

ProtoWorld Suite v1.0

Installation guide

&

Introduction to the functionalities of the framework

Miguel Ramos Carretero

July, 2016

Contents

1. INTRODUCTION	4
1.1. OVERVIEW TO PROTOWORLD.....	4
1.2. USERS AND PLAYERS	4
1.3. BRIEF NOTES BEFORE STARTING	5
2. HOW TO INSTALL PROTOWORLD.....	5
2.1. REQUIRED SOFTWARE	5
2.2. STEP BY STEP INSTALLATION.....	5
3. STARTING PROTOWORLD	8
3.1. INITIALIZING THE POSTGRESQL DATABASE	8
3.2. INITIALIZING THE WCF SERVICE	9
3.3. CONFIGURING SCRIPTS	9
3.4. FIRST TEST WITH PROTOWORLD.....	10
3.5. CREATING A NEW SCENE WITH PROTOWORLD	10
4. SCENARIO GENERATION MODULE	11
4.1. HOW TO ADD THIS MODULE	11
4.2. ELEMENTS IN THIS MODULE	11
4.3. HOW TO POPULATE POSTGRESQL WITH OSM DATA.....	11
4.4. GENERATING A NEW SCENE IN UNITY	12
4.5. CROPPING TOOL.....	13
4.6. GENERATING A NEW SCENE IN UNITY	13
5. NAVIGATION MODULE	13
5.1. HOW TO ADD THIS MODULE	14
5.2. ELEMENTS IN THIS MODULE	14
5.3. THE CAMERA OBJECT.....	14
5.4. THE TIME CONTROLLER.....	15
6. FLASH PEDESTRIAN SIMULATION MODULE	15
6.1. BEFORE STARTING: BAKING THE NAVIGATION MESH.....	16
6.2. ADDING THE MODULE: GENERATING SPAWNING POINTS AND DESTINATIONS	16
6.3. ELEMENTS IN THIS MODULE	18
6.4. RUNNING AN EXAMPLE.....	18
7. ARROW PUBLIC TRANSPORT SIMULATION MODULE	19
7.1. HOW TO ADD THE MODULE: GENERATING STATIONS AND LINES	19
7.2. ELEMENTS IN THIS MODULE	21
7.3. RUNNING AN EXAMPLE.....	22
8. TRAFFIC INTEGRATION MODULE	22
8.1. HOW TO ADD THE MODULE	23
8.2. SUMO INTEGRATION IN REAL TIME	23
8.3. SUMO INTEGRATION WITH FCD FILES	24
8.4. VISSIM INTEGRATION WITH FSP FILES.....	25
8.5. INTEGRATION WITH PWS FILES	25
8.6. MATSIM INTEGRATION	25

8.7.	DECISION TREE INTEGRATION	26
8.8.	ELEMENTS IN THIS MODULE	26
8.9.	RUNNING AN EXAMPLE	26
9.	DRAG AND DROP MODULE.....	27
9.1.	HOW TO ADD THIS MODULE	27
9.2.	ELEMENTS IN THIS MODULE	28
9.3.	HOW TO ADD MORE DRAG AND DROP ELEMENTS	28
9.4.	RUNNING EXAMPLE	29
10.	VVIS VISUALIZATION MODULE	30
10.1.	HOW TO ADD THIS MODULE	30
10.2.	ELEMENTS IN THIS MODULE	30
10.3.	PREPARING DATA FILES USING SQLITE	30
10.4.	PREPARING QUERIES USING AN XML FILE	31
10.5.	PREPARING MAP FOR VISUALIZATION.....	32
10.6.	RUNNING EXAMPLE	32
11.	LOGGER MODULE.....	33
11.1.	HOW TO ADD THIS MODULE	33
11.2.	ADDING NEW LOGGING LINES IN THE CODE	33
11.3.	SETTING LOG4NET IN THE RELEASE	33
12.	DECISION TREE MODULE (<i>BETA</i>).....	34
12.1.	HOW TO ADD THIS MODULE	34
12.2.	WRITING THE XML DECISION TREE FILE	34
12.3.	ELEMENTS IN THIS MODULE	35
12.4.	PREPARING THE MODULE FOR GAMEPLAY.....	36
12.5.	RUNNING AN EXAMPLE.....	36
13.	KPI MODULE (<i>BETA</i>)	36
13.1.	HOW TO ADD THE MODULE: ADDING CHARTS.....	37
13.2.	ELEMENTS IN THIS MODULE	38
13.3.	RUNNING AN EXAMPLE.....	38
14.	RELEASING A PROTOWORLD SCENE	38
14.1.	RELEASING PROCESS	39
14.2.	EXECUTING THE PROTO WORLD RELEASE	39
CODA	40

1. Introduction

This guide contains a step by step explanation about how to install the ProtoWorld suite and a dedicated chapter for each of the different modules that are integrated in this framework. ProtoWorld contains diverse functionalities for building virtual urban environments and distributed, integrated and interactive simulations. Given the multiple possibilities of such framework, new users might find a bit challenging to start working with this software. This guide tries to explain every part of the framework as clearly as possible, including screenshots and practical examples. The developers hope that, through this document, the users can find an ease on the learning and use of ProtoWorld.

ProtoWorld is an open source tool developed at the Gaming Participatory Labs (GaPSLabs) at the School of Technology and Health, KTH Royal Institute of Technology, Huddinge, Sweden.

1.1. Overview to ProtoWorld

The ProtoWorld framework can be divided into three main components:

- *ProtoWorld Unity Project*: The core of the framework and the place where the edition and the hub of functionalities take place. The ProtoWorld Unity Project, built over the Unity Game Engine, contains all the editor tools needed to build an urban scene from scratch and to add functionalities such as pedestrian and public transport simulations. It also include interfaces to enable integration of external simulations such as SUMO and Vissim.
- *PostgreSQL*: A database to hold all the information related to GIS data, needed to generate urban scenes using the ProtoWorld Unity Project. The database is populated with data from OpenStreetMap, a collaborative free project to create and edit GIS maps worldwide.
- *WCF Service*: A middleware software containing the services needed to access and retrieve GIS data from the database. The services provided by the WCF framework are used by the ProtoWorld Unity Project when generating new scenes.

1.2. Users and players

Throughout this documentation, the reader will see references to both *users* and *players* of the ProtoWorld framework. It is important to clarify the difference between these two. *Users* refer to those that are deploying and using the ProtoWorld framework to build and edit urban scenes. In other words, they are the ones to whom this documentation is addressed to. *Players*, on the other hand, refer to those that will only play the ProtoWorld scenarios gameplays generated by the users, not having interaction with the editor tools.

1.3. Brief notes before starting

In order to be able to use ProtoWorld in its full potential, the users need to have a fair knowledge of the basic functionalities of the Unity Game Engine. Also, although not essential, skills in programming and database management systems are recommended.

It is important to notice that the development of this software is a prototype of the whole concept behind the ProtoWorld framework. The developers hope that the users use this software not only as a final tool but also as a base for inspiration and further developing.

2. How to install ProtoWorld

This chapter contains a detailed guide about how to get the ProtoWorld framework and how to install it, step by step.

2.1. Required software

The following list contains all the software needed in order to be able to run ProtoWorld:

- Unity 5.3 Personal edition
- PGAdmin III with PostgreSQL 9.3
- Internet Information Services (IIS)
- VS Express for Web or Visual Studio 2013 with .NET libraries

ProtoWorld has been tested successfully in the software versions included in this list, always under the operating system Microsoft Windows 7 Professional. Newer versions of the software might also be compatible with ProtoWorld, but they have not been tested. In the following section there is a more detailed explanation about the installation of these software components.

2.2. Step by step installation

In the following lines there is a detailed explanation to help users install the ProtoWorld framework correctly and the software needed for it:

1) *Obtaining the ProtoWorld package*

The ProtoWorld package can be obtained from the github repository <https://github.com/SebastiaanMeijer/ProtoWorld.git>. The project is held in LFS (Large File Storage), so it is recommended pulling it using gitshell:

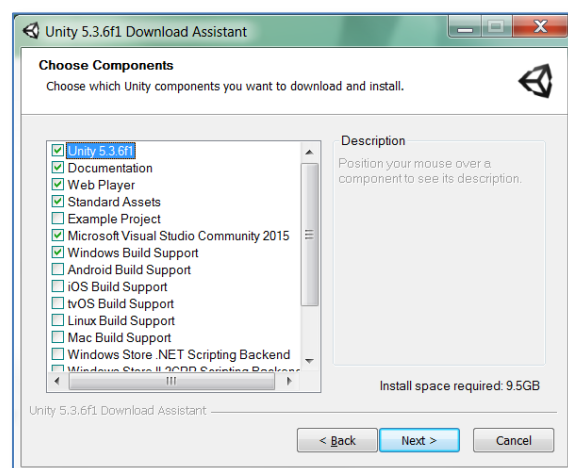
```
git clone --depth 1 https://github.com/SebastiaanMeijer/ProtoWorld.git --branch master
```

The project contains the following folders:

- *ProtoWorldUnityProject*: Full project for the Unity game engine.
- *WCFService*: Visual Studio solution containing the code needed to build the services that will allow communication between the ProtoWorld Unity Project and the PostgreSQL database.
- *ScriptsAndTools*: Auxiliary scripts and other software to automate several tasks while using ProtoWorld. This folder also includes testing tools.
- *Documentation*: Documents and tutorials of the ProtoWorld framework.

2) Installing Unity 5.3

The Unity game engine can be downloaded from its own webpage: www.unity3d.com. Users can choose the version that better adapts to their needs, but the free version (the personal edition) is enough to be able to run ProtoWorld. It is important to notice that all the default Unity components must be installed.



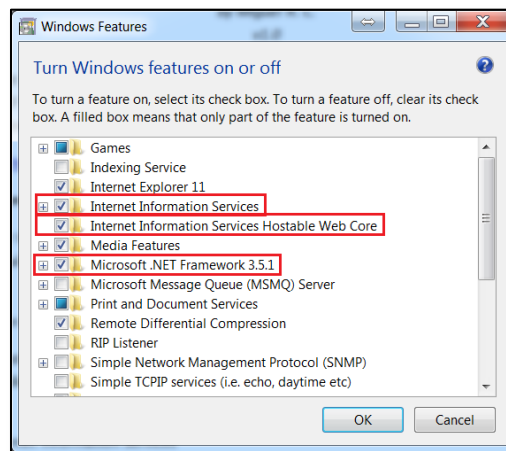
3) Importing ProtoWorld Unity Project

Once the installation is completed, the project contained in ProtoWorldUnityProject can be imported into the Unity Editor. Given the size of the project, it might take some time to complete the importing process. Once it is done, all the assets should appear in the project tab along with a new menu in the Unity Editor called ProtoWorld Editor. The users might see eventual warnings and errors related to inconsistent line endings, meshes and FBX imports, but they should be easily dismissible.

