|  |
| --- |
|  |
| Unity program |
| Workings of visualization |

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

Table of Contents

**No table of contents entries found.**

# 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.

In the FixedUpdate method, several things are done. The first thing is to keep track of the object that is first on the conveyor belt. This is constantly updated so that when sending data to the plc, this is easier to keep track of.

This is followed by 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.

## 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.

# Visualisation objects and what they do

Er zijn veel objecten verdeeld over de beide scenes. In de Mainmenu scene vindt je het canvas object met daarin alle items die relevant zijn voor de opmaak en inputs van het hoofdmenu. Daarnaast vind je het Directional Light, deze geeft een lichtinval zodat de hele scene belicht is. Als laatste bevat deze een camera, dit zodat het hoofdmenu natuurlijk ook zichtbaar is.

In de Gamescene scene heb je een aantal "bijzondere" objecten. Deze objecten bevatten te scripts/klasses die nodig zijn om "bijzondere" taken uit te voeren.

Het eerste object in de scene is de InGameCanvas. Dit is het "pauze" menu, hierop staan de instellingen wanneer de escape toets wordt ingedrukt. Daarna de Camera, deze bevat het OptionsPauseMenu script.

Dan volgt de Directional Light deze zorgt natuurlijk wederom voor de belichting in de scene.

# Scenes