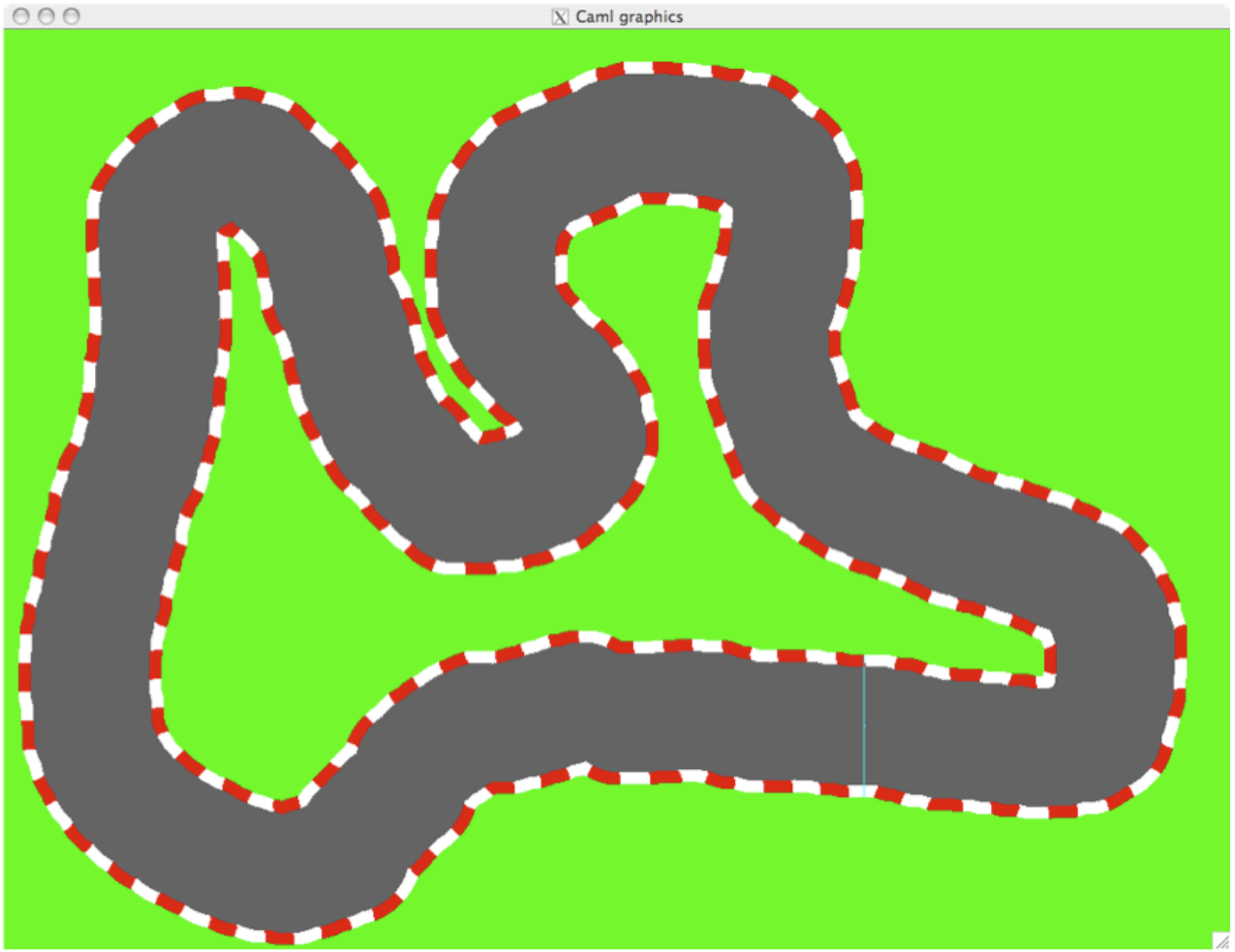
Cours 1 : Informations générales, fichiers

Vincent Guigue
UPMC - LIP6

Contexte

Quelques images pour comprendre le contexte Une compétition algorithmique (ICFP 2003)



Objectifs/ingrédients de ce projet

- Gérer de nouveaux outils JAVA (interfaces, fichiers, fenêtres...)
- Utiliser efficacement une IDE (Eclipse)
- Apprendre à mieux programmer (introduction aux design pattern)
- Coder (!)
- Gérer l'interface entre mathématiques et informatique
- Algorithmique
  - Géométrie dans l'espace
  - Algorithmes de plus court chemin
  - Algorithmes génétiques

Organisation partie 1

1. Fichiers, énumération, introduction à l'UML
2. Organisation du code & outils : eclipse, package, interface
3. Géométrie
   - Point, Vecteur
   - Modèle physique, inertie, déplacement, franchissement ?
4. Algorithmique & architecture
   - 1er algorithme : radar + système expert Architecture de la stratégie Système de décision

Fonctionnement de l'UE

- 1h45 TD/Cours le mercredi
- 3h30 TME le vendredi
- Possibilité de tutorat pour les personnes qui se sentent perdues à partir de la semaine 4/5
- Contacts par mail `vincent.guigue@lip6.fr`

Evaluation :

- CC : 70%
  - Partiel (0.25), Rapport et performance de votre code (0.6), participation (0.15)
- Exam : 30%
  - Soutenance orale et modification de code individuel