4) Installing IIS

The Internet Information Service (IIS) is needed in order to be able to host the WCF service to establish communication between the ProtoWorld Unity Project and the PostgreSQL database. IIS must be installed in the system before the WCF can be deployed. This can be done checking the features installed in Windows: *Start > Turn*

Windows features on or off. The features to install are: *Internet Information Services*, *Internet Information Services Hostable Web Core* and *Microsoft .Net Framework*. All the tools within these branches must be installed.



5) Installing Visual Studio

Visual Studio can be obtained at www.visualstudio.com. Users can choose the software option that suits better his/her needs, but we recommend using Visual Express for Web or Visual Studio 2013. ASP.NET and .NET libraries must be installed along with this software.

6) Importing WCF Service

If Visual Studio has been installed properly with all the needed libraries, users should be able to open the WCFService solution into the programming environment. Once the project is open, the whole solution needs to be compiled and built. Chapter 3 contains more information about how to use the WCF.

7) Installing PostgreSQL

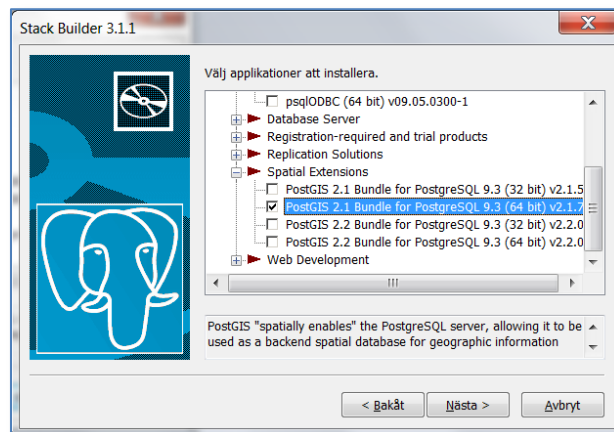
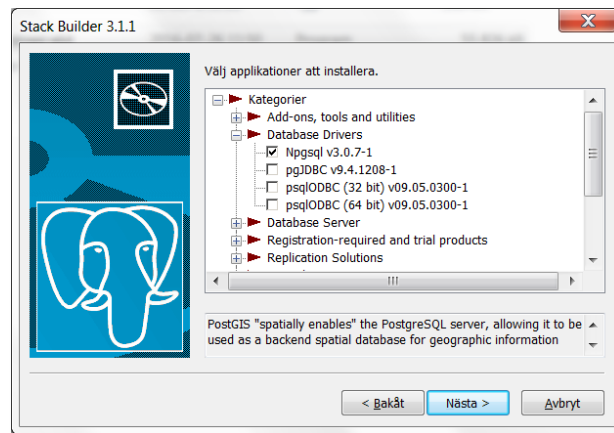
The packages to install PostgreSQL can be downloaded from www.postgresql.org. Along with this, the users will need a database management tool. The developers recommend using PGAdmin III, which can be obtained from www.pgadmin.org.

During the installation of PostgreSQL, the wizard installer will ask for a password, a port and a locale. In order to match the default setup of the ProtoWorld Unity Project, it is recommended to use the following values:

- Password: *test*
- Port: *5432*
- Locale: *default*

Once the basic installation of PostgreSQL is completed, some additional tool will need to be installed using the application Stack Builder (included with PostgreSQL). These tools

are the *npgsql* database drivers and the *PostGIS 2.1* spatial extension. They can be selected from the Stack Builder wizard. When asked by the wizard installer, users can ignore the option about creating a new spatial database.



3. Starting ProtoWorld

Once the ProtoWorld framework has been installed in our system, users will need to initialize the software. This chapter covers the steps to initialize the different components of the ProtoWorld framework before starting using it.

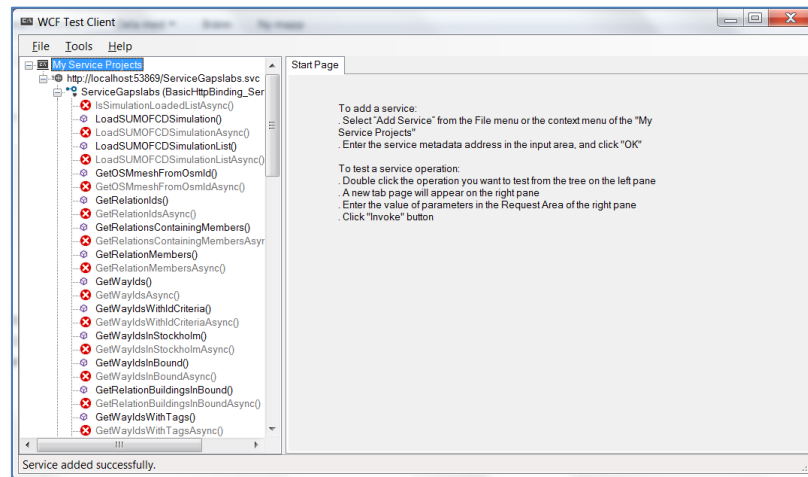
3.1. Initializing the PostgreSQL database

The PostgreSQL does not need any special initialization: it will start running in the background by default every time the Windows system is started.

When opening the PGAdmin III tool for the first time, it will ask for the password that was set during the installation. Once entered, the database management system will allow the users to see the list of postgres databases in the system and to manipulate them. It is here where the GIS data will be stored, which will be accessed by the ProtoWorld Unity Project to build urban scenes. More information about this will follow in Chapter 4.

3.2. Initializing the WCF Service

The WCF service can be started in Visual Studio from the GaPSLabWCFService project, contained in the WCFService solution. Once set as StartUp project, the services can be started from *ServiceGapslabs.svc.cs* (*Ctrl+F5* in Visual Studio). If IIS was set properly, the WCF Test Client window will open and, after a few seconds, the services will be ready to be called by the ProtoWorld Unity Project.



- ✓ *Developers note: some users might wonder why the WCF services were not included within the ProtoWorld Unity Project. The reason for this is because these services use functionalities from .NET framework 4.0, which are not supported by the Unity game engine. Also, a WCF middleware allows the ProtoWorld components to be distributed in different systems so, if needed, the PostgreSQL database and the ProtoWorld Unity Project could be deployed in separate machines.*

3.3. Configuring scripts

In order to make the scripts able to run in the system, users will need to adjust some parameters in the script *PopulateOSMtoDB.bat*, which can be found in the folder *ScriptsAndTools/PopulateOSMtoDB*. In particular, users need to set the path of the *bin* folder of PostgreSQL and some other parameters to connect to the PostgreSQL databases: url, port, host, username and password (by default, these parameters already match the default PostgreSQL parameters).

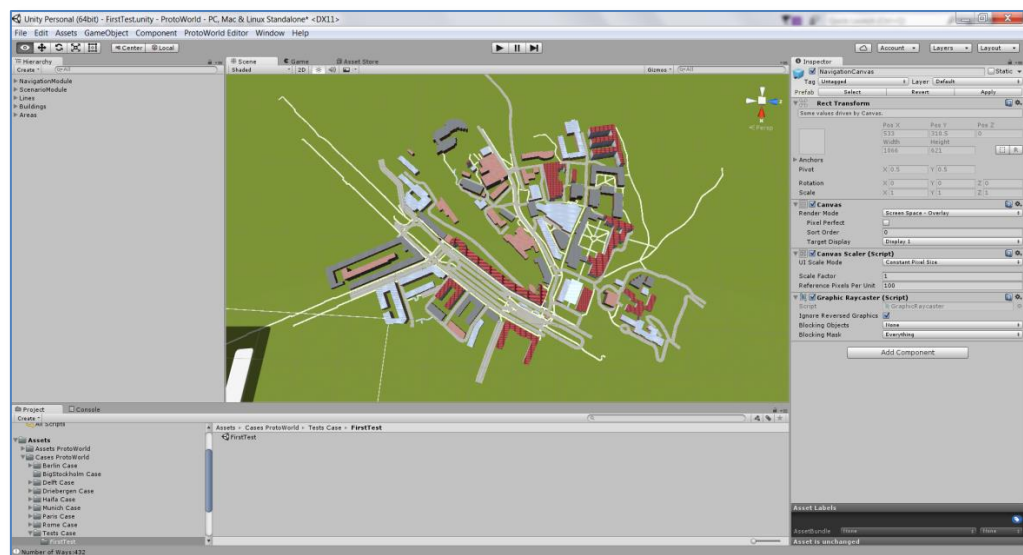
```
@set PATH=C:\Program Files\PostgreSQL\9.3\bin;C:\Program
Files\PostgreSQL\9.3\extra_dep
@set URL=127.0.0.1
@set PGPORT=5432
@set PGHOST=localhost
@set PGUSER=postgres
@set PGPASSWORD=test
```

3.4. First test with ProtoWorld

The ProtoWorld framework includes some testing tools to check if the installation of each component has been made properly. We recommend users to follow these steps before starting working with ProtoWorld:

- i. Initialize the WCF Service as explained in Section 3.2.
- ii. Navigate into the folder *ScriptAndTools/Test/FirstTest* and run the batch file *PopulateTestDB.bat*. This should create a database called *TestDB* in PostgreSQL.
- iii. Open the ProtoWorld Unity Project and navigate through the project assets to *Assets/Miscelanea/Tests/FirstTest*. There, open the scene *FirstTest.unity*.
- iv. In the Unity Editor, go to the menu *ProtoWorld Editor > ProtoWorld Essentials > Map Tools > Generate Map*. A new window will appear requesting some parameters, which should be already set by default.
- v. Click on the button *Generate map*. A progress bar should appear and the map should start being generated.

Once the process is finished, the users should see a 3D map of an urban area. If that is the case, ProtoWorld is set properly and it can start being used.



3.5. Creating a new scene with ProtoWorld

To get started with ProtoWorld in a new empty scene, users just need to navigate through the menu *ProtoWorld Editor > ProtoWorld Essentials > Add Essentials*. That will include into the scene hierarchy two new elements that contain the basic functionalities of ProtoWorld: the Scenario Module and the Navigation Module. These elements are needed before any other module is added. More details about these two modules follow in Chapters 4 and 5, respectively.

4. Scenario Generation Module

ProtoWorld allows the users to generate virtual urban scenarios from real GIS data, from small towns and cities to large districts and metropolis. This includes the generation of buildings, roads, highways, footways, cycleways and also water areas. This module creates the base layer on which other components of the framework, such as pedestrian and traffic simulation, will be built afterwards.

4.1. How to add this module

This module is part of the ProtoWorld Essentials tools. It can be added from the menu *ProtoWorld Editor > ProtoWorld Essentials > Add Essentials*. In the hierarchy of the scene, two new components will appear: the Navigation Module and the Scenario Module. The one this chapter focuses on is the Scenario Module, which will allow users to generate scenarios.

4.2. Elements in this module

This module contains two main elements:

- *AramGISBoundaries*: The core of the scenario module. The component MapBoundaries, attached to this game object, holds all the information needed in relation to the area that we want to generate, from the GIS coordinates, the materials and size of different roads, the height of buildings (including a customized skyline) and the parameters to connect to the PostgreSQL database. It is recommended not to modify these parameters.
- *Surface*: A simple plane with a grass material that represents the ground of the scenario. As an alternative, users that want further customization of the ground could use the *Unity's Terrain System*, included by default with the game engine.

