INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Universidad de Oviedo

Ejemplo avanzado de C++ Trayectorias + obstáculos en 2D

Ignacio Alvarez García

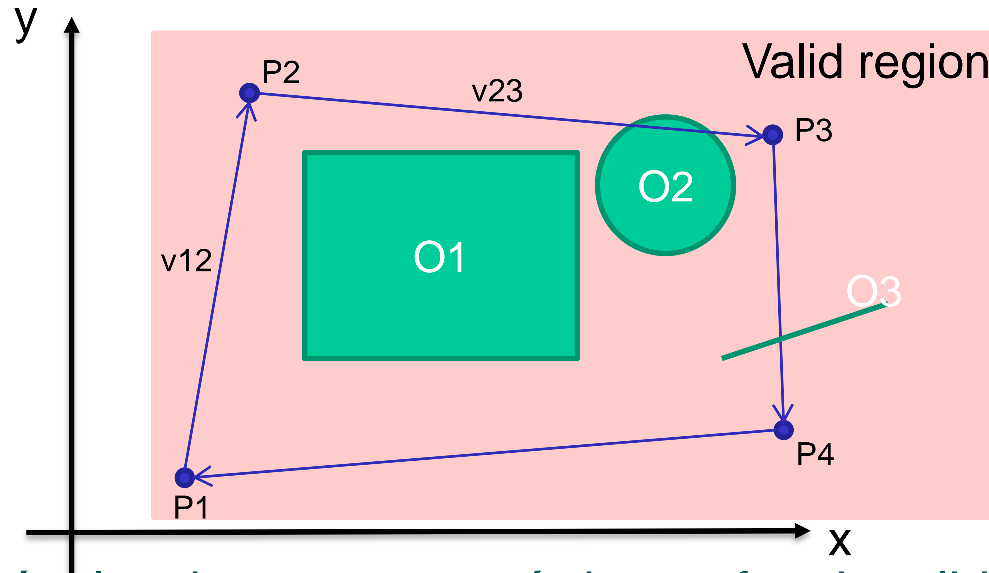
Sept - 2024

Universidad de Oviedo

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Ejemplo avanzado C++: trayectorias 2D

- Añadimos al ejemplo básico la posibilidad de que los obstáculos tengan formas diferentes:



- Los obstáculos tienen características y funcionalidades comunes:
  - Posición, tamaño, nombre
- Pero otras son específicas:
  - Forma (cada forma puede requerir variables diferentes para su definición)
- Y algunas comunes y específicas a la vez:
  - ¿Interfiere con segmento? (todas la implementan, pero de forma diferente)

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Clases base y derivadas

- Para el concepto de obstáculo:
  - Clase base MyShape con las características comunes
  - Clases derivadas MyShapeRect, MyShapeCircle, MyShapeLine con la especialización
  - Colección de obstáculos: QList<MyShape*>

Se precisarán punteros para asignación dinámica de cada objeto, porque las clases derivadas pueden tener tamaños distintos en memoria y funcionalidades virtuales sobrecargadas (se verá el detalle más adelante en el paso a paso)

Clase base

- La clase MyShape contendrá propiedades comunes:
  - Punto base (tipo MyPoint)
  - Ancho, alto (tipo float)
  - ¿Nombre? (puede heredar él de MyPoint)
  - Es válido (tipo bool)
- Y también funcionalidades comunes:
  - Constructor(es) y destructor (si es necesario)
  - Getters para las propiedades (si son privadas)
- Por ultimo, funcionalidades especializables en las clases derivadas:
  - GetBoundingBox() → retorna MyRect
  - CrossesWith(const MySegment& segment) → retorna bool

Clase base

MyShape.h

```
#ifndef MYSHAPE_H
#define MYSHAPE_H

#include <QString>
#include "MyPoint.h"
#include "MyRect.h"

class MyShape
{
private:
    MyPoint point;
    float width,height;
protected:
    bool isValid;
public:
    MyShape(const QString& i_name,float x,float y,float i_width,float i_height=-1);

    inline MyPoint GetBasePoint() const { return point; }
    inline float GetWidth() const { return width; }
    inline float GetHeight() const { return height; }
    inline bool IsValid() const { return isValid; }

    MyRect GetBoundingBox() const;
};

#endif // MYSHAPE_H
```

private: *Sólo accesibles desde funciones de MyShape*
public: *Accesible desde cualquier parte de código*
protected: *Sólo accesibles desde funciones de MyShape y funciones de clases derivadas de MyShape*

