Manual de Técnico – Nations Maneuver

Contenido

[Introducción 2](#_Toc180443676)

[Capítulo 1 Arquitectura, tecnologías y despliegue 3](#_Toc180443677)

[1.1. Arquitectura del sistema 3](#_Toc180443678)

[1.2. Tecnologías usadas 4](#_Toc180443679)

[1.2.1. Unity 4](#_Toc180443680)

[1.2.2. PlayerPrefs 4](#_Toc180443681)

[1.3. Despliegue del sistema 4](#_Toc180443682)

[Capítulo 2 Código del proyecto 5](#_Toc180443683)

[2.1. Git 5](#_Toc180443684)

[2.2. GitHub 6](#_Toc180443685)

[2.3. Estructura del sistema 6](#_Toc180443686)

[2.3.1. Directorios Unity 6](#_Toc180443687)

[Capítulo 3 Implementación del sistema 8](#_Toc180443688)

[3.1. Modelo de base de datos 8](#_Toc180443689)

[3.2. Clonación de proyectos con Git y GitHub 9](#_Toc180443690)

[Capítulo 4 Pruebas 10](#_Toc180443691)

[Anexos 12](#_Toc180443692)

[Documento de Python 12](#_Toc180443693)

# Introducción

Pdte introducción de proyecto.

# Capítulo 1 Arquitectura, tecnologías y despliegue

Nations Maneuver es una aplicación móvil gamificada que apoya e incentiva al usuario al aprendizaje del área de la macroeconomía, a través de una interfaz que permite calcular las jugadas y llevar un seguimiento de cada partida y de esta manera el ganador del juego se determina según el índice de prosperidad económica obtenido, el cual se calcula a partir de las variables de resultado, inflación, tasa de desempleo y balance presupuestal.

Por lo tanto, es clave identificar todas las tecnologías que se usaron para la construcción de esta, con el fin de proveer el mantenimiento adecuado cuando así se requiera. Además, saber la organización general del proyecto, resaltando la comunicación entre los diferentes módulos y sistemas que lo conforman para su correcta ejecución. Por último, es importante tener conocimiento de la manera en que todo el sistema es desplegado para que los usuarios finales puedan hacer uso del mismo.

A continuación, se muestra la arquitectura del sistema, enseñando el lugar que corresponde a cada una de las tecnologías. Por último, se realiza una descripción del despliegue del sistema.

## Arquitectura del sistema

Este diagrama proporciona una visión integral y práctica de las tecnologías utilizada en la arquitectura del sistema (Ver [Figura 1](#Figura_1)). Además, representa cómo los diferentes componentes del sistema se relacionan entre sí y cómo se comunican entre sí para lograr los objetivos del sistema.

Se plantea un juego desarrollado en Unity3D, el cual trabaja de manera local en el dispositivo Android. Finalmente, hace uso de la base de datos temporal y local llamada *PlayerPrefs* que ofrece Unity3D.

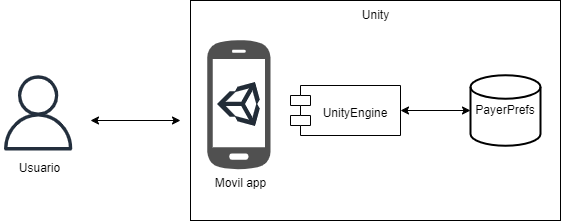


Figura 1: Diagrama de Arquitectura de Integración del sistema.

## Tecnologías usadas

### Unity

Unity es la plataforma de desarrollo 3D en tiempo real líder en el mundo para la creación y explotación de este tipo de contenidos. Es ampliamente utilizado para crear juegos 3D, experiencias de realidad virtual y aumentada, y aplicaciones interactivas. Entre otras cosas, facilita la implementación de contenidos digitales en más de 25 plataformas y tecnologías líderes del sector para llegar al mayor público posible. Finalmente, posibilita a artistas, diseñadores y desarrolladores trabajar juntos para crear experiencias inmersivas e interactivas.