4.3. How to populate PostgreSQL with OSM data

Before being able to run the scenario module, a PostgreSQL database needs to be populated with OSM data:

- i. Download a proper OSM file with data from an area of your choice. The webpage www.openstreetmap.org contains an export tool that allows exporting small areas, but for large areas it is recommended to check other OSM servers such as <http://extract.bbbike.org/>.
- ii. Once the file is downloaded (always in OSM format), go to the folder `ScriptAndTools/PopulateOSMToDB/`. There, open a command line and invoke the script `PopulateOSMToDB.bat` with two arguments: the first one is the name to give

to the database and the second one the path of the OSM file. Let the script populate the database.

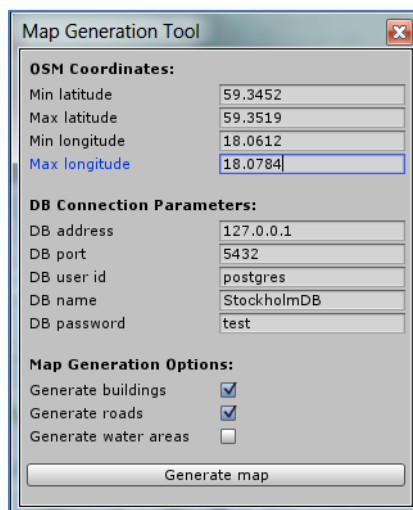
PopulateOSMToDB.bat <database name without spaces> <path to the osm file>

Before running the script, users should check that the parameters for connecting with PostgreSQL are right (see section 3.3 for more details). Depending on the size of the map, the whole process might take from a few minutes to several hours. This time also depends on the quality of the data and the level of detail within, which highly depends on the nature of the OSM file.

4.4. Generating a new scene in Unity

Once the database is populated properly following the steps of the previous section, the ProtoWorld framework will be ready to generate the map in Unity:

- i. Go back to ProtoWorld Unity Project and open the map generation tool from the menu *ProtoWorld Editor > ProtoWorld Essentials > Map Tools > Generate Map*.
- ii. Set the GIS boundaries of the map. Notice that it is possible to select a section of the total area hold into the database.
- iii. Set the right values to connect to the database: address, port, used id, database name and password.
- iv. Select the elements to be generated: buildings, roads and/or water areas. Notice that water areas refer only to fountains, lakes and other water areas within a city, but rivers and open sea areas must be drawn manually using the Unity's Terrain System.
- v. Check that the WCF service is running on IIS and click on the button *generate map*. This will run the process of generating the scenario into Unity. Depending on the size of the area, it could take from a few minutes to several hours. After that time, a 3D map of the area should appear in the scene.



The screenshot shows a Windows-style dialog box titled "Map Generation Tool". It contains three sections of configuration options:

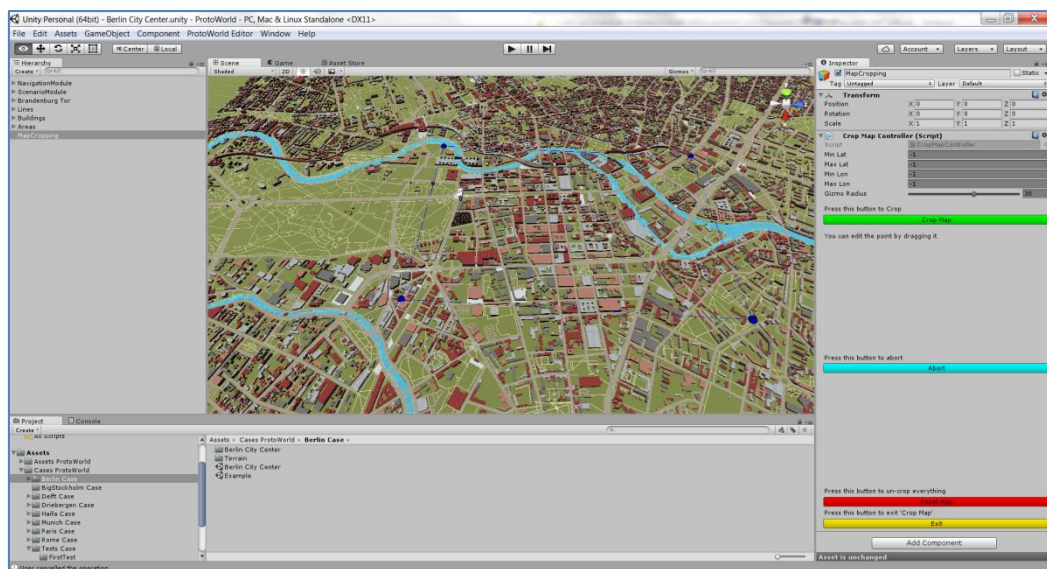
- OSM Coordinates:** Four text input fields for "Min latitude" (59.3452), "Max latitude" (59.3519), "Min longitude" (18.0612), and "Max longitude" (18.0784).
- DB Connection Parameters:** Five text input fields for "DB address" (127.0.0.1), "DB port" (5432), "DB user id" (postgres), "DB name" (StockholmDB), and "DB password" (test).
- Map Generation Options:** Three checkboxes: "Generate buildings" (checked), "Generate roads" (checked), and "Generate water areas" (unchecked).

At the bottom of the dialog is a "Generate map" button.

4.5. Cropping tool

As an additional tool to the map generation, there is a cropping tool for users that prefer to generate large areas and, afterwards, crop them in different scenes according to their needs:

- i. After the scenario is generated, activate this tool from the menu *ProtoWorld Editor > ProtoWorld Essentials > Map Tools > Crop Map*. A new window will appear in the inspector tab.
- ii. Click on the *start* button to start defining the boundaries of the area. To do so, click and drag on the map to generate a bounding box that will set the limits of the cropping.
- iii. Once the bounding box is drawn, click on the button *crop map*. That will crop all the urban elements outside the aforementioned bounding box. To undo the cropping, click of *reset map*. To exit the tool, click *exit*.



4.6. Generating a new scene in Unity

In the menu *ProtoWorld Editor > ProtoWorld Essentials > Map Tools > Advance*, users will find some advance options for alternative ways of map generation using a filter for roads and options for optimizing the map after being generated, such as grouping building and roads in combined meshes (recommended for big maps) or turning on and off building and road colliders.

5. Navigation Module

This module, also part of the ProtoWorld Essential tools, contains all the elements needed to implement the functionalities that allow users and players to navigate on the scenario during gameplay. It also controls other elements needed for the simulations such as the time and the event system.

5.1. How to add this module

This module is part of the ProtoWorld Essentials tools. It can be added from the menu *ProtoWorld Editor > ProtoWorld Essentials > Add Essentials*. In the hierarchy of the scene, two new components will appear: the Navigation Module and the Scenario Module. The one this chapter focuses on is the Navigation Module, which will allow the players to navigate the scenario during gameplay.

5.2. Elements in this module

The following list contains a brief description of each of the elements. Some of them will be explained further in following sections:

- *CameraObjects*: Contains all the elements needed to implement the behaviour of the camera within the scene during gameplay. Further explanation of this object follows in section 5.3.
- *Directional Light*: Basic light element to illuminate the scene.
- *EventSystem*: Internal object for Unity to control event systems during gameplay such as clicks and keys. Alteration in this object might cause unexpected malfunctioning.
- *Navigation Canvas*: Canvas that contains the elements to display the 2D menus for the time controller and the camera controller. More information about these elements will follow in section 5.3 and 5.4.
- *Loading Canvas*: Contains a hidden canvas that displays a splash animation for certain events during gameplay that require loading times.

5.3. The camera object

The camera element is the main navigation tool that players will use during gameplay, allowing navigation and zooming in/out through the scenario from an aerial point of view. The CameraObject holds three elements attached to it: the *Main Camera*, the *Target Point* and the *Overview Point*. The *Main Camera* element contains the default components that configure a camera in Unity. Along with that, the users can find there the Camera Control component, which allows setting certain parameters of the camera such as navigation boundaries, degrees of freedom or navigation velocity.

➤ *How to adjust the camera object for gameplay:*

By default, the camera focuses at the beginning of the game to the game object *Overview Point*. In order to adjust the camera properly for gameplay, the users can move

the *Overview Point* object to the center of the scenario or to any other place of the scene where the users want to focus the camera at the beginning. The *Target Point* is an internal game object used by the *Main Camera*, so it is not recommended to alter it.

➤ ***The camera change button:***

During gameplay, players can move the camera back to the starting overview point at any moment. This functionality is implemented on a button from the *Navigation Canvas*: the *CameraChangeUI* game object.

5.4. The time controller

This element implements the functionalities needed for ProtoWorld to control the internal time that will integrate all the other modules of the framework such as the pedestrian and the traffic simulations. This tool also allows, during gameplay, for pausing/resuming the simulations and for speeding them up and down.

➤ ***Adjusting the time controller for gameplay:***

Within the *TimeControllerUI* the users will find the *Time Controller* component, which will allow them to customize certain parameters of the time such as the maximum speed factor (for simulation accuracy and performance, the developers recommend not going higher than factor 3), or some visual aspects of the clock. Other parameters there refer to links with other game objects and other modules within the framework, so it is recommended not to alter them.

6. FLASH Pedestrian Simulation Module

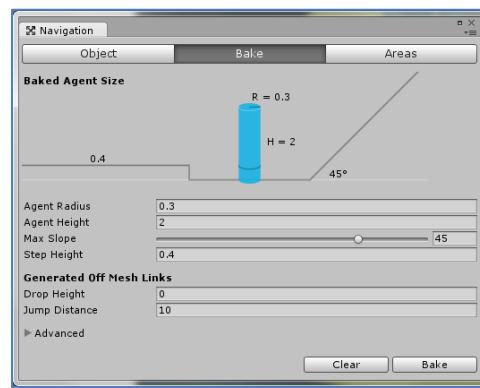
FLASH is a proprietary pedestrian simulation that implements the behaviour of pedestrians walking and commuting within the scene to reach a certain destination. This simulation has a very simple interface based on spawning and destination points that allows for an easy integration and flexibility for any kind of situation. The key elements of this simulation are two:

- *Spawners*: Points where the pedestrians will start their journey. A map can contain as many spawners as the users wish, each of them generating a different stream of pedestrians.
- *Destinations*: Points where pedestrians will tend to go to finish their journey. A map can contain as many destinations as the users wish, and each of them will be reached by a certain percentage of pedestrians depending on certain parameters.

Despite the simplicity of FLASH, the power of this simulation relies in its capability to work together and in an integrated way with other components of the ProtoWorld framework such as, for example, the public transport module and the integrated traffic simulations. More about this integration will be explained in the respective chapters of each of those modules.

6.1. Before starting: baking the navigation mesh

The FLASH pedestrian simulation module works on top of the Unity's Navigation system, a functionality of the game engine that allows pedestrians to walk the scene. All the layers for navigation are set **automatically during the map generation process**, but the navigation should be baked by the users from the Navigation window in Unity. Users just need to click on the *bake* button to make the process start. This process can take from minutes to hours depending on the size of the map.