Getters en línea

MyShape.cpp

```
#include "MyShape.h"

MyShape::MyShape(const QString &i_name, float x, float y, float i_width, float i_height) :
                                                       point(i_name,x,y),
                                                       width(i_width),
                                                       height(i_height)
{
    if (height<0)
        height=width;
    isValid=(width>=0) && (height>=0);
}

MyRect MyShape::GetBoundingBox() const
{
    if (! isValid)
        return MyRect();

    MyPoint topleft("tl",point.x-width/2,point.y-height/2);
    MyPoint bottomright("br",point.x+width/2,point.y+height/2);
    MyRect r(topleft,bottomright);
    return r;
}
```

Constructor: crea valores iniciales de las propiedades (las propiedades que tienen constructor por defecto se pueden inicializar al principio o dentro de las llaves)

Suponemos que, por defecto, el punto que se pasa como referencia es el centro de la forma

Clases derivadas

MyShapeCircle.h

```
1  #ifndef MYSHAPECIRCLE_H
2  #define MYSHAPECIRCLE_H
3
4  #include "MyShape.h"
5
6  class MyShapeCircle : public MyShape
7  {
8  public:
9      MyShapeCircle(const QString& i_name,float cenx,float ceny,float i_radius);
10
11     inline float GetRadius() const { return GetWidth()*0.5; }
12 };
13
14 #endif // MYSHAPECIRCLE_H
15
```

Derivada de MyShape (con acceso público a los contenidos que MyShape haya declarado public o protected)

MyShapeCircle.cpp

```
1  #include "MyShapeCircle.h"
2
3  MyShapeCircle::MyShapeCircle(const QString &i_name, float cenx, float ceny, float i_radius)
4                                     : MyShape(i_name,cenx,ceny,i_radius*2)
5  {
6  }
7
```

La base MyShape tendrá:
width=diámetro,
height=-1 → el constructor de MyShape hará height = width

Clases derivadas

MyShapeRect.h

```
1  #ifndef MYSHAPERECT_H
2  #define MYSHAPERECT_H
3
4  #include "MyShape.h"
5
6  class MyShapeRect : public MyShape
7  {
8  public:
9      MyShapeRect(const QString& i_name,float x,float y,float i_width,float i_height);
10
11     MyRect GetBoundingBox() const;
12
13 };
14
15 #endif // MYSHAPERECT_H
16
```

Derivada de MyShape (con acceso público a los contenidos que MyShape haya declarado public o protected)

MyShapeRect.cpp

```
1  #include "MyShapeRect.h"
2
3  MyShapeRect::MyShapeRect(const QString& i_name,float x,float y,float i_width,float i_height) :
4                                              MyShape(i_name,x,y,i_width,i_height)
5  {
6
7  }
8
9  MyRect MyShapeRect::GetBoundingBox() const
10 {
11     MyPoint bottomright("br",GetBasePoint().x+GetWidth(),GetBasePoint().y+GetHeight());
12     return MyRect(GetBasePoint(),bottomright);
13 }
14
```

MyShapeRect considera el punto referencia x,y como el top-left, por lo tanto debe especializar (sobrecargar) la función GetBoundingBox() de la clase base MyShape

Programa principal

main.cpp

```
1  #include "MyShapeRect.h"
2  #include "MyShapeCircle.h"
3  #include <iostream>
4
5  int main()
6  {
7      MyShapeRect obs1("OBS1",50,100,40,20);
8      MyShapeCircle obs2("OBS2",90,60,12.5);
9
10     std::cout << "OBSTACLE 1: ";
11     obs1.GetBoundingBox().Print();
12
13     std::cout << "OBSTACLE 2: ";
14     obs2.GetBoundingBox().Print();
15
16     return 0;
17 }
```

MyShapeRect ha sobrecargado la función GetBoundingBox() de la clase base MyShape , se usará la versión sobrecargada

MyShapeCircle no ha sobrecargado la función GetBoundingBox() de la clase base MyShape , se usará la versión de la clase base