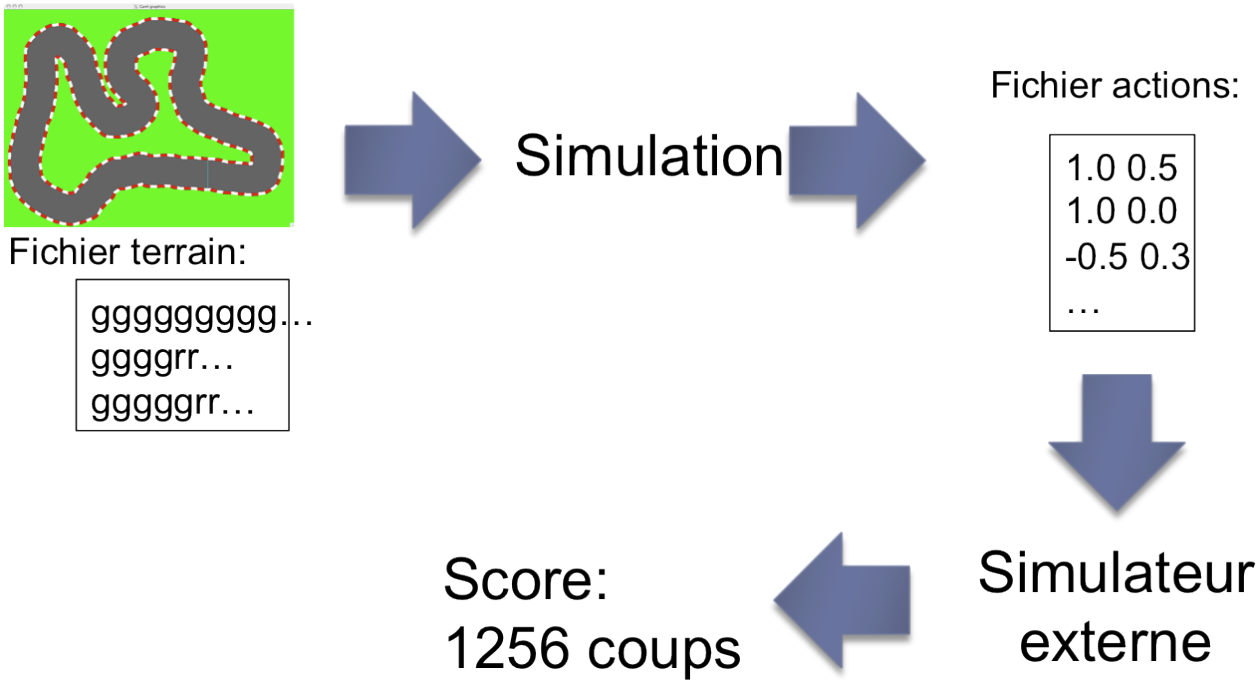
Quelques règles utiles

- Attributs TOUJOURS private (+ accesseurs éventuels)
- Code d'une classe < 2 pages en général, exceptionnellement un peu plus
- Penser à créer des sous-classe
  - Factory
  - Tools
  - ...
- Regarder la documentation quand on a un doute : `http://java.sun.com/javase/6/docs/api/`
- Utiliser les interfaces avant de coder les objets complexes (cf Cours 2)
- Utiliser les packages pour structurer le projet (cf Cours 2)

A QUOI ÇA SERT ?

Dialogue entre les composants / architecture robuste

- Lire le fichier de terrain
- Ecrire un fichier action
- Pour le simulateur externe : lire le fichier action



Dans un contexte plus général :

- Fichier = système de communication/spécification pour les projets collaboratifs
- Simplification de l'architecture des IHM multi-composants, réduction des dépendances

3 manières de gérer les fichiers. En JAVA, une logique de flux

- Approche générale, *à l'ancienne*
  - (1) Lecture/écriture ASCII
  - (2) Lecture/écriture binaire
- (3) Approche Objet
  - Serialization
  - Externalization

Plan général

1. **Fichiers** : lire les noms, vérifier l'existence, vérifier la possibilité d'écriture...
   - Equivalent des fonctions `dir, cd, ...` Mais à l'intérieur de JAVA
2. Une fois le fichier ciblé, **l'ouvrir** et **lire** ce qu'il y a dedans
3. **Créer** un fichier et/ou **écrire** dedans
4. ... D'autres choses se gèrent comme les fichiers
   - Clavier, Réseau...

Gérer les fichiers sur le disque dur

Classe File

Cette classe permet de gérer les fichiers :

- test d'existance
- distinction fichier/répertoire
- copie/effacement
- ...

- boolean canExecute()
- boolean canRead()
- boolean canWrite()
- boolean delete()
- boolean isDirectory()
- boolean isFile()
- File[] listFiles()
- boolean mkdir()

Nombreuses opérations très intéressantes concernant la manipulation des fichiers

Lecture de fichiers

1. `File` : désigner un fichier
2. `FileInputStream` : création de cet objet = ouverture en lecture du fichier
   - Des **exceptions** à gérer
   - Penser à **fermer** les fichiers ouverts

```
1  FileInputStream in = null;
2  File f = new File("xanadu.txt");
3   try {
4       in = new FileInputStream(f); // ouverture du fichier
5       // Throws: FileNotFoundException : => try/catch
6
7       // OPERATIONS DE LECTURE
8
9       }
10  } finally {
11      if (in != null) {
12          in.close();
13      }
14  }
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1 try { FileInputStream in = new FileInputStream(
2                     new File(filename));
3           ... // LECTURE
4           in.close();
5 catch(...){...}
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1 try { FileInputStream in = new FileInputStream(
2                     new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){...}
```

