en esta fase), pero cuando se enfrente a nuevos registros no se asegura su capacidad de generalización.

El árbol complejo que se ha creado debe simplificarse para que alcance esta capacidad de generalización. Se utiliza un método de podado del árbol. El procedimiento asegura que sólo se retiran los nodos que incrementan muy poco la precisión del árbol. Se utiliza una medida de coste-complejidad (que combina los criterios de precisión frente a complejidad en el número de nodos y velocidad de procesamiento), buscando el árbol que obtiene menor valor en este parámetro. Los árboles más sencillos (podados con este criterio) aseguran una mayor capacidad de generalización.

De todos los árboles podados posibles debe seleccionarse el mejor. El mejor árbol (árbol solución) será el que consigue menor error en el ajuste de los registros utilizados en su proceso de aprendizaje. Pero esta condición no es suficiente, debe ajustar bien la base de datos utilizada en su aprendizaje, pero también debe ajustar registros no empleados en esta fase. Para conseguir este objetivo hay diversos métodos.

El método más común (implementado en los programas de mayor utilización), que no precisa un conjunto de prueba independiente, se denomina validación cruzada. La validación cruzada es un método de Re-muestreo que aprovecha el total de la información disponible en la base de datos sin prescindir de una parte de sus registros.

En la validación cruzada se divide de forma aleatoria la base de datos utilizada en la fase de aprendizaje (conjunto de aprendizaje) en N partes (normalmente 10). De forma secuencial, cada uno de estos subconjuntos se reserva para emplearse como conjunto de prueba frente al modelo de árbol generado por los N-1 subconjuntos restantes. Obtenemos así N modelos diferentes, donde se puede evaluar la precisión de las clasificaciones tanto en el conjunto de aprendizaje (N-1) como en los subconjuntos de prueba (N), y podemos seleccionar el árbol óptimo cuando la precisión se alcance tanto en uno como en otro subconjunto.

Las diferencias principales entre los distintos algoritmos de construcción de árboles de decisión radican en las estrategias de poda y en la regla adoptada para partir los nodos. (Gaceta Sanitaria, n.d.)

* 1. Algoritmos
  2. Técnicas

1. Validación del modelo.
2. Ejecución del modelo.