6.2. Adding the module: generating spawning points and destinations

This module can be added from the menu *ProtoWorld Editor > Pedestrian Module > Add or Edit Pedestrian Points*. A menu will appear in the inspector where the users will be able to start adding the elements to implement this simulation:

- i. Click on the *add* button to begin.
- ii. Select an option from the dropdown menu that appears in the inspector: choose between *spawner* or *destination*.
- iii. Click on the map to place the point. A new menu will appear in the inspector with options to customize the point:
 - a. For spawners, customize the total amount of pedestrians to spawn, the spawning frequency and the min/max number of pedestrians spawned on each spawning iteration.
 - b. For destinations, write a name and set a destination priority (higher numbers in this parameter imply higher percentage of pedestrians going there).
- iv. Once the set of parameters is finished, click on *save and go back*. Follow the same process for each point that you want to add into the scenario. Once it is done, click on *exit*.

➤ **An important note about the position of spawners and destinations:**

By default, the surface of the scenario (the grass) is not set as walkable by the navigation mesh, so the spawning and the destination points should be set over a walkable element (footways, cycleways, areas and roadways), otherwise Unity will throw a navigation mesh error.

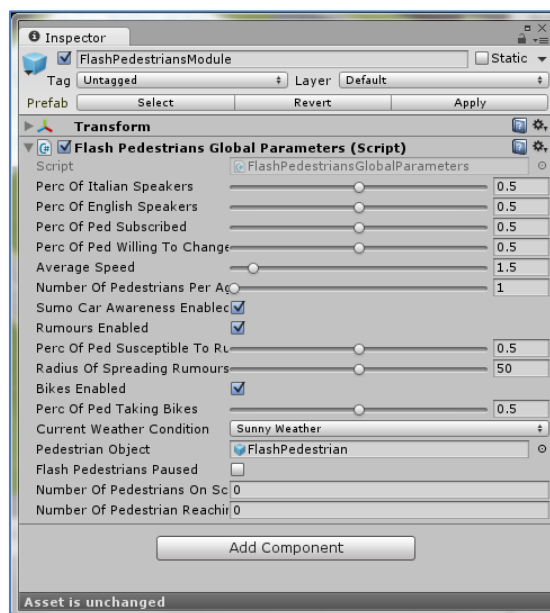
✓ *Developers note: Why is the surface (the grass) not set as a walkable area? Although this looks desirable for the pedestrian simulation, the activation of the grass surface as walkable area brings two main problems: first, it implies a large increase of computational power used by the navigation mesh to calculate paths (which highly impacts the performance in large scenarios) and, second, the building objects are ignored by the navigation mesh, allowing pedestrians go through buildings like ghosts!*

➤ **Editing points on a later stage:**

After the generation of spawners and destinations, the pedestrian editor can be opened again through the menu *ProtoWorld Editor > Pedestrian Module > Add or Edit Pedestrian Points*. A point can be edited clicking first on the *edit* button and, right afterwards, clicking on that point. That will show again the parameters of that point in the inspector.

➤ **Editing general parameters in the Flash Pedestrian module:**

The game element that holds the module (*FlashPedestrianModule*) contains a script called *FlashPedestrianGlobalParameters* that will allow the users to customize global parameters of the simulation before gameplay such as default weather, average speed of pedestrians, etc. It is recommended the modification of these parameters only to advance users.



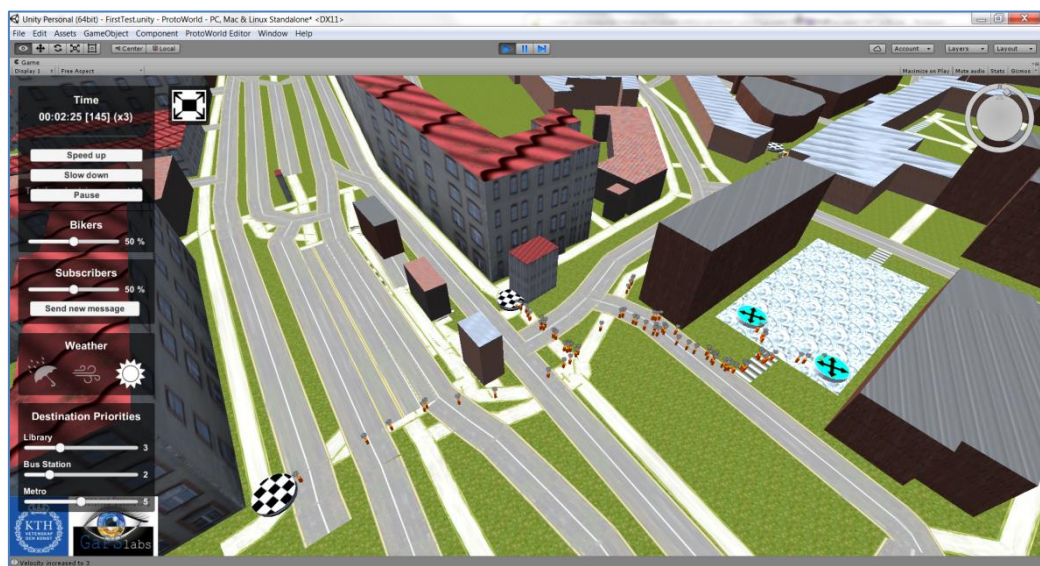
6.3. Elements in this module

In this module, the users can see the following elements:

- *SpawnerPoints*: Game object that holds all the spawner points generated in the scenario. Here the points can be edited as well.
- *DestinationPoints*: Game object that holds all the destination points generated in the scenario. Here the points can be edited as well.
- *FlashInformer*: This element controls the behaviour of the pedestrians to certain events happening during gameplay, such as changes in public transport and notifications from the player via informing messages. More about this will be explained in chapter 7.
- *FlashCanvas*: Canvas that contains all the UI elements displayed on the screen during gameplay for the player. It includes UI for changing destination priorities during gameplay, setting percentage of pedestrians informed (see chapter 7), setting percentage of bikers (see chapter 9) and changing weather.

6.4. Running an example

An example of a scene with the FLASH pedestrian simulation implemented can be found in *Assets/Miscelanea/Tests/PedestrianTest*. Once open, users can start running it pressing play. From the beginning of the gameplay, the users will see pedestrians appearing in the different spawners and going to the different destinations available in the map according to the priorities set. The users could play with some of the parameters in the UI to see how that affects the decisions of pedestrians.



7. ARROW Public Transport Simulation Module

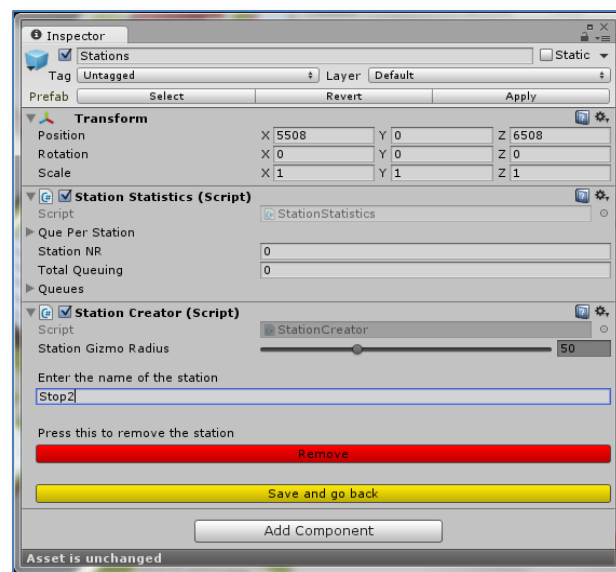
ARROW is a proprietary simulation that provides all the functionalities to build a public transport system within the ProtoWorld framework. This system is based in a simple logic of stations and lines, allowing the users to simulate metro, trams, buses and trains. As mentioned already in the previous chapter, this simulation has full integration with the FLASH pedestrian simulation, and it affects the behaviour of pedestrians on their decision to reach their destinations. The key elements of this simulation are two:

- **Stations:** Points where the public transports stop and pick/release FLASH pedestrians. A map can contain as many stations as the users wish. Also, during gameplay, these stations can be manipulated by the players, allowing to close them and to notify pedestrians around.
- **Lines:** These elements connect the different stations of the scenario, implementing the logic of the routes followed by the public transports. A map can contain as many lines as the users wish. Within this logic, the users can also specify the travelling time between stations.

7.1. How to add the module: generating stations and lines

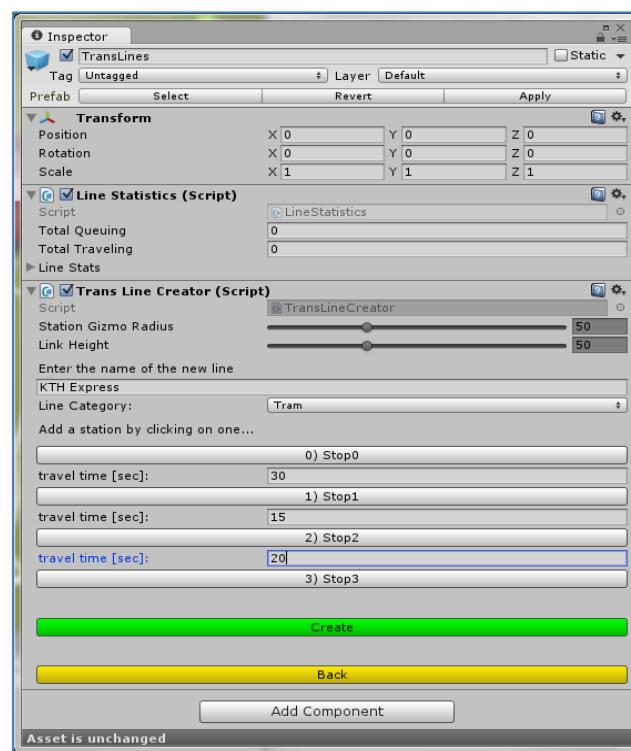
To add the module, the first step would be to add stations on the map:

- i. Open the station editor from the menu *ProtoWold Editor > Public Transport Module > Add or Edit Station*. A menu will appear in the inspector.
- ii. Click on the *start editing* button to begin, and then click on the map to place the station.
- iii. Once the station is placed, a new menu will appear asking for a name for the station. Give an appropriate name and click *save and go back*.
- iv. Go back to step *ii* to add more stations or click on the yellow button *exit* to finish.



After all the stations are added, next step would be to create lines to implement the routes:

- i. Open the line editor from the menu *ProtoWold Editor > Public Transport Module > Add or Edit Line*. A menu will appear in the inspector.
- ii. Click on *create new line* button to begin. A new menu will appear asking for a name for the line and for a category: choose between metro, bus, tram or train. There is also an option for walking lines, which are meant to represent pedestrian tunnels that connect stations below ground.
- iii. Once the category has been selected, click on the station where the line will start/end. The station will be added to the menu. Click on the next station. An arc should appear connecting both stations. Keep clicking on stations to define the line until the other end of the line is reached.
- iv. Once the line is completed, set the travelling times between the stations in the boxes that appear between them and, finally, click on the button *create*.
- v. Follow the same procedure to create other lines. Notice also that a station can have several lines. Once it is done, click on the yellow button *exit* to finish.



