Complément mémoire :

```
int a = 5;
int* pA = &a;
Float* pB = (Float*) &A;
```

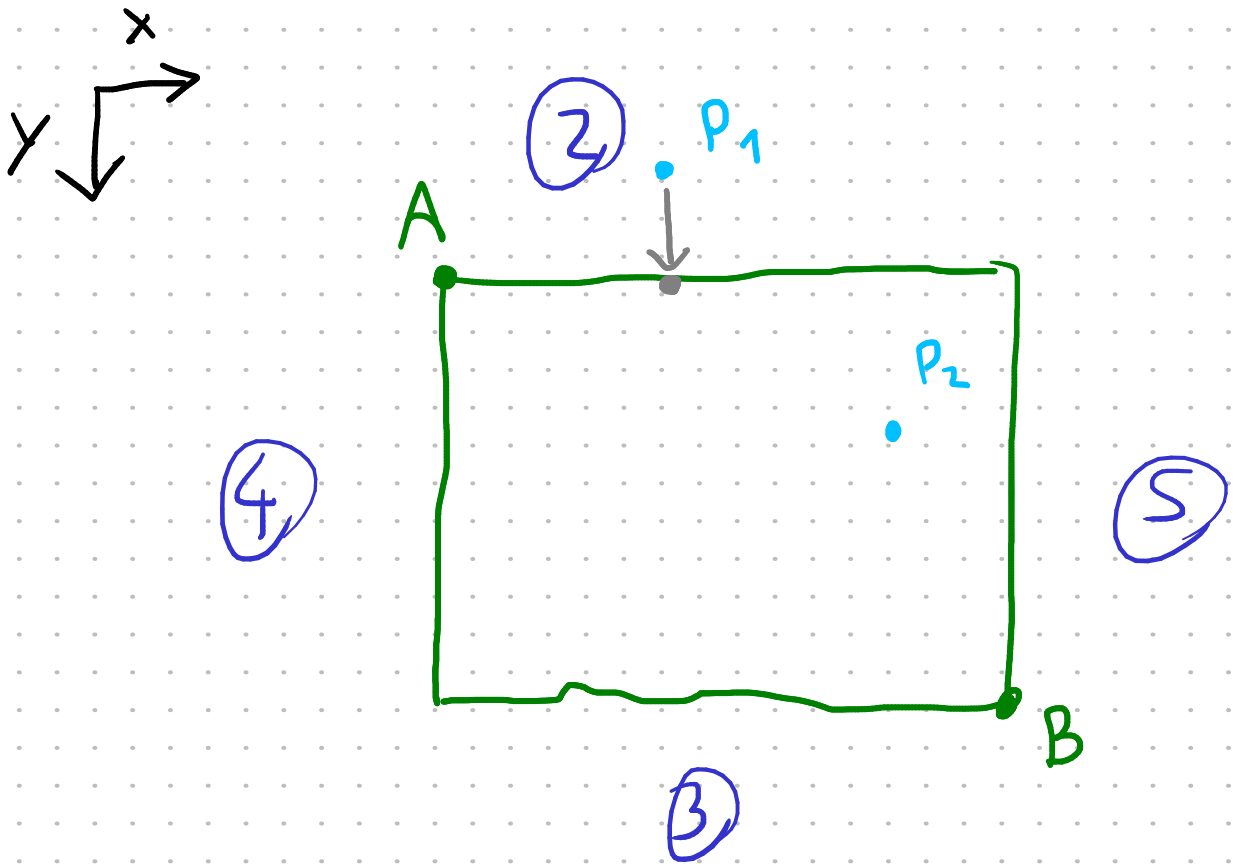
réinterprétation de pointeur
→ implique de la vigilance
car le compilateur ne pourra pas
détecter les erreurs.