2. Même s'il y a des erreurs pendant la lecture !

```
1 try { FileInputStream in = new FileInputStream(
2                                 new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){ in.close(); }
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1 try { FileInputStream in = new FileInputStream(
2                     new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){...}
```

2. Même s'il y a des erreurs pendant la lecture !

```
1 try { FileInputStream in = new FileInputStream(
2                                 new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){ in.close(); }
```

3. Mais ça ne compile pas !

```
1 FileInputStream in = null;
2 try { in = new FileInputStream(new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){  in.close(); }
```

Les petits pièges... La fermeture des fichiers

1. Toujours fermer un fichier ouvert...

```
1 try { FileInputStream in = new FileInputStream(
2                     new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){...}
```

2. Même s'il y a des erreurs pendant la lecture !

```
1 try { FileInputStream in = new FileInputStream(
2                                 new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){ in.close(); }
```

3. Mais ça ne compile pas !

```
1 FileInputStream in = null;
2 try { in = new FileInputStream(new File(filename));
3         ... // LECTURE
4         in.close();
5 catch(...){   in.close(); }
```

4. Plus élégant : lignes 4-5 $\Rightarrow$ `finally{in.close()}`

Les petits pièges... La fermeture des fichiers

5. La solution précédente ne marche pas encore !

```
1 FileInputStream in = null;
2 try { in = new FileInputStream(new File(filename));
3         ... // LECTURE
4 finally{  in.close(); }
```

Les petits pièges... La fermeture des fichiers

5. La solution précédente ne marche pas encore !

```
1 FileInputStream in = null;
2 try { in = new FileInputStream(new File(filename));
3         ... // LECTURE
4 finally{  in.close(); }
```

6. ... le `close` est susceptible de lever une exception si le fichier n'est pas ouvert (NullPointerException) !

```
1  FileInputStream in = null;
2  File f = new File("xanadu.txt");
3   try {
4       in = new FileInputStream(f); // ouverture du fichier
5       // Throws: FileNotFoundException : => try/catch
6
7       // OPERATIONS DE LECTURE
8
9       }
10  } finally {              // On est sûr de passer par là
11      if (in != null) { // vérifier que le fichier est ouvert
12            in.close();
13      }
```

Lecture de fichiers (byte stream)

public int read() throws IOException

Reads a byte of data from this input stream. This method blocks if no input is yet available.

Returns :

the next byte of data, or -1 if the end of the file is reached.

Throws :

IOException - if an I/O error occurs.

```
1     while ((c = in.read()) != -1) {
2        System.out.print(c);
3      }
4      // signifie en fait:
5      c = in.read(); // lecture d'un octet
6           // susceptible de lever IOException => try/catch
7      while (c  != -1) { // tant que fin de fichier non atteinte
8        System.out.print(c); // affichage dans la console
9        c = in.read(); // lecture du caractère suivant
10     }
```

Limites → Nouvelles classes

- Des classes supplémentaires enrichissent les `FileStream` :
- Processus de décoration d'objets (cf LI314)
- Fonctions de lecture

```
1  DataInputStream istream = null;
2  try {
3      istream = new DataInputStream(
4        new FileInputStream(new File("toto.dat")));
5
6      System.out.println(istream.readChar());
7      System.out.println(istream.readDouble());
8      System.out.println(istream.readInt());
9      System.out.println(istream.readChar());
10     // pas de fonctions pour les String...
11 } finally{
12     if(istream != null)
13         istream.close();
14 }
```

- ... Ces fonctions ont évidemment des fonctions symétriques pour l'écriture

Ecriture de fichiers (byte stream)

public FileOutputStream(String name) throws FileNotFoundException

Creates an output file stream to write to the file with the specified name.
Parameters :
name - the system-dependent filename
Throws :
FileNotFoundException - if the file exists but is a directory rather than a regular file, does not exist but cannot be created, or cannot be opened for any other reason

- La fonction est proche de celle d'ouverture en lecture... Avec une option supplémentaire : ajouter des choses dans un fichier...
- `public FileOutputStream(String name, boolean append) throws FileNotFoundException`

Ecriture de fichiers

- Exemple d'utilisation (Oracle Java Tutorials) :

```
1  FileInputStream in = null;
2  FileOutputStream out = null;
3
4  try {
5      in = new FileInputStream("xanadu.txt");
6      out = new FileOutputStream("outagain.txt");
7      int c = in.read();
8
9      while (c != -1) {
0          out.write(c);
1          c = in.read();
2      }
3  } finally {
4      if (in != null) {
5          in.close();
6      }
7      if (out != null) {
8          out.close();
9      }
0  }
```

- Que fait ce programme ?

Limites → Nouvelles classes

- 1 Octet = 1 char (en fonction du codage)... On voudrait lire des choses de plus haut niveaux
  - Entier/Double = 4 octets
  - String...
- Des classes supplémentaires enrichissent les `FileStream` :
  - `DataInputStream/DataOutputStream`

```
1  // Ecriture dans un fichier des types de base
2  DataOutputStream ostream = null;
3  try {
4      ostream = new DataOutputStream(
5        new FileOutputStream(
6          new File("toto.dat")));
7      ostream.writeChar('r');
8      ostream.writeDouble(2.5);
9      ostream.writeInt(6);
10     ostream.writeChars("toto");
11 } finally{
12     if(ostream != null)
13         ostream.close();
14 }
```

Ouverture en écriture...