➤ **An important note about the position of stations:**

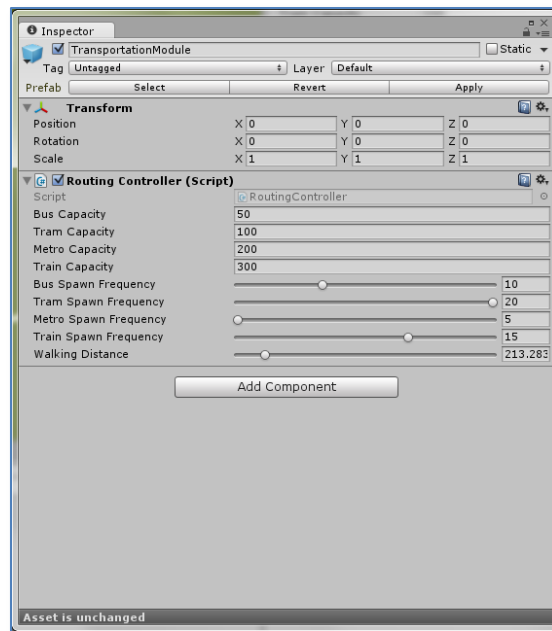
Same as in the Flash Pedestrian Simulation Module, the stations should be over walkable elements to allow pedestrians to reach them, so they should be positioned on footways, cycleways, areas or roadways. See section 6.2 for more details about this.

➤ **Editing stations and lines on a later stage:**

The station and line editors can be opened again through the menus *ProtoWold Editor > Public Transport Module > Add or Edit Station* and *ProtoWold Editor > Public Transport Module > Add or Edit Line*. A station can be edited clicking on the *start editing* button and, then, clicking on the station in the map. That will show the parameters of that station in the inspector. Lines can be edited clicking on the *edit line* button. That will show in the inspector a list of all the lines in the scene, arranged by color. Clicking on one of them will show all the parameters corresponding to that line.

➤ **Editing general parameters in the Arrow public transport module:**

The game element that holds the module (*TransportationModule*), contains a script called *RoutingController* that will allow the users to customize global parameters of the simulation before gameplay, such as default vehicle capacity, spawning frequency, etc. It is recommended the modification of these parameters only to advance users.



7.2. Elements in this module

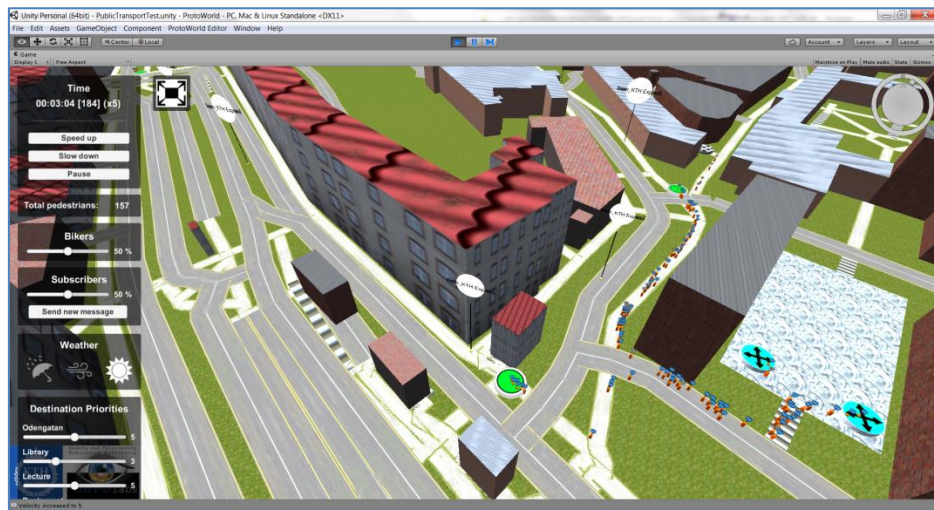
In this module, the users can see the following elements:

- **VehicleSymbol:** This element is used to visualize the position of the public transport in the scenario during gameplay. Due to the use of underground public transport, developers decided to represent the visualization of them with a balloon. Advance users can modify this visualization.
- **Stations:** This element holds all the stations generated in the scene, arranged by number. Here the stations can be edited as well.

- *TransLines*: This element holds all the lines generated in the scene, arranged by their category and name. Clicking on each of them allows for editing as well.
- *Travelers*: Internal element for Unity that keeps statistics about travellers. It is used by the KPI module (see chapter 13).
- *TransportationCanvas*: Canvas that contains the UI element to display station information for the player during gameplay. This UI also allows the player to modify the status of the station and to send messages to pedestrians around.

7.3. Running an example

An example of a scene with the ARROW public transport simulation implemented can be found in *Assets/Miscelanea/Tests/PublicTransportTest*. Once open, users can start running it pressing play. To illustrate the functionalities of this simulation, it runs along with a Flash Pedestrian Simulation. From the beginning of the gameplay, the users will see pedestrians being spawned and going to their destinations available in the map trying to use the public transport system to arrive faster. The users could click on some stations and close some of them, sending information to pedestrians to see their reactions and the changes in their decisions.



8. Traffic Integration Module

Unlike other modules of ProtoWorld that implement proprietary simulations, such as the pedestrian and the public transport modules, the Traffic Integration Module implements instead the functionalities needed to integrate well-known external traffic simulations within the ProtoWorld framework, allowing the users to integrate traffic simulations such as SUMO, VISSIM and MATSIM.

8.1. How to add the module

This module can be added from the menu *ProtoWorld Editor > Traffic Integration Module > Add Traffic Integration*. A new element will be created in the hierarchy with the name *TrafficIntegrationModule*. When selected, this element shows in the inspector a dropdown menu with all the different options available for traffic integration. Each of these options is explained in the following sections.

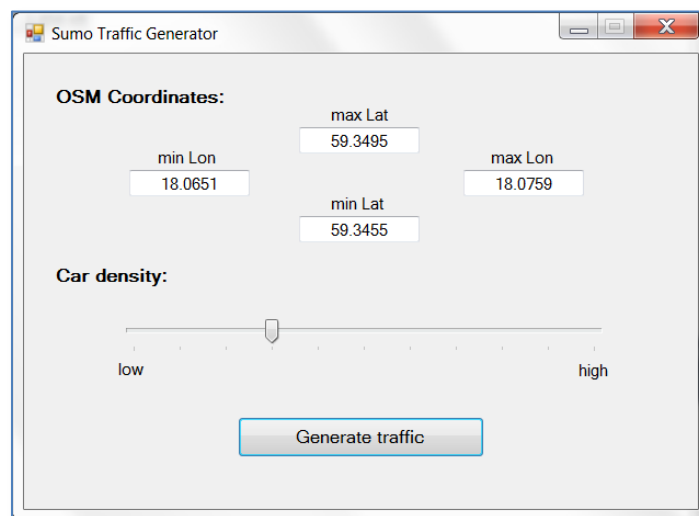
8.2. SUMO integration in real time

This kind of integration allows the users to integrate a real-time SUMO traffic simulation along with a running gameplay of ProtoWorld. SUMO 0.26 can be downloaded as an open source from the following [link](https://www.python.org/). It is important to notice that Python 3.x (<https://www.python.org/>) must be also installed along with the SUMO software.

➤ How to generate SUMO traffic for simulation

In the folder *ScriptAndTools*, users can find the source of an application called *SumoTrafficGenerator* that generates random traffic using the different tools provided by the SUMO software. In order to use it properly, users first need to compile and build this tool using Visual Studio. Once this is done, it can be executed:

- i. Run the *SumoTrafficGenerator* program. A window will open asking for the GIS coordinates and for a value for traffic density.
- ii. Adjust the GIS coordinates according to the map generated in Unity.
- iii. Adjust the value of the traffic density. Higher values imply more number of cars per second.
- iv. Click on the button *generate traffic* to generate all the necessary files needed to run a SUMO simulation in real time. A confirmation message will appear when the process is finished.



➤ Running SUMO

After all the SUMO files have been generated with the aforementioned tool, a new batch will appear in the folder destination called *start.bat*:

- v. Run the batch file. If all the SUMO environment variables were set correctly, SUMO will initialize its simulation.

➤ Integrating SUMO with ProtoWorld

For the final step, the Unity scene needs to be set for the integration:

- vi. Add the Traffic Integration Module and set it to *SUMO Live Integration* from the dropdown menu in the inspector.
 - vii. Adjust any additional parameter in the inspector, if needed. By default, all values of each parameter should have already the appropriate values for a standard integration.
 - viii. Start a gameplay. The module should establish a connection with SUMO and the integration should take place in real time. If everything is correct, some cars should be seen moving in the scene.
- ✓ *Developers note: There are no conventions about car angles between different traffic simulations, so the ProtoWorld framework might show wrong rotations of cars for certain cases. For advance users, please refer to the script `TrafficIntegrationVehicle.cs`, lines 164-170.*

➤ Interaction with other modules

The main advantage of this kind of integration is that it can run and interact in real-time with other simulations of the ProtoWorld framework such as, for example, the pedestrian simulation, meaning that the different agents of these simulations are aware of each other and react consequently. For the case of pedestrians, for example, cars will stop if pedestrians are blocking the road or passing through a zebra cross.

8.3. SUMO integration with FCD files

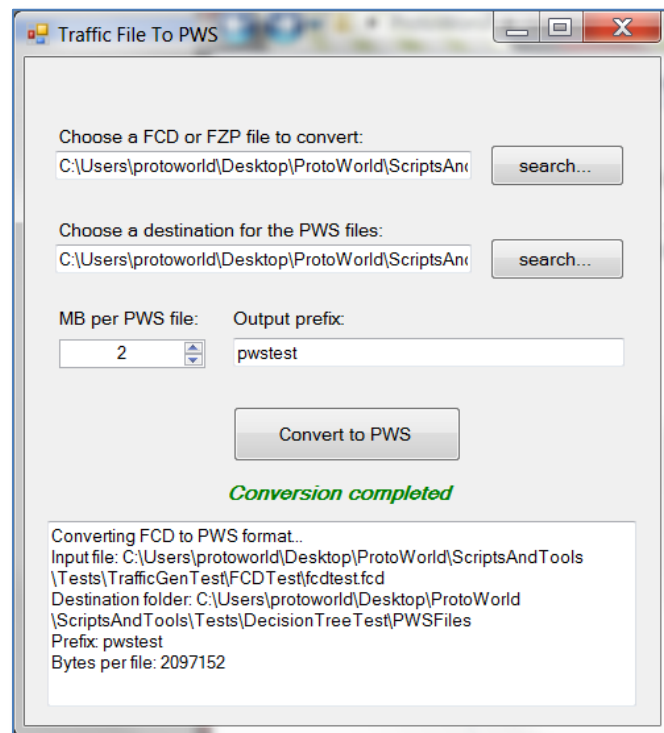
As an alternative to the live integration, the users can also input simulation data into the Traffic integration Module using an FCD file. The documentation provided by SUMO explains how to generate this kind of files. For the integration, the only thing the users would need to do would be to specify the path of such a file in the inspector menu and to run the gameplay. The cars should appear, but in this case there will not be interaction with any other module of ProtoWorld (cars will not stop for pedestrians, for example).

