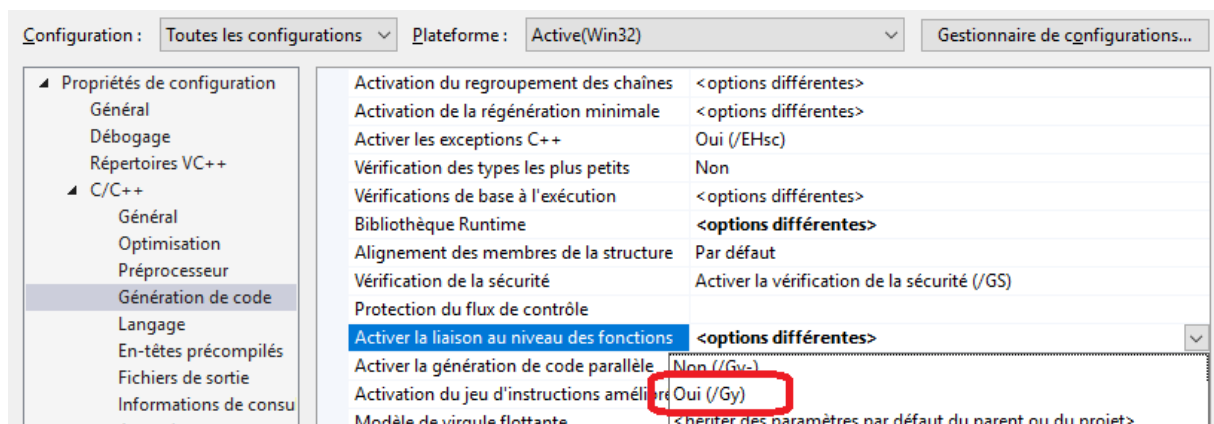


# Cours 8IAR125: Intelligence Artificielle pour le Jeu Vidéo

## Instructions et conseils Laboratoire #2 : Steering Behavior

### 1. Instruction d'installation de l'application

Suivez les mêmes étapes que pour le premier laboratoire jusqu'à la modification de l'option **Activer la liaison au niveau des fonctions** avec la valeur **Oui (/Gy)**. (C/C++ > Génération de code).



Le code a besoin d'être modifié pour pouvoir compiler. Ouvrez le fichier `misc/Path.h`. Supprimez `, curWaypoint(NULL)` à la ligne 36 et `assert(curWaypoint != NULL);` à la ligne 54 comme sur l'image suivante.