La elección de Unity dependió de las necesidades específicas del proyecto. Dada la popularidad en la industria, las facilidades a la hora de desarrollar juegos (2D y 3D) y la compatibilidad con diferentes plataformas, se optó por el uso Unity para crear el juego 2D para dispositivos móviles con sistema operativo Android. Por lo tanto, en esta se desarrolló la aplicación móvil (juego) para los pacientes.

### PlayerPrefs

PlayerPrefs es una clase de Unity que proporciona una forma sencilla de almacenar y recuperar datos persistentes en una aplicación. Utiliza un sistema de clave-valor para almacenar los datos. Cada valor se almacena asociado a una clave única que se utiliza para recuperarlo más tarde. Además, estos datos se almacenan en el dispositivo del usuario y pueden ser accesibles entre diferentes sesiones del juego o de la aplicación. Este tipo de almacenamiento permite guardar y recuperar información de manera local.

Es importante tener en cuenta que PlayerPrefs no es una base de datos completa y solo se recomienda para almacenar datos simples y de baja complejidad. Por ende, se utilizó para almacenar datos temporales de la partida, de modo que al comenzar una nueva partida no sea necesario tenerlos en cuenta.

## Despliegue del sistema

El despliegue del sistema se refiere a todas las actividades necesarias en pro de colocar el sistema en un estado en el que los usuarios interesados puedan usarlo. Nations Maneuver al ser una aplicación que trabaja de manera local, se debe descargar el APK para ser instalada en el dispositivo Android.

# Capítulo 2 Código del proyecto

## Git

Con el propósito de implementar el proyecto siguiendo buenas prácticas de desarrollo de software, se optó por llevar a cabo versionamiento del mismo. Esta elección implicaba la creación de un repositorio en el que se almacenasen los distintos archivos que conformaban el proyecto, así como el registro de cada modificación, por pequeña que fuese. Para llevar a cabo este versionamiento, se seleccionó Git como la herramienta adecuada.

Git es un software de control de versiones creado por Linus Torvalds, el mismo individuo que dio vida al sistema operativo Linux. Esta herramienta fue concebida con el objetivo de garantizar eficiencia y fiabilidad en la gestión de versiones de proyectos de software que contaban con un elevado número de archivos de código fuente. Más allá de monitorizar los cambios realizados en un proyecto, Git facilita la coordinación del trabajo de un grupo de desarrolladores sobre archivos compartidos. Git es un proyecto de software libre, liberado bajo la licencia GPL o Licencia Pública General de GNU.

Git ofrece varias características significativas, tales como:

* Gestión de ramas o "branches," que posibilita la realización de diversos cambios en diferentes partes del proyecto, evitando conflictos entre tales modificaciones.
* Gestión distribuida: una vez que se completan los cambios en las diferentes ramas, es posible fusionarlas o "merge" para integrar todas las actualizaciones realizadas en el proyecto.
* Eficiente gestión de proyectos de gran envergadura mediante el uso de "snapshots" para llevar a cabo el versionamiento.
* Capacidad para navegar a través de las diversas ramas y "commits" realizados, permitiendo visualizar las distintas actualizaciones.
* Interfaz amigable en comparación con otros sistemas de control de versiones, como Subversion.

Para instalar la herramienta abre tu navegador web y visita el sitio oficial de Git en <https://git-scm.com/>. Descarga el instalador correspondiente a tu sistema operativo y una vez que se haya descargado el archivo de instalación, ábrelo haciendo doble clic en él. Aparecerá una ventana de instalación. Sigue las instrucciones y haz clic en "Next" (Siguiente) para continuar hasta finalizar la instalación. Verifica la instalación y abre una terminal o línea de comandos. Escribe el siguiente comando y presiona Enter:

|  |
| --- |
| **$ git --version** |

Deberías ver la versión de Git instalada en tu sistema, lo que confirma que la instalación fue exitosa.