8.4. VISSIM integration with FSP files

Similar to the previous case, it is also possible to input FZP files to integrate VISSIM simulations in ProtoWorld. To generate this kind of files, refer to the VISSIM documentation. Same as before, in the inspector menu of the module users need to specify the path of the FZP file and then run the gameplay. There will not be any integration with other modules.

8.5. Integration with PWS files

For large FCD or FZP simulation files, there is an alternative to the integrations explained above. In ScriptAndTools, users can find a tool called *TrafficFiletoPWS* for converting FCD and FZP files into a more friendly format called PWS, which optimizes the structure of the traffic data and divide it in several files, allowing for a much faster loading time of traffic data during gameplay. The application is very straight forward: users just need to select the FCD or FZP file that they want to convert into PWS and specify the amount of MB that will have each PWS split. As an output, users will get a set of PWS files and a PWS.meta. In the Traffic Integration menu, the only thing to do would be to specify the location of the PWS.meta file and to run the gameplay. For large simulations, there should be a noticeable improvement in terms of loading time.



8.6. MATSIM integration

✓ *Developers note: This feature is under implementation. Future versions will cover this point.*

8.7. Decision Tree integration

The option for Decision Tree integration should be selected if users are also using the Decision Tree Module (see chapter 12 for more details).

8.8. Elements in this module

In this module, the users can see the following elements:

- *TrafficIntegrationData*: This element contains all the game objects and scripts needed to integrate external traffic simulations into Unity, including the graphic models for cars and other vehicles. Users can modify or add new models within the list Graphic Car contained into the component Traffic Integration Spawner.
- *SumoController*: Contains the functionalities to integrate and synchronize SUMO. It also implements an API for communication with the SUMO interface TraCI.
- *SimulationReader*: Internal element of the module that is in charge of reading FCD, FZP and PWS files in order to populate the traffic data used during gameplay.
- *SumoBusSpawner*: Internal element to spawn buses in a SUMO simulation in real-time from the gameplay. This is an experimental tool.
- *TrafficIntegrationCanvas*: Canvas that contains the UI element to display information related to the traffic simulation during gameplay. Only used if the *SumoBusSpawner* is active.

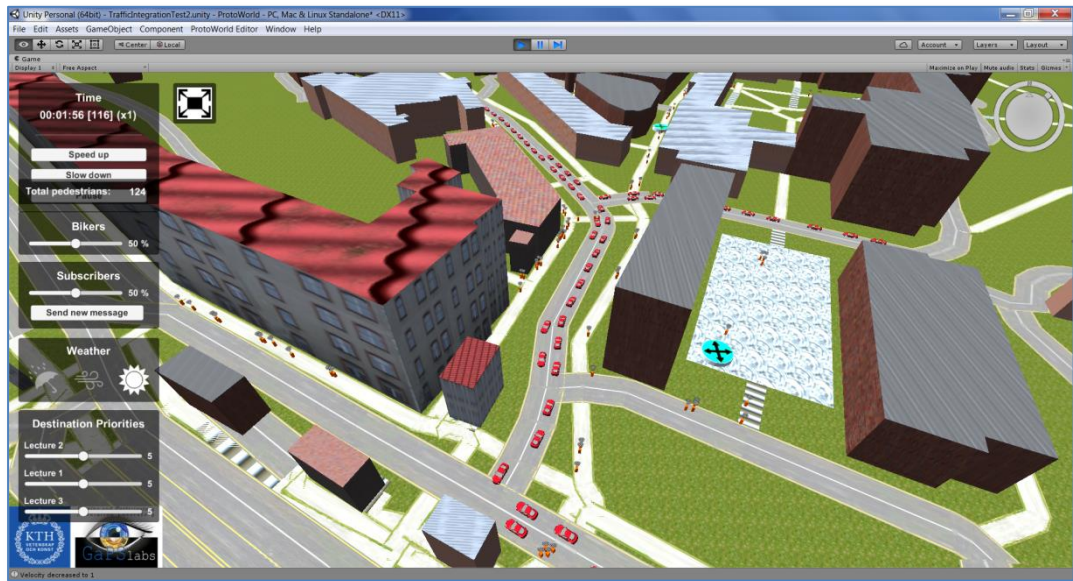
8.9. Running an example

Examples of scenes with the integration of traffic from SUMO can be found in *Assets/Miscelanea/Tests/TrafficIntegrationTest*. There are two examples, one with an FCD file and another with a live integration from SUMO:

- The first test (*TrafficIntegrationTest1.unity*) can be run directly just starting the gameplay. The FCD file provided for this test can be found in *ScriptAndTools/TestFiles/TrafficGenTest/FCDTest*. Before running the gameplay, users should check that the path to the FCD file is set properly in the Traffic Integration Module.
- For the second test (*TrafficIntegrationTest2.unity*), users first need to start a real-time SUMO simulation. The files of a ready-made SUMO simulation for this scene can be found in the folder *ScriptAndTools/TestFiles/TrafficGenTest/LiveTest*. There, users just

need to execute the batch file *start.bat* to start the traffic simulation and to start the gameplay in Unity afterwards.

To illustrate the functionalities of this integration, these examples run along with a pedestrian simulation. From the beginning of the gameplay, the users will see cars and pedestrians moving around the scenario. For the second case, the users could navigate around and see how the interaction works: the cars will stop when pedestrians take over the roads.



9. Drag and Drop Module

This module provides a tool for the users to generate an in-game menu where players will be able to drag and drop different elements into the scene during gameplay. This is a tool to enhance the interaction between the players and the simulations running within the ProtoWorld framework.

9.1. How to add this module

This module can be added from the menu *ProtoWorld Editor > Drag and Drop Module > Add Drag and Drop system*. A new element will be created in the hierarchy with the name *DragAndDropModule*.

➤ ***A drag and drop example: public bike stations for pedestrians***

By default, the drag and drop system contains already an element in the menu that is fully functional with the module: an element to drag and drop public bike stations for pedestrians. This element, integrated along with the FLASH pedestrian simulation module, allows the players to place bike stations on the scene, giving pedestrians the

possibility to take also bikes to reach their destinations, as an alternative to walking or public transport. This element, called *FlashBikeStation*, can be found as a prefab in *Assets/Assets ProtoWorld/PedSimulation/Prefabs/FlashPointsPrefabs*. This is a perfect example and a good template for the users to start creating their own drag and drop objects. More details about this can be found in section 9.3.

9.2. Elements in this module

In this module, the users can see the following elements:

- *InstantiatedObject*: This element holds during gameplay the drag and drop objects that have been placed in the scene by the player.
- *DragAndDropCanvas*: Canvas that contains the UI elements that are part of the Drag and Drop module. It is here where the users can add and edit the drag and drop elements.

9.3. How to add more drag and drop elements

There are two main steps that need to be followed in order to add a new drag and drop element:

➤ **First step: making the game object draggable**

An game object needs to be draggable before it can be taken as part of the Drag and Drop system:

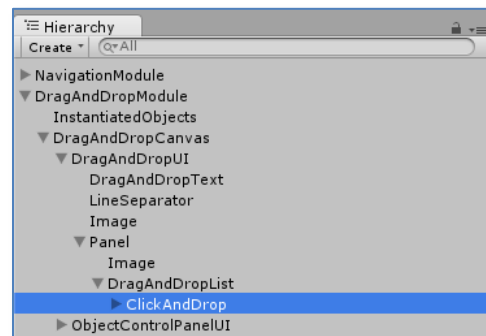
- i. Create a prefab out of the game object that is going to be draggable.
- ii. Add the *DragTransform* component to it. This component contains the functionalities to make the object draggable and it provides event functions that can be filled with the functionalities that the users wishes for actions such as selected, deselected, dropped, etc.
- iii. Add the *ObjectData* component to it. This component displays information about the game object in the UI canvas when it is selected, including a slider and several buttons that the users can fill with customized functionalities. In the case of the *FlashBikeStation* object, the buttons have been filled with functionalities to add and remove bikes from the station.

➤ **Second step: adding the dragabble object to the drag and drop menu**

Once the game object is dragabble, it can be added to the drag and drop menu:

- iv. Go to the hierarchy and navigate inside the module: *DragAndDropCanvas > DragAndDropUI > Panel > DragAndDropList*. There, there should be an element called *ClickAndDrop*.

- v. Make a copy of the *ClickAndDrop* element.
- vi. Inside the *ClickAndDrop* element, find the component called *ClickAndSpawn*. There, there is a field called *ObjectToInstantiate* that needs to be filled with the prefab of the draggable game object.
- vii. In the image component of *ClickAndDrop*, customize the sprite that will represent the draggable game object. The name to display in the menu can be edited in its child object. The game object should be now ready to be dragged and dropped during gameplay.



9.4. Running example

An example of a scene with the Drag and Drop module can be found in *Assets/Miscelanea/Tests/DragAndDropTest*. This scene is set along with the pedestrian simulation module to illustrate better the concept of the *FlashBikeStation* object. From the beginning of the gameplay, the users will see pedestrians being spawned and going to the different destinations available in the map. If the users drag and drop some bike stations around, some pedestrians will change their mind and go for a bike. That behaviour will also depends on the percentage of bikers set for pedestrians in the menu. The users could click on the bike stations and play with the parameters that the UI offers, seeing how that affects the decisions of pedestrians.



10. VVIS Visualization Module

VVIS is a tool that allows the users to visualize road data from a SQLite database over a scenario. Although it is not part of the integration of simulations, it provides tools to the users to facilitate data visualization based on trajectories (sets of OSM roads) and points (sets of GIS coordinates), taking the scenario generation and the navigation of the ProtoWorld framework as a base for this visualization. Also, it provides tools to allow the players draw in the map, functionality that is integrated along with the Drag and Drop Module.

10.1. How to add this module

This module can be added from the menu *ProtoWorld Editor > Visualization Module > Add VVIS Module*. A new element will be created in the hierarchy with the name *VVisDataSQLiteModule*.

10.2. Elements in this module

In this module, the users can see the following elements:

- *VvisVisualizer*: Contains the visualization functionalities of the module. The users can customize some parameters in it, such as the color for the drawing tool or the gradients for the visualization of trajectories.
- *VVisDataSQLite*: Responsible of the connection between the module and the SQLite files.
- *VvisCamScreenshot*: Provides the functionalities of taking screenshots of the whole map during gameplay. The size of the output image can be set here.
- *VvisCanvas*: Canvas containing all the UI elements related with this module. It includes the menu with the drawing tools and the query tools. It is recommend it not to modify it. Within the child element *VVisUI*, there is a game object called *BlockingWindow* where a key can be set to unblock the canvas for queries during gameplay.
- *VvisFixPoint*: Internal to the module, it is used to generate points.

