



# Tecnológico de Monterrey

## MOVILIDAD URBANA

M5

Documento final

### Integrantes

Luis Enrique Bojórquez Almazán	A01336625
Marco Antonio Bosquez González	A01653247
Christian Jesús González Ramírez	A01657929

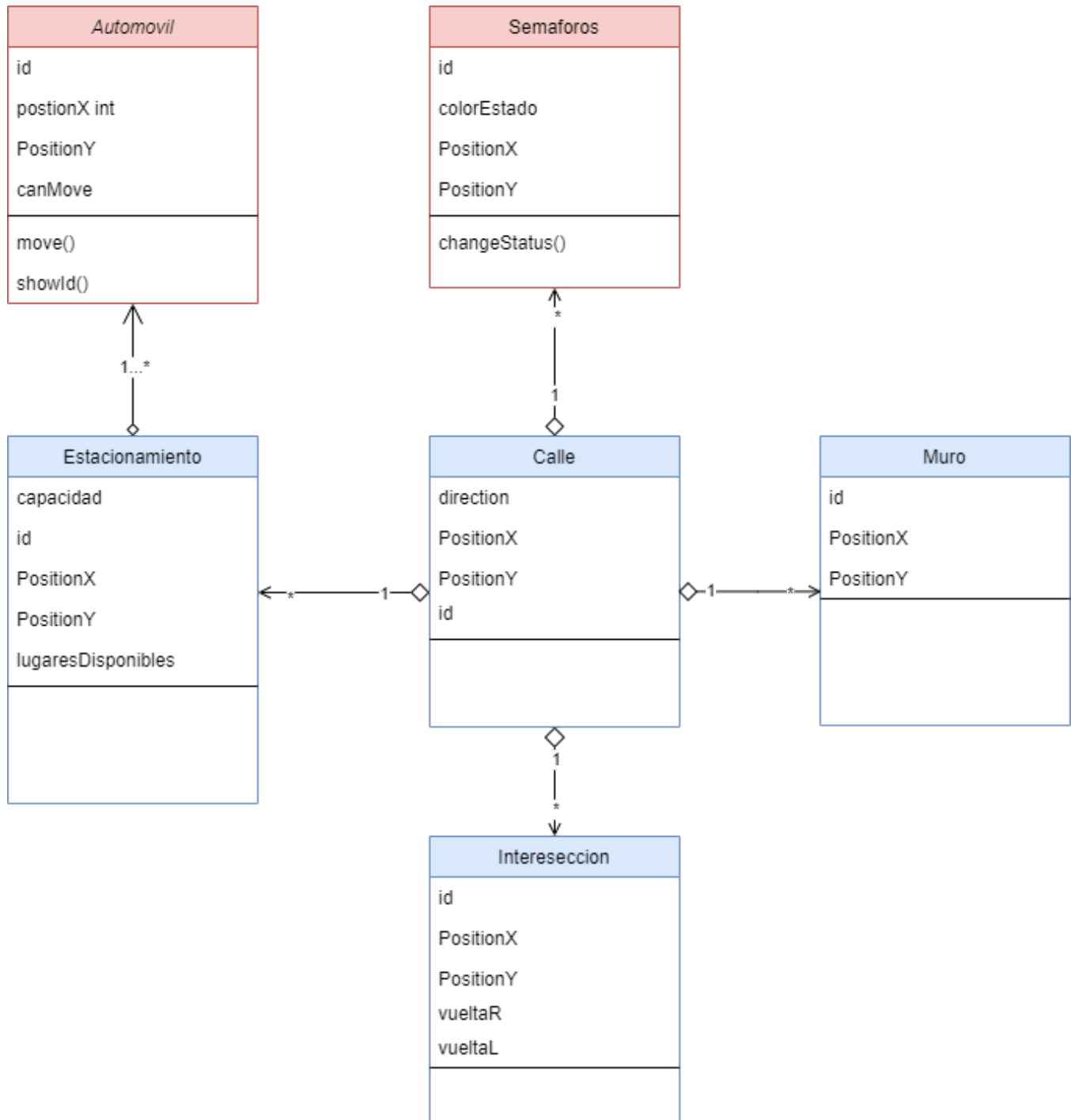
### Profesores

Sergio Ruiz Loza  
David Christopher Balderas Silva

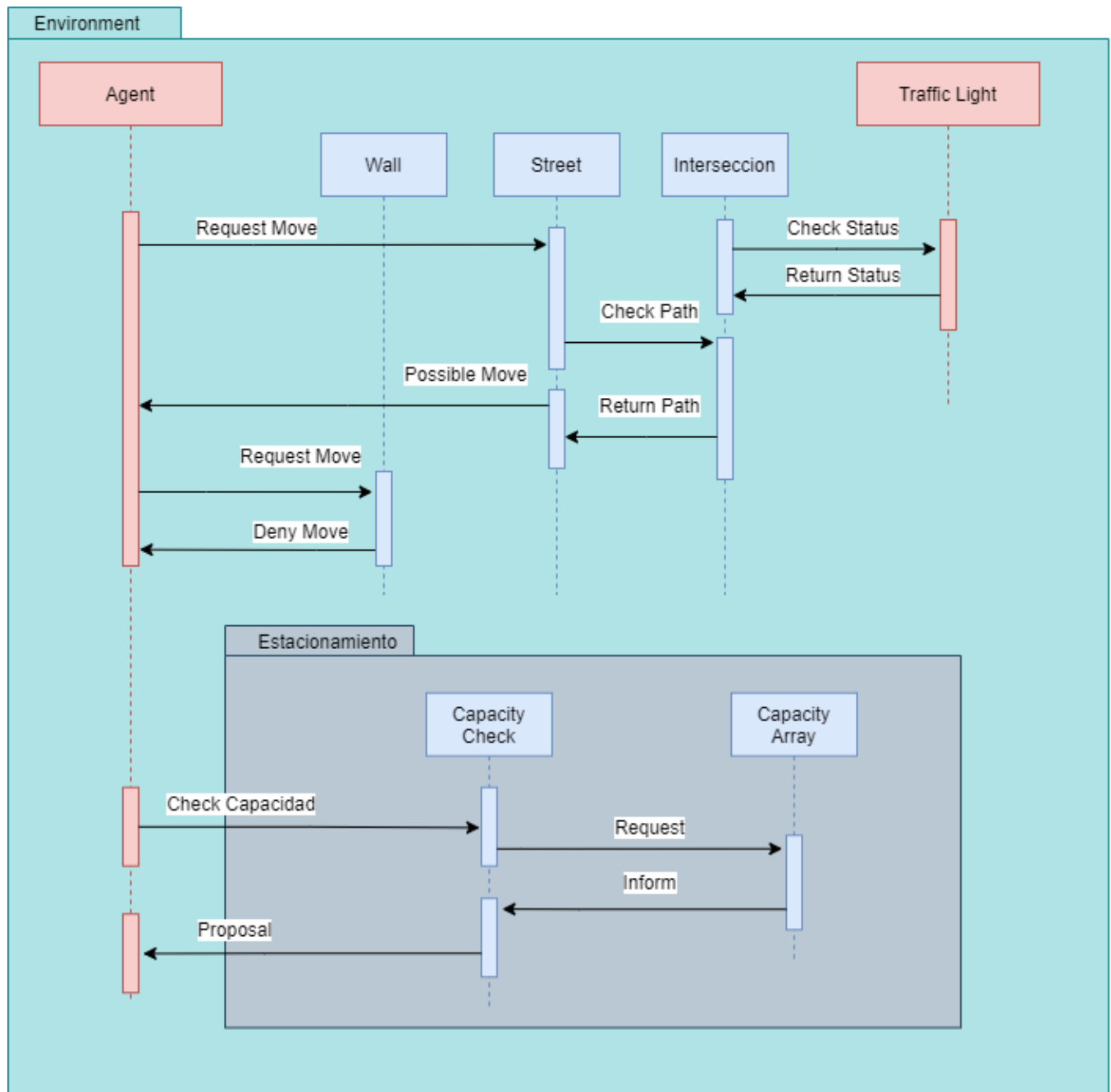
Diagramas	2
Implementación de los agentes	3
Implementación de la interfaz gráfica de la simulación	3
Documentación	4
Análisis de la solución desarrollada	4
Reflexión sobre el proceso de aprendizaje	4
Referencias	5