## GitHub

Github es una plataforma web en la que se puede llevar a cabo el versionamiento remoto de proyectos de manera gráfica. Ofrece la ventaja de tener una versión del proyecto almacenada en la nube. Esto resulta especialmente útil en situaciones donde surgen problemas con la versión local, ya que la versión remota siempre está disponible para su acceso y continuación del trabajo.

Una característica importante de Github es su capacidad para trabajar en proyectos en colaboración y en equipos distribuidos. Esto se logra mediante la clonación de repositorios a través de Git, lo que permite a múltiples personas trabajar simultáneamente en el mismo proyecto. Además, Github también utiliza conceptos como "branches" (ramas) para evitar conflictos en los cambios realizados por diferentes colaboradores. De esta manera, cada persona puede trabajar en su propia rama de desarrollo y luego fusionar los cambios de manera ordenada. El enlace del repositorio es <https://github.com/leezym/Nations_Maneuver>.

## Estructura del sistema

En esta sección se presenta la estructura de directorios de cada uno de los proyectos pertenecientes a UBICU mostrados en la Figura 3. Gracias a la estructura de directorios los proyectos se pueden ejecutar de manera correcta.

### Directorios Unity

Al crear un proyecto en Unity, la estructura de carpetas y archivos sigue ciertas convenciones para organizar el código de manera efectiva (Ver [Figura 2](#Figura_6)).

* **Assets:** Esta es la carpeta principal donde se almacenan todos los activos de tu juego. Dentro de esta carpeta, puedes organizar los activos en subcarpetas según su tipo y función:
  + **Animations:** En esta carpeta van elementos relacionados con la animación de objetos 2D o 3D.
  + **Mockups**: En esta carpeta se colocan los bocetos de las interfaces, los cuales sirven de guía.
  + **Packages:** En esta carpeta, Unity almacena las bibliotecas y paquetes externos que has importado a tu proyecto.
  + **Scenes:** Aquí es donde se almacenan las escenas de tu juego. Cada escena representa un nivel, menú u otra parte jugable de tu juego.
  + **Scripts:** En esta carpeta, puedes almacenar todos los scripts de C# que controlan la lógica de tu juego.
  + **Sprites**: En esta carpeta se colocan elementos relacionados con la interfaz de usuario, como imágenes y prefabs de elementos de UI.
  + **TextMesh Pro**: Puedes colocar los archivos y recursos relacionados con el plugin TextMesh Pro aquí.
  + **Typography**: Se almacenan los elementos gráficos de las fuentes de texto.
* **ProjectSettings:** Contiene archivos de configuración específicos de tu proyecto, como configuraciones de calidad, configuraciones de input y otros ajustes globales.
* **UserSettings:** En esta carpeta, se almacenan los archivos de configuración personalizados del usuario. Se pueden tener subcarpetas o archivos individuales para organizar diferentes tipos de configuraciones.
* **Library:** Esta carpeta contiene archivos generados automáticamente por Unity para administrar el estado del proyecto. No necesitas editar ni modificar estos archivos directamente.
* **Temp:** Unity utiliza esta carpeta para almacenar archivos temporales. No necesitas preocuparte por esta carpeta en términos de organización manual.
* **.gitignore:** Si estás utilizando control de versiones (como Git), asegúrate de tener un archivo .gitignore en la raíz de tu proyecto para evitar que archivos y carpetas innecesarios se incluyan en el repositorio.

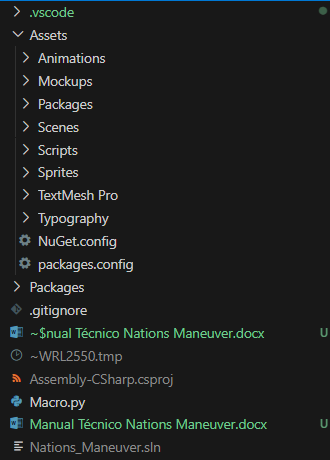


Figura 2: Directorio raíz de Unity

# Capítulo 3 Implementación del sistema

En este capítulo se describen los detalles importantes y a tener en cuenta relacionados a la implementación de cada una de las partes del proyecto Nations Maneuver. Se explica la codificación base con la tecnología correspondiente. Además, se muestra el modelo de base de datos y la forma en la que se pueden clonar los proyectos para trabajar en ellos.