10.3. Preparing data files using SQLite

There are two kinds of data that can be visualized with this module: trajectories and points. For each of them, a SQLite file needs to be generated, which will be read later on by the module during gameplay:

➤ **Trajectories:**

They represent complete trajectories of vehicles following a route, that is, a collection of OSM roads. To create a SQLite database with this data, the schema of the table should match the one shown below this paragraph. The SQLite file can be generated using free software tools such as SQLiteMan (<http://sqliteman.yarpen.cz/>). Once generated, the SQLite file should be named *trajectoriesDB* and it should be placed inside the ProtoWorld Unity Project within the folder *Assets/StreamingAssets/SQLiteDB*. Other names for the file could be chosen, but that will imply editing also the parameters in the *VVisDataSQLite* game object. This is the schema of the table:

```
CREATE TABLE trajectory_paths (  
    "tid" INTEGER NOT NULL,  
    "vid" INTEGER,  
    "seq" INTEGER NOT NULL,  
    "nodeid" INTEGER,  
    "edgeid" INTEGER,  
    "cost" REAL,  
    "id" INTEGER,  
    "osm_id" INTEGER  
);
```

➤ **Points:**

They represent points on the map. To create a SQLite database with this data, the schema of the table should match the one shown below this paragraph. Once the SQLite file has been generated, the file should be named *pointsDB* and it should be placed inside the ProtoWorld Unity Project within the folder *Assets/StreamingAssets/SQLiteDB*. Other names for the file could be chosen, but that will imply editing also the parameters in the *VVisDataSQLite* game object. This is the schema of the table:

```
CREATE TABLE "points" (  
    "no" INTEGER,  
    "count" INTEGER PRIMARY KEY,  
    "long" REAL,  
    "lat" REAL  
);
```

10.4. Preparing queries using an XML file

In addition to the aforementioned files that contain the SQLite data, the users can also define a list of queries to give to the players during gameplay. These queries can be stored in an XML file that will be read by the module during game time, showing a dropdown menu with all of them. The XSD file with the schema of it can be found at *Assets/Assets ProtoWorld/VinVisualization/Schemas/*. These are the elements of the XML file:

- Root *vvis_queries*: Root of the XML file.
- Element *queries*: Nested in the *vvis_queries* root. Holds the list of queries.
- Element *query*: Nested in the *queries* element. Defines a query for the SQLite database of trajectories.
 - Attribute *name*: Name of the query that will appear in the dropdown menu.
 - Attribute *query*: SQL query to be executed when running this.
 - Attribute *info*: Information to be displayed in the UI while selecting this query.

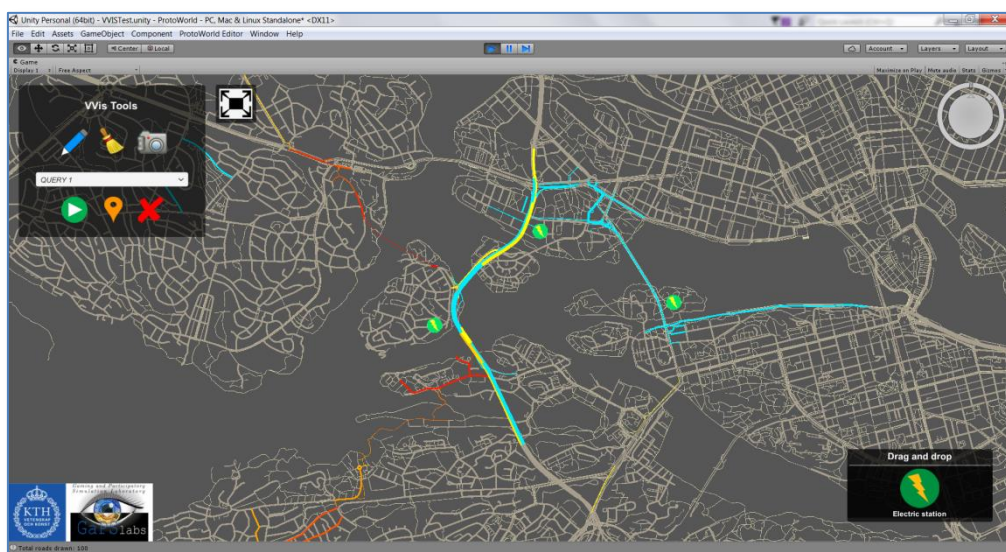
Once the XML file has been defined, it should be named *vvis_queries.xml* and placed inside the ProtoWorld Unity Project within the folder *Assets/StreamingAssets/SQLiteDB*. Other names for the file could be chosen, but that will imply editing also the parameters in the *VvisCanvas > VvisUI* game object.

10.5. Preparing map for visualization

For a proper visualization of the data, a map must be first generated using the Scenario Generation module explained in chapter 4. Also, for a better visualization, it is recommended to use an orthographic camera in top-bottom view (see section 5.3)

10.6. Running example

An example of a scene with the VVIS module can be found in *Assets/Miscelanea/Tests/VVISTest*. The SQLite and XML files used for this test can be found in *Assets/StreamingAssets/SQLiteDB*. From the beginning of the gameplay, the users will be able to select drawing tools and to move around the map. By default, the query tools will be blocked, but they can be unblocked clicking on the key icon and inserting the key *123* within the menu that pops-up. Once this is done, the users can try to select and run a query and to play around with the different tools.



11. Logger Module

The ProtoWorld framework allows the users to include a logger in their scenes to track the interaction of the players with the final game. This logger, based on the Log4Net software (<https://logging.apache.org/log4net/>) creates a log file of gameplay actions such as buttons clicked, parameters changed, etc. It is important to notice that, due to the nature of Log4Net, this logger only works for standalone releases of scenes, but not within the Unity Editor (see chapter 14 for more information about how to make releases of scenes).

11.1. How to add this module

This module can be added from the menu *ProtoWorld Editor > Logger Module > Add Logger Module*. A new element will be created in the hierarchy with the name *LoggerAssembly*.

11.2. Adding new logging lines in the code

Every module in the ProtoWorld framework has been prepared to work along with the Logger module to log the most essential data. Advance users that know c# and that want to log additional data in the scripts need to follow these simple steps:

- i. Check if the script has the following global variable. If it does not, add it:

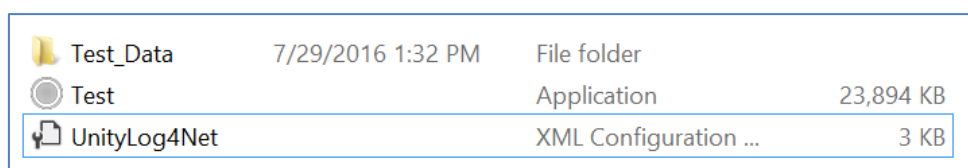
```
private static readonly log4net.ILog log =  
log4net.LogManager.GetLogger(System.Reflection.MethodBase.GetCurrentMethod().Declaring  
Type);
```




- ii. Use the following line in the code to create a new log message:

```
log.Info(<you Log message>);
```

11.3. Setting Log4Net in the release

As mentioned before, the logger module only works for final releases. Once the release has been made (see chapter 14), all the users will have to do is to include a log4net config file beside the executable. A ready-made log4net config file can be found in *ScriptAndTools/LoggerConfig*. Users just need to copy the file *UnityLog4Net.config* in the same folder where the executable of the release is. For more information about how to modify this config file, please refer to the documentation of Log4Net.



	Test_Data	7/29/2016 1:32 PM	File folder	
	Test		Application	23,894 KB
	UnityLog4Net		XML Configuration ...	3 KB

12. Decision Tree Module (*beta*)

This module was conceived for a specific case along with the PWS traffic simulation files (see section 8.5). The idea of this module is to allow the users to implement events in the scene before gameplay such as pop-up questions, information boxes and changes in PWS traffic simulations. This brings the possibility to steer the players through the gameplay and also allow them to take decisions at certain points in time, altering the PWS traffic simulations with their decisions. This is done via an XML file that will store all the information needed.

- ✓ *Developers note: This module still needs further testing, so some of its functionalities might not work as expected.*

12.1. How to add this module

This module can be added from the menu *ProtoWorld Editor > Decision Tree Module > Add Decision Tree Module*. A new element will be created in the hierarchy with the name `DecisionTreeModule`.

12.2. Writing the XML decision tree file

The decision tree XML file defines all the information needed to display events during gameplay. The following lines cover the structure of this file and its elements. The XSD file with the schema can be found in *Assets/Assets ProtoWorld/DecisionTree/Schemas*.

- Root *decision_tree*: Root of the XML file.
- Element *scenarios*: Nested in the *decision_tree* root. Holds all the scenarios.
- Element *scenario*: Nested in the *scenarios* element. Specifies a whole gameplay scenario, containing information about which PWS traffic simulation to run and all the questions that will appear through time. This element contains two attributes:
 - Attribute *id*: Identifies the scenario with a number. Scenario with *id*=0 will be the first one to run by default.
 - Attribute *name*: Name of the scenario. It will be displayed on the UI during gameplay.
- Element *simulation*: Nested in the *scenario* element. There can only be one of these elements for each scenario. Specifies the PWS traffic file to run:
 - Attribute *path*: Path of the PWS metafile that holds the information of the traffic simulation that should be run for this scenario.
- Element *question*: Nested in the *scenario* element. It defines pop-up questions and information boxes that will appear to the player during gameplay. There can be several

question elements nested in a *scenario* element. Each *question* element contain three attributes:

- Attribute *id*: Identifies the question with a number.
 - Attribute *time*: Specifies with a float at which time the question should appear during gameplay.
 - Attribute *text*: Text of the question.
 - Attribute *landmark*: This is an optional element. If defined, specifies where the camera should focus during gameplay when the question pops-up. See next section for more details.
- Element *choice*: Nested in the *question* element. Represents a possible answer/choice of the *question* element where it is nested:
 - Attribute *text*: Text with the answer/choice.
 - Element *execScen*: Nested in the *choice* element. Specifies the next scenario to run in case that choice is taken during gameplay. The id of the next scenario to run should be specified in the xml text of this element.