# Diagramas

## Diagramas de clase



## Diagrama de protocolos de interacción



# Implementación de los agentes

Código en Replit: [Movilidad Urbana](#)

## Implementación Elementos del Ambiente

```
1  from mesa import Agent
2  import random
3
4  # Visual Node
5  class VisualNode(Agent):
6      def __init__(self, x, y):
7          super().__init__(x, y)
8
9  # Visual Board
10 class Road(Agent):
11     def __init__(self, unique_id, model, x, y):
12         super().__init__(unique_id, model)
13         # direccion aqui
14         self.dir = [] # U D L R
15
16 # Visual Board
17 class Wall(Agent):
18     def __init__(self, x, y):
19         super().__init__(x, y)
20
21 # ESTACIONAMIENTO
22 class ParkingLot(Agent):
23     def __init__(self, pos, capacity=10):
24         super().__init__(pos, capacity)
25         self.pos = pos
26         self.capacity = capacity
27         #self.used = 1
28         self.used = random.randint(0,2)
29
30     def hasCapacity(self):
31         if self.capacity > self.used:
32             return True
33         else:
34             return False
35
36     def park(self):
37         self.used += 1
38
39         if self.used == self.capacity:
40             return 'Last Car'
41
```

## Implementación Agente Semáforo

```
# LUCES DE TRAFICO
class TrafficLight(Agent):
    def __init__(self, u_id, model, pos, horiz=True):

        super().__init__(u_id, model)
        self.pos = pos
        self.horiz = horiz
        self.lights = []
        self.current_cycle = 1
        self.counter = 0

    def status(self):
        if self.lights[0].current_cycle == 0:
            self.current_cycle = 1
        else:
            self.current_cycle = 0

    def check(self):
        # Checamos status de las luces paralelas
        self.status()

        # Checamos direccion de calle para saber donde checar a los agentes
        dirRoad = self.model.grid.get_cell_list_contents((self.pos[0], self.pos[1]), True)[0].dir

        self.x = 0
        self.y = 0

        if 'D' in dirRoad:
            self.y = 1
        elif 'R' in dirRoad:
            self.x = -1
        elif 'U' in dirRoad:
            self.y = -1
        elif 'L' in dirRoad:
            self.x = 1

        vehicle_cell = self.model.grid.get_cell_list_contents((self.pos[0] + self.x, self.pos[1] + self.y), True)

        if self.counter == 3:
            self.current_cycle = 0
            self.counter = 0
            self.lights[0].current_cycle = 1
```

## Implementación Agente Vehículo

```
# VEHICULOS
class Vehicles(Agent):
    def __init__(self, u_id, model, pos):
        super().__init__(u_id, model)
        self.original_pos = pos
        self.pos = self.original_pos
        self.x = self.pos[0]
        self.y = self.pos[1]
        self.canMove = True # Para semaforo
        self.isParked = False

    def step(self):
        self.move()

    def move(self):
        # Cada que se mueva, checar si hay algun estacionamiento alrededor con capacidad
        if not self.isParked:
            self.checkNearby()

        # Checar contenidos de la celda en la que esta el agente
        cell = self.model.grid.get_cell_list_contents((self.x, self.y), True)[0].dir

        # Si la celda en la que esta el agente tiene mas de una direccion, elegir una dir random a ir
        if len(cell) > 1 and self.canMove and not self.isParked:

            pick = cell[random.randint(0, len(cell)-1)]

            if pick == 'R':
                self.x += 1

            elif pick == 'D':
                self.y -= 1

            elif pick == 'L':
                self.x -= 1

            elif pick == 'U':
                self.y += 1

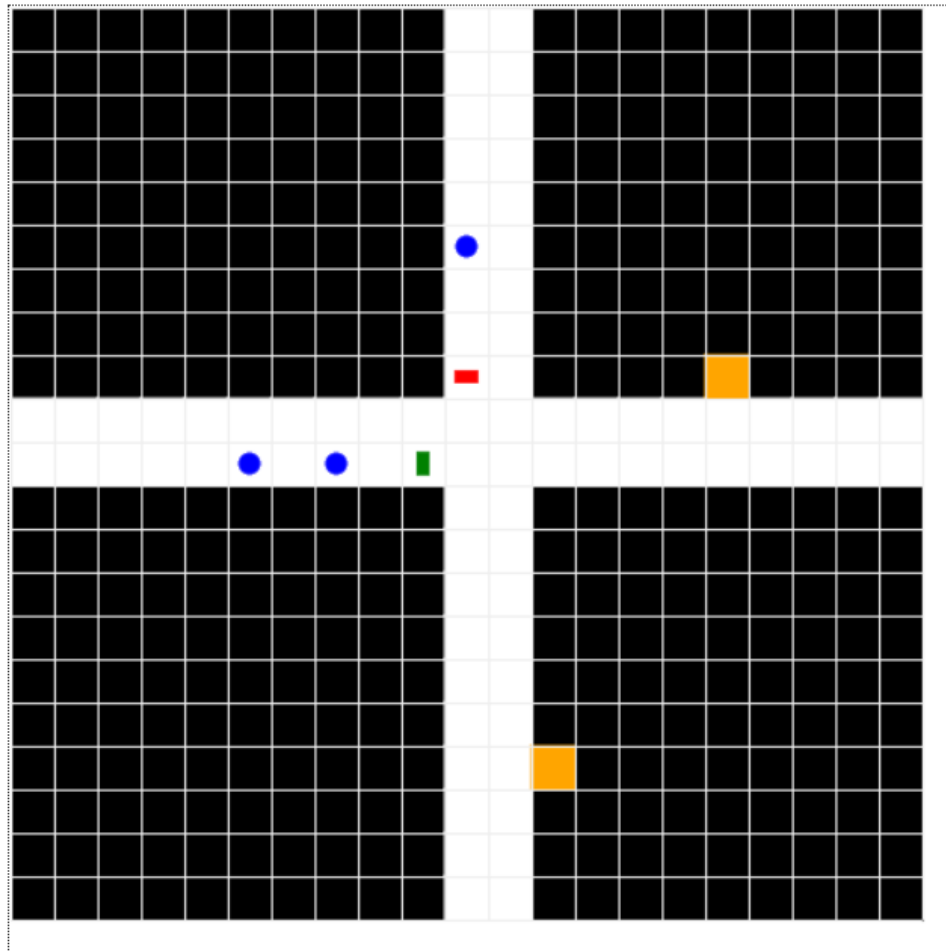
        # Ir a direccion
        if not self.isParked:
            self.model.grid.move_agent(self, (self.x, self.y))
```