Par défaut : remplacement des fichiers existants...

Gare aux catastrophes !

Il existe une option pour ajouter des choses dans un fichier **sans** écraser le contenu :

```
1 try {
2     ostream = new DataOutputStream(
3       new FileOutputStream(
4         new File("toto.dat"), £true£));
5                   // le boolean correspond à l'option append
6 ...
```

ASCII or not ASCII...

Dans les opérations précédentes, voici le fichier manipuler (ouvert avec emacs) :

```
^@r@^@^@^@^@^@^@^@^@F^@^@^@t^@o^@t^@o
```

Ce qui n'est pas très convivial...
En fait il y a des avantages et des inconvénients.

Gestion de l'ASCII

Distinction entre ASCII et données brutes

Il faut distinguer un fichier **ASCII** d'une ***donnée brute*** :

- On peut pas lire directement les int/double sauver par la méthode précédente...
- On peut tricher :

```
1 ostream.writeChars(((Double) 2.5).toString());
```

  $\Rightarrow$ lisible mais perte de précision

Problème de lecture des String

- Lecture caractère par caractère fastidieuse
- Problème des sauts de ligne (codage différent selon le système)

$\Rightarrow$ Choisir en fonction des situations (possibilité de mélanger)

ASCII or not ASCII...

ASCII

- Fichiers de Textes
- Entêtes des fichiers
- XML, HTML...
- Parfois pour quelques chiffres
  - lorsque la précision n'est pas importante
  - pour les fichiers excels...

not ASCII

- Fichiers de chiffres
  - Volume
  - Précision

Gestion en lecture de fichier ASCII

- Encore une nouvelle classe : `BufferedReader`...

`public String readLine() throws IOException`

Reads a line of text. A line is considered to be terminated by any one of a line feed or a carriage return.
Returns : A String containing the contents of the line, not including any line-termination characters, or null if the end of the stream has been reached
Throws : IOException - If an I/O error occurs

```
1  BufferedReader in = null;
2  try {
3      in = new BufferedReader( new InputStreamReader(
4          new FileInputStream( // décorations multiples
5            new File("xanadu.txt"))));
6
7      String buf = in.readLine();
8      while(buf != null){
9          System.out.println(buf);
10         buf = in.readLine();
11     }
12 }...
```

Stratégie de lecture des fichiers ASCII

Une fois la ligne lue, il est nécessaire de la traiter...

- Séparer les mots : `StringTokenizer`
  - Choix des séparateurs (espace, tabulation, virgule...), accès aux sous-chaines
- Conversion en Int, Double...
  - `public static Double valueOf(String s) throws NumberFormatException`
  - `public static Double parseDouble(String s) throws NumberFormatException`

Stratégie alternative

Idée

Combiner la lecture de fichier avec un Tokenizer... Ca donne un `Scanner`

```java
Scanner s = null;
try {
    s = new Scanner(
      new BufferedReader(
        new FileReader("xanadu.txt")));

    while (s.hasNext()) {
        System.out.println(s.next());
    }
} finally {
    if (s != null) {
        s.close();
    }
}
```

Par défaut, le séparateur est `",\s*"` mais on peut le changer...

```java
s.useDelimiter(",\\s*");
```

Ecriture de fichiers ASCII

BufferedReader $\Rightarrow$ BufferedWriter

```
1 BufferedWriter writer =
2       new BufferedWriter(
3             new FileWriter(
4                   new File("monFichierASCII.txt")));
5
6  writer.write("toto");
7  writer.close();
```

Fonctions utiles de formatage du texte

JAVA propose des outils de formatage du texte issu directement de la syntaxe C : c'est utile pour faire des affichages tabulés, ou pour améliorer la lisibilité

- Syntaxe usuelle (implicite) de conversion Double → String :

```
1 int i = 2;
2 System.out.println("i = "+i);
```

- Syntaxe explicite :

```
1 int i = 2;
2 System.out.println("i = "+((Integer) i).toString());
```

- Formatage (disponible dans la classe String, entre autre) :
  `static String format(String format, Object... args)`

Exemples de formatages

```java
double[] tab = new double[10];
for(int i=0; i<10; i++)
    tab[i] = Math.random()*1000;
for(int i=0; i<10; i++)
    System.out.println(tab[i]);
```

```
30,781400
453,926246
213,175566
18,565873
940,045852
...
```

```java
for(int i=0; i<10; i++)
    System.out.println(
      String.format("%12f", tab[i]));
```

```
  357,600960
  832,277086
   89,423706
  124,335349
 ...
```

```java
for(int i=0; i<10; i++)
    System.out.println(
      String.format("%10.3f", tab[i]));
```

```
   870,771
   735,952
    44,770
 ...
```

```java
for(int i=0; i<10; i++)
    System.out.println(
      String.format("%010.3f", tab[i]));
```

```
000825,579
000611,987
000013,986
 ...
```

Ca marche aussi avec les entiers (%d) et les String (%s)

Outils

Vincent Guigue
UPMC - LIP6

Tableaux, ArrayList

- Taille fixe & connue à l'avance :
  tableau classique : `int[]`, `double[]` ?
- Sinon : `ArrayList`, la même chose en plus flexible.
  Pas de taille déclarée
  Utiliser `add` pour ajouter, `get` pour accéder (cf Javadoc)
  Pour la syntaxe, il faut penser à déclarer ce que l'on va mettre dans l'`ArrayList` :

```
ArrayList<Vecteur> vec = new ArrayList<Vecteur>();
```