12.3. Elements in this module

In this module, the users can see the following elements:

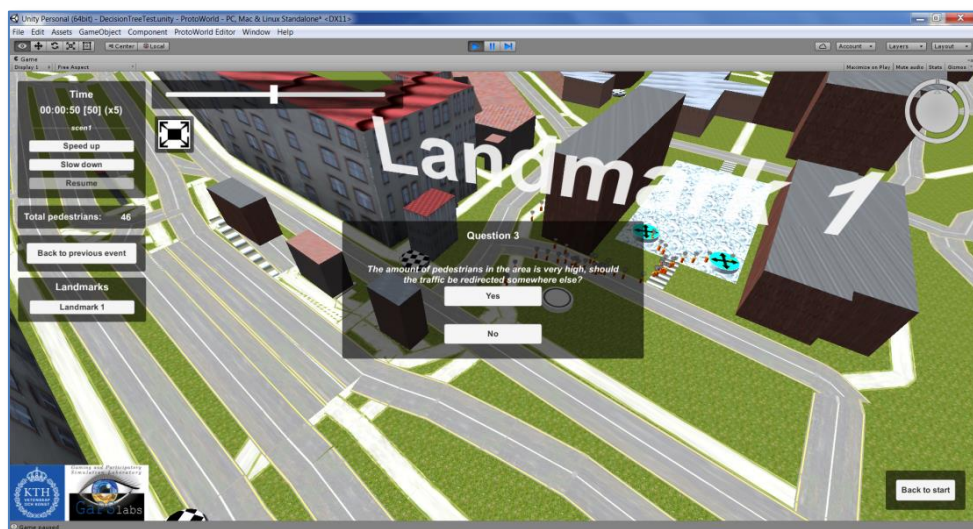
- *ScenarioController*: Internal element that holds the script to control the scenarios in the decision tree. It is recommended not to modify this.
- *DecisionTreeCanvas*: Canvas that contains the UI elements related to the decision tree, such as the question canvas and the navigation buttons to allow the player to go back and forth in the events.
 - ✓ *Developers note: when the decision tree module is running, the player can also activate a time slider at the bottom of the screen. This UI element can be activated in the navigation module. The users could go back and forth in time using this slider.*
- *LandmarkPoints*: This element holds the game objects that represent the landmarks in the scenario, defined in the *question* elements to focus the camera when a new question pops-up. Each of this elements holds a script *LandmarkController* that defines an id (the one that identifies the landmark and the one that should be used in the XML file) and an optional tag. The users can have as many landmarks as he/she wished on the scenario. To add new landmarks, just make duplicates of the Landmarks that are already there.

12.4. Preparing the module for gameplay

Once the XML has been correctly generated, the path of the file should be specified in the ProtoWorld Unity Project before starting the gameplay. On the element *DecisionTreeModule* in the hierarchy, the users will see a script called *DecisionTreeController* with a string field called *Path Decision XML File*. Here the users should write the path of the decision tree XML file that will be used to define the decision tree in the gameplay. The *TrafficGenerationModule* should also be there, with the option *Decision Tree Module* selected.

12.5. Running an example

An example of a scene with the decision tree module implemented can be found in *Assets/Miscelanea/Tests/DecisionTreeTest*. An example of an XML decision tree file along with a PWS simulation can be found in *ScriptsAndTools/TestFiles/DecisionTreeTest*. Before running the scene, users should check that the path of the XML file is set properly in the *DecisionTreeModule*. During the gameplay, the users will see some pop-up questions and some changes in the traffic simulation according to the choices taken.



13. KPI Module (beta)

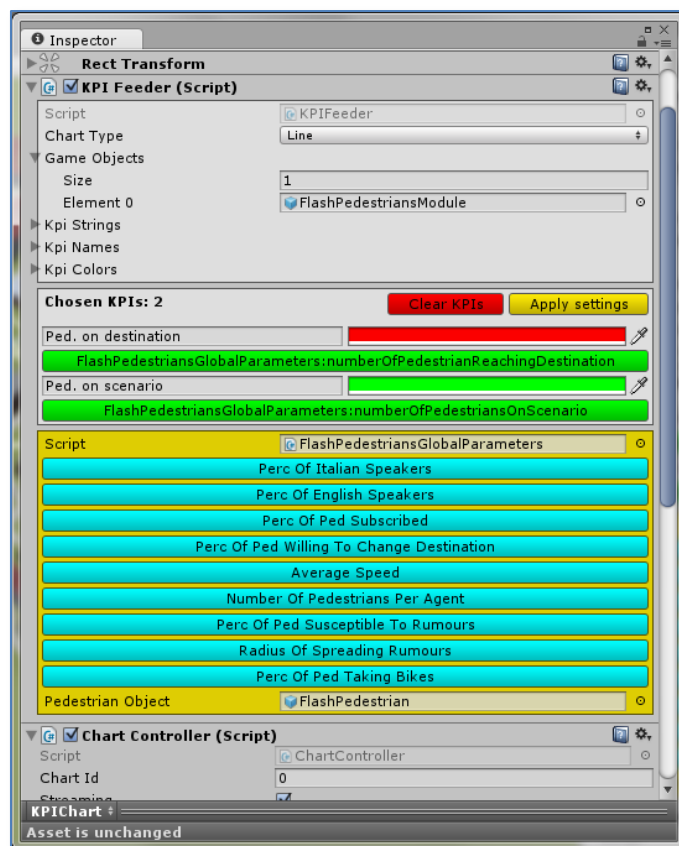
This module provides functionalities to display graphs, charts and other kind of KPIs to plot statistics of different parameters of the ProtoWorld framework during gameplay. These parameters can be extracted from any other of the modules explained in the previous sections. For example, with the KPI module the users will be capable of plotting real-time information of pedestrians queuing on a station, reaching their destinations or using bikes.

- ✓ *Developers note: This module still needs further testing, so some of its functionalities might not work as expected.*

13.1. How to add the module: adding charts

The KPI module will be added automatically the first time the users request adding a KPI chart. This can be done from the menu *ProtoWorld Editor > KPI Module > Add KPI Chart*. A new element will be created in the hierarchy with the name *KPIModule*, and a menu will appear on the inspector window to customize the new chart in a component called KPI Feeder:

- i. From the dropdown menu in the inspector, select one of the types of charts: choose between bar, pie, line or stacked area.
- ii. Select the elements that will feed the KPI. To do so, drag the modules that you are interested in tracking inside the parameter *Game Objects* in the KPI Feeder. That will open a new tab within the inspector with a list of parameters that can be tracked from that module.
- iii. Choose the parameters to track and assign a color to each of them. Notice that the tags of the chart can be also modified from the parameters *KPI strings* and *KPI Names* in the KPI Feeder component.
- iv. Once is finished, click on the yellow button *apply settings*. Follow the same process to generate more charts.

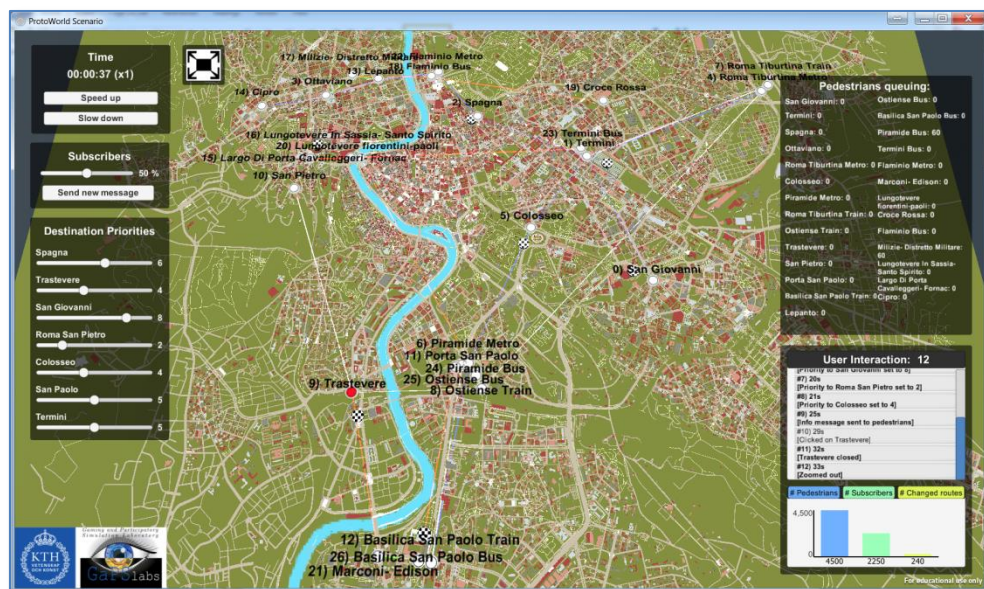


13.2. Elements in this module

This module contains only one element: the *KPI Canvas*, which contains the UI elements needed to be able to portraint the KPI charts on the screen during gameplay. Within this module, each KPI chart will have its own game object, where the users will be able to edit the chart information.

13.3. Running an example

An example of a scene with the KPI module can be found in *Assets/Miscelanea/Tests/KPITest*. Other modules have been also added to showcase the monitoring of parameters from FLASH pedestrian simulation and ARROW public transport. When the gameplay starts, the users will see how the charts built with the KPIs are feeded with the data from the simulations in real time, and how the player interaction with the game alters the values of the data plotted.



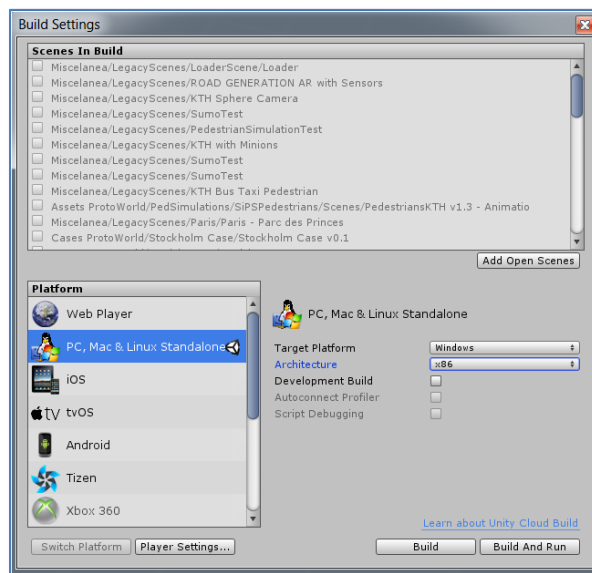
14. Releasing a ProtoWorld Scene

After the users have added and set up all the desired modules in the scene, and after testing the gameplay in the Unity editor, the last step would be to generate a release of the scene. This release will create a standalone executable that allows the portability of the scene to any other computer without the need of having the ProtoWorld framework built within. It is important to notice, though, that external live simulations, such as SUMO, would still be needed to run along with this release.

14.1. Releasing process

The process for releasing the scene is all done through the Unity editor. It is recommended to save the scene generated before starting the release:

- i. In the Unity Editor, go to the menu *File > Build Settings*. There, select a platform to build the executable and an architecture. All the developer tests have used Windows with x86 architecture. Other platforms might work as well, but they have not been tested.
- ii. Click on the *Build* button and select a destination folder to make the release. The process will take some minutes.
- iii. Once completed, an executable with a folder tagged as *_Data* should appear in the destination folder selected.



14.2. Executing the ProtoWorld release

For Window platforms, the folder with the release of the scene should contain a folder with the tag *_Data* at the end of the name, and an executable with the name choosen for the release. The aforementioned folder contains all the files and libraries related to Unity and the ProtoWorld framework. In this folder, the users will also find the folder *StreamingAssets* and any picture taken with the VVIS tool for screenshots. Running the executable will start the gameplay of the scene.

If the Logger Module was included in the release, users should include the log4net config file beside the executable file before running it. Also, it is important to notice that, if the scene contains a Live SUMO integration, the SUMO simulation should be started before running the executable.

Coda

This is the end of this guide. The developers hope that users find this document useful and that it helps them to take the most of this software. Now it is time to build new scenes and keep playing with ProtoWorld.

Have fun!