# Implementación de la interfaz gráfica de la simulación

Visualización de la implementación usando librería Mesa:  
Traffic Flow (Visualización en Mesa)

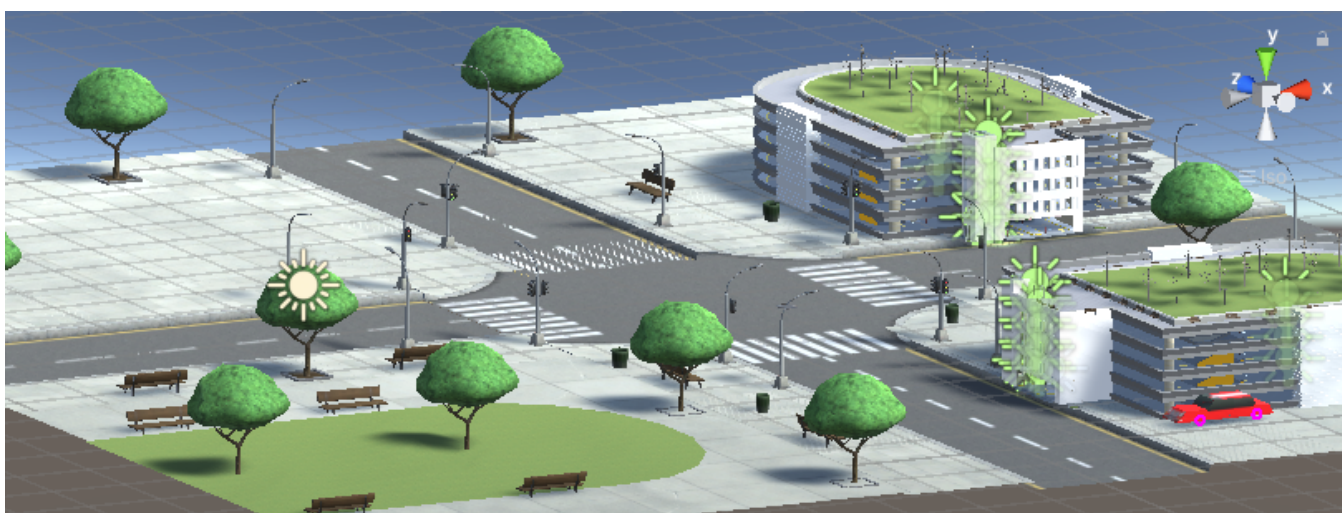
- Círculos Azules: Agente Vehículo
- Rectángulos Verdes y Rojos: Agente Semáforo
- Rectángulos Negros: Muros para bloquear camino
- Rectángulos Blancos: Camino donde se pueden mover los vehículos
- Rectángulos Naranja: Estacionamientos con Capacidad
- Rectángulos Morados: Estacionamientos sin Capacidad

Current Step: 4





Visualización de la implementación en Unity:

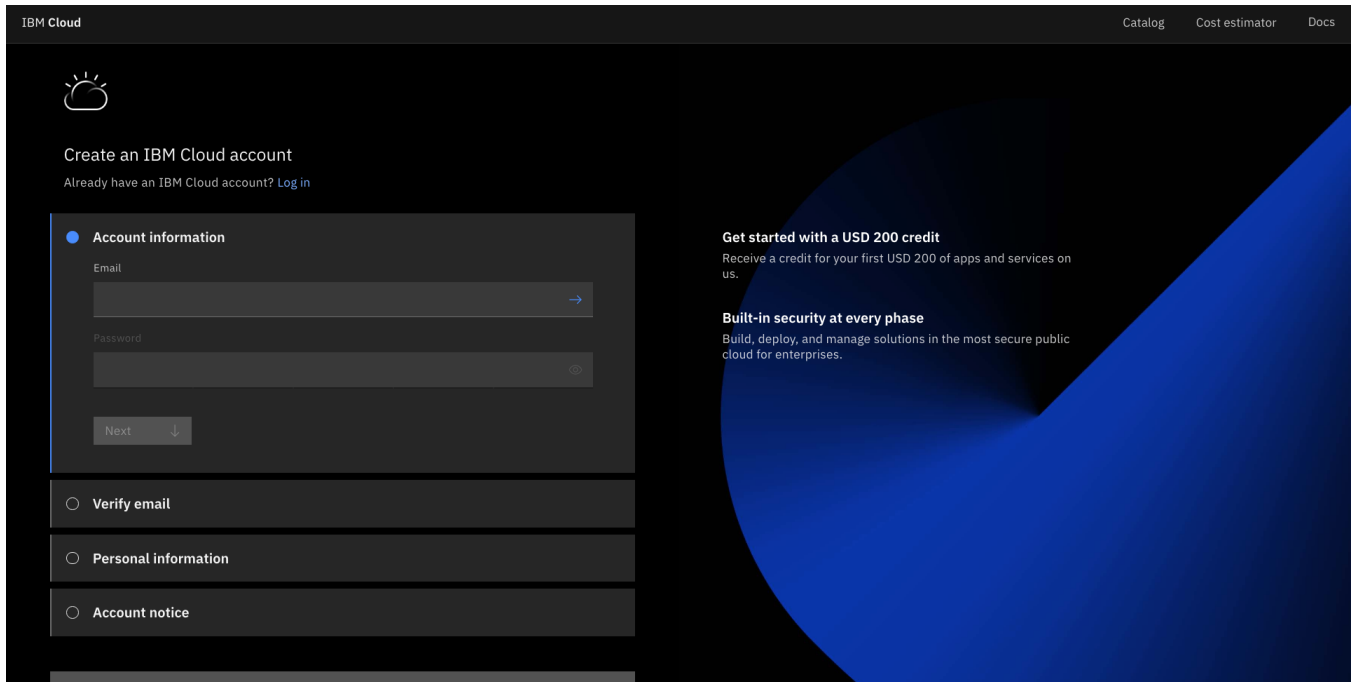


# Documentación

## Instalación

### Paso 1:

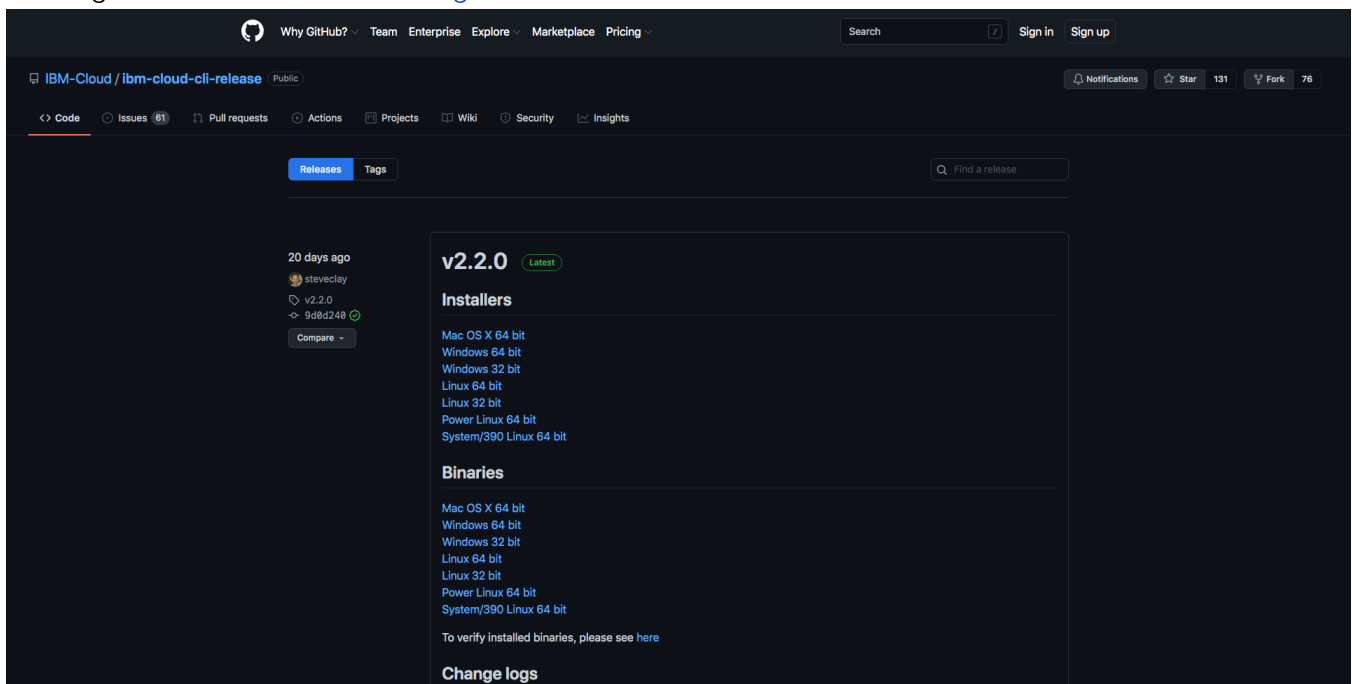
Crear cuenta en IBM Cloud: [cloud.ibm.com/registration](https://cloud.ibm.com/registration).



The screenshot shows the IBM Cloud registration page. At the top, there's a navigation bar with 'IBM Cloud', 'Catalog', 'Cost estimator', and 'Docs'. Below the navigation bar is a large blue and black graphic. The main content area is titled 'Create an IBM Cloud account'. It includes a link 'Already have an IBM Cloud account? Log in'. The registration form is divided into four steps: 'Account information' (selected), 'Verify email', 'Personal information', and 'Account notice'. The 'Account information' step has fields for 'Email' and 'Password', and a 'Next' button. To the right of the form, there are two promotional messages: 'Get started with a USD 200 credit' and 'Built-in security at every phase'.

### Paso 2:

Descarga e instalar IBM Cloud CLI: [github.com/IBM-Cloud/ibm-cloud-cli](https://github.com/IBM-Cloud/ibm-cloud-cli).



The screenshot shows the GitHub repository page for 'IBM-Cloud/ibm-cloud-cli-release'. The page is dark-themed. At the top, there's a navigation bar with 'Why GitHub?', 'Team', 'Enterprise', 'Explore', 'Marketplace', and 'Pricing'. Below the navigation bar is a search bar and 'Sign in' and 'Sign up' buttons. The main content area is titled 'IBM-Cloud / ibm-cloud-cli-release'. It includes a 'Releases' tab and a 'Tags' tab. The 'Releases' tab is selected, showing a list of releases. The latest release is 'v2.2.0', which is marked as 'Latest'. Below the release list, there are sections for 'Installers' and 'Binaries'. The 'Installers' section lists the following operating systems and architectures: Mac OS X 64 bit, Windows 64 bit, Windows 32 bit, Linux 64 bit, Linux 32 bit, Power Linux 64 bit, and System/390 Linux 64 bit. The 'Binaries' section lists the same operating systems and architectures. Below the 'Binaries' section, there is a link 'To verify installed binaries, please see here'. At the bottom, there is a 'Change logs' section.

### Paso 3:

Descargar e instalar Unity: [unity3d.com/download](https://unity3d.com/download).

Unity

Productos

Solutions

Casos de estudio

Conocer

Support & Services

Obtener Unity

Asset Store

Q

Productos

Mi cuenta

Revendedor

Educación

Premium Support

Precios

## Descargar Unity

¡Bienvenido! Está aquí porque desea descargar Unity, la plataforma de desarrollo más popular del mundo para crear juegos multiplataforma y experiencias interactivas 2D y 3D.

Antes de descargar, elija la versión de Unity que sea adecuada para usted.

Elige tu Unity + descargar

Descarga Unity Hub

[Descubrir más acerca del nuevo Unity Hub aquí.](#)

## Descargar Unity Beta

Obtén acceso antes que los demás a nuestras funciones más recientes, y ayúdanos a mejorar la calidad haciéndonos conocer tus valiosos comentarios.

Descargar versión beta

## ¿Ya eres cliente?

Excelente. Descarga Unity aquí si tienes una suscripción Plus o Pro.

Descargar

## Requisitos del sistema

**OS:** Windows 7 SP1+, 8, 10, 64-bit versions only; Mac OS X 10.12+; Ubuntu 16.04, 18.04, and CentOS 7.

**GPU:** Tarjeta de video con capacidad para DX10 (shader modelo 4.0).

Averiguar más

## Lanzamientos

- Long Term Support (LTS) releases
- Learn about Unity 2020 LTS + Unity 2021.1 Tech Stream
- Pre-release technology

## Recursos

- Documentación
- Base de conocimiento
- Soporte

### Paso 4:

Descargar unitypackage y el modelo de multiagentes: [drive.google.com](https://drive.google.com)

Google Drive no puede analizar este archivo en busca de virus.

[integradora.unitypackage.zip](#) (97M) supera el tamaño máximo de archivo que Google puede analizar en busca de virus. ¿Quieres descargar el archivo de todos modos?