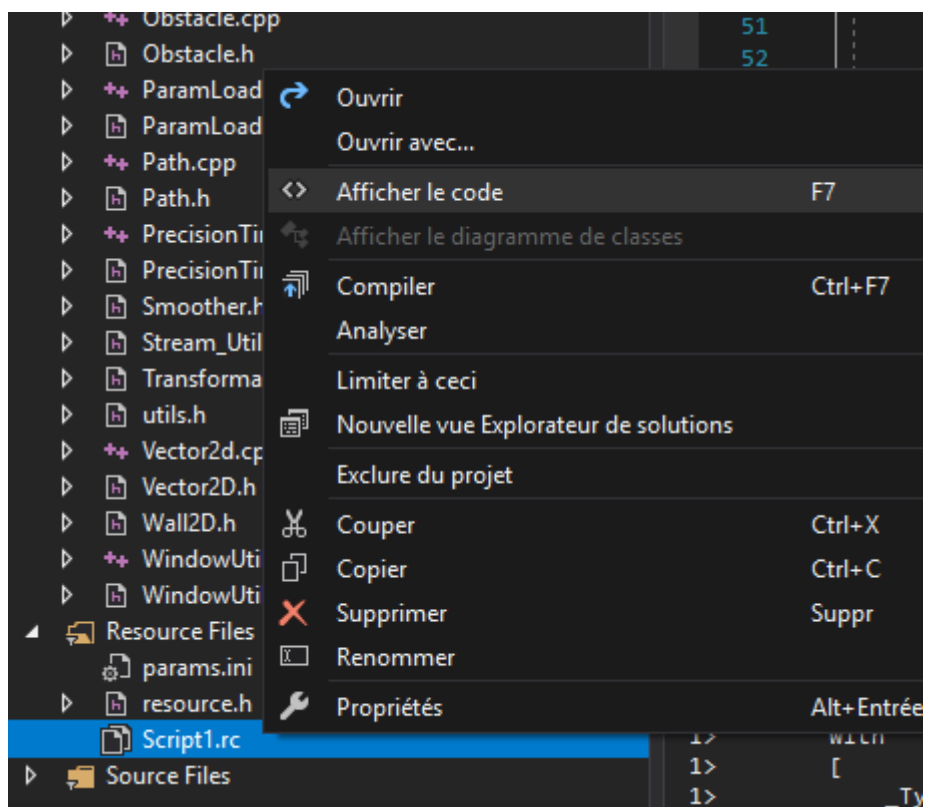
The screenshot shows a C++ IDE with a project named 'Steering'. The left sidebar displays the project structure, including 'Header Files' and 'misc'. The main editor shows the implementation of the 'Path' class in 'Path.cpp'. The code includes a private member 'curWaypoint' of type 'std::list<Vector2D>::iterator', a 'm\_bLooped' boolean, and several public methods: 'Path()', 'Path(int NumWaypoints, double MinX, double MinY, double MaxX, double MaxY, bool looped)', 'CurrentWaypoint()', and 'Finished()'. The 'Path()' constructor initializes 'curWaypoint' to NULL. The 'Path()' constructor with parameters calls 'CreateRandomPath' and sets 'curWaypoint' to the beginning of 'm\_WayPoints'. 'CurrentWaypoint()' returns 'curWaypoint' with an assertion. 'Finished()' returns true if 'curWaypoint' is at the end of 'm\_WayPoints'.

```

27 //points to the current waypoint
28 std::list<Vector2D>::iterator curWaypoint;
29
30 //flag to indicate if the path should be looped
31 //(The last waypoint connected to the first)
32 bool m_bLooped;
33
34 public:
35
36 Path():m_bLooped(false), curWaypoint(NULL){}
37
38 //constructor for creating a path with initial random waypoints. MinX/Y
39 //& MaxX/Y define the bounding box of the path.
40 Path(int NumWaypoints,
41      double MinX,
42      double MinY,
43      double MaxX,
44      double MaxY,
45      bool looped):m_bLooped(looped)
46 {
47     CreateRandomPath(NumWaypoints, MinX, MinY, MaxX, MaxY);
48     curWaypoint = m_WayPoints.begin();
49 }
50
51 //returns the current waypoint
52 Vector2D CurrentWaypoint()const{assert(curWaypoint != NULL); return *curWaypoint;}
53
54 //returns true if the end of the list has been reached
55 bool Finished(){return !(curWaypoint != m_WayPoints.end());}
56
57
58

```

Ouvrez le fichier **Resources Files/Script1.rc** avec clic droit > Afficher le code.



A la ligne 10, remplacez **#include "afxres.h"** par **#include "windows.h"**.

```

7 //
8 // Generated from the TE
9 //
10 #include "afxres.h"
11
12 //////////////////////////////////////////////////
13 #undef APSTUDIO_READONLY

```

```

7 //
8 // Generated from the TEXTI
9 //
10 #include "windows.h"
11
12 //////////////////////////////////////////////////
13 #undef APSTUDIO_READONLY_SY

```

Avant de générer, modifiez le nombre d'agents dans le fichier **Resources Files/params.ini** à la ligne 2 avec un nombre plus raisonnable comme 30.

Lors de la génération, vous aurez plusieurs « warning ». N'en tenez pas compte.

## 2. Conseils pour réaliser le laboratoire

- 1) Observer le constructeur de la classe **GameWorld**. Voyez comment il crée des instances de **Vehicle** et définit leur comportement à l'aide de la méthode **pVehicle->Steering()->FlockingOn()**, à la ligne 70 de **GameWorld.cpp**.
- 2) Pour les classes d'agent poursuiveur et d'agent leader, faites les hériter de **Vehicle**. Il vous suffira ensuite de correctement définir les comportements attendus à l'aide des mêmes méthodes que dans **GameWorld** mais cette fois dans le **constructeur** de vos nouvelles classes.
- 3) Pour les touches de clavier afin de contrôler le leader, référez-vous à la méthode **HandleKeyPresses ()** qui se trouve dans la classe **GameWorld** à l'intérieur du fichier **GameWorld.cpp**.

*Bonne continuation !*