`void*` → Pointeur sans type associé

```
int a = 5;
void* pA = &a;
int* pB = pA;
int* pB = (int*) pA;
```

Réinterprétation implicite
Erreur : si le pointeur est sans Type,
le compilateur ne peut pas vérifier.
OK : Réinterprétation explicite

TD 7



(2) $P_1.y < A.y$

$P_1.y = A.y$

(3) $P_1.y > B.y$

$P_1.y = B.y$

(4) $P_1.x < A.x$

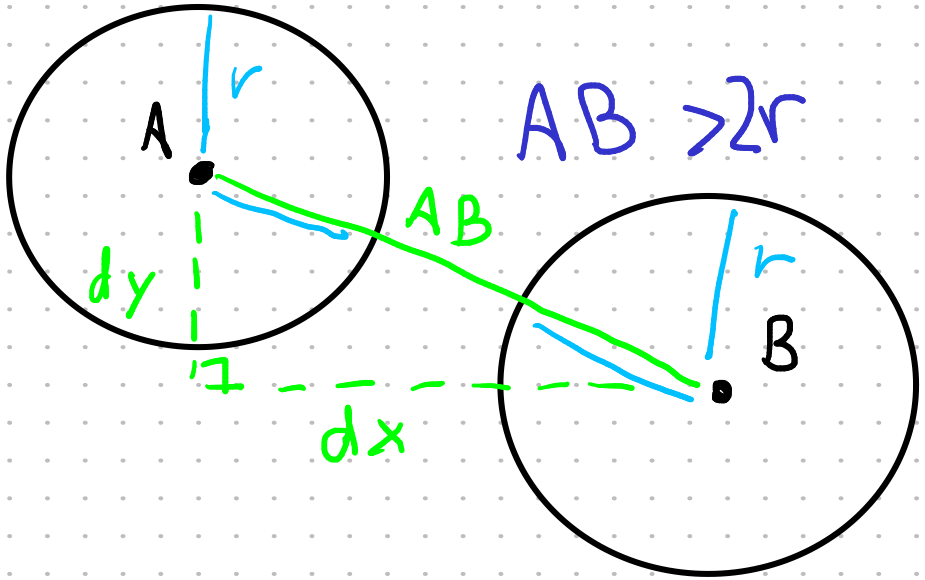
$P1.x = A.x$

(5) $P_1.x > B.x$

$P1.x = B.x$

Collisions

Dans le projet, $r = 8px$



$AB > 2r$

$$AB = \sqrt{dx^2 + dy^2}$$

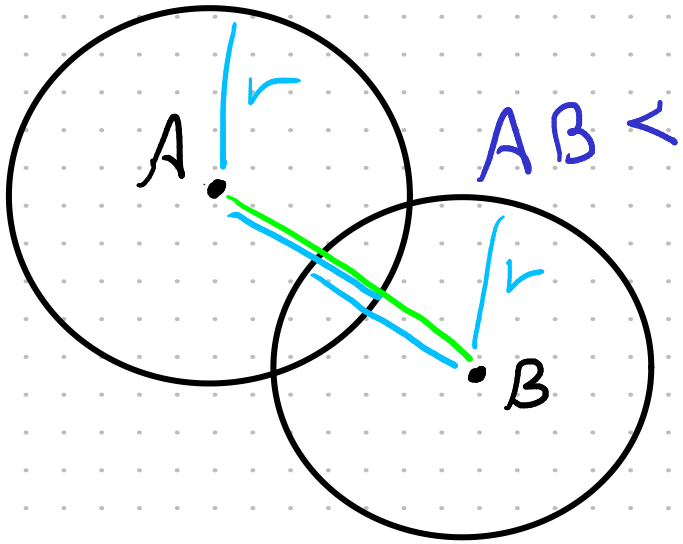
$$AB = \sqrt{(A.x - B.x)^2 + (A.y - B.y)^2}$$

$$\sqrt{(A.x - B.x)^2 + (A.y - B.y)^2} < 2r \ ?$$

$\square^2 \downarrow \square^2$

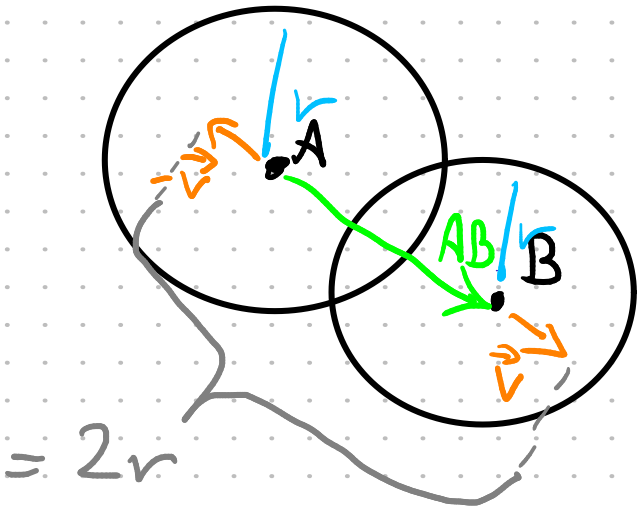
$$(A.x - B.x) \times (A.x - B.x) + (A.y - B.y) \times (A.y - B.y) < 4 \times \frac{r^2}{8}$$