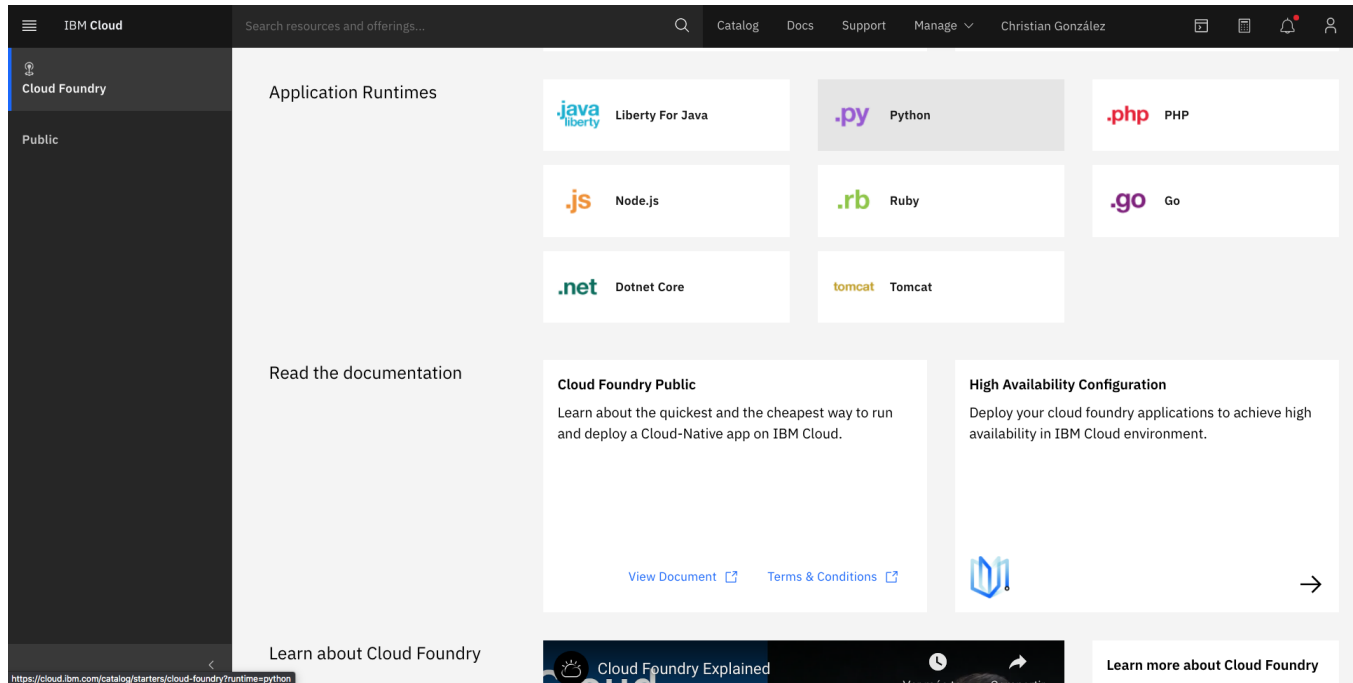
Descargar de todos modos

© 2021 Google - [Ayuda](#) - [Privacidad y condiciones](#)

## Configuración

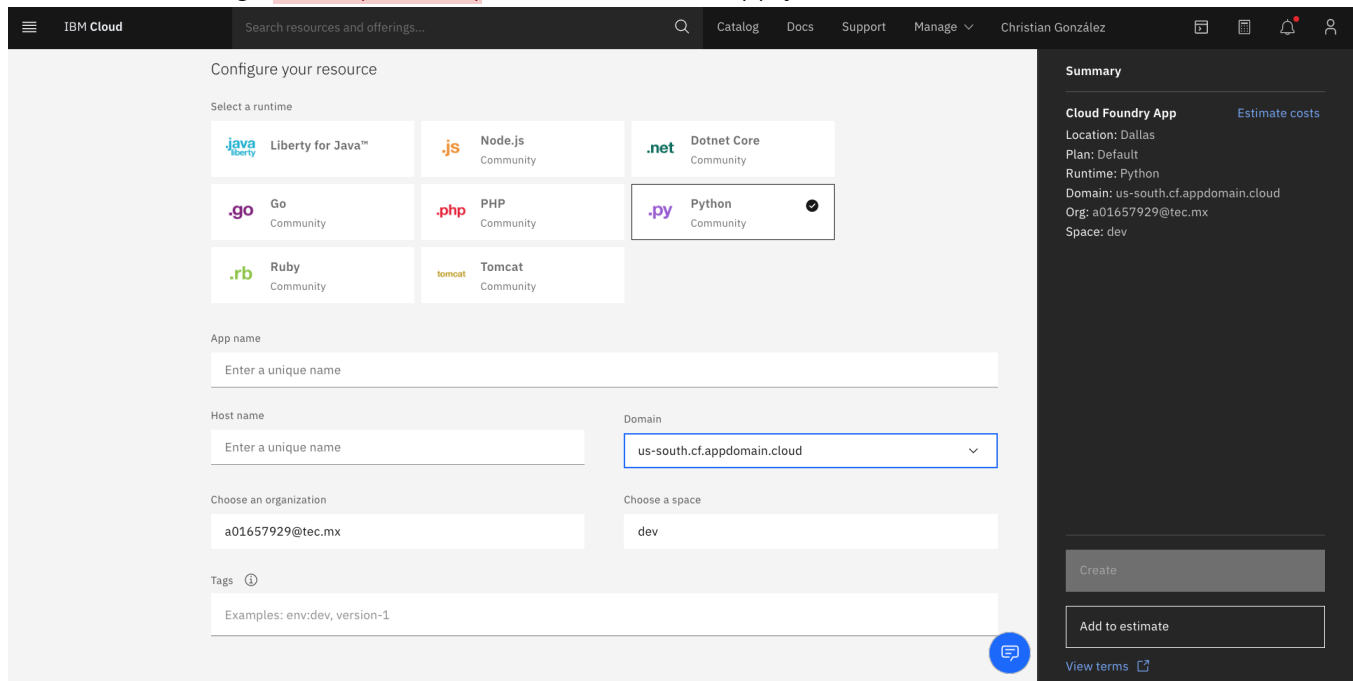
### Paso 1:

Entrar a IBM Cloud y seleccionar nueva app de Cloud Foundry en Python: [cloud.ibm.com/cloudfoundry](https://cloud.ibm.com/cloudfoundry).



### Paso 2:

En localización elegir **Dallas (us-south)**, darle nombre a la app y dar clic en crear.



### Paso 3:

En el CLI iniciar sesión con el comando `ibmcloud login` e introducir mail y contraseña de IBM Cloud y seleccionar la región: `us-south`

### Paso 4:

Ejecutar el comando `ibmcloud cf install`

### Paso 5:

Configurar API endpoint con el comando `ibmcloud api https://api.ng.bluemix.net`

### Paso 6:

Configurar target con el comando `ibmcloud target -cf`

### Paso 7:

Ejecutar el comando `ibmcloud.exe cf apps`

### Paso 8:

Editar el archivo `manifest.yml` y cambiar name por el servidor creado en la nube