Enumération

Nous n'utiliserons que quelques types de terrain.
Bonne idée : créer une structure de données associée à ce nouveau type.
Mauvaise idée : utiliser un `int`.

```
1 public enum Terrain {
2     Route, Herbe, Eau, Obstacle, BandeRouge,
3     BandeBlanche, StartPoint, EndLine, Boue;
4 }
```

- utiliser un `int` est une source de bug :
  - introduction/gestion de codes inexistants
  - code moins lisible (test d'égalité...)
  - mélange de types
- Perte de mémoire conséquente

Enumération : enrichissements

Sauvegarde, Chargement, Conversion...

```
1  public enum Terrain implements Serializable {
2     Route, Herbe, Eau, Obstacle, BandeRouge, BandeBlanche,
3       StartPoint, EndLine, Boue;
4
5     public static char[] conversion =
6             {'.', 'g', 'b', 'o', 'r', 'w', '*', '!', 'm'};
7
8     public static Color[] convColor = {Color.gray, Color.green,
9         Color.blue, Color.black, Color.red, Color.white,
10        Color.cyan, Color.cyan, new Color(200, 150, 128)};
11 }
```

Enumération : enrichissements (2)

Propriétés :

- Chaque Terrain est associé à un chiffre :

```
1 public static char charFromTerrain(Terrain c) {
2    return Terrain.conversion[c.ordinal()];
3 }
```

- On peut récupérer toutes les valeurs pour les faire défiler

```
1 public static Terrain terrainFromChar(char c)
2          throws TerrainException{
3       Terrain[] values = Terrain.values();
4       for(int i=0; i<values.length; i++)
5          if(c == Terrain.conversion[i])
6             return values[i];
7
8    throw new TerrainException("Terrain inconnu : "+c);
9 }
```

- S'utilise comme un type de base pour les `switch` et les ==

```
1 Terrain t = Terrain.Herbe
2 if( t == Terrain.Route)
3 ...
```

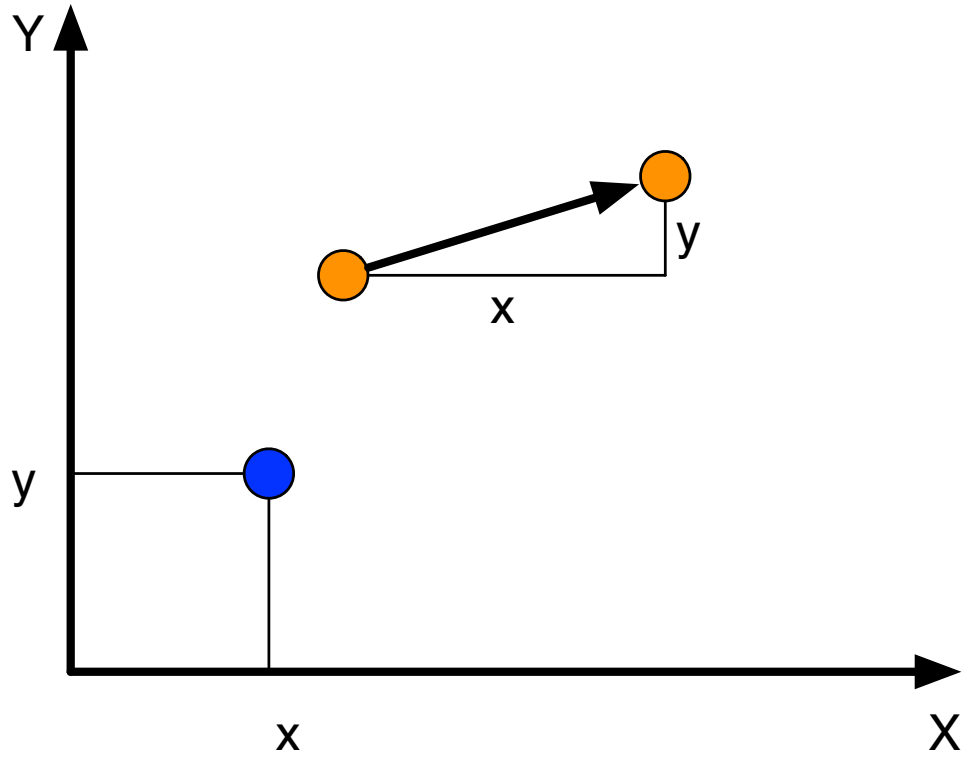
Gestion des images

- Image : classe abstraite
- Nous utiliserons la classe `BufferedImage`
- Chargement :
  `BufferedImage im = ImageIO.read(new File(filename))`
- Savegarde :
  `ImageIO.write(im, "png", outputfilename);`
- Creation d'une image vide :
  `BufferedImage im = new BufferedImage(ncol,nligne, BufferedImage.TYPE_INT_ARGB);`
- Manipulation :
  `im.setRGB(i,j, color); int color = im.getRGB(i,j);`
- Création d'un pixel : `Color c = new Color(200, 150, 128)` (RGB entre 0 et 255)