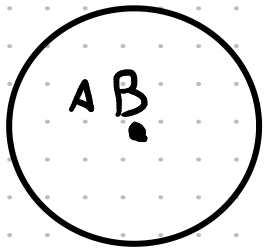
$$< 256$$



$AB < 2r$

Résolution de collisions





$$\vec{v} = \alpha \vec{AB}$$

$$(\vec{A} - \vec{v}) \quad (\vec{B} + \vec{v}) = 2r$$

$$2v + AB = 2r$$

$$2\alpha AB + AB = 2r$$

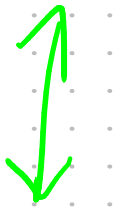
$$2\alpha AB = 2r - AB$$

$$\alpha = \frac{2r - AB}{2AB}$$

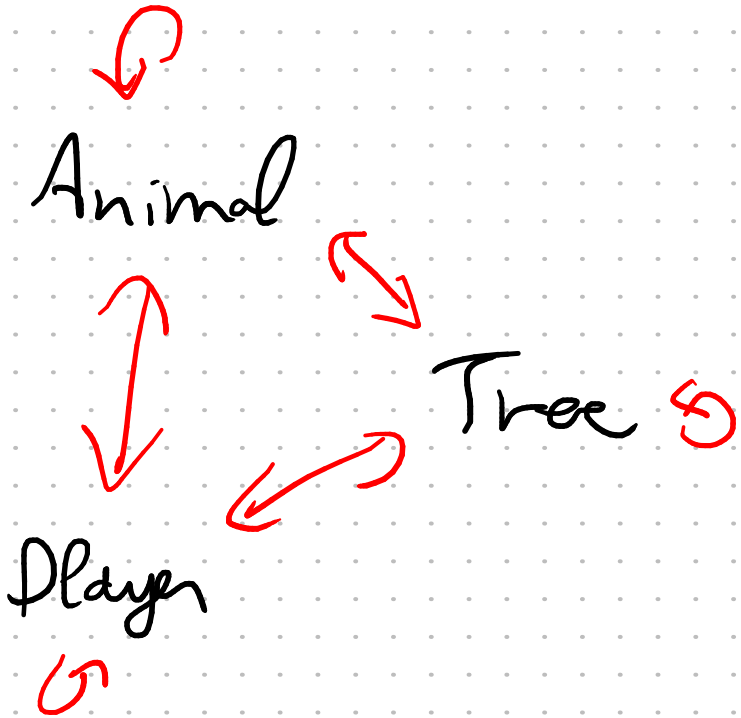
$$\vec{v} = \alpha \vec{AB} = \frac{2r - AB}{2AB} \vec{AB}$$

```
ABsq = (A.x-B.x)*(A.x-B.x)
      +(A.y-B.y)*(A.y-B.y
rsq = 8*8
if (ABsq > 0   &&
    ABsq < 4*rsq){
// Collision
// #include <cmath>
AB = sqrt(ABsq)
alpha = (2*8 - AB)
        / 2*AB
v.x = alpha
      * (B.x-A.x)
v.y = alpha
      * (B.y-A.y)
A.x -= v.x   A.y -= v.y
B.x += v.x   B.y += v.y
}
```

3 boucles pour gérer les collisions



6 boucles pour gérer les collisions



→ Problème de mise à l'échelle :
À chaque fois que je rajoute un nouveau type d'élément,
Je double la taille du code (copier/coller de boucles)

```
class DinoEntity
```
← Classe de base / basique

m_pos
SetPos() GetPos()
// gestions des collisions ?

