Unity program

Workings of visualization

Kersten,Jorn J.L.V.
5-26-2023

Table of Contents

About this document

In this document you will find more information or how the Unity application works. The application is a visualization of an assembly line, it communicates with a Beckhoff TwinCAT plc. The application sends the location data of the blocks on the conveyor belt to the plc so that a robot arm can go to that position in real life.
In the first section of this document you will find the different classes that are out there and what they do. Then you find which objects in the visualization contain these classes.

Classes

ConveyorBelt.cs

This class contains the code for the conveyor belt. In the Start method, all relevant settings from the main menu are retrieved and set correctly in the object.
First the speed is retrieved and set, then the length and width are retrieved and set. It is first checked whether both length and width are set in the main menu, if so then the length and width of the conveyor belt are adjusted accordingly.
If this comparison is not true, the length of the conveyor is set first if it is filled, otherwise the conveyor gets a default length of 20.
Then the width is checked, if it is set then the width of the conveyor belt is adjusted accordingly. If it is not set, then the width of the conveyor belt is set to the standard width of 5.
After the speed, length and width are set, the scene is unloaded from the main menu and the camera is changed to the correct camera. Also, there is no need to keep that scene loaded and it is unloaded for less lag.

The Update method we check if a cube has entered a detection area on the conveyor belt. Then if it has entered the area and the robot arm is ready for picking up the cube, the conveyor belt is slowed down to 0 speed. Then the location of the cube is sent to the PLC, along with changing a bool so the arm knows there is a new coordinate to go to. Then the arm “picks up” the cube on the belt, and the belt is set to be moving again.

Next the SlowDownBelt method, this method does exactly what you think it does: it slows down the conveyor belt. It does this off of a timer, so it doesn’t abruptly stop. It incrementally slows it down by 1 speed per loop, the whole function is an async function. There is a await Task.Yield() at the bottom of the method, this tells the unity program to asynchronously wait for the task in the method to be finished before moving on.

Then the SpeedUpBelt method, this method does exactly what you think it does again: it speeds up the conveyor belt. It does this off of a timer, so it doesn’t abruptly start. It incrementally speeds it up by 1 speed per loop, the whole function is an async function. There is a await Task.Yield() at the bottom of the method, this tells the unity program to asynchronously wait for the task in the method to be finished before moving on.

In the FixedUpdate method, several things are done. The first thing is the code of the conveyor belt itself, this "moves" the conveyor belt. What this code does is somewhat special, in one frame the position of the rigidbody (the object to which the code is attached) is moved in a fixed direction. This is done by multiplying the direction by the speed and time span of 1 frame. Then the object is returned to the position it was at before it was moved.
Doing this moves an object across the conveyor belt.

The OnCollisionEnter method keeps track of when an object lands on the conveyor belt, then this method adds that object to a list of objects.

The OnCollisionExit method does the opposite of the OnCollisionEnter. It keeps track of when an object goes off the conveyor belt, then this method removes that object from the list of objects.

DestroyObjects.cs

This class handles object destruction as described. The Start method retrieves all relevant settings from the main menu and sets them correctly in the object. The length and width are retrieved and set. It is first checked whether both length and width are set in the main menu, and if so, the position of a surface to be placed after the conveyor belt is adjusted accordingly. The width is divided by two because, of course, the plane must be in the middle of the conveyor belt.
If this equation is not true, first position length of the plane is set if it is filled, otherwise the plane is given a default length of 25.
Then the width is checked, if it is set, then the width position of the plane is adjusted accordingly. If it is not set, then the width of the conveyor belt is set to the default width of 2.5.

The OnTriggerEnter method keeps track of whether an object enters the plane's trigger field. If so, the object entering the trigger is destroyed.

DetectCube.cs

This class detects if a cube enters a plane. The Start method retrieves all relevant settings from the main menu and sets them correctly in the object. The length and width are retrieved and set. It is first checked whether both length and width are set in the main menu, and if so, the position of a surface to be placed a little past half way of the conveyor belt. If none of the settings are entered, then the width and length of the surface are 5 by 5. last, it is calculated what the end of the conveyor belt is so that it can be used in calculations for sending to the Beckhoff PLC.

The Update method look whether a list object contains 1 or more objects. Namely, this means that a cube has come inside the detection plane. It then sets a boolean to true so that a method in the ConveyorBelt.cs can respond to it. If there are no objects in the list object, the boolean is set to false.

The OnTriggerEnter method is called by Unity when an object enters the trigger. When this happens, the method adds the object entering the trigger to a list that keeps track of whether any objects have entered the trigger.