### Paso 9:

Ubicarse en la carpeta donde está el zip descargado anteriormente y ejecuta el comando `ibmcloud cf push` y verificar que el resultado diga estado en ejecución

```
Consider adding this directory to PATH or, if you prefer to suppress this warning, use --no-warn-script-location.
WARNING: The script cookiecutter is installed in '/tmp/contents388693823/deps/0/python/bin' which is not on PATH.
WARNING: The script flask is installed in '/tmp/contents388693823/deps/0/python/bin' which is not on PATH.
Successfully installed Flask-2.0.2 Jinja2-3.0.3 MarkupSafe-2.0.1 Werkzeug-2.0.2 arrow-1.2.1 binaryornot-0.4.4
certifi-2021.10.8 chardet-4.0.0 charset-normalizer-2.0.8 click-8.0.3 cookiecutter-1.7.3 idna-3.3 itsdangerous-2.0.1 jinja2-time-0.2.0 mesa-0.8.9 networkx-2.6.3 numpy-1.21.4 pandas-1.3.4 poyo-0.5.0 python-dateutil-2.8.2 python-slugify-5.0.2
pytz-2021.3 requests-2.26.0 six-1.16.0 text-unidecode-1.3 tornado-6.1 tqdm-4.62.3 urllib3-1.26.7
Exit status 0

Waiting for app to start...

name: controlTrafico
requested state: started
routes: controlTrafico.us-south.cf.appdomain.cloud
last uploaded: Tue 30 Nov 22:04:02 CST 2021
stack: cflinuxfs3
buildpacks: python

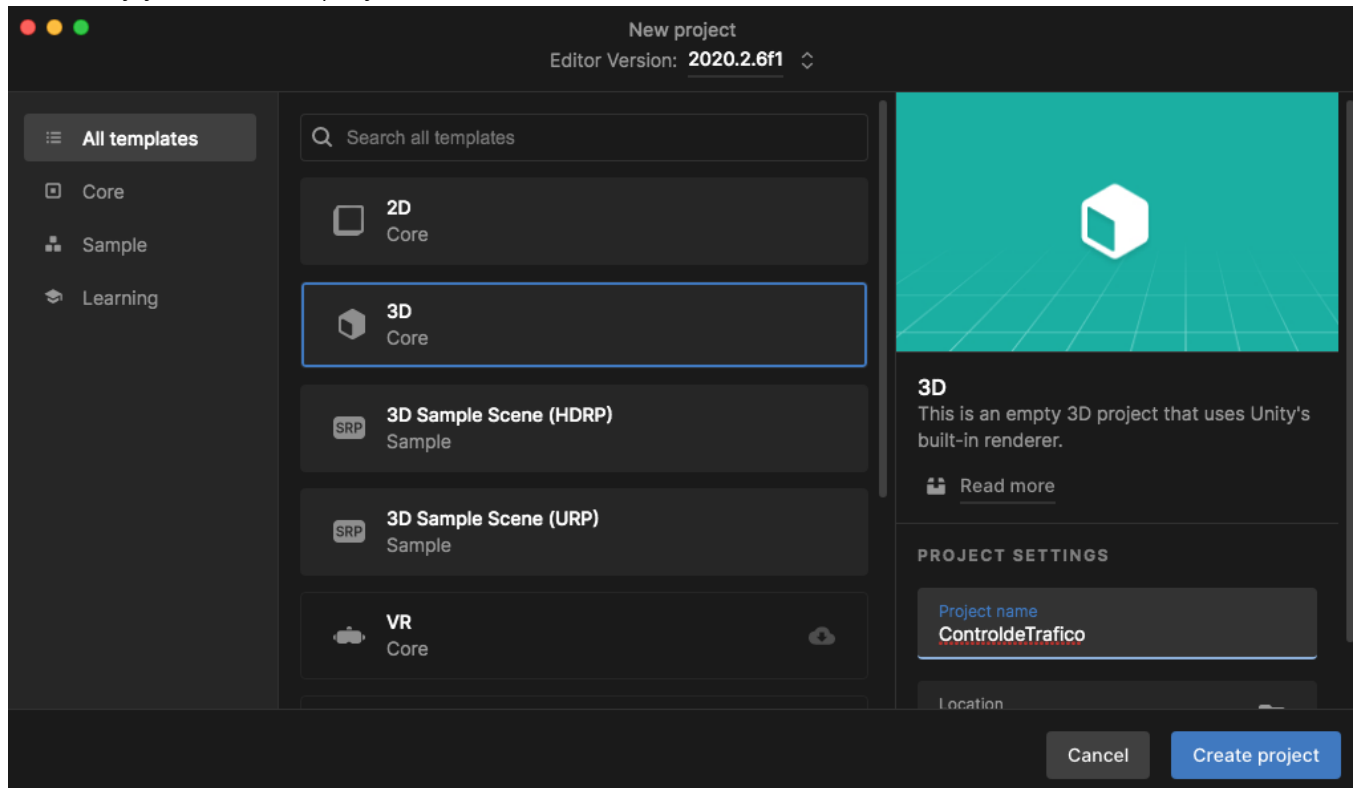
type: web
instances: 1/1
memory usage: 128M
start command: python hello.py

state since cpu memory disk details
#0 running 2021-12-01T04:04:28Z 0.0% 0 of 0 0 of 0
```

## Ejecución

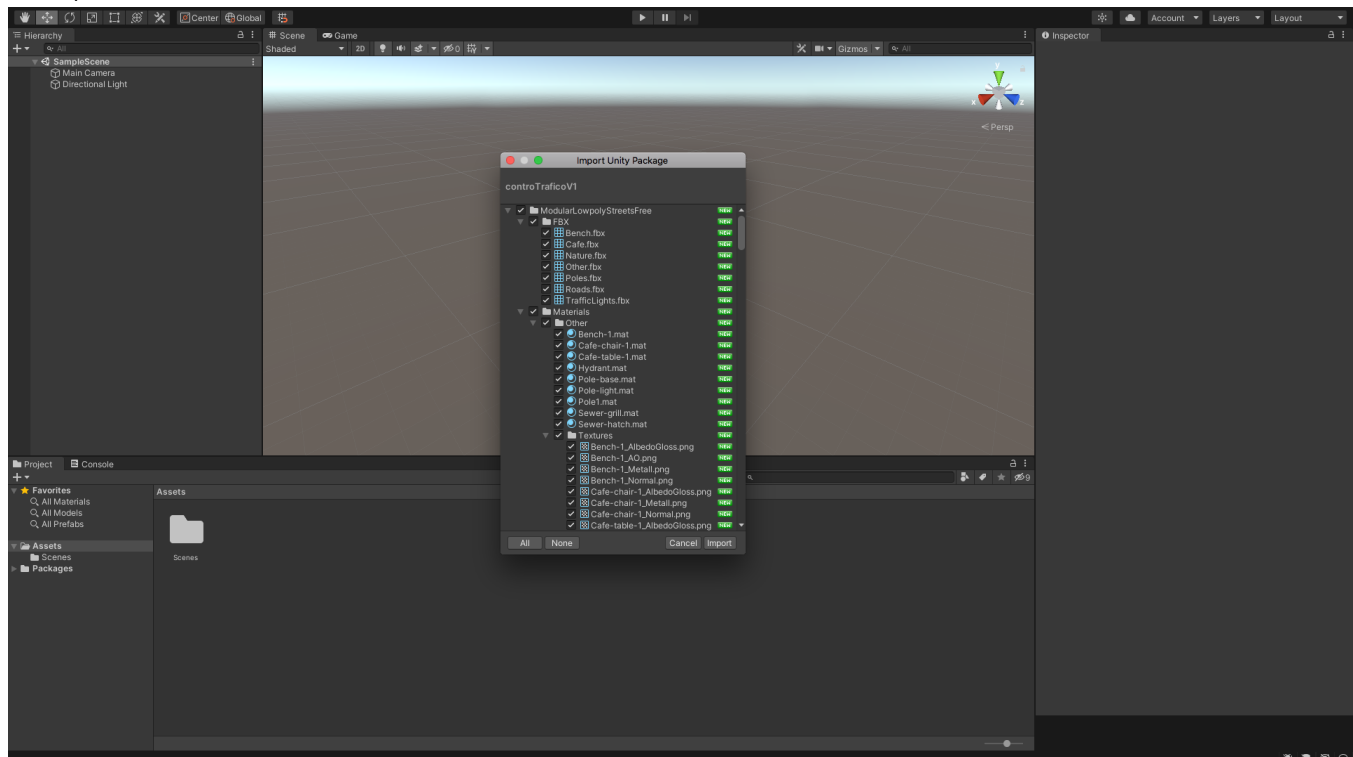
### Paso 1:

Abrir Unity y crear nuevo proyecto en 3D



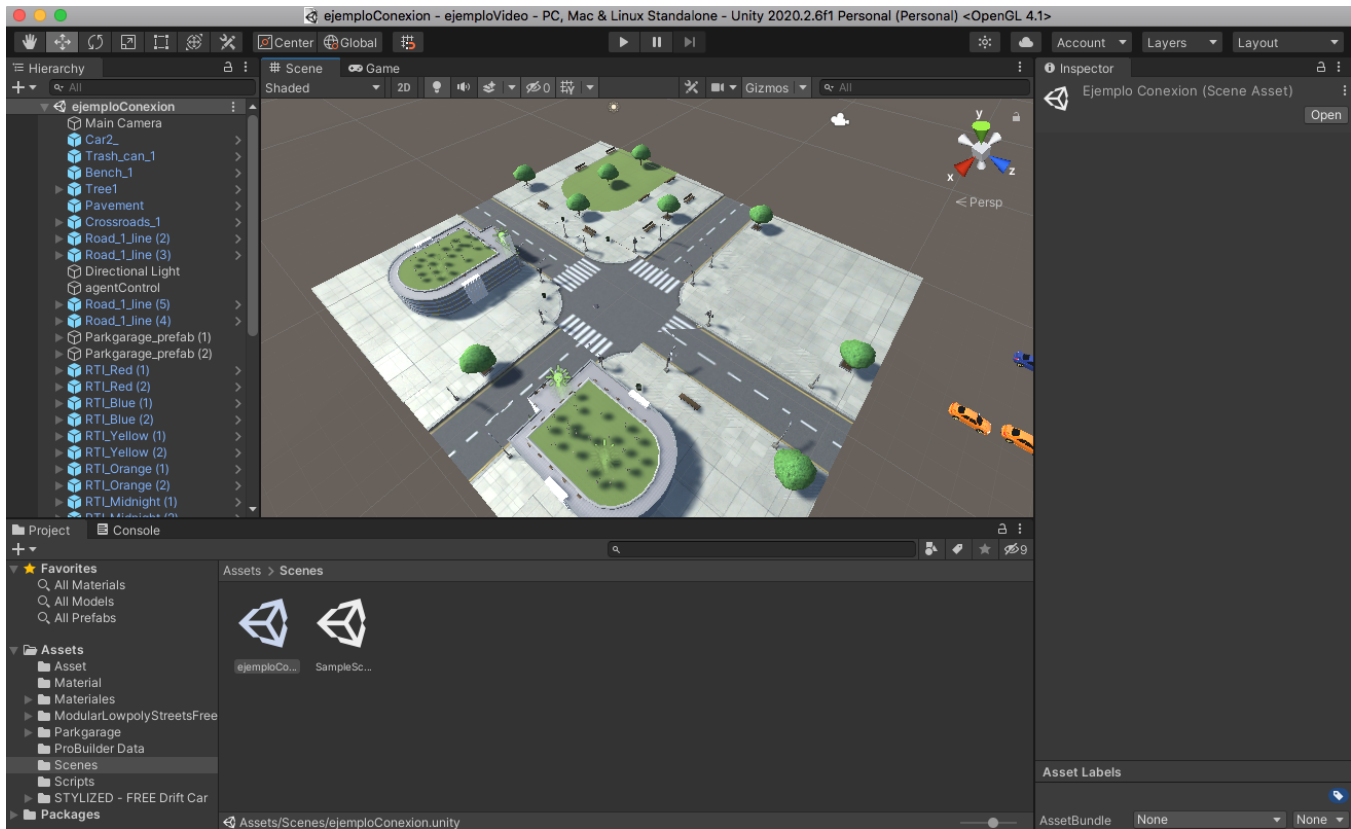
### Paso 2:

Assets > Import Package > Custom Package... > seleccionar archivo unitypackage descargado y dar clic en import



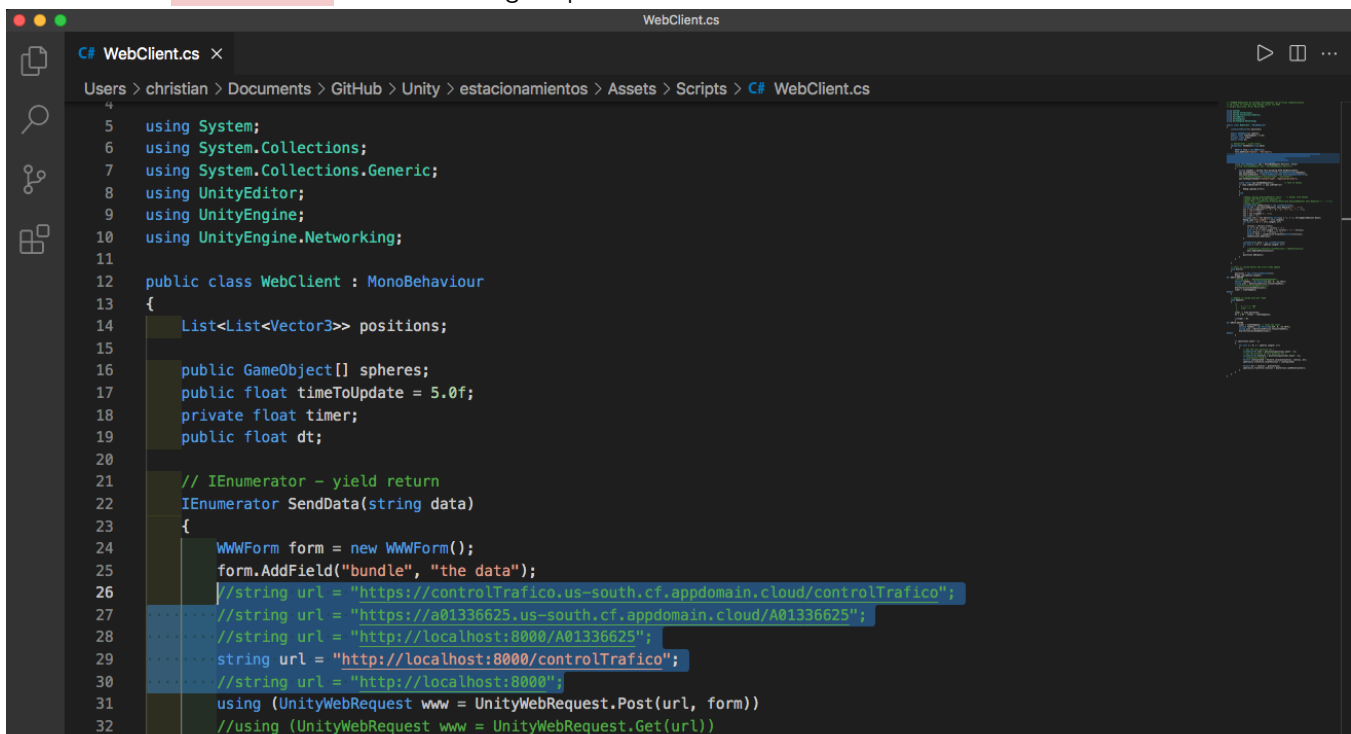
### Paso 3:

Cargar la escena principal



### Paso 4:

En el archivo `webClient.cs` cambiar string url por el url de IBM Cloud





## Paso 5:

Correr la simulación dentro de Unity.



## Análisis de la solución desarrollada

### Luis Enrique Bojórquez Almazán

En este caso cabe mencionar que para poder llevar a cabo la solución de este reto fue necesario utilizar herramientas como Unity para el modelado 3D de nuestro ambiente con agentes. Y también, herramientas como IBM Cloud para la implementación de un servidor web capaz de recibir y mandar información por medio de formatos JSON. Además, es importante mencionar que antes de llevar a cabo la implementación en Unity realizamos una simulación gráfica 2D en Python con la ayuda de la librería “Mesa”. Esto con el objetivo de poder visualizar de primera mano cómo es que se iban a colocar los objetos existentes en el ambiente y lo más importante observar cómo iban a interactuar los diferentes agentes que existen con el ambiente en general y cómo es que ve dicho comportamiento.