## Modelo de base de datos

A continuación, en la [Figura 3](#Figura_15) se muestra la función *SaveLocalData* que permite guardar variables de juego como: Gasto, Tasa impositiva, Bonos, Oferta monetaria, entre otras. En primer lugar, por cada turno del jugador se deben guardar dichos datos para que a lo largo del juego se actualice su información. Así, el ganador del juego se determina según el índice de prosperidad económica obtenido, el cual se calcula a partir de las variables de resultado, inflación, tasa de desempleo y balance presupuestal.

Para almacenar datos se utiliza la sintaxis, donde XXX representa el tipo de dato que se quiere almacenar (por ejemplo, “Int” para un número entero):

PlayerPrefs.SetXXX(“clave”, valor);

Para recuperar los datos, se utiliza la sintaxis, donde XXX representa el tipo de dato que se quiere recuperar:

PlayerPrefs.GetXXX(“clave”);

public void SaveLocalData()

    {

        PlayerPrefs.SetInt("shift", shift);

        PlayerPrefs.SetString("G", textGasto.text); // gasto

        PlayerPrefs.SetString("t", textTasaImp.text ); // tasa impositiva

        PlayerPrefs.SetString("M", textOma.text); // bonos

        string formattedInitialGuess = "";

        for (int i = 0; i < initialGuess.Length; i++)

        {

            formattedInitialGuess += initialGuess[i].ToString().Replace(",", ".");

            if (i < initialGuess.Length - 1)

            {

                formattedInitialGuess += "-";

            }

        }

        PlayerPrefs.SetString("initialGuess", formattedInitialGuess);

    }

Figura 3: Colecciones MongoDB.

## Clonación de proyectos con Git y GitHub

La clonación con las herramientas Git y GitHub se utiliza principalmente para obtener una copia de un proyecto remoto y ubicarla en un repositorio local. Con el fin realizar un desarrollo colaborativo al poder trabajar en el proyecto desde diferentes equipos, y así, crear diferentes versiones del proyecto hasta unir todos los cambios y obtener la versión final.

Se requiere tener instalado Git y la URL del repositorio en Git. Abre tu navegador y visita la página del repositorio en GitHub que deseas clonar. Haz clic en el botón "Code" (o "Código") que se encuentra en la parte superior derecha del repositorio. Selecciona la opción "HTTPS" y copia la URL que se muestra. Debería tener un formato similar a: <https://github.com/usuario/nombre-repositorio.git>.

Abre una terminal o línea de comandos en tu sistema y navega al Directorio de Destino utilizando el comando cd para navegar al directorio en el que deseas clonar el repositorio. Por ejemplo, si deseas clonar el repositorio en tu directorio de inicio:

|  |
| --- |
| **$ cd ~** |

En la terminal, utiliza el siguiente comando para clonar el repositorio:

|  |
| --- |
| **$ git clone URL\_DEL\_REPOSITORIO** |

Reemplaza *URL\_DEL\_REPOSITORIO* con la URL que copiaste al inicio. Presiona Enter para ejecutar el comando. Git descargará todos los archivos y el historial del repositorio en el directorio seleccionado.