Géométrie dans l'espace 2D

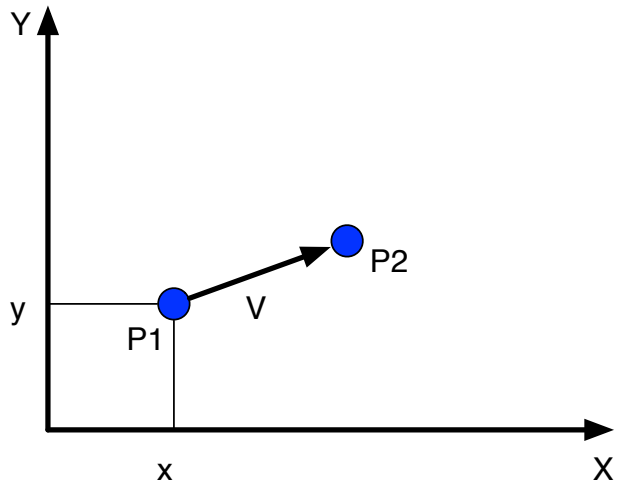
Vincent Guigue
UPMC - LIP6

Repérage dans l'espace 2D



- A l'aide de la classe `Point`
  - Attributs `double` x et y
- La même classe nous permet de gérer les points et les vecteurs

Gestion des déplacements

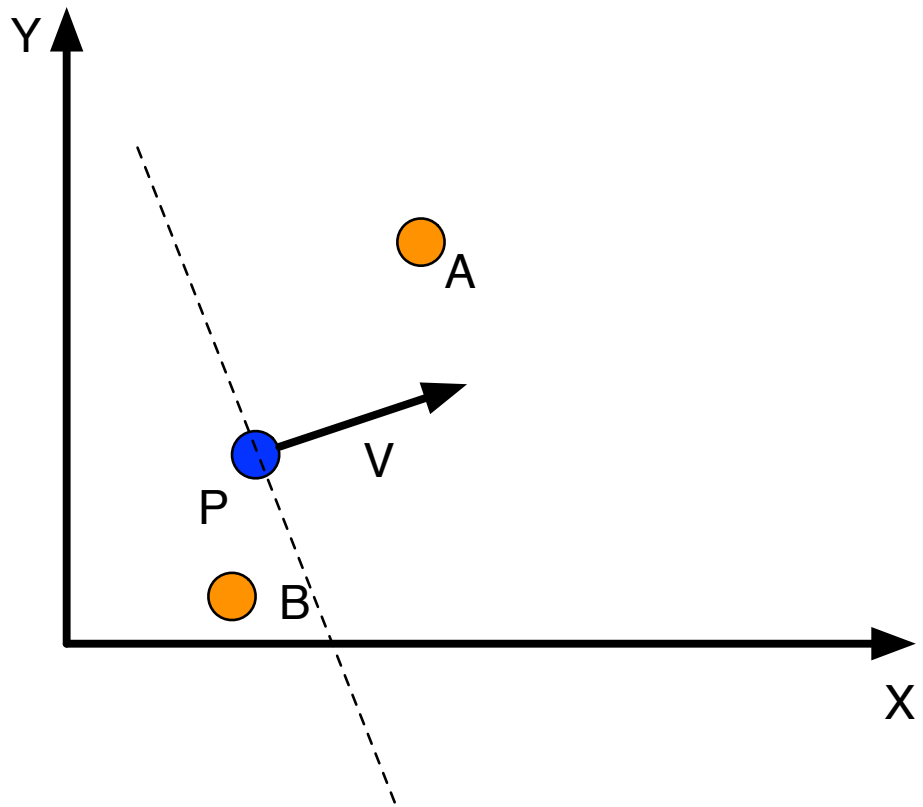


- Déplacements discrets (P : position, V : vitesse) :

$$P_2 = P_1 + V, \quad \begin{cases} P_2.x = P_1.x + V.x \\ P_2.y = P_1.y + V.y \end{cases}$$

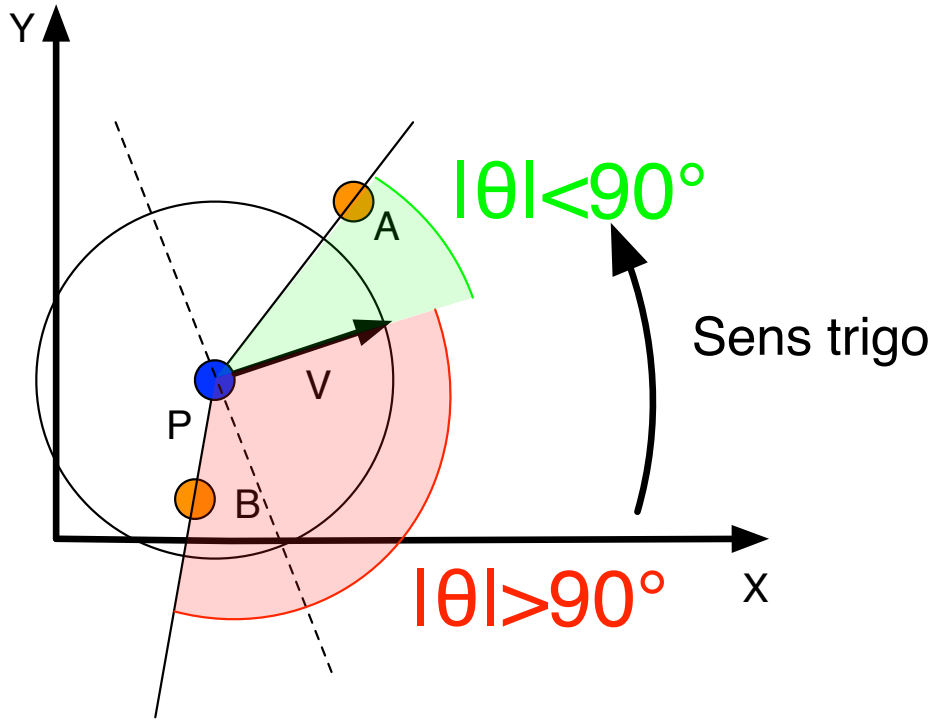
- En physique : $\vec{v} = \dot{\vec{x}} \approx \frac{\vec{x}_{t+1} - \vec{x}_t}{\delta_t}$, pour nous : $\delta_t = 1$ (unité arbitraire)
- En utilisant des vecteurs suffisamment petits : modélisation d'un déplacement continu