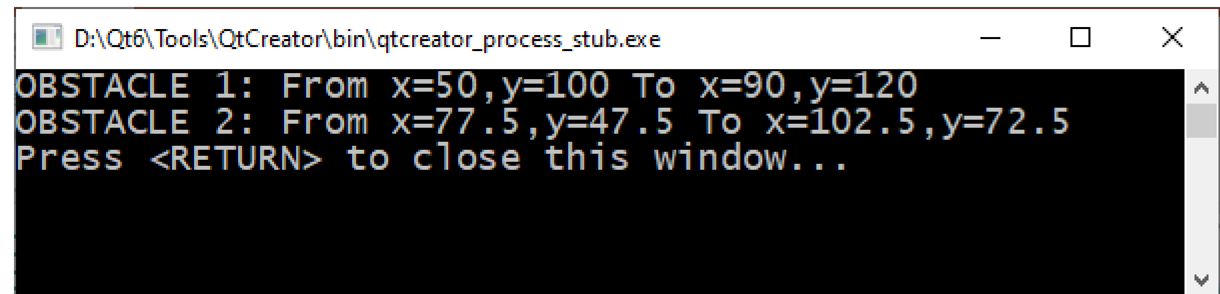
Memoria (stack)

obs1

x	50.0
y	100.0
width	40.0
height	20.0
name	"OBS1"

obs2

x	90.0
y	60.0
width	25.0
height	25.0
name	"OBS2"



Utilizar Debugger para ver contenidos de variables

Universidad de Oviedo

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Punteros a objetos

- Antes de seguir, debemos revisar el concepto puntero y añadir nuevas características de C++ para punteros a objeto:
  - Una variable tipo puntero contiene la dirección de otra variable
  - Una variable tipo puntero se declara con:

    tipoVbleApuntada* nombreVblePuntero;
  - Si el puntero es a un objeto tipo clase, se accede a sus contenidos con el operador **->** (flecha) en lugar de **.** (punto)
  - Los punteros se pueden asignar con &vble (dirección de):

main.cpp

```
17    MyShapeRect* ptObs1;
18    ptObs1= &obs1;
19
20    std::cout << "OBSTACLE 1 FROM POINTER: ";
21    ptObs1->GetBoundingBox().Print();
```

D:\Qt6\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
OBSTACLE 1: From x=50,y=100 To x=90,y=120
OBSTACLE 2: From x=77.5,y=47.5 To x=102.5,y=72.5
OBSTACLE 1 FROM POINTER: From x=50,y=100 To x=90,y=120
Press <RETURN> to close this window...
```

Utilizar Debugger para ver contenidos de variables

Punteros a objetos

- Si se usa un puntero con tipo ClaseBase* :
  - Se puede apuntar a cualquier objeto de clases derivadas, pero …
  - … solamente se tiene acceso a los elementos de la clase base

main.cpp

```
11      std::cout << "OBSTACLE 1: ";
12      obs1.GetBoundingBox().Print();
13
14      std::cout << "OBSTACLE 2: ";
15      obs2.GetBoundingBox().Print();
16
17      MyShape* ptObs;
18      ptObs= &obs1;
19      std::cout << "OBSTACLE 1 FROM POINTER: ";
20      ptObs->GetBoundingBox().Print();
21
22      ptObs= &obs2;
23      std::cout << "OBSTACLE 2 FROM POINTER: ";
24      ptObs->GetBoundingBox().Print();
```

Seleccionar D:\Qt6\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
OBSTACLE 1: From x=50,y=100 To x=90,y=120
OBSTACLE 2: From x=77.5,y=47.5 To x=102.5,y=72.5
OBSTACLE 1 FROM POINTER: From x=30,y=90 To x=70,y=110
OBSTACLE 2 FROM POINTER: From x=77.5,y=47.5 To x=102.5,y=72.5
Press <RETURN> to close this window...
```

*Resultado **diferente** para OBSTACLE 1:*

- obs1 *es de tipo* MyShapeRect → obs1.GetBoundingRectangle() *usa* MyShapeRect::GetBoundingRectangle(), *que hemos sobrecargado para calcular correctamente en el caso de rectángulo.*
- ptObs *es de tipo* MyShape* → *aunque apunta a* obs1, *que es de tipo* MyShapeRect, *sólo puede acceder a los contenidos de* MyShape, *por lo que* ptObs->GetBoundingRectangle() *usa* MyShape::GetBoundingRectangle(), *que asume que el punto referencia es el centro*