Una vez completada la clonación, utiliza el comando cd para ingresar al directorio del repositorio clonado:

|  |
| --- |
| **$ cd nombre-repositorio** |

Reemplaza *nombre-repositorio* con el nombre del repositorio clonado

# Capítulo 4 Pruebas

Para el desarrollo de las pruebas matemáticas se hace uso del Framework de Unity llamado Unity Test. El proyecto hizo entrega de los insumos y fórmulas ([Anexo: Documento de Python](#_Documento_de_Python)) a tener en cuenta en el desarrollo del juego, y de esta manera se creó código que permita calcular y probar estas fórmulas de manera automática el script *EconomicModel.cs*.

using System;

using System.Collections;

using NUnit.Framework;

using UnityEngine;

using UnityEngine.TestTools;

using MathNet.Numerics.RootFinding;

public class TestEconomicModel

{

    private EconomicModel economicModel;

    [SetUp]

    public void Setup()

    {

        GameObject gameObject = new GameObject();

        economicModel = gameObject.AddComponent<EconomicModel>();

        economicModel.SetShift(1);

        economicModel.initialGuess = new double[] { 800, 0.05, 1.1 };

        economicModel.finalGuess = new double[] { 900, 0.06, 1.2 };

        economicModel.G = 150;

        economicModel.t = 0.1;

        economicModel.M = 5000;

        economicModel.a = 160;

        economicModel.f = 100;

        economicModel.w = 50;

        economicModel.ka = 30000;

        economicModel.L = 225;

        economicModel.A = 1;

        economicModel.x1 = 0.2;

        economicModel.x2 = 30;

        economicModel.m1 = 0.06;

        economicModel.m2 = 10;

        economicModel.y\_ex = 20000;

        economicModel.r\_ex = 0.05;

        economicModel.c = 0.6;

        economicModel.b = 1500;

        economicModel.k = 0.2;

        economicModel.h = 1000;

        economicModel.alpha = 0.5;

        economicModel.rho = 10;

        economicModel.e\_0 = 2;

    }

    [Test]

    public void TestDefineSimplifications()

    {

        double G = 150;

        double t = 0.1;

        double M = 5000;

        economicModel.DefineSimplifications(G, t, M);

        Assert.AreEqual(160 + 100 + G + 0.2 \* 20000, economicModel.omega);

        Assert.AreEqual(1 - 0.6 \* (1 - t) + 0.06, economicModel.gamma);

        Assert.AreEqual(1 / 0.5, economicModel.q);

        Assert.AreEqual((1 - 0.5) / 0.5, economicModel.u);

        Assert.AreEqual(0, economicModel.pi\_e);

        Assert.AreEqual(30 - 10, economicModel.sigma);

    }

    [Test]

    public void TestSolveEquations()

    {

        double G = 150;

        double t = 0.1;

        double M = 5000;

        double[] initialGuess = new double[] { 800, 0.05, 1.1 };

        double[] expectedSolution = new double[] { 8030.99, 1.41, 26.76};

        economicModel.DefineSimplifications(G, t, M);

        double[] solution = economicModel.SolveEquations(initialGuess);

        Assert.AreEqual(expectedSolution[0], solution[0], 0.01); // PIB (y)

        Assert.AreEqual(expectedSolution[1], solution[1], 0.01); // Tasa de interés (r)

        Assert.AreEqual(expectedSolution[2], solution[2], 0.01); // Precio (p)

    }

    [Test]

    public void TestCalculateReport()

    {

        economicModel.CalculateReport();

        Assert.AreEqual(12.50f, Math.Round(economicModel.variacionPIB, 2)); // variación PIB

        Assert.AreEqual(9.09f, Math.Round(economicModel.inf), 2); // inflación

        Assert.AreEqual(0.06f, Math.Round(economicModel.r), 2); // tasa de interés

        Assert.AreEqual(-60.0f, Math.Round(economicModel.BF), 2); // balance fiscal

    }

}

La función *Setup()* permite inicializar y configurar los valores que existen en *EconomicModel.cs* y que se van a trabajar en el archivo de pruebas *TestEconomicModel.cs*. Además, tenemos tres funciones que permiten probar de manera independiente las fórmulas inicializando un valor de inicio y un valor esperado: *TestDefineSimplifications()*, *TestSolveEquations()* y *TestCalculateReport()*.

# Anexos

## Documento de Python