En lo desarrollado en Python, podemos encontrar nuestro modelo de simulación 2D que realizamos con la librería “Mesa”. Además, podemos observar que es aquí donde se inicializan cada uno de los elementos que van a existir dentro de nuestra aplicación podemos encontrar estacionamientos, calles, autos, y semáforos. Cabe mencionar que esta propuesta de solución se puede optimizar utilizando una estructura basada en grafos, de esta manera se podría simplificar el desarrollo del sentido de las calles y el manejo de la información del comportamiento de los agentes se haría más eficiente, tanto en memoria como en tiempo de ejecución.

Por la parte de Unity, fue importante saber utilizar esta herramienta para poder hacer una simulación 3D que representara de manera satisfactoria nuestra simulación en un formato 3D. Dentro de esta, fue muy importante el uso de la herramienta ProBuilder, la cual nos permitió poder crear modelos 3D desde 0. Y también, utilizando esta misma herramienta podemos personalizar y crear un modelo de automóvil, desde su material hasta su carrocería.



Por último, en el desarrollo de IBM Cloud fue necesario en primera instancia crear una cuenta IBM Cloud. Después, se requiere instalar una paquetería de IBM para poder ejecutar los comandos desde la consola. Una vez instalado, se ingresaron todos los comandos necesarios en consola para crear la aplicación, asignarle su endpoint y realizar el push de todos los archivos necesarios para que nuestra aplicación funcione de manera correcta en la nube.

## Marco Antonio Bosquez González

En cuanto al modelo que seleccionamos para la simulación, para este utilizamos la librería Mesa de Python, que nos ayuda a visualizar un Grid en el que movemos y manejamos a todos los agentes y elementos del ambiente. Lo seleccionamos porque es sencillo de utilizar y por lo menos de la clase, estamos más acostumbrados a este. Para manejar el modelo gráfico, escogimos usar Unity, ya que este podría verse correctamente en una simulación y, agregando animaciones, podemos lograr que la simulación alcance otro nivel de realismo. Utilizando Python con Mesa, mandamos la información de los vehículos en un archivo JSON por la plataforma de IBM Cloud y al final recibiendo estas coordenadas en Unity.

Para este modelo, tomamos en cuenta 5 elementos, los cuales fueron los vehículos, semáforos, caminos, muros y estacionamientos. Estos 5 elementos, de los cuales 3 son elementos del ambiente, lo que quiere decir que interactúan con dos agentes, los cuales fueron el semáforo y los vehículos para poder realizar su función de viajar por el ambiente y las calles disponibles de forma correcta. De manera que, pudieran alcanzar su objetivo final, el cual es estacionarse en un punto específico del ambiente.

Al final, elegimos el diseño gráfico de Unity porque este tenía una mejor representación del movimiento de los vehículos y su función para estacionarse, de misma manera, las animaciones y prefabs visuales tenían una vista más estética y preferible para la representación de un automóvil.

En nuestra solución, hay bastantes funcionalidades que nos ayudan a representar mejor la simulación, de manera que, los coches revisan sus alrededores para saber a dónde dirigirse, los semáforos checan el movimiento y actividad de las intersecciones, los estacionamientos revisan su capacidad antes de dejar pasar a un vehículo y no importando el ambiente, se puede modificar las calles y agregar más elementos para seguir funcionando. A pesar de que nuestra solución sea bastante capaz y adecuada, esta tiene algunas desventajas como el control de la simulación únicamente por Mesa y el movimiento aleatorio de los coches en intersecciones. Para modificar esto, podríamos implementar una mejor solución con nodos en la que obtengamos mejor el movimiento y podamos implementar optimización de rutas incluyendo todos los elementos mencionados anteriormente.

## Christian Jesús González Ramírez

Para la realización del reto usamos 3 herramientas principales, la primera fue la librería Mesa en Python, aquí creamos el modelo de multiagentes, la librería de Mesa cuenta con muchas funciones que nos fueron de utilidad y nos facilitaron, tanto la creación del sistema como de la visualización previa a la simulación, ya que al correr Mesa se genera un grid y con tuvimos una visualización más clara de los vehículos, semáforos, calles, muros y estacionamientos.

Otra herramienta importante que se usó durante el desarrollo del reto fue IBM Cloud, con esta herramienta creamos un app de Cloud Foundry para que nuestro servidor se desplegará en línea, la app de Cloud Foundry envía las coordenadas en formato JSON con las nuevas posiciones de los automóviles y posteriormente Unity las recibe.

Por último para el modelado y la visualización de la simulación utilizamos Unity, aquí creamos un ambiente usando objetos descargados de internet y otros hechos por nosotros con la herramienta de ProBuilder con la que cuenta Unity, también empleamos los conceptos de iluminación y texturas para darle más realismo a la simulación final.

## Reflexión sobre el proceso de aprendizaje

### Luis Enrique Bojórquez Almazán

La modelación de sistemas multiagentes es un tema completamente nuevo para mí, por lo que al principio me pareció que iba a ser un tema sumamente interesante y complejo. Aprender la diferencia entre un agente y el ambiente fue una tarea fácil de comprender, sin embargo, me di cuenta de que la cantidad de respuestas o comportamientos que puede tener el agente en base a lo que ocurre a su alrededor es exponencialmente mayor. Además, es importante conocer que existen diferentes tipos de agentes, y estos se catalogan de acuerdo al funcionamiento que uno desea en ellos. Por otro lado, me pareció muy interesante el hecho de haber tenido la oportunidad de montar un servidor en IBM Cloud para ver el funcionamiento de los archivos python desde la nube y no solo de manera local (Local Host).

### Marco Antonio Bosquez González

Al principio del semestre, al escuchar que la clase iba estar relacionada con agentes, esperaba aprender lo básico en cuanto a Inteligencia Artificial, agentes y su interacción con el ambiente. Durante todo el bloque, aprendimos a utilizar estos elementos para solucionar problemas, e incluso aprendimos de algoritmos que nos ayudan a optimizar nuestra solución y hacerla más rápida y sencilla. Estoy agradecido que pudiéramos aprender las bases de una área que me parece bastante interesante y a pesar de que aprendí eso, esperaba trabajarlo más, pero debido al tiempo, solo pudimos ver los temas generales, aunque también eso me pareció correcto. La simulación quedó como esperaba y aprendí bastante de todos los elementos que hacen el funcionamiento de una simulación por agentes y su ambiente.

### Christian Jesús González Ramírez

Al principio del bloque nunca había escuchado el tema de “agentes”, al principio me pareció un tema muy complejo pero conforme fueron pasando las clases, los temas comenzaron a verse más claros, ya que vimos desde los conceptos hasta algoritmos para optimizar.

En cuanto a la parte de Unity también aprendí mejor cómo modelar, ya que al principio Unity me pareció difícil y complejo, pero al verlo desde las bases y la teoría, me quedaron más claros temas como las texturas, iluminación, sombras, cámaras, etc. Me hubiera gustado que este bloque fuera de mínimo 10 semanas ya que considero que 5 semanas no fueron suficientes y solo pudimos ver lo básico en cuanto a agentes, simulaciones e inteligencia artificial.

## Referencias

Medina Ramírez, Salvador. (2012). *Transforming Urban Mobility in Mexico: Towards Accesible Cities Less Reliant on Cars*. Institute for Transportation and Development Policy (ITDP Mexico). Retrieved on August 7, 2019, from [mexico.itdp.org/TransformingUrbanMovility](http://mexico.itdp.org/TransformingUrbanMovility)

Schteingart, M. & Ibarra, V. (2016). *Desarrollo urbano-ambiental y movilidad en la Ciudad de México : evolución histórica, cambios recientes y políticas públicas*. Ciudad de México, México: EL COLEGIO DE MEXICO.