Se repérer dans l'espace



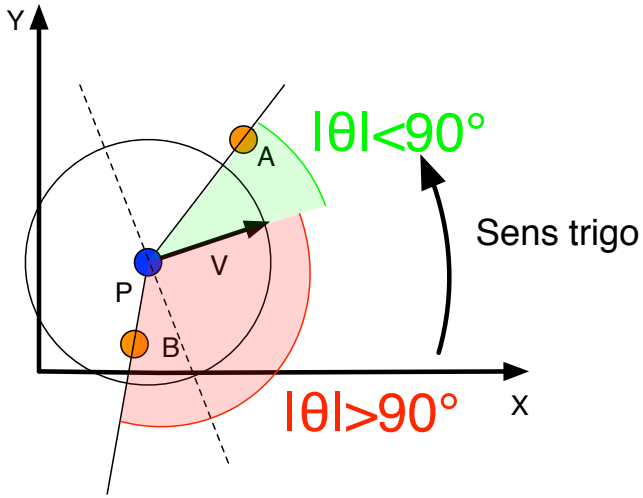
- Un objet est caractérisé par sa position P et sa vitesse V
- Qu'est ce qui est devant, qu'est ce qui est derrière l'objet ?
- Qu'est ce qui est à droite, qu'est ce qui est à gauche ?

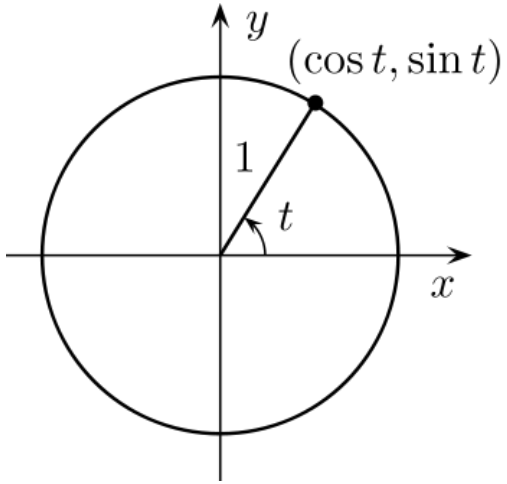
Solution complète : calcul d'angle



- Devant/Derrière : calculer les angles $\widehat{V, PA}$ et $\widehat{V, PB}$
- Gauche/Droite : calculer les angles $\widehat{V, PA}$ et $\widehat{V, PB}$ avec le signe !

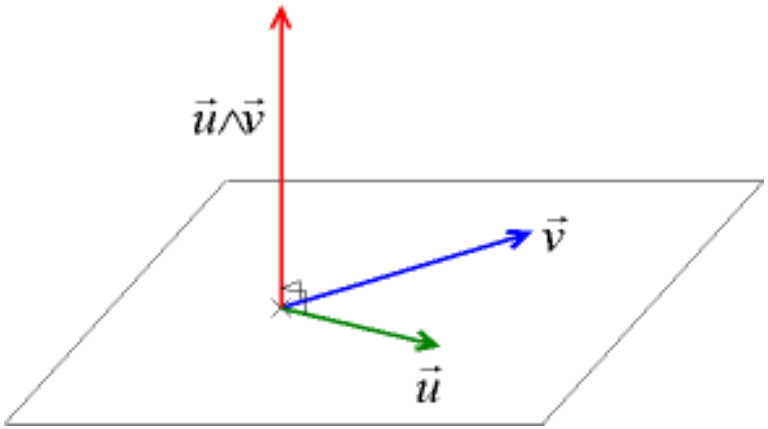
Détail du calcul d'angle

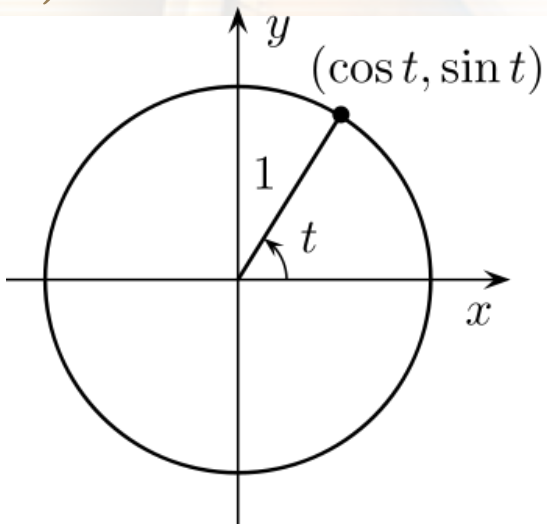




- Produit scalaire
  $U \cdot V = \langle U, V \rangle = \|U\| \|V\| \cos(\widehat{U, V}) = U_x V_x + U_y V_y$
- Corollaire :
$$\widehat{U, V} = \text{acos}\left(\frac{U \cdot V}{\|U\| \|V\|}\right)$$
- Attention : $\widehat{U, V} \in [0,\ \pi]$, pas de signe ici...
- Le signe de $U \cdot V$ permet de résoudre le pb devant/derrière

Détail du calcul d'angle (suite)





- Produit vectoriel : $\|U \wedge V\| = \|U\|\|V\| \sin(\widehat{U, V})$
- Corollaire :

$$\widehat{U, V} = \text{asin}\left(\frac{U \wedge V}{\|U\|\|V\|}\right) \qquad U \wedge V = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

- L'étude de $u_1 v_2 - u_2 v_1$ permet de connaitre le signe de l'angle...