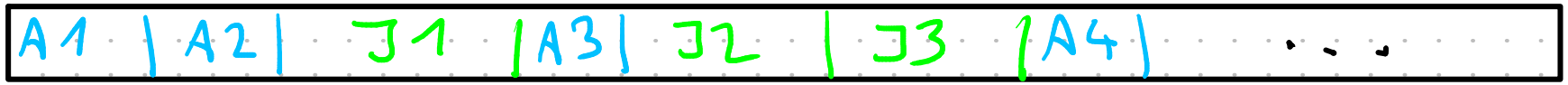
Héritage

```
class DinoAnimal
    : public DinoEntity
{
    ...
};
```

```
class DinoPlayer
    : public DinoEntity
{
    ...
};
```
← Classe dérivée

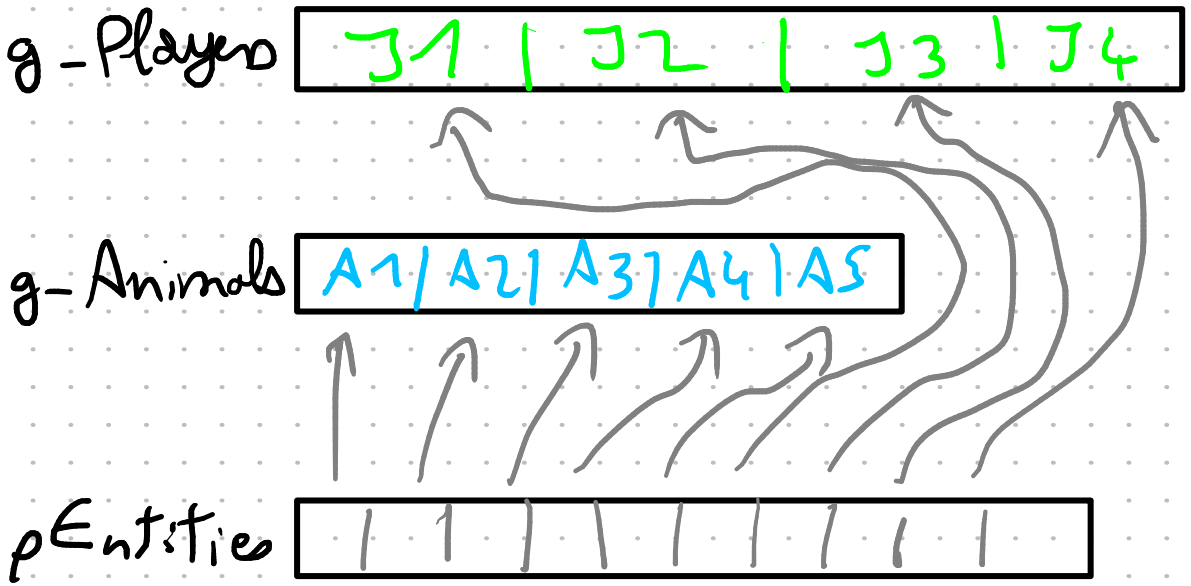
⟹ Gères tous les cas de collisions en un seul endroit, dans une même liste

Imaginons



std::vector ne pourrait pas trouver l'emplacement du n^ième élément
(pas de multiplication par sizeof(T) possible)

Solution



Mise en commun → Réduire la taille de bug

On a parfois besoin de point "customizable"