The OnTriggerExit method is called by Unity when an object exits the trigger. When this happens, the method removes the object entering the trigger from the list that keeps track of whether any objects have entered the trigger.

MainMenu.cs

The MainMenu class only keeps track of when the button at the bottom of the main menu is clicked. When this is clicked, the "Gamescene" is added to the current scene. This is done this way on purpose, if the "Gamescene" scene is just loaded, then the "Mainmenu" scene is unloaded immediately.

OptionsPauseMenu.cs

This class opens a menu to change some settings during visualization. In the Start method, the canvas is retrieved and immediately set to inactive, this makes it invisible to the camera.

In the Update method, it checks whether the escape key on the keyboard is pressed during the visualization. If so, a Boolean is modified to true or false to open or close the menu.

In the PauseGame method, the canvas is set to active, then the Boolean that keeps track of whether the game is "paused" is set to true.

In the ResumeGame method, the text from the menu is retrieved to adjust the speed, this is done only if the user has actually entered something. Next, the canvas is set to inactive so that it is no longer visible to the camera. Finally, the Boolean that keeps track of whether the game is "paused" is set to false.

Spawner.cs

In this class, objects are spawned at a designated location. In the Start method, some settings are retrieved from the main menu. It starts by setting the maximum width in which the spawner object can spawn. This is width of the conveyor belt - 0.5.
Then the amount of objects that are spawned is set. Then the time between spawning of the objects is set and finally the weight of the objects. All these things are set only if a value is actually entered in the main menu.
After everything is set, the SpawnObject method is called.

The SpawnObject method first creates a random width position between minimum and maximum width where an object can spawn between. Then a random height position and width position are generated in the same way.
Then a copy of the object to be spawned is made. It is then renamed with the number of the amount of spawned objects. Then the weight of the object is set.
Then it is spawned, this is done by creating an instance of it. Here the object, random position, rotation and transform are given. Directly after creating a new instance the object is no longer made kinematic so that gravity and other forces have an effect on the object. Finally, the counter of amount of spawned object is increased.

The Update method calls the SpawnObject method when less than the specified number of objects are spawned. The Update method does this only when the correct amount of seconds has elapsed between spawning objects. Besides that, the cubes are also spawned only if a variable is set to true. This is the doSpawn boolean, which is updated by the ConveyorBelt class when the conveyor stops or starts.

Visualisation objects and what they do

There are many objects distributed between the two scenes. In the Mainmenu scene you will find the canvas object containing all items relevant to the layout and inputs of the main menu. Next you will find the Directional Light, this provides a light so that the whole scene is illuminated. Finally, it contains a camera, this so that the main menu is also visible, of course.

In the Gamescene scene you have some "special" objects. These objects contain scripts/classes that are needed to perform "special" tasks.
The first object in the scene is the InGameCanvas. This is the "pause" menu, this contains the settings when the escape key is pressed. Next is the Camera, which contains the OptionsPauseMenu script. Then follows the Directional Light this of course again takes care of the lighting in the scene. Next you see the spawner object, which contains the Spwaner.cs script. Next is the Object Destroyer object, which contains the Destory Objects.cs script this object destroys cubes when they come into contact with the object. Next we see the Terrain object, this object is self-explanatory and has no additional scripts. Then we come to Conveyor belt (unused) this is a copy of the used Conveyor Belt as a backup in case something goes wrong when setting/adjusting the Conveyor Belt. Then we have the SpawnerCube this is the cube used by the Spwaner.cs script, it copies this cube. Last is the Conveyor belt scaler object, it makes sure that the conveyor belt is scaled correctly. Inside the Conveyor belt scaler is the regular Conveyor blet, it contains the script for moving cubes and communicating via ADS with the Beckhoff plc.

Scenes

The visualization consists of two scenes, the Main menu scene and the Games scene. In the Mainmenu scene, the settings are completed prior to actually starting the visualization and thus the visualization can actually be started.

The Gamescene scene is not loaded before the visualization is started, this can be seen by the lighter gray color and the text: "(not loaded)". If this is not the case, it must be unloaded before the visualization is started. This can be done by right-clicking on the scene and selecting unload scene.
In the Games scene the visualization takes place, here the conveyor belt is set to the correct length and width derived from the given settings on the Main menu scene. The speed is also taken from the Main menu scene here.

What is needed to run the visualization

To run the visualization, Unity 2022.1.21f1 must be installed. It is important to use exactly this version; an older or newer version may cause the Unity application to stop working. In addition, the PC with the visualization must be connected to the PLC so that they can communicate directly with each other, another option is by running TwinCAT locally. If you want to run TwinCAT locally, TwinCAT 3.1 is required. You can also run the visualization without being connected to TwinCAT.