Universidad de Oviedo

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Punteros a objetos

- Solución al problema anterior: funciones virtuales
  - Una función virtual se resuelve en tiempo de ejecución (no de compilación), teniendo en cuenta el tipo real del objeto apuntado en lugar del tipo del puntero
  - Una fn virtual se debe declarar como tal en la clase base (solamente en el .h) y en las derivadas que la quieran sobrecargar (solamente en el .h)

MyShape.h

```
 8  class MyShape
 9  {
10  private:

22
23      virtual MyRect GetBoundingBox() const;
24  };
```

MyShapeRect.h

```
 6  class MyShapeRect : public MyShape
 7  {
 8  public:
 9      MyShapeRect(const QString& i_name,float x,float y,float i_width,float i_height);
10
11      virtual MyRect GetBoundingBox() const override;
12
13  };
```

D:\Qt6\Tools\QtCreator\bin\qtcreator_process_stub.exe

```
OBSTACLE 1: From x=50,y=100 To x=90,y=120
OBSTACLE 2: From x=77.5,y=47.5 To x=102.5,y=72.5
OBSTACLE 1 FROM POINTER: From x=50,y=100 To x=90,y=120
OBSTACLE 2 FROM POINTER: From x=77.5,y=47.5 To x=102.5,y=72.5
Press <RETURN> to close this window...
```

*Resultado **igual** para OBSTACLE 1:*

- obs1 *es de tipo* MyShapeRect → obs1.GetBoundingRectangle() *usa* MyShapeRect::GetBoundingRectangle()*, que hemos sobrecargado para calcular correctamente en el caso de rectángulo.*
- ptObs *es de tipo* MyShape* → *como apunta a* obs1*, que es de tipo* MyShapeRect*, y la función a llamar se ha declarado virtual, detecta (en tiempo de ejecución) que la llamada* ptObs->GetBoundingRectangle() debe *usar* MyShapeRect::GetBoundingRectangle()

Universidad de Oviedo

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Creación y eliminación dinámica

- Todas las variables declaradas en una función (incluidos sus argumentos y el valor retornado) tienen una vida temporal:
  - Se crean automáticamente cuando se invoca la función como copias de los argumentos
  - Se eliminan automáticamente cuando se retorna de la función
- Con los operadores new y delete se pueden conseguir variables con tiempo de vida no ligado a la función en que se usan:
  - new MyClass(argumentos del constructor) → crea espacio en memoria (heap) para un nuevo objeto de tipo MyClass, llama al constructor según los argumentos, y devuelve un puntero al nuevo objeto creado
  - El objeto permanenece en memoria hasta que se llame a delete, y es accessible a través del puntero devuelto
  - delete puntero → llama al destructor de MyClass para el objeto creado, y a continuación elimina de la memoria (heap) el objeto. El puntero ya no debe usarse
  - Las clases con funciones virtuales (base y derivadas necesitan tener declarado destructor también virtual, aunque esté vacío)

Creación y eliminación dinámica

- Ejemplo : creación y eliminación dinámica de objetos

main.cpp

```
 5  int main()
 6  {
 7      MyShape *ptObs1,*ptObs2;
 8      ptObs1 = new MyShapeRect("OBS1",50,100,40,20);
 9      std::cout << "OBSTACLE 1 FROM POINTER: ";
10      ptObs1->GetBoundingBox().Print();
11
12      ptObs2 = new MyShapeCircle("OBS2",90,60,12.5);
13      std::cout << "OBSTACLE 2 FROM POINTER: ";
14      ptObs2->GetBoundingBox().Print();
15
16      delete ptObs1;   ptObs1=nullptr;
17      delete ptObs2;   ptObs2=nullptr;
18
```

MyShape.h

```
 8  class MyShape
 9  {
10  private:

17      virtual ~MyShape() { };
```

MyShapeRect.h

```
 6  class MyShapeRect : public MyShape
 7  {
 8  public:
 9      MyShapeRect(const QString& i_name,float x,float y,float i_widt
10      virtual ~MyShapeRect() { }
```

MyShapeCircle.h

```
 6  class MyShapeCircle : public MyShape
 7  {
 8  public:
 9      MyShapeCircle(const QString& i_name,float cenx,float ceny,float
10      virtual ~MyShapeCircle() { }
```

Universidad de Oviedo

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Lista de objetos de tipos variados