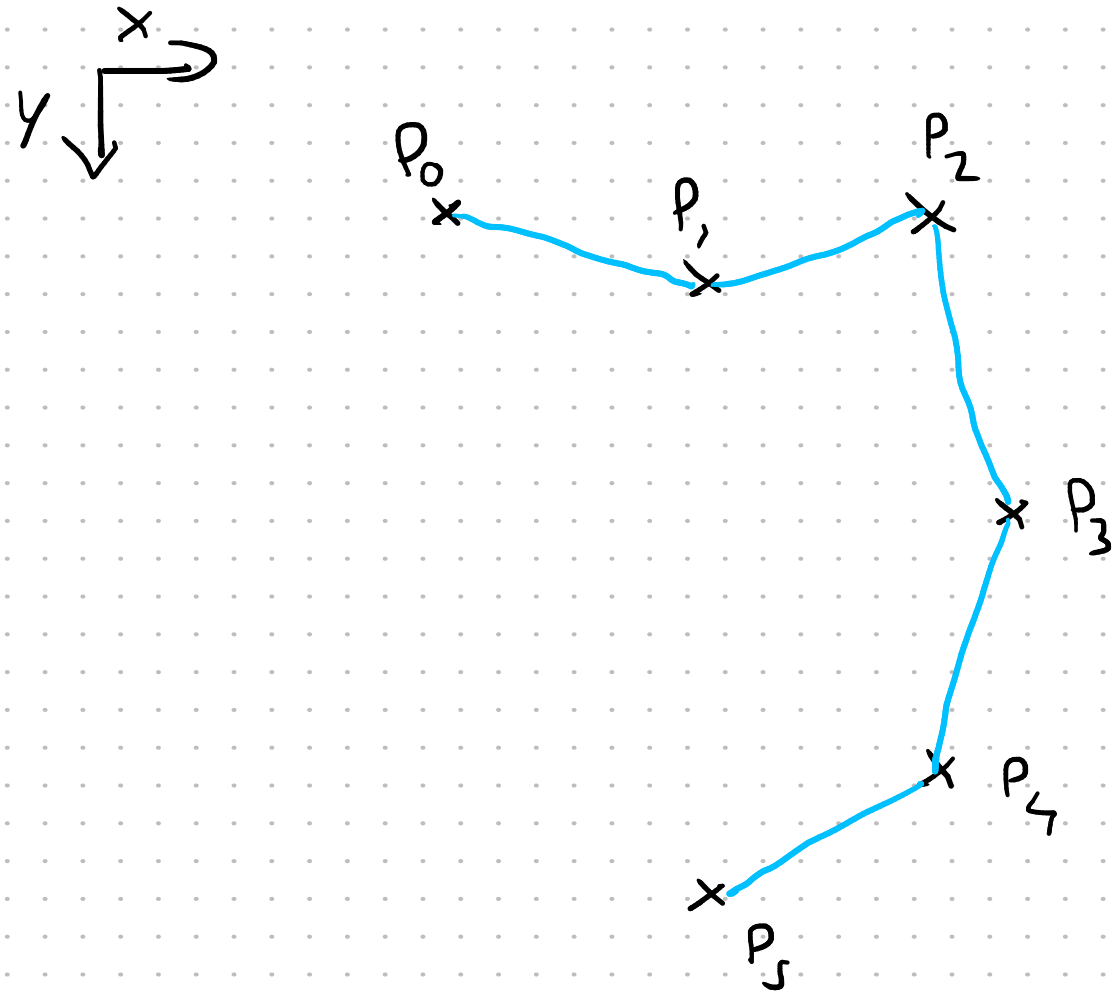
Comportement à peu près identique dans les grandes lignes, mais quelques détails différents.

⇒ Méthodes virtuelles / overridables

CLASSE DE BASE : `virtual void OnTerrainBorder();`

CLASSE DÉRIVÉE : `void OnTerrainBorder() override;`

<ASSO>



Implémentation :