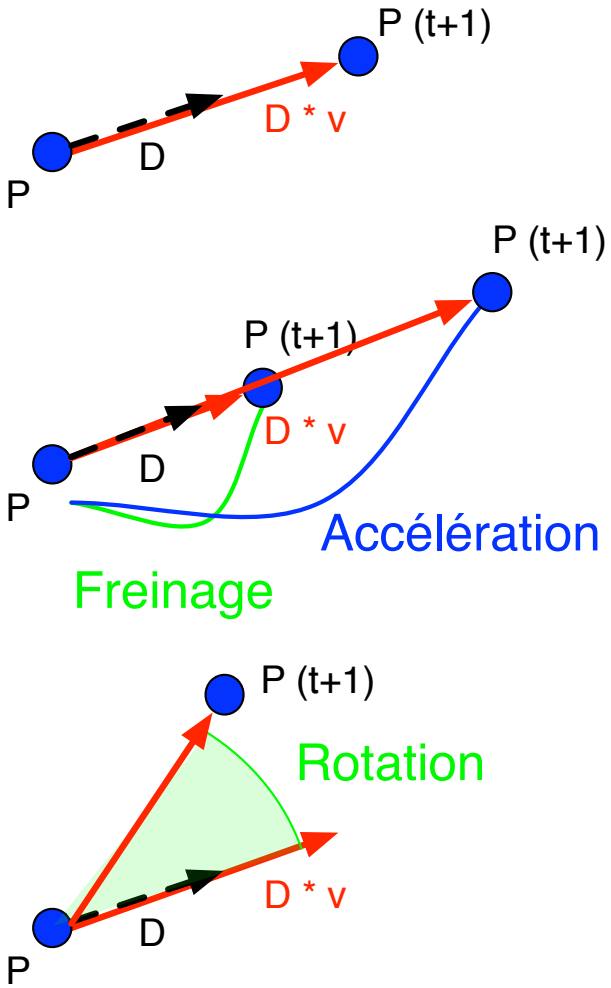
Pré-définition de la voiture

La voiture sera définie géométriquement par :

- Sa position : P
- Sa direction (vecteur unitaire) : D
  Conservation de la direction même à l'arrêt
- Sa vitesse (scalaire) : $v \in [0,\ vmax]$

La commande de la voiture se fera sur 2 axes :

- Accélération/Freinage : modification de v
- Commande de direction : modification de D



Rotation sur un vecteur

Définition de la rotation d'un vecteur :

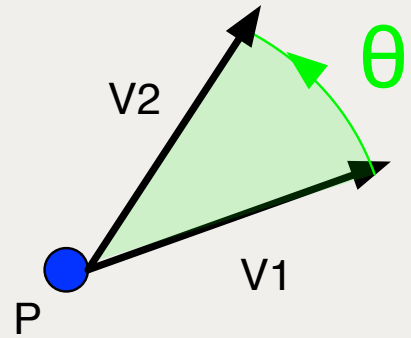
Soit un vecteur $V = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, la rotation d'angle θ est obtenu en utilisant la matrice de rotation :

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad V' = RV$$

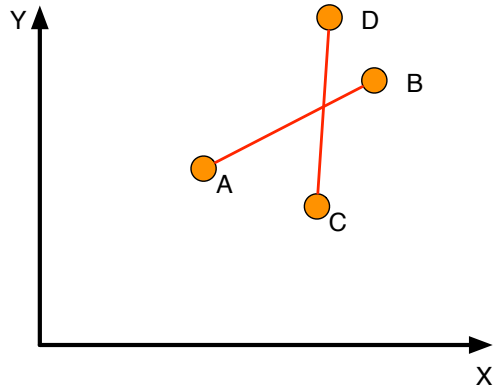
C'est à dire en utilisant la mise à jour :

- $v'_x = v_x \cos(\theta) - v_y \sin(\theta)$
- $v'_y = v_x \sin(\theta) + v_y \cos(\theta)$

ATTENTION à ne pas modifier v_x avant la seconde ligne



Approfondissement : collisions



(Problématique de base dans les cartes graphiques/moteur physique)

Comment détecter la collision de deux vecteurs ?

- Si C et D sont à gauche et à droite de AB
- ET que A et B sont à gauche et à droite de CD

Résultat :

S'ils sont de part et d'autre, l'un des produit vectoriel est positif, l'autre négatif...

$$(AB \wedge AC)(AB \wedge AD) < 0 \text{ ET } (CD \wedge CA)(CD \wedge CB) < 0$$

Résumé des méthodes de la classe Vecteur

- Addition, soustraction
  - génération d'un nouveau vecteur
  - auto-opérateur
- produit scalaire
- produit vectoriel (composante en z)
- multiplication par un scalaire
- rotation
- calcul de la norme
- clonage
- test d'égalité (structurelle)