- Lista de objetos que comparten clase base:
  - Variable: QList<ClaseBase*> myList; → colección de punteros a la clase base
  - Para añadir un nuevo objeto de una clase derivada:

    myList.append(new ClaseDerivada(*args del contructor*)); → añade al final
    ó
    myList.prepend(new ClaseDerivada(*args del contructor*)); → inserta al inicio
    ó
    myList.insert(i,new ClaseDerivada(*args del contructor*)); → inserta en la posición i
  - Acceso a los elementos:

```
for (int i=0 ; i<myList.size() ; i++) {
        result=myList[i]->función_de_clase_base(argumentos);
}
```

  - Eliminar elemento de la posición i :

    delete myList[i]; → primero hay que eliminar el objeto creado dinámicamente
    myList.remove(i); → ahora ya se puede quitar el puntero de la lista
  - Eliminar todos los elementos:

```
for (int i=0 ; i<myList.size() ; i++) {
    delete myList[i]; → primero hay que eliminar cada objeto creado dinámicamente
}
myList.clear(); → ahora ya se puede vaciar la lista de punteros
```

Universidad de Oviedo

INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

Lista de objetos de tipos variados

- Sugerencia:
  - Si el constructor de todas las clases (base y derivadas) toma un único argumento tipo QString que contenga los parámetros en un formato determinado (XML, jSon), se simplifica mucho el código.
  - Esto sólo se debe hacer con funciones (como el constructor) que se invocan ocasionalmente: procesar el texto consume tiempo.
  - Además, se puede realizar una nueva función static CreateElementFromXml() que revise el texto, compruebe el tipo del nuevo elemento, y añada el elemento del tipo adecuado.

MyShape.h

```cpp
class MyShape
{
...
public:
    MyShape(const QString& init_xml);
    static MyShape* CreateElementFromXml(const QString& init_xml);
```

main.cpp

```cpp
int main()
{
    QList<MyShape*> listObs;
    bool readingObs=true;

    while (readingObs) {
        QString xmlInfo=GetObstacleInfoFromXXX();
        if (xmlInfo.isEmpty()) {
            readingObs=false;
        }
        else {
            MyShape* shapeRead=MyShape::CreateElementFromXml(xmlInfo);
            if (shapeRead!=nullptr)
                listObs.append(shapeRead);
        }
    }
```

MyShapeRect.h

```cpp
public:
    MyShapeRect(const QString& init_xml);
```

MyShapeCircle.h

```cpp
public:
    MyShapeCircle(const QString& init_xml);
```

Universidad de Oviedo

Lista de objetos de tipos variados

- Funcionalidades para XML:

init_xml (texto obtenido de archivo, comm, gui, …)

```
<Shape>
    <name>Obstacle 1</name>
    <type>Rect</type>
    <topleft>
        <x>25.2</x>
        <y>12</y>
    </topleft>
    <bottomright>
        <x>45.2</x>
        <y>17</y>
    </bottomright>
</Shape>

<Shape>
    <name>Obstacle 2</name>
    <type>Circle</type>
    <center>
        <x>44.5</x>
        <y>18.9</y>
    </center>
    <radius>12.5</radius>
</Shape>
```

MyShape.cpp

```
5  QString XmlGetContents(const QString& all_xml,const QString& item_to_find)
6  {
7      QString contents;
8      int startItem=all_xml.indexOf("<"+item_to_find+">");
9      if (startItem>=0) {
10         startItem += 1+item_to_find.size()+1;
11         int endItem=all_xml.indexOf("</"+item_to_find+">",startItem);
12         if (endItem>=0) {
13             contents=all_xml.mid(startItem,endItem-startItem);
14         }
15     }
16
17     return contents;
18 }
19
20 MyShape *MyShape::CreateElementFromXml(const QString &init_xml)
21 {
22     QString contents=XmlGetContents(init_xml,"Shape");
23     QString type=XmlGetContents(contents,"type");
24     if (type=="Rect") {
25         return new MyShapeRect(contents);
26     }
27     if (type=="Circle") {
28         return new MyShapeCircle(contents);
29     }
30
31     return nullptr;
32 }
```

Ejemplo avanzado C++: clases

- Ejercicios propuestos

6) Añadir función base y derivadas de tipo virtual para todas las clases de MyShape:

```
bool IntersectsWith(const MySegment& segment);
```

7) Añadir al programa ya realizado una lista de obstáculos variados que se obtienen de un archivo XML, y el chequeo de la realizabilidad de la trayectoria (no choca con ningún obstáculo)