std::vector<DinoVec2>

Poly ligne → Par joueur

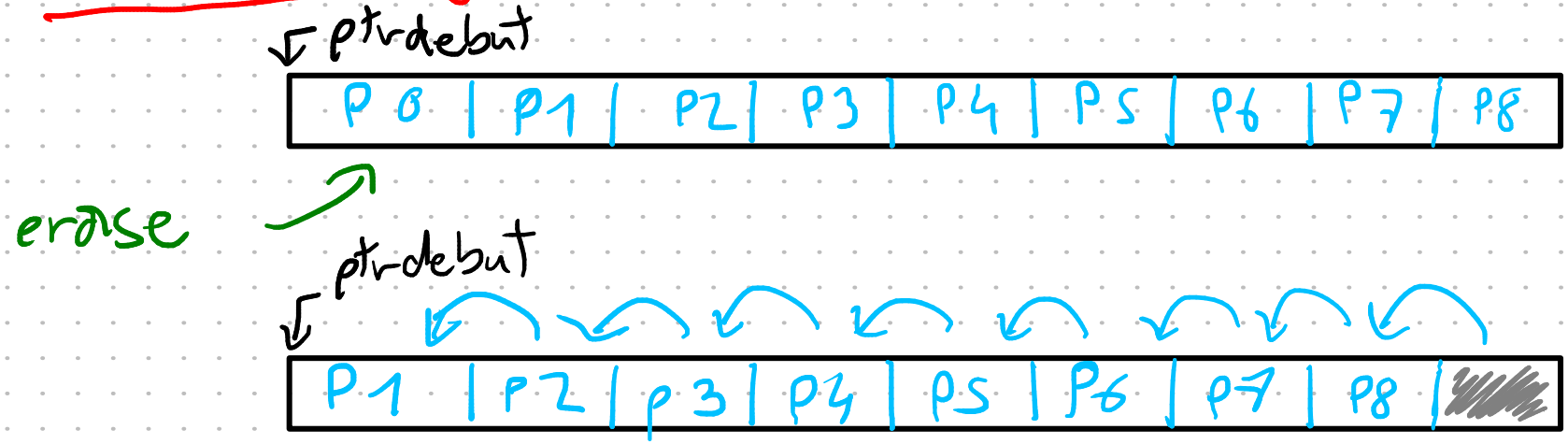
ligne cassée

1 point par Frame

60 Fps → 60 points / seconde

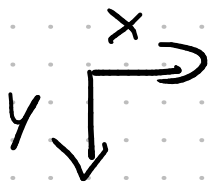
2 secondes → 120 points

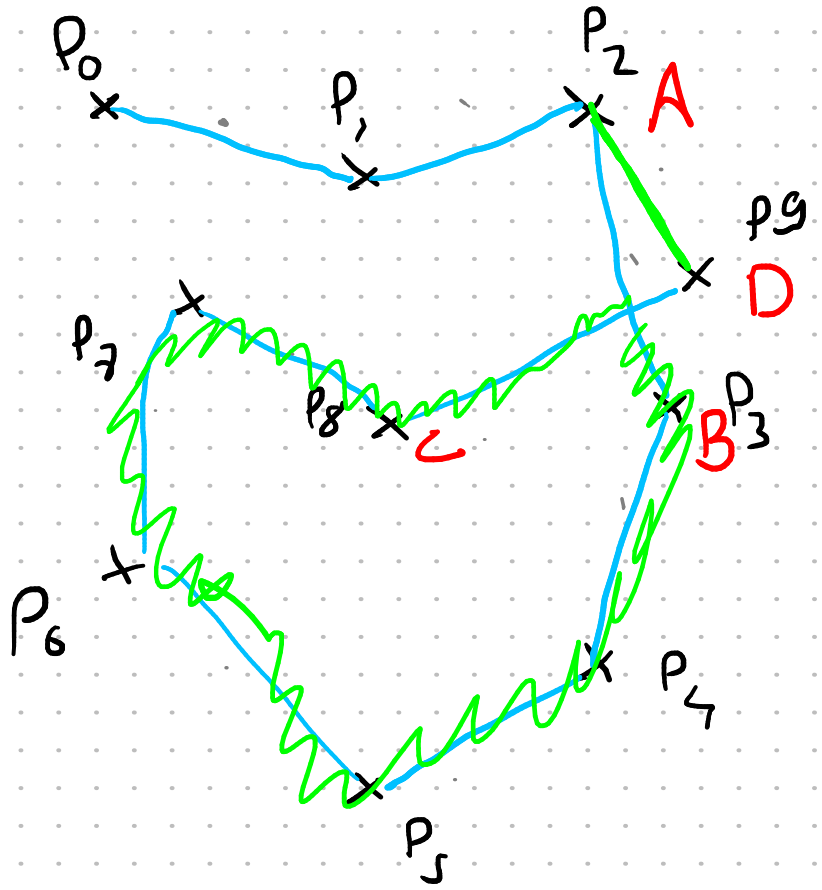
.erase(.begin())



emplace_back()

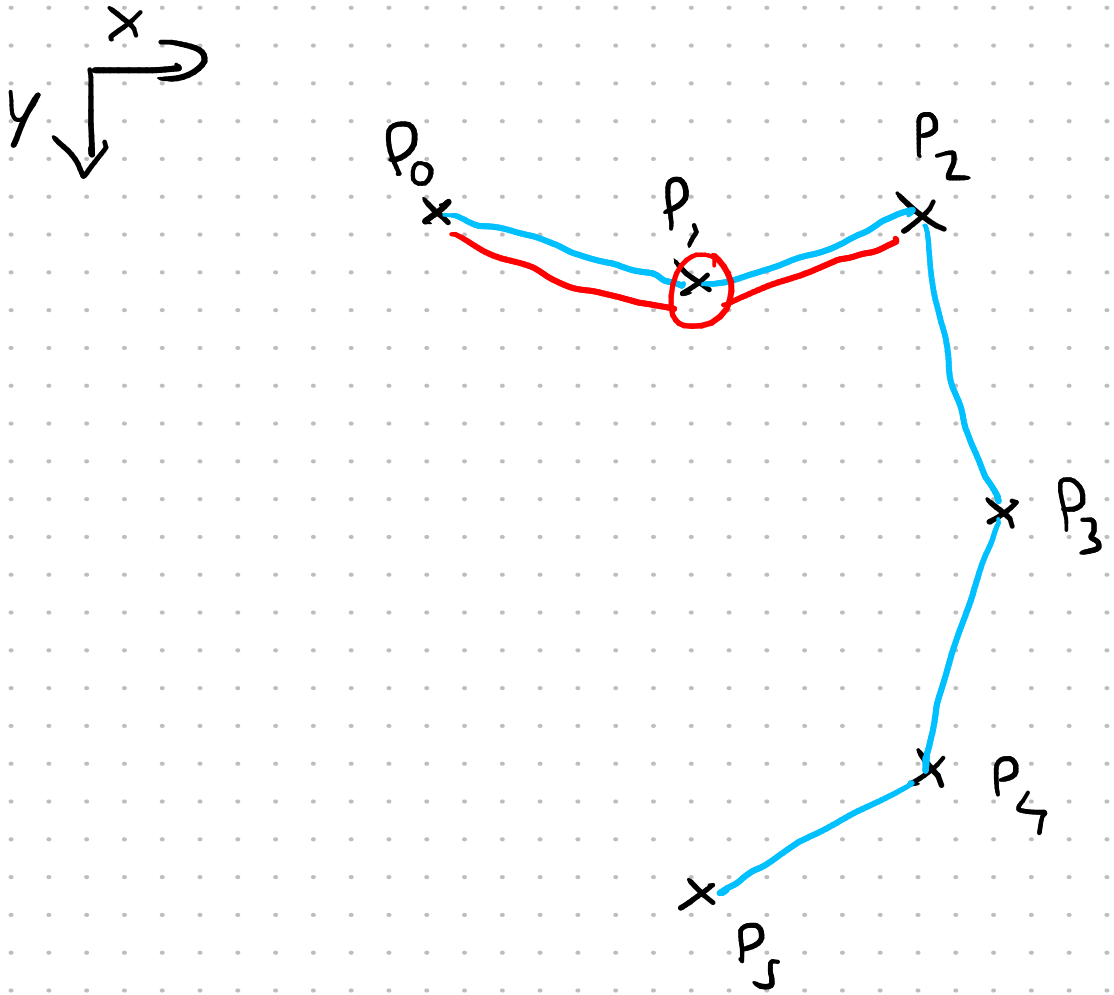
std::vector va garder cet emplacement en interne





[P2; P3] et [P8; P9] s'intersectent

→ P0 P1 P2 P9



✗ $[P_0P_1]$ et $[P_1P_2]$

P_0P_1 et P_2P_3

P_0P_1 et P_3P_4

P_0P_1 et P_4P_5

✗ P_1P_2 et P_2P_3

P_1P_2 et P_3P_4

P_1P_2 et P_4P_5

✗ P_2P_3 et P_3P_4

P_2P_3 et P_4P_5

✗ P_3P_4 et P_4P_5

2 secondes d'historique

→ 120 points

→ 119 segments

119 Fois

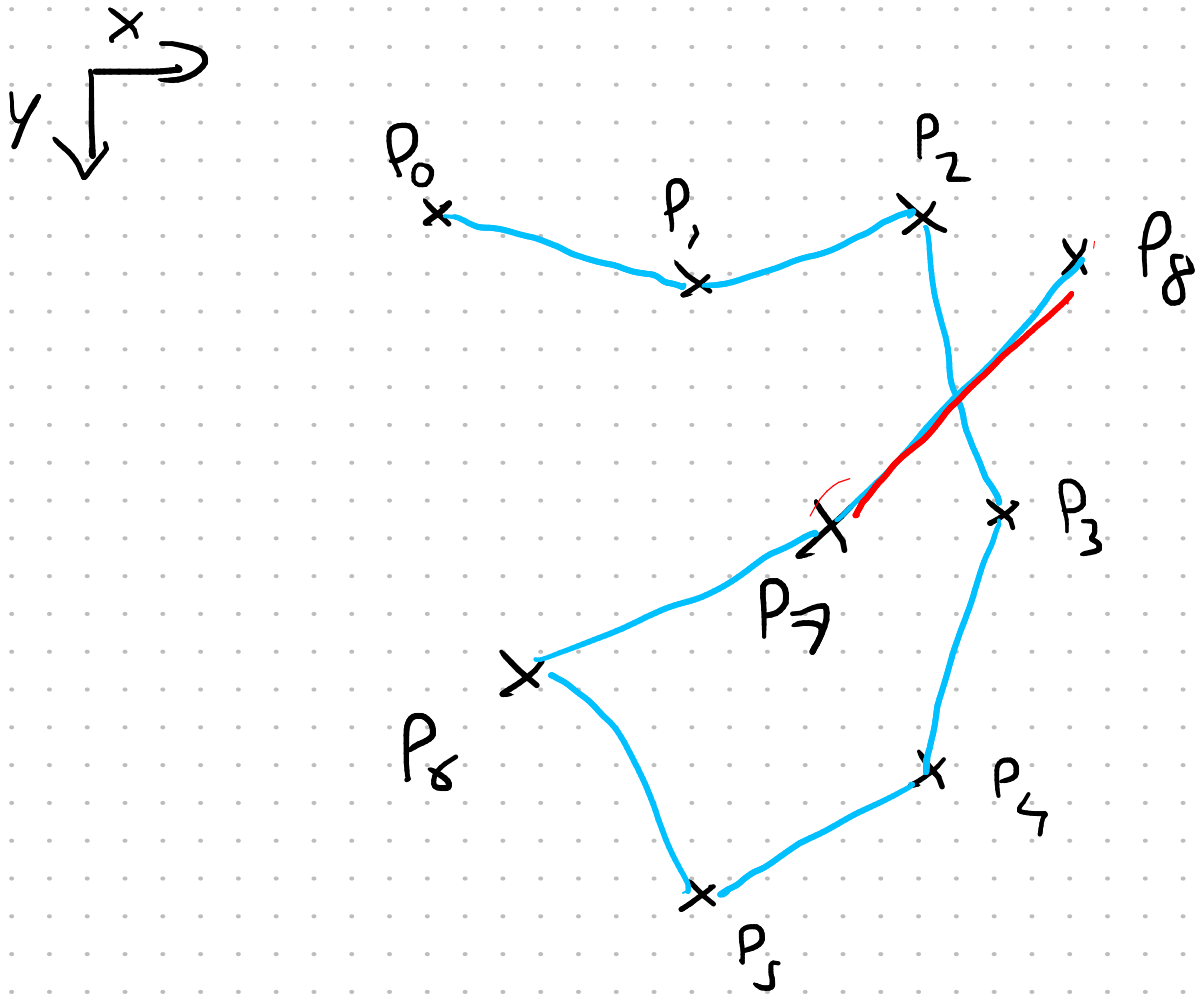
```
for (int idxSegment1 = 0; idxSegment1 < m_lasso.size() - 1; ++idxSegment1) {
    DinoVec2 A = m_lasso[idxSegment1];
    DinoVec2 B = m_lasso[idxSegment1 + 1];

    for (int idxSegment2 = idxSegment1 + 2; idxSegment2 < m_lasso.size() - 1; ++idxSegment2)
        DinoVec2 C = m_lasso[idxSegment2];
        DinoVec2 D = m_lasso[idxSegment2 + 1];

        if (Dino_IntersectSegment(A, B, C, D)) {
```

idxSegment1	idxSegment2	
0	→ [2; 118]	→ 117 appels
1	→ [3; 118]	→ 116
2	→ [4; 118]	→ 115
3	→ [5; 118]	→ 114
⋮		⋮
116	→ [118; 118]	→ 1
117	→ ∅	
118	→ ∅	

= 117 + 116 + 115 + 114 ... + 1

= 6786 appels par joueur

= 27 144 appels par frame



Juste le dernier segment peut créer une intersection !